Haplotype phasing from long reads with ASP : a flexible optimization approach

Clara Delahaye

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Clara DELAHAYE

Haplotype phasing from long reads with ASP : a flexible optimization approach

Thèse présentée et soutenue à Rennes, le 15 décembre 2022
Unité de recherche : Équipe GenScale, Univ Rennes, Inria, CNRS, IRISA

Rapporteurs avant soutenance :

Gunnar KLAU	Professeur, Heinrich-Heine-Universität, Düsseldorf
Jean-François FLOT	Professeur, Université Libre de Bruxelles, Bruxelles

Composition du Jury :

Président :	Sébastien FERRÉ	Professeur, Univ Rennes, IRISA, Rennes
Examinateurs :	Gunnar KLAU	Professeur, Heinrich-Heine-Universität, Düsseldorf
	Jean-François FLOT	Professeur, Université Libre de Bruxelles, Bruxelles
	Anne FRIEDRICH	Maîtresse de conférences, GMGM, Univ Strasbourg
	Sébastien FERRÉ	Professeur, Univ Rennes, IRISA, Rennes
Dir. de thèse :	Jacques NICOLAS	Directeur de recherche, Univ Rennes, INRIA, Rennes
Co-dir. de thèse :	Dominique LAVENIER	Directeur de recherche, Univ Rennes, INRIA, Rennes

Acknowledgement

Je tiens tout d'abord à remercier Gunnar Klau, Jean-François Flot, Anne Friedrich et Sébastien Ferré pour avoir accepté leurs rôles de rapporteurs et de jury lors de ma soutenance. C'est une belle [1] façon de conclure cette thèse que de l'éprouver face à des spécialistes du domaine.

Merci également à mes encadrants, Jacques Nicolas et Dominique Lavenier. Tout particulièrement, merci à Jacques, pour la patience et la persévérance dont tu as fait preuve pour me $\lim_{n\to\infty}(re)^n$-expliquer les choses, notamment quand il s'agit de notions d'informatique et d'algorithmique que je n'avais jamais eu l'occasion d'aborder auparavant. Pour m'avoir fait découvrir l'ASP. C'est vrai que c'est assez rigolo, voire puissant. Même si je suis consciente de n'avoir effleuré qu'un pouième des capacités de ce langage. Et enfin, pour le nombre innombrable de nombreuses corrections et fautes. Avec tous les *the* que tu as apporté dans ce manuscrit, on aurait pu faire une sacré omelette. Cela dit, si on prend le temps de les compter, il n'y en avait que : $1 + 2 + 3 + ...$ Soit seulement $\frac{-1}{12}$. C'est relativement raisonnable, au final.

Je tiens à remercier chaleureusement l'ensemble de Symbiose, dans laquelle j'ai eu la chance de travailler depuis pas mal de temps déjà : depuis mon entrée dans le monde de la bio-info, et plus précisément dans l'équipe Genscale depuis mon stage de M2. Ça a été un réel plaisir de travailler en votre compagnie, et j'espère pouvoir retrouver un cadre de travail avec une aussi bonne entente dans le futur.
Merci en particulier à Marie, de nous protéger avec ta bonne humeur de ce qui nous effraie le plus : la bureaucratie.
Merci à Émeline pour avoir pris le temps de me faire découvrir le processus de séquençage (même si cette manip' n'a pas vraiment fonctionné comme prévu), et pour tes retours sur les aspects biologiques.
Merci enfin à Garance, pour le soutien sur ces dernières semaines, les coups de mains, les petites discussions fort sympathiques [2]. Tu n'es arrivée que tardivement dans le bureau, et c'est bien dommage !

1. Mi-stressante, mi-belle. Et re mi-stressante derrière.
2. et les cookies !!

Je remercie aussi *Tortinator* (également connu sous le pseudonyme Torsten Schaub) pour avoir mis à disposition un cours et des slides très détaillées sur l'ASP, qui m'ont aidé à mieux appréhender ce langage.

Merci à LaTeXd'exister. Je n'ose imaginer l'horreur que ça aurait été de rédiger ce presse-papier géant avec ton simili-équivalent-pas-libre-et-pas-pratique. Bien que tu restes à la ramasse quand il s'agit de faire des présentations, et que ton nom oblige systématiquement à faire une petit gymnastique mentale pour sélectionner soigneusement les mots-clés qu'on utilise dans les moteurs de recherche quand on ne comprend pas comment utiliser une de tes fonctionnalités.
Merci à Lucas Bourneuf pour m'avoir incité à perdre plusieurs mois à apprendre à taper en bépo [3]. C'est super cool, je ne m'en passe plus [4]. Mais je soupçonne la présence d'une once de vilenie de la part de ses concepteurs, qui ont placé les touches Z et W côte à côte [5].

Merci et bravo à Olivier et Téo pour leurs contributions à Sciences en Cour[t]s, à la fois pour la réalisation du film et l'organisation du festival, et à l'association Nicomaque. Merci également à ceux qui ont repris le flambeau !

Un grand merci aux personnes diverses et avariées (qui se reconnaîtront sans peine, je n'en doute pas), pour toutes les discussions sur tout et n'importe quoi, les chiens, les jeux de mots foireux, les blagues vaseuses, les photos de chiens, les drinkscords, les balades canines, les coup de main, les randos, les toutous, ...

Merci également à ma famille pour leur soutien. Vous qui étiez même venus à Sciences en Cour[t]s dans l'espoir d'en savoir un peu plus sur le sujet de ma thèse, et qui n'avez obtenu qu'un dessin animé étrange avec un chat qui hurle... Courage, cette fois au moins il y aura un pot à la fin. :)

Viimeisenä, mutta ei vähäisempänä...
Merci à toi, Camille, pour ton soutien inconditionnel, pour m'avoir autant facilité cette fin de thèse. C'est ton tour maintenant, et je compte bien te rendre la pareille !

Kiitos suklaasta.

3. C'est quoi ce truc ?
4. Vous aussi, perdez du temps en vous arrachant les cheveux !
5. Je vous laisse regarder l'effet d'un Ctrl + la seconde touche en pensant appuyer sur la première...

List of Figures

List of Tables

Table of Contents

Résumé de la thèse - French summary of the thesis

0.1. Introduction

0.1.1 Structure d'un génome : chromosomes et haplotypes

L'information génétique d'un individu est contenue dans son *génome*, sous la forme de macromolécules d'ADN (acide désoxyribonucléique), et contient toutes les informations nécessaires au développement d'un organisme, sa croissance et sa survie. À une échelle plus fine, c'est également ce qui va déterminer le fonctionnement des cellules.

La taille d'un génome est variable selon les organismes : environ 3,4 milliards de paires de bases pour l'humain, plusieurs millions pour les bactéries [6]. Un génome peut être représenté sous la forme d'une chaine de caractères sur l'alphabet $\sum = \{A, C, G, T\}$, en référence aux noms des quatre acides nucléiques (aussi appelé *paires de bases*) qui composent l'ADN : **a**dénine, **c**ytosine, **g**uanine et **t**hymine (Figure 1). La plupart du temps, l'ADN se présente sous la forme d'une double hélice, chacune constituée d'un enchaînement de nucléotides, et est alors appelé "ADN double brin". Bien qu'il existe de l'ADN simple brin (par exemple chez certains virus, ou lors du processus de réplication de l'ADN), c'est sous la forme double brin que nous considérons l'ADN dans ce manuscrit.

Une macromolécule d'ADN peut se présenter sous une forme condensée appelée *chromosome*. Certains organismes ne présentent chaque chromosome qu'en un seul exemplaire, et sont dits *monoploïdes*. À l'inverse, beaucoup d'êtres vivants possèdent p (p fixé pour un organisme donné) exemplaires quasiment identiques de chaque chromosome : ils sont dits *diploïdes* si $p = 2$, et *polyploïdes* si $p > 1$. Dans de tels cas, les exemplaires d'un même chromosome sont appelés des *chromosomes homologues*, on parle également d'*haplotypes*.

6. Cependant, la taille d'un génome n'est pas corrélée avec la taille d'un individu ni sa complexité : le génome de *Polychaos dubium*, une amibe microscopique, a une taille d'environ 675 milliard de paires de bases.

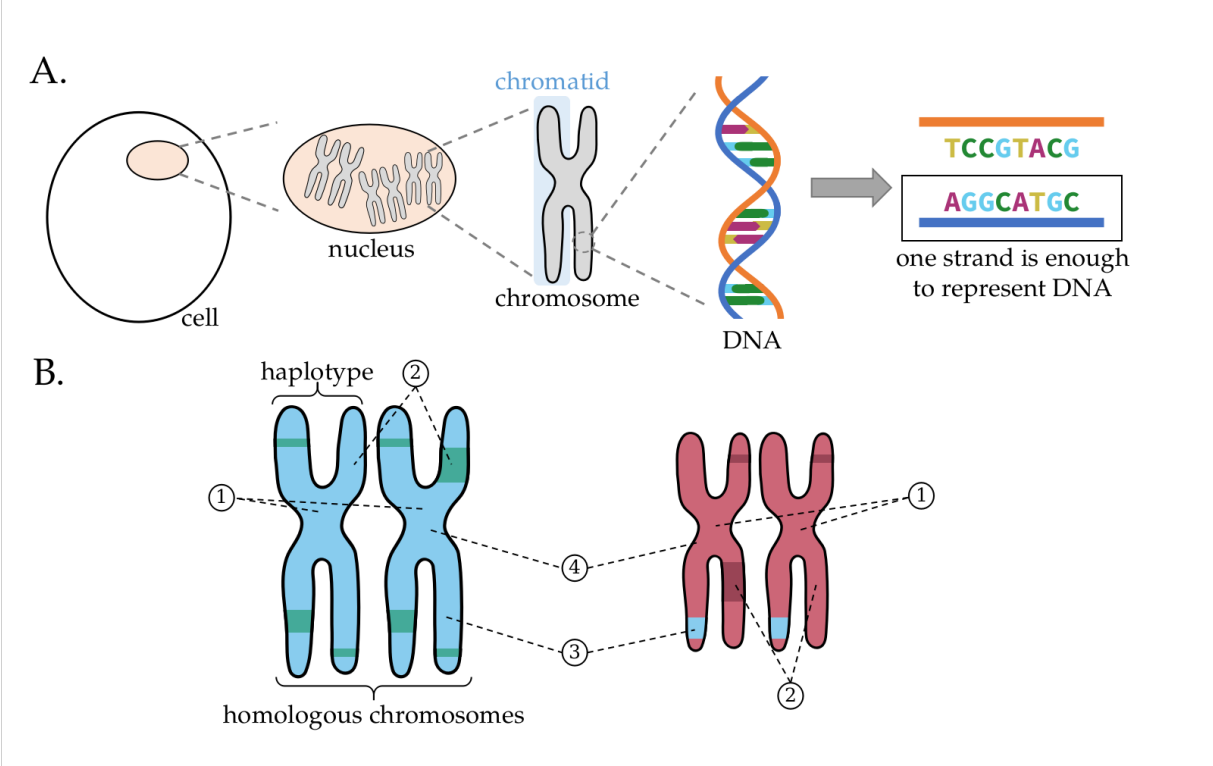


Figure 1 – **ADN, chromosomes et haplotypes. A)** Dans les cellules (ici une eucaryote, mais le principe serait le même pour une cellule procaryote, mis à part qu'il n'y aurait pas de noyau), l'ADN est contenu au sein du noyau, sous la forme de chromosomes. L'ADN est une molécule faite de deux brins qui contiennent une information complémentaire : on peut donc représenter une macromolécule d'ADN par une simple chaine de caractères. **B)** Deux paires d'haplotypes. Les haplotypes d'un même chromosome sont globalement très similaires (1), mais peuvent présenter des différences locales (2). À l'inverse, deux chromosomes différents peuvent être localement identiques (3) mais sont globalement différents (4).

L'étude des haplotypes a plusieurs champs d'application. Elle est utile en médecine où elle peut servir à étudier les causes de maladies génétiques [Squitieri *et al.* 1994; Warby *et al.* 2011; Kerem *et al.* 1989], en agronomie dans la sélection de caractères des cultures [Abed and Belzile 2019], ou dans l'étude des populations [Lawson *et al.* 2012; Leitwein *et al.* 2020; Yang *et al.* 2017].

0.1.2 Représenter un génome à partir de données longues lectures

Il existe différentes technologies de séquençage, qui permettent d'accéder à l'information contenue dans des macromolécules d'ADN.

Récemment, le développement des technologies dites de troisième génération (en particulier PacBio et Oxford Nanopore Technologies, ONT) a permis d'obtenir des longues lectures, et donc de séquencer des grandes régions du génome, souvent de plusieurs dizaines de milliers de nucléotides à la fois (les technologies précédentes ne permettaient de générer des lectures que de quelques centaines de nucléotides). Cependant, ces technologies sont associées à des taux d'erreurs assez élevés [Dohm *et al.* 2020], ce qui rend difficile la distinction entre les véritables variations du génome (par exemple, localement entre deux haplotypes) et des erreurs de séquençage.

Il est possible de reconstruire un génome à partir des lectures séquencées. Plusieurs approches existent, notamment basées sur le paradigme OLC (*Overlap*, *Layout*, *Consensus*) ou des graphes de de Bruijn. Cependant, les génomes assemblés souffrent de plusieurs problèmes. Un, en particulier, nous intéresse : la quasi totalité des génomes assemblés ne fournissent qu'une seule séquence par chromosome, et ne distinguent pas les haplotypes. La séquence fournie n'est d'ailleurs souvent pas un des haplotypes, mais une chimère entre les différents haplotypes du chromosome. Par ailleurs, les algorithmes d'assemblage fournissent une unique séquence assemblée en guise de solution. Or, assembler un même jeu de données avec différents algorithmes, conduira probablement à des solutions différentes, du fait d'heuristiques, de méthodologies et de paramètres différents. Bien qu'il n'y ait qu'une seule réalité biologique, il semble raisonnable de penser que nous ne l'atteindront sans doute pas exactement, et qu'il serait plus pertinent de proposer un ensemble de solutions qui pourraient résoudre le problème donné, à savoir retrouver les séquences des haplotypes.

0.1.3 Objectifs de la thèse

L'objectif principal de cette thèse est de reconstruire les séquences d'haplotypes d'un organisme donné à partir de données de séquençage longues lectures (PacBio ou Nanopore). Pour cela, nous nous plaçons dans un cadre de travail particulier :

- cette méthode de phasage a pour cible des organismes diploïdes et polyploïdes (en pratique, on se limite à une ploïdie de 12) ;

- nous nous limitons à l'utilisation des données longues lectures, sans avoir recours à d'autres données de séquençage ni à un génome de référence.

Plusieurs difficultés sont à prendre en compte pour le problème de phasage d'haplotypes à partir de longues lectures.

Les données de séquençage longues lectures ont un taux d'erreurs relativement élevé. Une pratique courante pour les corriger est d'utiliser leur redondance : une même portion du génome est représentée par plusieurs lectures. On s'attend alors à ce que les erreurs de séquençage, a priori aléatoires, apparaissent moins fréquemment que la véritable séquence d'ADN. Il est donc possible de se baser sur des méthodes de consensus qui retiennent, pour chaque position du génome séquencé, le nucléotide majoritaire dans les lectures.
Cela peut cependant se révéler problématique pour la détection de variants rares, qui apparaîtront donc peu fréquemment dans les lectures, et pourraient être confondus avec des erreurs de séquençage. Dans ce cas, l'information de tels variants serait perdue, et l'assemblage résultant constituerait une représentation moins précise de la réalité.

Dans le cas d'un génome polyploïde, les haplotypes ne sont pas tous équidistants : chaque paire d'haplotypes n'aura pas nécessairement le même nombre de différences. Par exemple, dans le cas d'un organisme triploïde issu de la fécondation entre un gamète diploïde et un monoploïde, deux de ses haplotypes (A et B, provenant du gamète diploïde) seront plus proches entre eux que du troisième haplotype (C, provenant du gamète monoploïde). Il serait alors facile de produire un phasage erroné en recréant seulement deux haplotypes : un représentant une fusion entre A et B, et second étant le C (voir la Figure 3.10 pour un exemple sur un organisme tétraploïde).
Par ailleurs, la distance entre haplotypes peut également varier selon les régions du génome considérées : deux haplotypes A et B peuvent êtres proches dans une région donnée du génome, tandis que dans une autre région l'haplotype B sera plus proche du C et distant du A.
De plus, les haplotypes, bien que globalement différents, peuvent êtres localement identiques dans certaines portions du génome. La ploïdie locale est alors plus faible que celle attendue, ce qui peut provoquer des erreurs de phasage.

Pour finir, le problème de phasage d'haplotypes est un problème hautement combinatoire. Cela revient à un problème de partitionnement : un ensemble de N lectures doit être divisé en p groupes distincts. L'espace des partitions d'un ensemble de N éléments

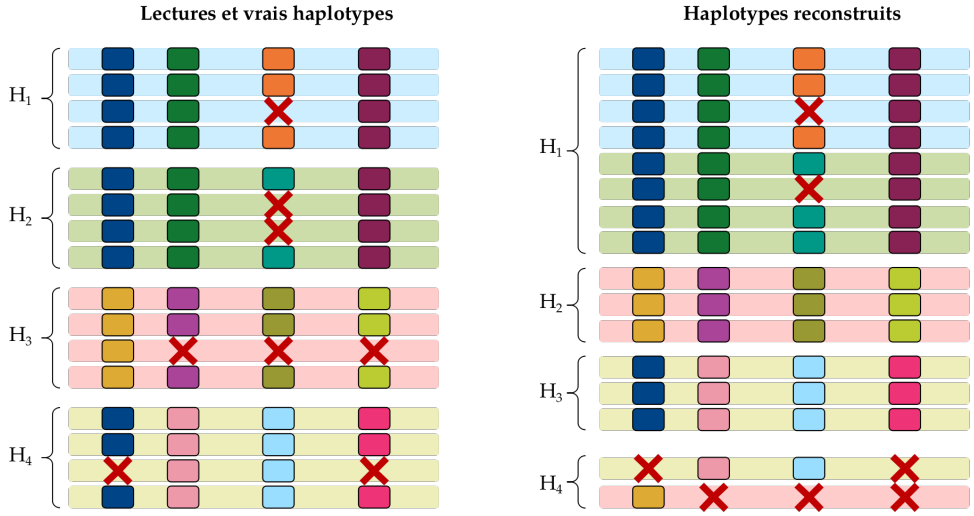


Figure 2 – **Fusion d'haplotypes similaires.** Considérons un ensemble de lectures, chacune provenant d'un haplotype donné (couleur d'arrière plan des lectures), et un ensemble de quatre loci. La couverture est supposée être égale entre les haplotypes. Les lectures contiennent différents variants (carrés colorés) et peuvent comporter des erreurs de séquençage (croix rouges). Le calcul des haplotypes en forçant une 4-partition sur ces lectures peut résulter en la création d'haplotypes chimériques : les haplotypes H_1 et H_2 sont assez similaires et peuvent être fusionnés en une seule partie, tandis que les lectures erronées sont rassemblées dans un haplotype spécial. De plus, la couverture obtenue n'est plus uniforme pour les haplotypes reconstruits. La couverture attendue est d'environ quatre lectures par haplotype. Elle est presque atteinte pour les haplotypes H_2 et H_3, mais H_1 a deux fois plus de lectures, et H_4 en a seulement la moitié.

en p parties est un nombre de Stirling de second ordre :

$$S(N,p) = \frac{1}{p!}\sum_{i=0}^{p}(-1)^{p-i}\binom{p}{i}i^N$$

pour lequel une borne supérieure a été définie dans [Rennie and Dobson 1969]:

$$S(N,p) \leq \frac{1}{2}\binom{N}{p}p^{N-p}, \text{ pour } n \geq 2 \text{ et } 1 \leq p \leq N-1$$

Ici, la valeur de p n'a pas une grande influence sur la combinatoire, car en pratique p ne prendra pas de grandes valeurs. Cependant, l'influence du nombre de lectures N est exponentielle, et N prend des valeurs de l'ordre de grandeur de 10^5 voire 10^6. Une approche naïve d'énumération des solutions possibles est donc infaisable.

Il ne s'agit là que d'une description naïve du problème, donnant une idée de la combinatoire associée : en pratique, les méthodes de phasage d'haplotypes exploitent des approches adaptées pour réduire la complexité, ou bien utilisent des heuristiques.

Un de nos objectifs dans l'élaboration de notre méthode de phasage, est de ne pas fournir une unique solution, mais un résultat basé sur l'ensemble des solutions possibles. Cela pourra notamment servir de support dans l'estimation de la robustesse du résultat proposé, et permettra d'ouvrir la voie à une recherche interactive d'une solution satisfaisante.

0.2. Contribution - profil d'erreur des données de séquençage Nanopore

Les données de séquençage Nanopore ont un taux d'erreurs élevé, et beaucoup de recherches ont été réalisées dans l'optique de corriger ces erreurs. Cependant, peu d'articles se sont intéressés au profil d'erreur sous jacent. En parallèle, ONT ne fournit pas (ou rarement) de publications concernant leurs benchmarks ni les détails de leurs logiciels. Les informations sont souvent annoncées lors de conférences annuelles, et là encore, il y a peu de communication sur le profil d'erreur de séquençage.

Pour pallier ce manque dans la littérature, nous avons tenté de décrire les différents types d'erreurs présents dans les données Nanopore [Delahaye and Nicolas 2021], afin de fournir une base pour les prochaines méthodes de correction de ces données. Par ailleurs, les données Nanopore devenant rapidement obsolètes (du fait de fréquentes mises à jour des algorithmes de basecalling et des chimies), nous avons développé un outil, SeqFaiLR[7] permettant de réaliser ces analyses sur de futurs jeux de données.

Nous nous sommes pour cela basés sur un jeu de données comprenant des données bactériennes et humaines. Nous nous sommes intéressés à différents aspects des données et erreurs : le taux de GC des lectures, leurs scores qualité, les erreurs systématiques et celles associées à des régions de faible complexité, ainsi qu'une étude en parallèle du signal électrique brut associé.

7. `https://github.com/cdelahaye/SeqFaiLR`)

0.3. Élaboration d'une méthode de phasage d'haplotypes à partir de lectures longues

Nous avons divisé notre méthode de phasage d'haplotypes en plusieurs étapes (Figure 4.7). À chaque étape du processus, nous nous permettons de ne pas prendre de décision (c'est-à-dire de phaser une lecture) s'il apparaît qu'il n'existe pas d'informations suffisantes sur lesquelles s'appuyer.

Construire une pseudo-référence. Nous proposons une méthode *de novo*, ne s'appuyant pas sur un génome de référence, mais sur un sous ensemble de lectures les plus longues (par un paramètre de longueur minimale). Ces lectures sont utilisées en tant que graines pour délimiter des instances de sous-problèmes associées à des régions comparables dans chaque haplotype. La subdivision en problèmes indépendants permet de réduire la complexité associée au problème de phasage d'haplotypes. Pour ce faire, les lectures-graines sont alignées entre elles, afin de détecter des relations de similarité entre elles. Cette opération est d'une complexité quadratique et nous devons donc nous assurer que l'ensemble des lectures n'est pas trop grand. Ensuite, un graphe de similarité est construit à partir des résultats de l'alignement. Ses composantes connexes - représentant des groupes de lectures graines - sont identifiées. Au sein de chaque groupe, les graines sont phasées sur la base de leur similarité, à condition que des informations suffisantes soient disponibles. Cela revient à partitionner les lectures en p parties, en s'appuyant sur des contraintes strictes pour réduire l'espace de recherche. Ces résultats servent par la suite à guider le phasage global.

Phaser les lectures restantes. Les lectures restantes sont affectées à différents groupes de graines, d'après leurs alignements (limités aux meilleurs pour réduire les ressources de calcul). Elles sont ensuite phasées en accord avec les résultats de phasage des graines, en exploitant à la fois la similarité et la cohérence avec les graines.

Construction du phasage global. Une fois que toutes les lectures ont été phasées, le résultat global est produit en reliant les résultats de chaque groupe de graines.

Nous proposons également une représentation des haplotigs (contigs d'haplotypes) reconstruits via un graphe basé sur les k-mers, mettant en évidence les régions similaires et les régions spécifiques de chaque haplotype.

Nous avons évalué notre méthode en l'appliquant à des données PacBio HiFi d'un organisme diploïde, *Adineta vaga*, qui présente un génome complexe, et pour qui un

génome de référence phasé était disponible. Nous sommes également en train d'appliquer notre méthode sur un jeu de données polyploïde de pomme de terre tétraploïde, séquencé en lectures ONT.

0.4. Conclusion et perspectives

Dans cette thèse, nous définissons les difficultés associées au problème de phasage d'haplotypes à partir de lectures longues uniquement. Nous posons les rudiments d'une méthode de phasage basée sur l'exploration et l'exploitation d'un ensemble de solutions optimales. Plusieurs aspects seront à prendre en compte dans la suite de son élaboration, notamment :

- réduire le temps de calcul associé aux étapes d'alignement, en utilisant à la place des recherches de chevauchements, moins précises mais plus rapides ;
- permettre l'intégration de données supplémentaires lorsqu'elles sont disponibles (par exemple des données de génotypage), qui permettraient de restreindre l'espace des solutions possibles et de fournir des résultats plus précis ;
- mettre en place une stratégie d'interaction avec l'utilisateur et de flexibilité de la recherche, en facilitant la formulation de contraintes et d'optimisations personnalisées ;
- poursuivre les recherches sur la création et la représentation d'un graphe d'haplotigs, permettant de mettre en évidence leurs points communs et leurs différences.

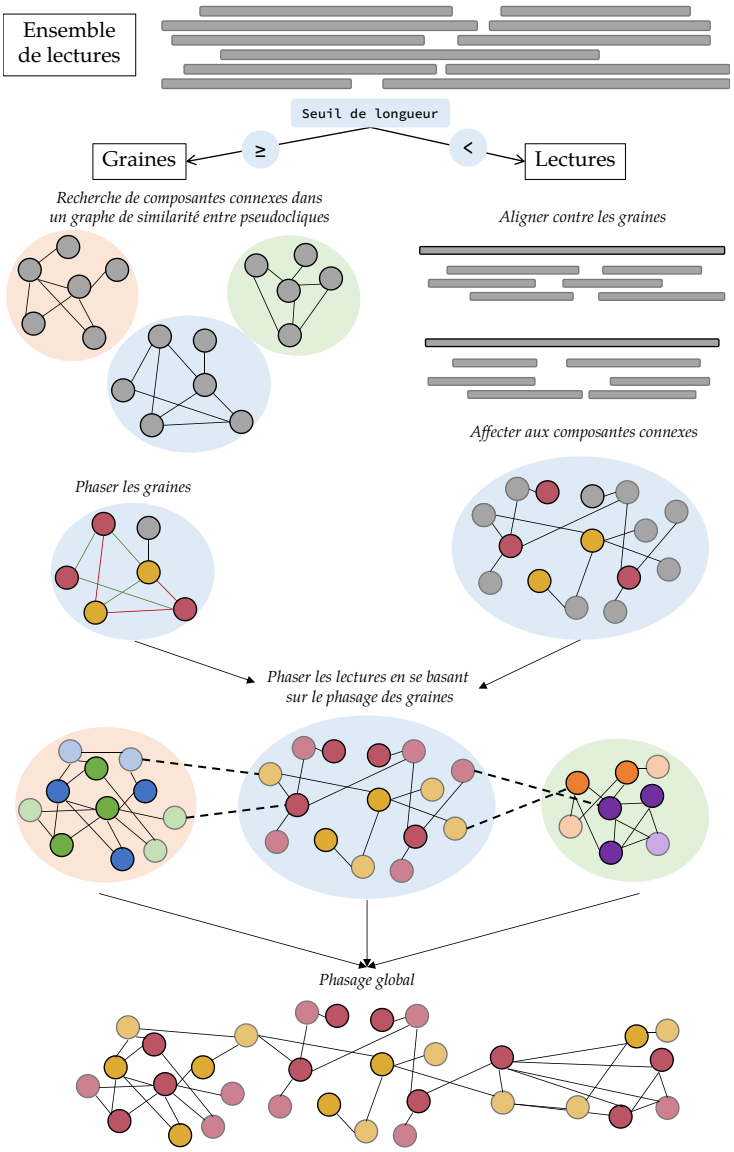


Figure 3 – **Vue d'ensemble de la méthode de phasage.** À partir d'un ensemble de lectures, nous sélectionnons les plus longues qui seront utilisées comme graines. Elles sont ensuite alignées les unes contre les autres afin de créer un graphe de similarité, dans lequel nous recherchons des composantes connexes, représentant des groupes de graines hautement similaires (susceptibles de provenir d'une même région d'un haplotype). Au sein de chaque composante connexe, les graines sont phasées fonction de leurs relations de similarité. Les lectures restantes sont ensuite affectées aux composantes connexes, et phasées sur la base de leur similarité avec les graines et de la cohérence du phasage. Enfin, les haplotigs peuvent être construits à partir des résultats de phasage locaux des composantes connexes.

Chapter 1

Overview: building haplotype aware reference genomes from long reads

Contents

Preamble: This chapter aims at giving an overview of the characteristics and difficulties of the haplotype phasing problem addressed in this thesis. The points mentioned here will be further developed in the following chapters. We start by introducing the basic genomic notions that will be used in this document. Then, by comparing the biological structure of genomes with their usual representation, we highlight the limitations of the latter, justifying the need for reference genomes integrating haplotype information. We end by stating the challenging points as well as the complexity of the problem.

1.1 Why should reference genomes incorporate haplotype information?

1.1.1 Low level representation: genomes as DNA macromolecules

An organism can be defined by its genome, *i.e.* all its genetic information. For most organisms, the genome is encoded in DNA (deoxyribonucleic acid) macromolecules (or RNA - ribonucleic acid - for some viruses). They contain all the needed information for the organism's development, growth, and life, and at a smaller scale, the proper functioning of each cell.

DNA is stored in every cell of an organism, whether in a nucleus for eukaryotes, or in the cytoplasm for prokaryotes. At certain points of the cell life, during division process, the DNA is condensed and can be observed in a compact form, called a *chromosome*. However, in addition to chromosomal DNA, organisms can have extrachromosomal DNA macromolecules: plasmids for bacteria, mitochondrial DNA for eukaryotes, chloroplastic DNA for plants and algae. DNA sequences are also characterized by their epigenetic modifications [1], but this will not be addressed in this work. In this thesis, we will limit our definition of a genome to its core, *i.e.* to its chromosomal DNA sequence. To give an idea of genome sizes: human and mouse genomes are around 3.4 Gbp (giga base pairs, *i.e.* billions of nucleotides), bacterial genomes are often around a few Mbp

1. Epigenetic modifications can be seen as particular tags on some nucleotides that impact DNA expression

(mega base pairs), and viral genomes are often around several tens of kbp (kilo base pairs).

The primary structure of a DNA strand can be described as a chain of nucleotides tied together by phosphodiester bonds. There are four nucleotides composing DNA: adenine, cytosine, guanine and thymine. Thus, in genomics, a DNA molecule can be straightforwardly represented as character strings over the alphabet $\{A, C, G, T\}$ - that are also commonly called *bases* (for "nitrogenous bases").

While DNA can exist as a single chain of nucleotides (for example in some viruses, or during the stages of DNA replication), it mainly appears as a macromolecule. DNA macromolecules are made up of two sequences (called *strands*) of nucleotides - that are bound by electrostatic bonds between hydrogen atoms - and has the secondary structure of an helix. Between the two strands, nucleotides mate in pairs: an adenine on one strand matches a thymine on the other strand, and a cytosine matches a guanine. As the two strands are complementary, the information from only one is enough to represent a DNA macromolecule (Figure 1.1 A).

DNA vocabulary

Chromosome. The structure taken by DNA macromolecules in a cell. A chromosome can be either in a filament state, or in a compact shape. In their most condensed form, chromosomes become visible: this is the structure to which we refer, and which we represent in this document.
Chromatid. A chromosome is shaped as a "double stick", in the shape of an X. The two "sticks" are identical, and called chromatids.
Locus. A position in a chromosome, associated to a gene or a genetic marker.
Variants. The different versions of a genomic sequence at a given locus. In their simplest form, they only concern one base variations and are then called Single Nucleotide Polymorphism (SNP).

1.1.2 Higher level representation: genomes as sets of chromosomes and haplotypes

Whereas bacteria have only one chromosomal DNA macromolecule and can thus be represented by a single sequence, most organisms have a more complex organization of

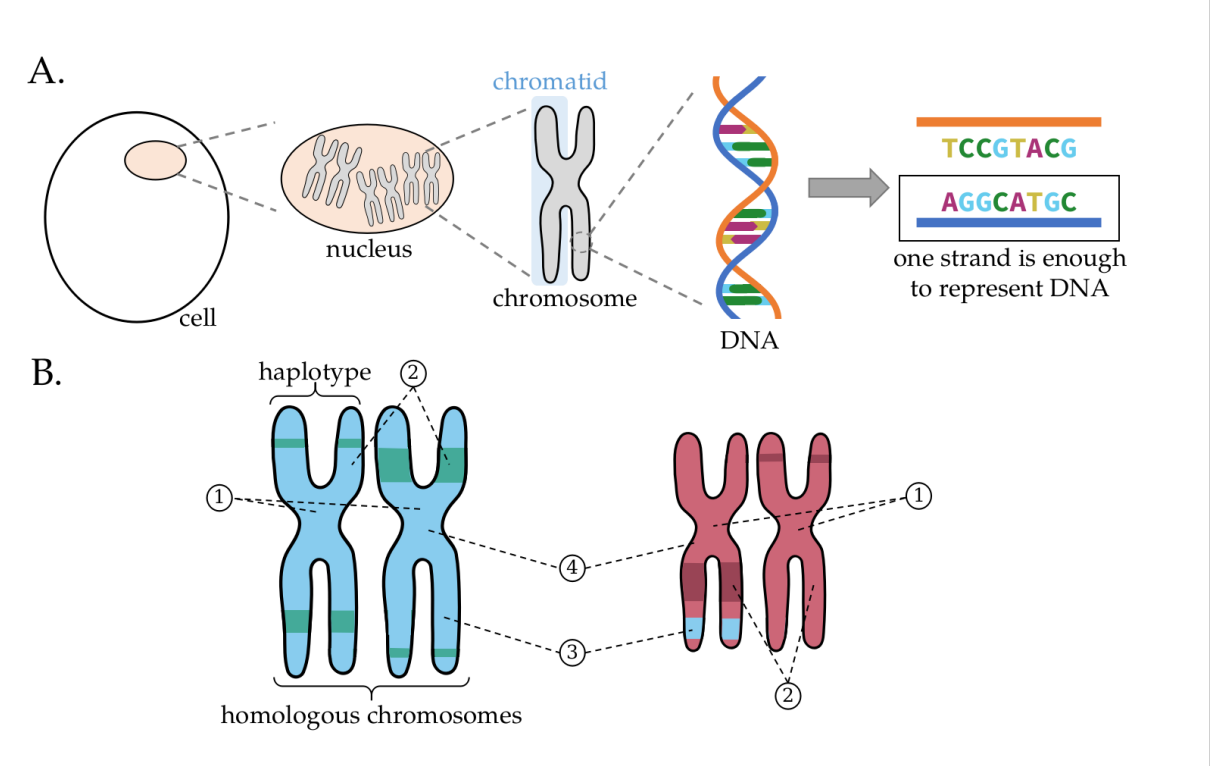


Figure 1.1 – **DNA, chromosomes and haplotypes. A)** In cells (here an eukaryote, but same principles apply for prokaryotes, except there would be no nucleus), DNA is contained into the nucleus, and is shaped in chromosomes. DNA is double stranded, and made of four nucleotides A, C, G and T, that match pairwise (A with T, C with G). Thus, in genomics, the information of a single strand is kept, as the second one is redundant. **B)** Two pairs of homologous chromosomes (*i.e.* two pairs of haplotypes). Within a given set of homologous chromosomes, haplotypes are very similar (1), yet may have some local differences (2). Non homologous chromosomes may share some local similarities (3), but are globally different (4).

their genome. For example, *Homo sapiens* (human) has 46 chromosomes, *Triticum aestivum* (common wheat) 42, *Canis lupus familiaris* (dog) 78, and *Drosophila melanogaster* (drosophila) 8. In addition to the number of chromosomes, an important parameter of genome's complexity is its *ploidy*. In fact, within a set of chromosomes, one can observe fixed-size subsets of chromosomes that are quite similar to each others. In the case of humans, chromosomes are organized in pairs: it is a *diploid* genome. Common wheat's genome can be represented by groups of 6 chromosomes: it is *hexaploid*. Generally speaking, an organism whose chromosomes can be grouped in subsets of p chromosomes is said *p-ploid*, and chromosomes in a given subset are called *homologous chro-*

mosomes. In contrast, organisms having a single copy of each chromosome, like bacteria, are said *monoploid* [2].

Haplotypes of an organism are tightly associated with sexual reproduction, and more specifically the with the meiosis and fertilization processes. During meiosis, gametes are derived from a double division of germ cells: the homologous chromosomes are separated into two cells, then the chromatids of each chromosome are separated, forming four cells. As a result, each cell contains one chromatid of each chromosome. In addition to such segregation, events of local exchanges between homologous chromosomes - called crossing-over - may occur. This leads, from a unique germinal cell, to four gametes in which chromosomes and chromatids are separated randomly (Figure 1.2). The process occurs both for maternal and paternal gametes. Then, during fertilization, one maternal gamete will randomly associate with one paternal one (which represents $4^2 = 16$ possibilities), leading to a new organism made up of half of maternal chromosomes and half of paternal ones.

In this thesis, we focus on diploid and polyploid species, *i.e.* species for which homologous chromosomes are present in at least two copies. Each copy within a set of homologous chromosomes is called an *haplotype*. To be more accurate, haplotypes of a given chromosome are not exact copies. They share a large ratio of information - which makes two haplotypes closer to each other than to any other non homologous chromosome - but are not identical (Figure 1.1 B). Haplotypes have different variants on certain loci, which can be single (difference on one nucleotide) or across several bases (differences can be of more than a thousand nucleotides).

1.1.3 Origins of polyploidy

During the history of evolution, several events of whole genome duplication occurred and led to the creation of numerous lineages, including the one that resulted in the vertebrates [Dehal and Boore 2005]. While polyploidy is a common phenomenon found in plants, it is more rare in animals, and most of the time lethal for higher vertebrates.

A polyploid can be produced by an improper segregation of chromosomes in the

2. The term "haploid" (from Greek "haploos", meaning "single") can be used with two distinct definitions: either meaning "exactly one set of chromosomes" or "half the number of sets of chromosomes compared to somatic cells". Using the latter definition, an "haploid" cell could be in fact triploid for example. To avoid ambiguity, we use the term "monoploid" to refer to a genome for which homologous chromosomes are present in only one copy each.)

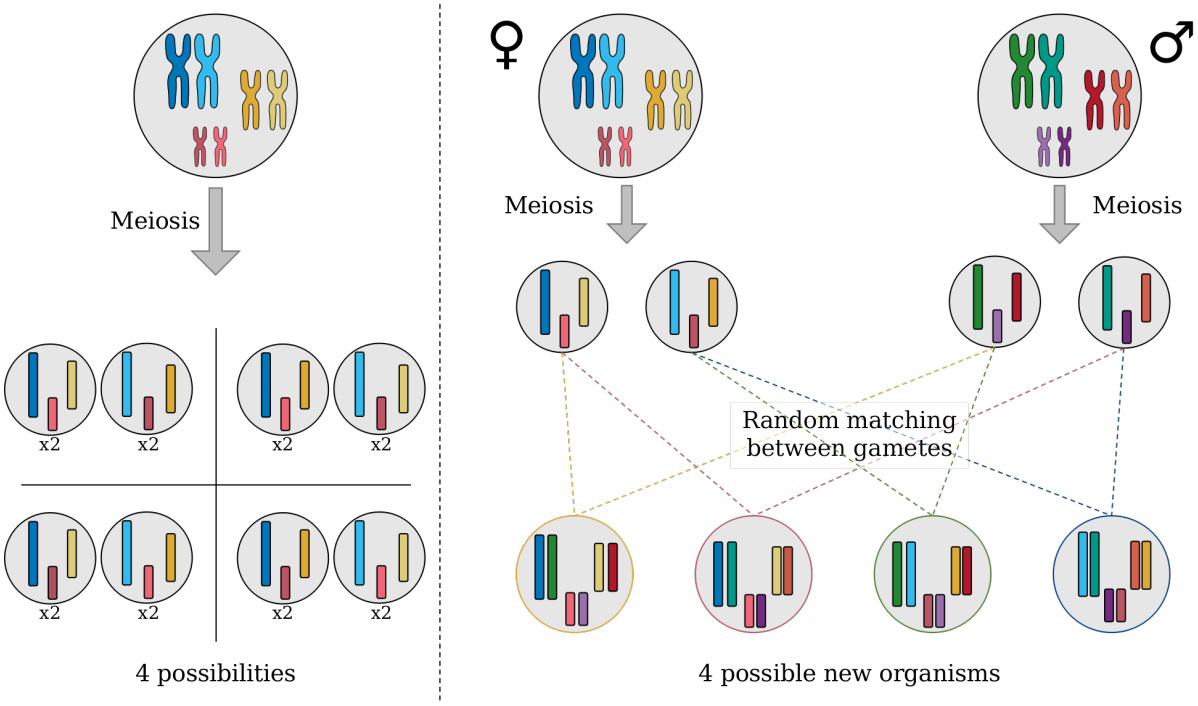


Figure 1.2 – **Diversity of gametes created from a single germinal cell.** During meiosis, a germinal cell undergoes two successive divisions producing four gametes. During these divisions, homologous chromosomes and chromatids are separated randomly in the new cells. This represents 4 possible distributions of chromatids in the gametes for a diploid organism with 3 chromosomes. During fertilization, one maternal gamete randomly matches with one paternal gamete, creating a new organism. Overall, this represents a huge diversity of possible combinations, each one resulting in a new organism containing half of maternal haplotype and half of paternal one.

two daughter cells during a cell division. If the failure appears during meiosis, it will impact the generated gametes. In this case, a diploid organism would produce a diploid gamete, which can match with a regular haploid gamete, and thus create a triploid zygote. The triploid state is unstable for most species: a triploid organism is likely to be not viable or sterile (triploid organisms produce aneuploid [3] gametes that are sterile most of the time), but it also can lead to the creation of polyploid organisms [Ramsey and Schemske 1998]. Indeed, a diploid gamete of a triploid organism can associate with another diploid one, leading to tetraploid organisms, which are viable. Some sterile polyploid organisms can maintain a triploid population via asexual reproduction. In

3. An aneuploid cell contains an unusual number of chromosome, that is not a multiple of the species' haploid number.

such cases, a whole genome duplication is likely to occur, leading to a new species of viable polyploids.

Two types of polyploidy can be distinguished: *autopolyploidy* when events of genome duplication occur within a same species, and *allopolyploid* when a genome duplication occurs after the hybridization of two different species.

A polyploid genome presents advantages over its diploid equivalent: the duplicated state of the genome brings gene redundancy, and is associated to heterosis [4] in the case of allopolyploidy. Gene redundancy enables the emergence of new traits and protects from deleterious effects of mutations, as one copy of a gene can undergo variations without causing harmful effects for the organism. For instance, the rapid adaptation of some bacteria and yeast to changing environment has been associated to their polyploidy [Eberlein *et al.* 2017].
Yet, polyploidy makes the proper separation of chromosomes more complex, and may produce aneuploid cells [Borel *et al.* 2002]. This has led to the emergence of cellular mecanisms to maintain correct segregation of chromosomes during cellular divisions, such as the gene *pairing homeologous I* in wheat [Sears 1976].

1.1.4 Why is haplotype information crucial?

As information on each haplotype of a given chromosome is almost identical, one may wonder if keeping just one consensus sequence would be sufficient for all practical purposes. Knowing information at the haplotype scale is relevant and even necessary for multiple applications and analyses, which we outline below.

An obvious application is *to study the causes of genetic diseases*. In human health, they are linked to the presence of specific variations in the genome. Thus, identifying variants at a given genetic locus (*i.e.* genotyping) helps to understand and detect such diseases. When one wants to find the genetic causes of a disease, one compares the differential expression of genes' variants between a control set (individuals exempt from this disease) and a set of patients, thus detecting loci and variants having potential influence on the disease. However, as medical research advances, it appears that complex genetic diseases, especially those that are not yet well understood, are more likely to be caused by multiple non-independent variant loci, occurring simultaneously (this is called *epistasis*). For example, disease susceptibility and gravity can be linked to a

4. Also called *hybrid vigor*, it refers to the cases in which an hybrid benefits from amplified traits from its parents.

specific variant whose expression is regulated by another region in the genome (often, promotors) [Joosten *et al.* 2001; Drysdale *et al.* 2000]. Moreover, if a disease manifestation is under the influence of several variant loci, it would require substantial resources to study their combinatorics. Yet, in practice few combinations of variants do appear as they are globally grouped in a limited number of haplotypes that create dependencies between variants. In such context, separately detecting possible variants at each locus is not sufficient: the concurrence of variants should also be detected, *i.e.* the haplotypes. Such information revealed mutations associated with the Huntington disease and markers of this disease [Squitieri *et al.* 1994], as well as its prevalence among populations [Warby *et al.* 2011], and highlighted haplotypes associated with cystic fibrosis [Kerem *et al.* 1989]. In order to extend these researches to other diseases, efforts have been made to establish a map of human haplotypes, leading to the HapMap project [International HapMap Consortium 2005; The International HapMap Consortium 2007].

Haplotype aware information is also *useful in agronomy*. Plants, and especially crops, are often polyploid species. Their genome can thus undergo numerous mutations on its haplotypes, bringing up new traits that can benefit the plant (stress resistance, growth, reproduction, ...). For a while, crops have been selected by humans according to their phenotypes (*i.e.* observed characteristics), breeding only crops showing traits of interest to keep or even emphasize them in the next generations. Nowadays, genetic knowledge is used to improve efficiency of such selection. A desired phenotype is often influenced by multiple genes, and considering variants independently is unsuitable. In particular, statistical tests of association with traits become difficult since the multiple tests on the set of variant loci need to be properly corrected. Depending on the conservativeness of the test used, it can increase the false negative rate. Knowing haplotypes for such crops can be useful to help understanding specific resistance (*e.g.* in the context of climate change, a better use of water resource, or better resistance to bugs, without needing pesticides), to select traits when breeding, or for better selection of crops to cultivate. Having information regarding their haplotype is more accurate and relevant than just having variants information, to associate genetic markers with characteristics of interest: [Abed and Belzile 2019] highlights the benefits of using haplotype information over SNP to detect quantitative traits of interest in barley, and [Bhat *et al.* 2021] reviews methods to exploit haplotype information for crop breeding in the context of climate change.

Finally, another major use of haplotype information is for *studying populations*. Com-

paring haplotypes among individuals, populations or species gives information on: their structure [Lawson *et al.* 2012]; selection pressure, evolution and demographic history [Leitwein *et al.* 2020]; or large event affecting population such as polyploidisation [Yang *et al.* 2017].

In all these cases, the haplotype information is used to look for *linkage disequilibrium* [Wall and Pritchard 2003], *i.e.* when, for given loci, a set of alleles occurs with a frequency that diverges from the expected one under hypothesis of loci independence and random association of alleles. Haplotype sequences provide direct access to variants that are co-occuring. By identifying the haplotypes associated with a particular phenotype (such as a disease or a desired characteristic for crops) it is possible to determine the variants and associations of variants that are likely responsible for it.

1.2 The representation of reference genomes

Nowadays, one can find more or less complete reference genomes in public databases (the main one being the NCBI [Sayers *et al.* 2022]) for many organisms. How are these sequences obtained?

A DNA macromolecule cannot be treated by sequencers at once, as it is too large for available technologies. Thus, one starts by fragmenting DNA into smaller pieces, which are then sequenced (they are then called *reads*) and assembled to retrieve the original genome sequence (see chapter 3 for details on the assembly process).

1.2.1 Sequencing a genome

The first genome sequencing methods date back to mid 70s, among which the Sanger [Sanger and Coulson 1975] method, which soon became predominant. The principle is to synthesize a complementary strand from a denatured DNA macromolecule, using a polymerase. The reaction is stopped at a random time, and a sensor recognizes the last nucleotide of this new strand. Such reactions are performed a huge number of times, enabling to deduce the full DNA sequence. While producing long reads (up to 700 bp) with high accuracy [Kircher and Kelso 2010], the prohibitive cost and preparation time required restrict their use to small scale sequencing projects.

Next generation sequencer - short reads

In the mid 2000's, the Next Generation Sequencing (NGS) introduced sequencing parallelization, providing high throughput. We focus here on the current leading technology: Illumina sequencing.

The rough principle is the following. DNA is extracted then split into few hundreds (often 100 or 150) bases long fragments, to which adapters are added before fragments are amplified (by PCR, followed by bridge PCR after being attached to a flowcell), in order to increase the signal before sequencing. Then, a solution of dNTPs with fluorescent markers is added, and DNA complementary strands are synthesized. As new nucleotides are added, fluorescent signal is recorded, which enables to reconstruct the DNA sequence.

Sequencing vocabulary

PCR (Polymerase Chain Reaction). Succession of DNA polymerization and denaturation cycles, for *in vitro* enzymatic amplification of an initial sequence.
dNTP (deoxynucleotide triphosphates). The elementary units that constitute DNA macromolecules, that are used during DNA replication. There are four dNTP, one for each possible nucleotide: dATP, dCTP, dGTP, and dTTP.
Read. A character string output from a sequencer, corresponding to a small fragment of a strand of DNA.

The limitations of this technology are the small size of reads (which leads to assembly problems, see chapter 3) and the inability to find epigenetic modifications in the DNA sequence, lost by PCR amplifications. In addition, sequences whose GC content is not even suffer from coverage bias [Benjamini and Speed 2012; Y.-C. Chen *et al.* 2013].

NGS data achieve very low error rate reads, about 99.9% accuracy [Glenn 2011], although still higher than for Sanger sequencing. Their main asset is the much higher speed of sequencing projects: whereas Sanger's method sequences only one molecule at a time, NGS enable to simultaneously sequence millions of fragments.

Third generation sequencer - long reads

A new major step in sequencing technologies development, Third Generation Sequencer (TGS), considerably increased read length, now reaching several thousands of bases. We give a quick overview of the two major TGS technologies. For more details, we invite the reader to refer to chapter 2. In the context of this thesis, we will build our method and base our experiments on these technologies.

PacBio sequencers. PacBio sequencing principle shares some similarities with Illumina sequencing, as it is based on measuring fluorescent activity of added nucleotides during synthesis of a DNA strand, from an initial one-stranded initial molecule. However, unlike Illumina for which the synthesis is done in multiple rounds, each molecule is sequenced all at once. Moreover, the template DNA can be circularized and can then be sequenced multiple times, which provides replicates of a same molecule. This enables to build a consensus corrected sequence for a given molecule, and thus achieves very low error rates (99.9% accuracy, according to PacBio).
PacBio sequencing has about 13% of raw (*i.e.* before any correction among reads nor consensus from circular molecule) error rate [Dohm *et al.* 2020], most of errors being insertions, followed by few deletions and substitutions. Reads are much more longer than for Illumina sequencing: up to 25kbp according to PacBio, and with mean length slightly below 10kbp [Dohm *et al.* 2020]. Epigenetic modifications of DNA can be detected, as sequenced DNA macromolecules are native: the color of the fluorescent marker gives the base, and the kinetics of the signal indicates potential methylation events.

Oxford Nanopore Technologies (ONT) sequencer. This sequencing approach measures electrical current variations while DNA macromolecules pass through nano-scale holes. As the four nucleotides have different shapes, they will generate specific patterns of electrical current variations.
A major asset of ONT sequencing is portability: the first released sequencing device, the MinION, is palm-sized, thus enabling on site sequencing projects such as the Tara Ocean metagenomic project [Carradec *et al.* 2020] or sequencing in space [Castro-Wallace *et al.* 2017].
ONT technology has evolved significantly over the years. Starting from a very high error rate (around 38%) and read average length of a few kb when released, [Laver *et al.* 2015], error rate of ONT data is now around 5% (although it depends on many parameters), almost evenly distributed over insertions, deletions and substitutions, with slight

dominance of insertion and deletion (indel) errors. Current read length is a bit higher than for PacBio sequencing, often of 10kbp, with longer reads with the ultra long read sequencing kit (> 100kb), even up to several megabases. Like for PacBio sequencing, it is possible to sequence native DNA and thus to detect epigenetic events (in particular methylations), as the electrical signal will be altered in such cases.

Both PacBio and ONT technologies enable to sequence in real time, meaning that the sequenced data can be accessed even if the sequencing process is not finished yet.

1.2.2 Build a reference genome from sequencing data

Once the sequencing of a genome is done, the generated reads can be used directly for various specific applications such as looking for markers indicating the presence of a gene of interest. The main downstream use of sequenced reads in Whole Genome Sequencing (WGS) is to recreate the original genome from the set of reads. This is called an *assembly* and consists in ordering reads along the genome based on their similarity, and building the genome sequence from the concatenation of overlapping fragments. The assembled genome may be used as a *reference* for other researches and analyses: aligning two references to each other, or a set of reads against a reference, enables to detect variations between populations, individuals, or different conditions.

The assembly can be "complete" when chromosome length assembly is achieved or, more frequently, partial *i.e.* fragmented in a set of *contigs*.

Assembling methods can be split into two categories. *Reference based* ones use an existing reference genome of the sequenced organism in addition to read data. It enables to achieve higher accuracy and resolve more complex regions, but requires to have a sufficiently reliable reference, and comes with the risk of biasing the new assembly. When no (or unreliable) reference genome is available (which is often the case for non model organisms) the assembly is said *de novo*. In the following, we present the principle of two commonly used methods for assembly, which can be addressed as searching an Eulerian or Hamiltonian path in an overlap graph.

Genome assembly: Overlap Layout Consensus (OLC) approach

This method, elaborated for sequencing data of the first generation, has been adapted for TGS datasets, exploiting the greater length of reads. The OLC paradigm has three main steps (Figure 1.3).

Overlap. Determine read pairs that share a common subsequence, and build an overlap graph with nodes representing reads, and edges connecting overlapping reads.
Layout. This step aims at cleaning the graph, removing redundant information and erroneous nodes or edges due to supposed sequencing errors. Once the graph has been cleaned, the assembly is obtained by searching for an Hamiltonian [5] path in the graph. Some unsolvable regions may still be present, causing the assembly graph to be fragmented: the assembled parts are then called *contigs*.
Consensus. Once a contig is built, it can be corrected by aligning on it all reads used for its creation, and keeping at each position the majority nucleotide, creating a consensus contig.

As finding an Hamiltonian path is NP-complete, heuristics are needed to solve the assembly problem using the OLC approach. OLC is more resilient to repeats than the second approach dBG (see below), yet is still impacted by repetitions larger than read length. Finally, the search of overlapping pairs of reads represents a computational bottleneck, as the number of reads can be very high, and often reaches millions.

Genome assembly: de Bruijn Graph (dBG) approach

OLC based methods used to assemble data sequenced with sequencers of first generation were not suitable for NGS, as they were not able to process the volume of data. A widely adopted scalable approach to assemble genomes from sequenced fragments is to use a de Bruijn Graph. The dBG structure enables to considerably speed up the computation steps compared to OLC, by performing k-mer comparison instead of full reads alignments.

The global algorithm is the following (Figure 1.4): one splits each read into small words of length k - called *k-mers* - then builds a graph.
A first option is to build a graph with k-mers as nodes, and edges between two nodes if they overlap on $k-1$ characters. Then, the assembly is done by searching for an Hamiltonian path, like in the OLC approach. This has been implemented in short reads assemblers like SOAPdenovo [Luo *et al.* 2012] or Velvet [Zerbino and Birney 2008]. However, searching for an Hamiltonian path is exponential in the number of branching nodes [Thomason 1989] and thus requires proper handle of such nodes.
A more appropriate construction is to set k-mers as edges, connecting two nodes repre-

5. Finding an Hamiltonian path in a connected graph consists in finding a path passing exactly once through each node. This problem is NP-complete.

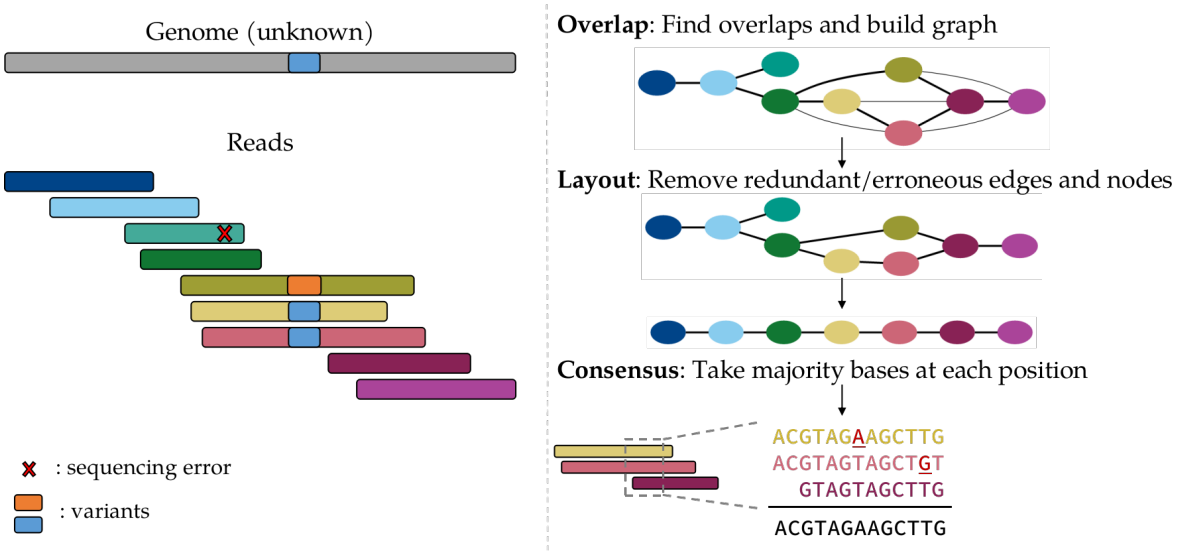


Figure 1.3 – **Principle of the OLC based assembly.** *Overlap:* From a set of input reads, compute pairwise comparisons to find overlap relations between reads. Then build a graph with reads as nodes and put edges between two overlapping reads (according to chosen thresholds such as a minimal overlap and a minimal identity). Here, stronger overlaps are in bold. *Layout:* The graph is cleaned by removing redundant and/or likely erroneous edges and nodes. *Consensus:* Finally, a consensus sequence is computed by taking the most frequent base at each position of reads.

senting the two (k-1)-mers of the overlap, and to look for an Eulerian path [6] in the graph, for which the complexity is therefore quadratic in the number of nodes (if no simplification operations are made on the graph) [Compeau, Pevzner, and Tesler 2011]. This basic approach for assembly is not well suited for long and erroneous reads [Z. Li *et al.* 2012], yet it has been implemented as adapted versions in assemblers such as wtdbg2 (now Redbean) [Ruan and H. Li 2020] and Flye [Kolmogorov *et al.* 2019].

The value of k depends on the desired sensitivity: for small values the extracted k-mers will have numerous occurrences but the graph may be highly fragmented (especially in presence of repeats, see next section); whereas larger values of k will generate more unique k-mers which are then more likely to contain an error, reducing chances to overlap with other k-mers. The choice of k is not straightforward, and depends in particular on the assembler used and the input data. The need to try several values of k, possibly relying on distribution of k-mer abundances, to obtain the best results can

6. Finding an Eulerian path in a connected graph consists in finding a path passing exactly once through each edge. To find out if a given graph has an Eulerian path can be resolved in polynomial time, based on nodes degrees, unlike finding an Hamiltonian path.

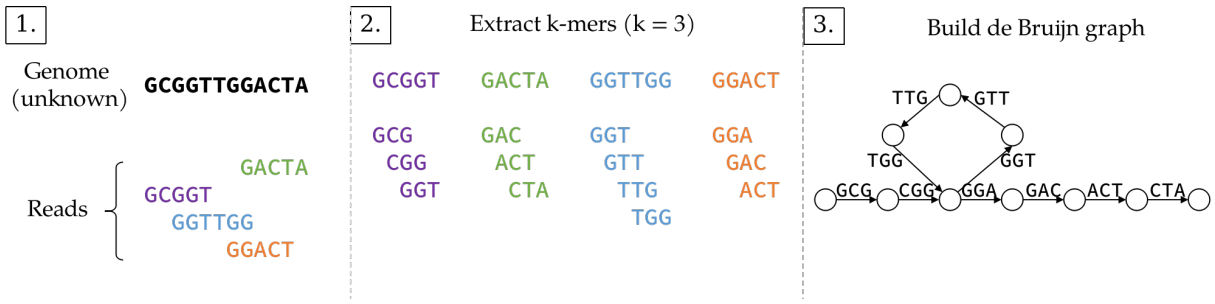


Figure 1.4 – **Principle of the dBG based assembly.** (1) From a set of input reads, (2) extract k-mers. Here, for the sake of simplicity, k is overly small. Then build the graph, with k-mers as edges, connecting the two nodes ((k-1)-mers) of the overlap (3). The initial genome can be retrieved by the concatenation of the first k-mer and the last base of next k-mers, following the Eulerian path in the graph.

be highly time consuming. Some tools implement efficient algorithms to count k-mer occurrences (Jellyfish [Marçais and Kingsford 2011]) or provide guidance for choosing a suitable k value (KmerGenie [Chikhi and Medvedev 2014]).

This approach is mainly used for short accurate sequencing data, because splitting long reads into short k-mers would deprive them of their main asset, being their wide length.

Splitting reads into k-mers enables to detect sequencing errors. Indeed, assuming sequencing errors occur randomly in reads, one can filter out k-mers with a very low abundance as they are likely to be erroneous. Moreover, using the dBG approach enables alignment-free comparison, thus considerably speeding up the costly step of overlap computation in OLC approach.

Yet, the dBG approach is not perfect, and one of its major limitations is when dealing with repeated regions. In case of a repeat of larger length than the k-mer size, then the k-mers occurring within this repeat will not contain surrounding information, and will be hard to distinguish from k-mers of the other repetition occurrences. This is discussed in more details in the next section about limits of assemblies.

How to tell if an assembly is good or not?

Numerous measures exist to assess the quality of an assembly. We present here only a few of the most common ones, present in software such as QUAST [Gurevich *et al.* 2013], one of the reference tools used to evaluate assemblies. A more detailed presentation of assembly metrics is presented in chapter 3.

When no reference is available, common metrics are: the number of assembled parts (contigs) as well as their total and maximum length, and the N50 value. The latter represents the maximum contig length for which the set of contigs of size at least N50 covers at least 50% of the assembly. For example, the latest human reference genome (GRCh38.p14) has a N50 of 67,794,873. The N50 gives an idea of the contiguity of an assembly, but does not enable comparison between assemblies. A similar metric, the NG50 is computed according to the size of the genome (or an estimation) instead of the size of the assembly. One can also use a third-party tool (*e.g.* Busco [Simão *et al.* 2015]) to predict genes and then count how many were predicted in the assembly.

An accurate reference is helpful, allowing to produce additional indicators: number of errors (substitutions, insertions, deletions), number of genes covered by the assembly, number of unaligned contigs, and misassemblies (inversions, translocations,...), k-mer content and distribution (with Kat [Mapleson *et al.* 2017]).

1.2.3 How an accurate reference genome can actually be incorrect

Common limits of assemblies

Assembled genomes are never perfect, and suffer from shortcomings that would impact all downstream analyses.

A first source of mistakes, is the *presence of sequencing errors*. Depending on the technology, such errors can be more or less abundant, and complex to be fixed. The profile of sequencing errors also matters: if errors are randomly distributed then a simple consensus may be sufficient to correct them, whereas systematic errors are much harder, or even impossible, to correct.

Another limitation comes from the *length of sequencing data*, even more if they are split into k-mers. Indeed, genomes often contain repeated regions of different sizes, and if their length is greater than read length, it will cause ambiguities in the assembly graph (Figure 1.5). In case of a tandem repeat (a pattern having several consecutive occurrences), if the pattern length is greater than k-mer length, then it will not be possible to correctly determine on which occurrence of the repetition the k-mers align. This would likely results in an assembly with under-estimated repetition length. A similar problem arises in the case of interspersed repeats, for which non-repeated sequences occur between each repeated pattern. In such case, there is no way to determine the correct order of the non-repeated sequences surrounded by the repeated pattern.

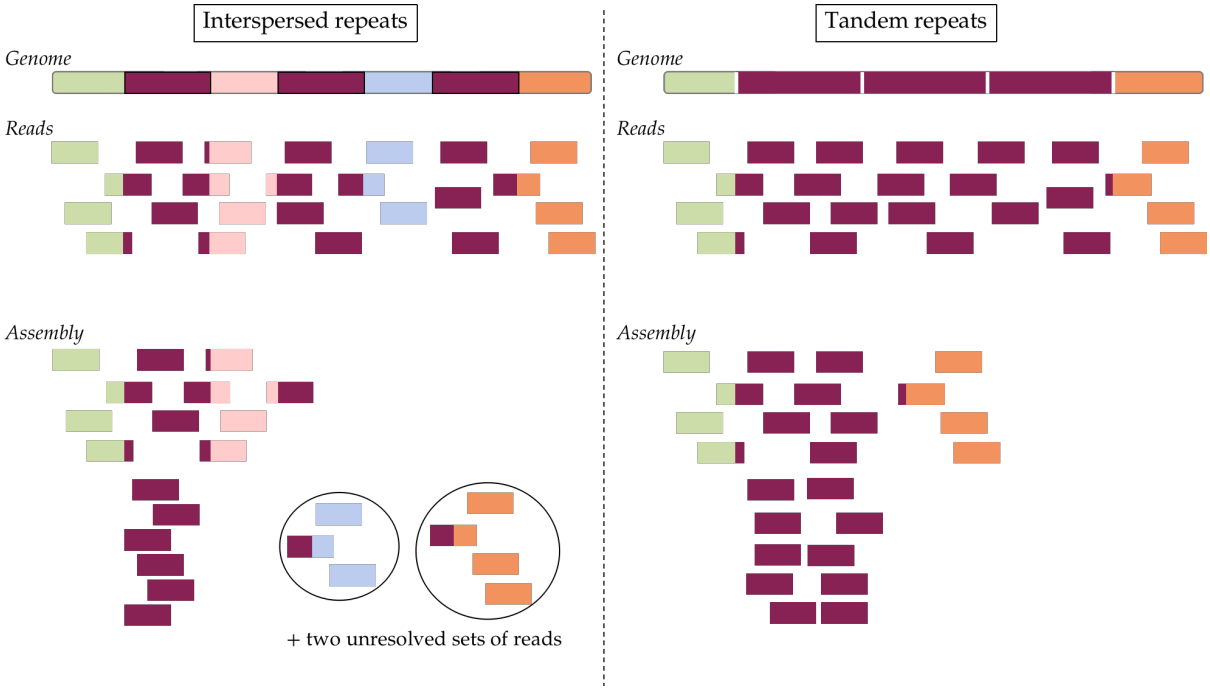


Figure 1.5 – **Large repeats cause issues in genome assemblies.** Repeated sequences that are larger than reads (or k-mers) lead to ambiguities in the final assembly. In the case of an interspersed repeat the correct order of non repetitive sequences cannot be retrieved, while the length of tandem repeats is likely to be underestimated.

Finally, another frequent issue with assembly, is its *incompleteness*. When sequencing data are lacking, or some regions of the genome are hard to assemble, this results in gaps of varying sizes in the assembly. Thus, the resulting genome is not a fully assembled sequence, but a set of assembled contigs or scaffolds (longer genome portions made of contigs and gaps). For instance, the human reference genome GRCh38.p14 has 996 contigs, and 470 scaffolds separated by 349 gaps (scaffolds contain 526 additional gaps). In comparison, the assembly from 2001 human genome sequencing data[7] contained 211 493 contigs, and 4 940 scaffolds separated by 382 gaps (scaffolds contained 206 553 additional gaps).

An assembly showing good results with regard to the above criteria (low error rate, resolved repeats, completeness) could be considered an accurate assembly. Yet, these usual criteria do not depict the full reality of what assembled genomes should integrate, and are not sufficient to define an accurate assembly.

7. `https://www.ncbi.nlm.nih.gov/assembly/GCA_000002115.2`

Ploidy related assembly limits

Most of the reference genomes available today are monoploid, even for polyploid organisms. Thus, the reference either represents a single haplotype (others being absent), or the reference is a chimera built from a consensus of all haplotypes. In both cases, this represents a huge loss of information, and such references are not representative enough of the true genome. For example, in the human reference genome GRCh38, the region coding for the ABO blood type had only the A allele (other alleles are provided as annotations, but are not directly present in the genomic sequence).

Thus, one should not limit to the previously mentioned sources of errors, and should also consider the haplotype related information when assembling a genome.

Combinatorics related limits

One should notice that assembling tools usually provide only one solution. Even if there is only one underlying biological truth, it is likely that we probably will not be able to reach it with computational methods. Considering the errors in input data, and heuristics needed to build up a genome from these data, it seems more reasonable to consider that one could have ended up with equally acceptable multiple solutions. Plus, defining what would be the optimal solutions may vary depending on the algorithmic, biological, and computational goals.
Thus one should not aim at proposing only a single solution, but rather at providing a set of all optimal solutions found.

1.3 Complexity of building an haplotype-aware reference genome

We previously mentioned some reasons why usual monoploid assemblies of polyploid species are not accurate enough, and might miss crucial information.

Here, we will briefly discuss of the major challenges to build reference genomes with haplotype information. This can be achieved by *haplotyping* or *phasing*. Haplotyping consists in segregating genomic information to distinguish the different haplotypes. The phasing problem is close, but aims in addition at providing an assembly of the haplotypes. As assembling a genome is a problem in its own right, we will not address it in this document, and focus on the haplotyping problem.

One should also note that the phasing problem is somehow similar to metagenomic assemblies. In such studies, one has a mix of many micro-organisms' DNA (such as bacteria, viruses, yeast, ...) from a given environment, like sea water or intestine, and aims at distinguishing then assembling their genomes. But these two problems diverge in one crucial point. In metagenomics, one has *a priori* no idea of how many - the order of hundreds or thousands - organisms to find. In contrast, for phasing, one often knows the expected ploidy, or has at least a rough idea of the organism ploidy - in most cases the organism is diploid or has a ploidy number up to about a dozen. Thus, even if parts of the methods can be similar, they are not switchable. This thesis focuses only on the haplotype phasing problem.

1.3.1 Distinguish sequencing errors from (rare) variants

A major issue in assemblies and haplotype phasing, especially with noisy reads such as TGS ones, is distinguishing between sequencing errors and true variants.

How a sequencing error and a true variant are discriminated?

Depending on the sequencing technology used, the error rate of reads, and the distribution of errors may vary (see dedicated chapter 2). The more accurate the reads, the less risks one has of confusing between errors and variants.

But accuracy is not the only way to discriminate real variations from spurious ones. The sequencing depth represents how many times a given region of the genome has been sequenced. The higher the sequencing depth, the more reads we have for a region, and the less likely it is to be wrong. Indeed, when sequencing errors are assumed to be randomly distributed, and with a reasonable error rate one would expect that for a given position in the genome a sequencing error would occur less often than the true nucleotide. Then, a common practice is to set a threshold (depending on technologies used and targeted sensibility) under which a sequenced nucleotide at a given position (or a k-mer) will be considered an error.

Variant frequency in a p-ploid organism

When considering a p-ploid organism, and as sequencing is assumed to be more or less even along a genome, one would expect to obtain a near balanced (*i.e.* $1/p$) distribution for each variant of haplotypes.

In practice, it is a bit more complex when considering $p > 2$. First, because all haplotypes are not equidistant. For example, considering a triploid organism created after the association of a diploid gamete with an monoploid one, having three haplotypes A, B, C, with A and B being closer to each other than to C. It would then be easy to mistake and recreate only two haplotypes: one collapsed A/B and C. Also, the distance between two haplotypes may vary depending on the genome region considered: two haplotypes A and B may be close in a region, whereas in another region B would be closer to another haplotype C and highly different from A. And finally, there can be regions of the genome where two haplotypes are locally identical, making the actual local number of haplotypes lower than the global one, and again may mislead the haplotype phasing. Hence, even if the ploidy number is a relevant information, it should be treated as an upper bound.

1.3.2 Haplotype phasing is a highly combinatorial problem

A simple formulation of the haplotype phasing problem consists in a partitioning issue: a set of N reads has to be split into p distinct groups. The space of partitions of a set of N elements in p parts is a Stirling number of the second kind:

$$S(N,p) = \frac{1}{p!}\sum_{i=0}^{p}(-1)^{p-i}\binom{p}{i}i^N$$

for which an upper bound has been defined in [Rennie and Dobson 1969]:

$$S(N,p) \leq \frac{1}{2}\binom{N}{p}p^{N-p}, \text{ for } n \geq 2 \text{ and } 1 \leq p \leq N-1$$

The p value is not having the greatest influence here, especially given that p will not take really high values. However, the influence of the number of reads N is exponential, and N can reach an order of magnitudes around 10^5 to 10^6 in practice. The naive approach of enumerating is therefore unfeasible in all practical cases.

Obviously, this is only a naive description of the problem, to give a glimpse at combinatorics. Existing methods for haplotype phasing use some approaches to reduce complexity, or proceed with heuristics (see dedicated chapter 3 for more details on complexity and methods to handle it).

1.4 Goals of this thesis

In this brief overview of the haplotype phasing problem, we pointed out the main associated challenges that we will try to handle in this work. The objective is to reconstruct haplotypes of a p-ploid organism $(2 \leq p \leq 12)$, based on TGS long but erroneous reads. We aim at achieving this without reference genomes, as they may be source of errors or bias, or simply be absent. Finally, we will pay attention to provide not a single solution, but a result based on the set of all optimal solutions found. This will provide insight into the proposed solutions' robustness and pave the way for a genuine interactive search for a satisfactory solution.
These aspect will be detailed in the following chapters 2 and 3.

Chapter 2

Long reads, a new generation of genomic data

Contents

Preamble: This chapter details the functioning of third generation sequencers, focusing on the two currently dominant technologies: Oxford Nanopore Technologies and Pac Bio sequencers. We will present the error profile of both, as well as a first contribution of this thesis focusing on characterizing the sequencing error profile of Nanopore sequencers. Finally, we will discuss about the different strategies employed to correct the sequencing errors in reads with dedicated correction tools.

2.1 Long read sequencing: principles, strengths and weaknesses

Most popular Third Generation Sequencing (TGS) technologies, are proposed by Pacific Biosciences (PacBio) and Oxford Nanopore Technology (ONT), which are based on different principles. In a nutshell, the main advantages of these technologies are an improved read length compared to the previous generation, as well as the ability of sequencing native DNA; but these come with an increased error rate.

2.1.1 Nanopore sequencing

Roughly, the Nanopore sequencing technology is based on measuring the variation of an electric current while a DNA molecule is passing through a pore: as the four nucleotides have different sizes and shapes, their passing will more or less obstruct the ion flux. Then these electric variations are converted to DNA sequences. The concept dates back to the 1980s (see [Deamer, Akeson, and Branton 2016] for details and milestones achieved before its selling), and has undergone many improvements until the first commercialization in 2014 of the MinION, the first ONT sequencer (see [M. Jain *et al.* 2016] for a description of the MinION's features at its release). ONT provides several variants of sequencers depending on the size of the sequencing project. The MinION and Flongle contain a single flowcell [1] and are well suited for small sequencing projects. In

1. The flowcell is the core device, where actual sequencing happens (not only for ONT sequencing technologies, but also PacBio and NGS ones).

contrast, the GridION and PromethION have respectively 5 and 48 flowcells, and therefore are suited for large scale sequencing projects. The Flongle and MinION have the additional advantage of being portable: they are about 10x3 cm and weight less than 100 grams.

Principle

The first step consists in extracting the desired DNA and preparing it for the sequencing process. ONT provides different extraction kits depending on data and desired characteristics of reads: DNA or RNA, whole genome or targeted sequencing, rapid kit, kit for ultra long reads, PCR amplification kit for low amount of DNA - to mention but a few.

Once DNA preparation is loaded in the flowcell, the sequencing is performed as follows. The flowcell contains a membrane on which there are hundreds of channels [2], each one made up of four nanopores, and there is at most one active pore at a time. A basal electric current is applied across the membrane. This drives negatively charged DNA molecules through the pores. There, a motor protein attached to the DNA binds to the pore and then start to denature the DNA double stranded molecule. One strand stays outside the pore, while the other one is driven through the pore by the protein. The translocation speed of DNA through the pore is regulated by this motor protein and is dependent on the protein used. It is of about 450 bases per second for current R9.4 (although not exactly constant over time), in a ratcheting motion. According to ONT, the theoretical output for one flowcell could be expected at up to 50 Gb for 72 hours of run (depending on sequencing conditions).

The passing of the DNA molecule within the pore will affect the measured electric current measured in a narrow sensing region. The changes will be different depending on the bases in the pore at a given time. In the ideal case, the electric current would be affected only by one base in the pore, always with the same variation. However, in practice, the electrical signal is impacted by the 5 to 6 bases in the pore, which generates much more patterns and induces noise in the signal.

Finally, a designed tool, called the *basecaller* proceeds to the translation of the electrical signal variations into a A, C, G, T-based DNA sequence.

2. The MinION and GridION devices use flowcells made of 512 channels, representing up to 2,560 channels in total. The PromethION device uses flowcells containing 2,675 channels, gathering up to 128,400 channels.

> **Basecall.** Computational process of deriving the nucleic sequence from raw sequencing information such as light intensities for Illumina and PacBio, or electrical signal for ONT.

The sequencing process is summarized in Figure 2.1.

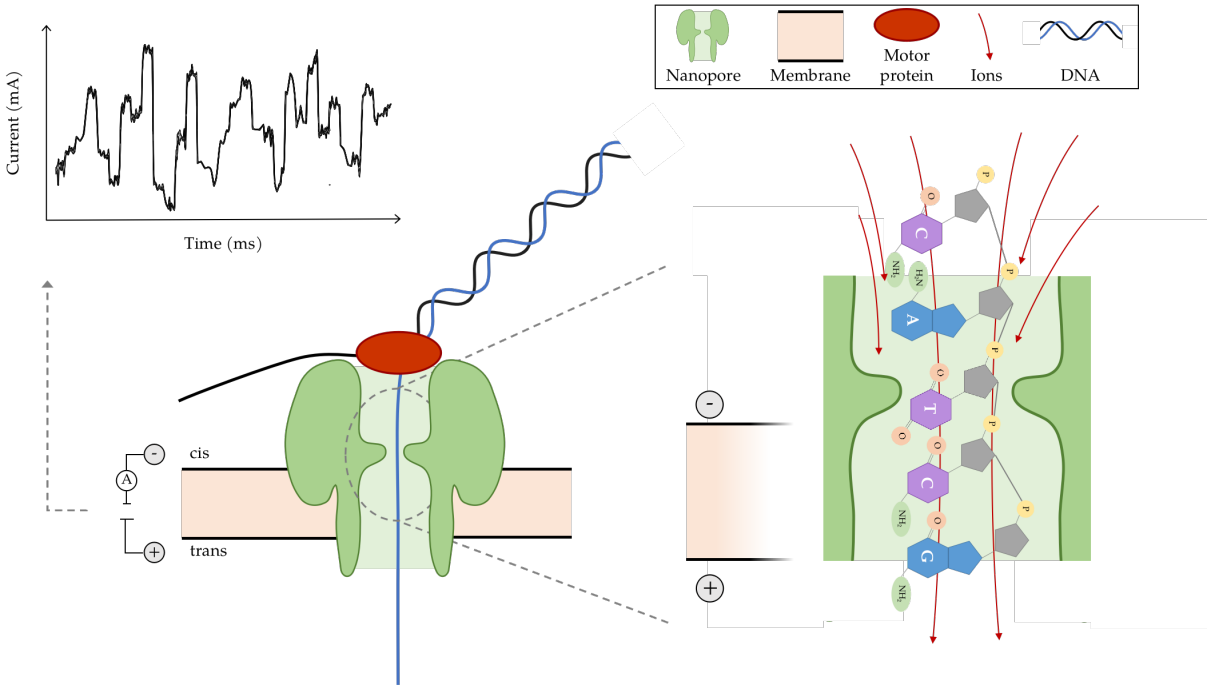


Figure 2.1 – **Nanopore sequencing principle.** (Bottom left) An electrical current is applied across the membrane. When the DNA arrives to the pore, the motor protein anchors to the pore, and start denaturing DNA. One strand is staying out while the other passes through the pore. (Bottom right) As DNA passes through the pore, the ion flux is more or less impacted. The sensing region - *i.e.* the narrow region - detects these variations. (Top left) Electrical variations are recorded, and will then be converted to a DNA sequence via the basecalling algorithm.

A crucial preparation step

The preparation of the DNA sample is a major step, as this will determine the length and characteristics of the basecalled sequences. Here we will focus on two key features of ONT sequencing: PCR-free sequencing, and ultra-long read sequencing.

Unlike with NGS sequencing, PCR amplification is not required (although it can be done for low input DNA amounts). This enables to retain base modification and

therefore to perform epigenetic sequencing on native DNA.

What makes TGS sequencers distinctive, especially ONT ones, is the produced read length: 10kb is now quite common. Plus, ONT now offers a dedicated sequencing kit to generate ultra long reads, whose announced length is of more than 50-100kb. Reads of several Mb length have already been sequenced [Payne *et al.* 2018].

Basecalling: a key factor impacting read accuracy

At a given time, the electrical signal is associated to the k bases near the sensing region of the pore. As DNA steps forward in the pore, one thus obtains electrical signals corresponding to overlapping k-mers of the read. This represents a challenging machine learning problem: considering the DNA alphabet, and that approximately 5 bases influence the signal, there is a range of $4^5 = 1024$ values that can be obtained from the signal. The first basecaller algorithms were designed to segment raw signal data and then to infer k-mers (Metrichor [3] and Nanocall [David *et al.* 2017] were based on Hidden Markov Model, Nanonet [4] and DeepNano [Boža, Brejová, and Vinař 2017] on neural network).
Later on, basecallers (Scrappie - in Albacore and Guppy [5] - and Chiron [Teng *et al.* 2018], based on neural network) stopped to segment the signal before computing actual basecalling, as it represented a supplementary source of errors, and deducted sequences directly from the raw unsegmented signal, which resulted in an improvement of sequencing quality.
The current Nanopore basecaller is Guppy, which enables the detection of methylation events, as well as to train the basecaller on custom datasets. This basecaller offers three different modes (at the time of writing, Guppy v6): a fast mode, a High ACcuracy (HAC) mode that achieve better accuracy (5 to 8 times longer computation time than FAST, according to Nanopore documentation), and a super accurate (sup) mode that is even more accurate and computationally intensive (3 times longer than HAC). In addition, one has the possibility to keep the raw electrical signal data to take into account updates in basecalling algorithms and increase the accuracy of sequences without new sequencing. The usual ONT basecaller, Guppy, is trained on a set of DNA from various organisms (plant, animal, bacteria and viruses) to be suitable for a wide range of

3. `https://metrichor.com/`
4. `https://github.com/ProgramFiles/nanonet`
5. Available on ONT website `https://community.nanoporetech.com`, for their customers only.

applications. A recent ONT basecaller, Bonito [6], has been implemented and enables to train the basecaller model on custom datasets in order to improve the accuracy of the basecalling on specific organisms.

The improvements in the motor protein used also helped increasing accuracy of sequencing, as a slower passing of the DNA through the pore makes it easier to distinguish events. However, slower sequencing comes with reduced throughput, so it is a matter of balancing between accuracy and data quantity.

There is still room for improvement, as raw read accuracy is still very low compared to NGS. For example, one of the key issues is the sequencing of homopolymers, *i.e.* sketches of a same nucleotide. As the DNA translocation is not constant, due to the lack of variation in such regions, and as the pore can only detect 5 to 6-bases length subsequences (for R9.4 flowcells), basecaller struggles in determining the exact length of homopolymers when their length is over few bases long.

Ongoing improvements

ONT sequencing is a rapidly evolving technology, and has experienced very frequent updates, whether concerning chemistry, flowcells or basecaller algorithms. For instance, there has been about one new flowcell release each year between 2015 and 2021, and the Guppy basecaller underwent 39 released versions from its first release in 2019. The latest ongoing key improvements are the new R10 flowcell, and a new sequencing kit enabling duplex sequencing.

The main point of R10 is to use a pore with two sensing regions (reader) instead of only one: at a given time the electrical signal is influenced by more bases, a given base will be sensed twice, and the overall sensing region is wider. The goal is to enhance sequencing accuracy with better homopolymer basecall: by improving the consensus call and by increasing the limit length of homopolymer detection. It is also expected that errors will be more random and thus easier to correct afterwards.

The R10 flowcells are meant to be use with a new sequencing kit (Kit12) that has been released simultaneously, and enables the duplex sequencing: sequencing both strands sequentially to increase accuracy. Yet, this technology is still under development, and first user feedbacks reported that although these reads achieved high raw

6. `https://github.com/nanoporetech/bonito`

accuracy (modal accuracy of 99%), only a small ($< 10\%$) fraction of reads actually were actually in duplex [Sanderson *et al.* 2022]. Efforts are now made by ONT to increase that ratio.

2.1.2 PacBio sequencing

PacBio sequencing technology has been developed few years before the release of MinION. They aim at combining accuracy of short read sequencing and strength of long read. Indeed, the process is similar to NGS sequencing, as it is based on measuring fluorescent activity during polymerization of a DNA strand. However, unlike NGS sequencing, PacBio enables to sequence native DNA (no PCR amplification required), and achieves longer reads.

Principle

The core point of PacBio sequencing is the use of "zero-mode waveguides" (ZMW), that are nanoscaled wells that enable to isolate molecule and capture its fluorescent activity. Each PacBio cell is made of 150 000 ZMW, at the base of each of them polymerase will be tethered, and also contain the four types of nucleotides with different fluorescent markers.
During DNA preparation, a hairpin adapter is added to both ends of the double stranded molecule, forming a circular DNA called Continuous Long Read (CLR, see Figure 2.2). A polymerase is also bound to the CLR. During sequencing, the polymerase adds fluorescent nucleotides, leading to a fluorescence output that is recorded and then converted to DNA sequence. As DNA is circular, the polymerase duplicates a strand, then adapters, then the other strand, and so on. Hence, a same DNA molecule could be sequenced multiple times. The obtained sequencing result depends on the polymerase lifetime: if the read is long then the polymerase will have time to only sequence the molecule once, whereas if the polymerase last long enough it can sequence the circular molecule multiple times before becoming inactive. In the latter case, one can identify adapter sequences and split the CLR into *subreads* from which a consensus can be derived, leading to a Circular Consensus Sequence (CCS), also called HiFi (High Fidelity) read. Hence, either one ends up with a longer but more erroneous CLR, or a shorter but accurate CCS. The length of reads is bounded by the lifespan of the polymerase, and currently enables to get reads of average length around 30kb. Figure 2.2 summarize the

PacBio sequencing principle.

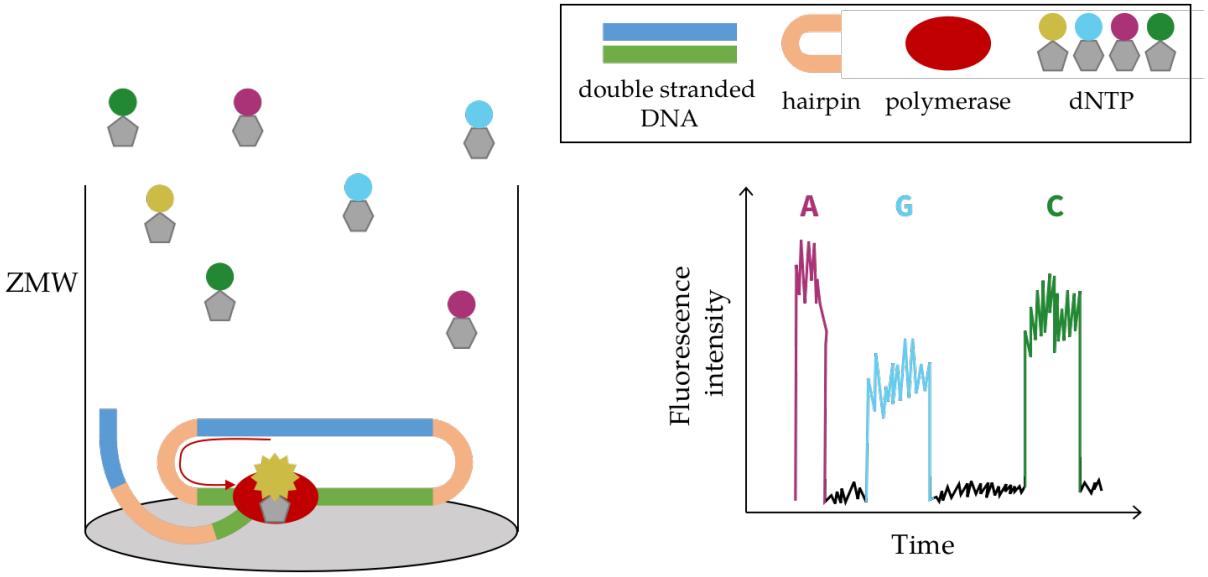


Figure 2.2 – **PacBio sequencing principle.** Before sequencing, double stranded DNA is circularized by adding hairpins at both ends. The built CLR will be then polymerised within a ZMW. Surrounding dNTP with specific fluorescent tag attached are added during this process. When a base is added, it produces a fluorescent signal that is captured. Finally, one can derive the DNA sequence based on these light signals.

Basecalling

The current PacBio basecaller is CCS[7]. The signal captured during the sequencing process carries two types of information. The first one is the wavelength related to the fluorescent dyes attached to incorporated nucleotides during DNA polymerization. This gives information on which one of the four nucleotides were added. The second one is the kinetics, *i.e.* time between two nucleotide incorporation by the polymerase, that indicates methylation events.

CCS basecaller also handles the consensus building for CCS reads. Basically, the more times the CLR has been sequenced (*i.e.* the higher the depth of sequencing), the higher the accuracy, as errors are randomly distributed. It is estimated in [Amarasinghe *et al.* 2020] that it requires a depth of respectively 4 and 9 to achieve 99% and 99.9% of accuracy.

7. `https://github.com/PacificBiosciences/ccs`

2.1.3 Error profile of long read data

Long read sequencing is commonly associated to a higher error rate than NGS data. Indeed, PacBio sequencing error rate of CLR in 2015 was around 11 to 15% [8], and Nanopore one was up to almost 40% [Laver *et al.* 2015]. However, TGS technologies had improved a lot, and are now very close to NGS's accuracy: ONT sequencing now achieve median accuracy around 96% with R9.4 flowcells [Sanderson *et al.* 2022] and a modal accuracy of 99% with the new R10.4 flowcells [Sereika *et al.* 2022]; and PacBio CCS reached more than 99.9% accuracy [Wenger *et al.* 2019].

Another difference lies in the error profile. While NGS data exhibit mostly substitution errors, TGS data are more subject to insertion and deletion (*indel*) errors [Dohm *et al.* 2020]. For ONT, sequenced data are about as sensitive to insertions as deletions and substitutions, although deletions are slightly prevalent. PacBio data, in contrast, are mostly prone to insertion errors that can represent two third of errors.

Finally a major difference between PacBio and ONT is the fact that PacBio errors are random, which, coupled with the high depth of coverage, is why the consensus correction for CCS works that well. In contrast, Nanopore sequenced data suffer from systematic errors, in particular in low complexity such as in homopolymers - whose length is thus hard to correctly estimate.

Overall, the PacBio reads currently have the lowest error rates thank to their HiFi technology. Yet, the recent development of the R10 ONT flowcells promise future improvement in Nanopore sequencing accuracy. In addition, Nanopore sequencers enable to produce longer reads, with their ultra long read sequencing kit. A final advantage of ONT is that they provide considerably smaller sequencing devices which make them suitable for a wider range of applications.

2.2 Contribution: sequencing error profile of Nanopore data

Nanopore sequencing data are error prone, and many researches have tried to correct these errors. Yet, there are only few articles addressing the description of the error

8. www.pacb.com/wp-content/uploads/2015/09/Perspective_UnderstandingAccuracySMRTSequencing1.pdf

profile. An open-source software, NanoOK [Leggett *et al.* 2016], has been developed and provides an analysis of errors and quality of reads compared to a reference, based on indel and substitution errors and k-mer analysis (those which are error-free, and those which are under- or over-represented preceding an erroneous position). Another study [Wick, Judd, and Holt 2019] also depicts the errors of nanopore sequencers based on quality and identity at read and consensus levels.
In addition, ONT does not - or very rarely - provide publications about their benchmarks nor software details. Information are often revealed during annual conferences, and here again, there is little communication about the error profile.

In order to compensate for this lack, we attempted to depict the different type of sequencing errors occurring in Nanopore data [Delahaye and Nicolas 2021] to provide a background for future correction methods aiming at distinguishing true variants from sequencing errors. This is especially crucial for applications requiring to identify true variants, *e.g.* for genotyping or haplotyping, where the use of consensus-corrected sequences would provide erroneous information.
Besides, as Nanopore sequencing data quickly turn out to be obsolete - due to the frequent updates of basecallers and chemistries - we developed a tool (SeqFaiLR [9], see last section) to replicate the main analyses on custom data sets.

We worked DNA sequencing data of 12 bacterial datasets (of different species, and strains) of various GC content, as well as two human datasets:

- two strains of *Streptococcus thermophilus* (CNRZ1066 and LMD-9) were obtained from our own team sequencing experiments;
- ten other bacterial species and strains came from [Wick, Judd, and Holt 2019]: *Acinetobacter pitii*, *Haemophilus haemolyticus*, *Klebsiella pneumoniae* (four strains: INF032, INF042, KSB2 1B and NUH29), *Staphylococcus aureus*, *Stenotrophomonas maltophilia*, *Serratia marcescens* and *Shigella sonnei*;
- two human datasets, from [Shafin *et al.* 2020].

All these datasets had been sequenced with R9.4 (or R9.4.1) ONT flowcell, without prior PCR amplification.

The dataset were basecalled with Guppy v4.2.2 (the most recent version available at that time) with both the High ACcuracy (HAC) and FAST modes. One of our goal was to characterize the improve of basecalling quality between these two modes. We

9. `https://github.com/cdelahaye/SeqFaiLR`

chose to use a quality score threshold of 10 instead of 7 (recommended by ONT): we obtained a set of selected reads for which more than 98% of reads had less than 10% of error rate, while the recommended parameter yielded to only a quarter of reads having less than 10% errors. We worked on raw, uncorrected reads.
We used as reference genomes those which were available on NCBI [Pruitt *et al.* 2012] for the *Streptococcus* strains: NC_006449.1 for *Streptococcus thermophilus* CNRZ1066, and NC_008532.1 for *Streptococcus thermophilus* LMD-9; and the available sequences provided in [Wick, Judd, and Holt 2019] for the remaining bacterial species and [Shafin *et al.* 2020] for human datasets. An overview of datasets characteristics is provided in Table 2.1.

Table 2.1 – **Main characteristics of studied species.** Size and GC content are computed from the reference genome. Mean read length is computed for aligned reads basecalled with Guppy 4.2.2. Sequencing depth is computed as the sum of all aligned sequenced bases divided by the size of the reference genome.

Species (strain)	Size (non-N bases)	GC %	Flowcell	Mean read length (bp)	Sequencing depth (bp)	Source
A. pittii	3,814,719	38.78	R9.4.1	25,487	25X	[Wick, Judd, and Holt 2019]
H haemolyticus	2,042,591	38.45	R9.4.1	9,557	14X	
K. pneumoniae (INF032)	5,111,537	57.63	R9.4	35,886	69X	
K. pneumoniae (INF042)	5,337,491	57.41	R9.4	47,39	63X	
K. pneumoniae (KSB2 1B)	5,228,889	57.59	R9.4	23,62	46X	
K. pneumoniae (NUH29)	5,134,281	57.61	R9.4	15,851	33X	
S. aureus	2,902,076	32.85	R9.4.1	21,591	77X	
S. maltophilia	4,802,733	66.28	R9.4.1	30,694	69X	
S. marcescens	5,517,578	59.13	R9.4.1	7,729	19X	
S. sonnei	4,829,160	51.03	R9.4	20,313	65X	
Human HG002	3,110,720,511	41.04	R9.4	19,29	29X	[Shafin *et al.* 2020]
Human HG00733	3,110,720,511	41.04	R9.4	14,731	28X	
S. thermophilus (CNRZ1066)	1,796,226	39.08	R9.4	6,049	330X	Own data
S. thermophilus (LMD-9)	1,856,368	39.08	R9.4	7,332	258X	

We addressed several aspects of the error profile: quality scores, GC content, types and abundance of errors, systematic errors. We have also refined some of our analyses by exploiting the raw electrical signal of the sequencing, using the ONT's tool Tombo [10]. Based on fast5 files containing raw electrical signals, a reference genome and fastq files of basecalled reads, it matches the signal on the bases of the read and associate the expected sequence from the reference genome.

All these analyses were based on the alignment of reads against their respective reference genome, using minimap2 [H. Li 2018]. In fasta files, a sequence is represented in its forward orientation. As the orientation of reads is not known, the both cases of

10. `https://github.com/nanoporetech/tombo`

aligning a read R on the forward representation of a genome G and on its reverse complement $rev(G)$ has to be considered, in order to find the true orientation of the read. Actually, most aligners reverse complement the read instead of the genome, as it is less computationally costly. This approximation is relevant for usual applications such as genome assembly, for which one only needs to know the relative orientation of reads and genome. Yet, studying the error profile requires to use the exact sequenced reads to correctly identify patterns of errors. In addition, due to heuristics used in alignment algorithms, it is likely that aligning R against $rev(G)$ and $rev(R)$ against G will not lead to equivalent results.
Example: Align the read $R = GCCAAGACCT$ against a reference $G = ACGTCATTGC$, with R and G having opposite orientations:

```
G       ACGTCATTGC                rev(G) GCAATGACGT
        | |||  | ||                      ||  | |||  |
rev(R)  AGGTCTTGGC                R      GCCAAGACCT
```

In both cases the error and overlap rates are the same (even if they could differ in practice due to heuristics). The crucial point here is that the errors are not identical: in the left alignment the errors are C/G, A/T and T/G and are A/C, T/A and G/C in the right one (X/Y meaning a substitution of the base X with the base Y).

Therefore, we payed attention to keep the original read orientations when computing alignments against reference genomes.

2.2.1 Comparison of FAST and HAC basecalling modes

We first compared the two basecalling modes HAC and FAST, in order to assess if the computation time saved using the FAST mode is worth the loss of accuracy compared to the HAC mode.

To do so, we compared the total error rates (sum of insertions, deletions and substitution errors) for the bacterial datasets with the two basecalling modes, and also measured the associated computation time (Table 2.2). Globally, the HAC and FAST basecalling modes show similar profiles among the species, with the FAST mode having about 2% additional error rate. In addition, the HAC mode is not excessively slower, as it only takes about twice more time to compute.

2.2.2 Error distribution and translocation speed

We then checked if some genomic regions were more prone to certain types of errors.

Table 2.2 – **Global error rate and runtime for bacterial species, in FAST and HAC basecalling modes of Guppy 4.2.2.** For each species and each basecalling mode, runtime is computed as the median time over five basecalling runs. Highest and lowest values of each column are in bold.

	Error rate (%)		**Runtime (min)**	
Species (strain)	FAST	HAC	FAST	HAC
A. pittii	8.02	5.37	**0.6**	**1.1**
H. haemolyticus	7.24	5.72	0.73	1.36
K. pneumoniae (INF032)	8.21	6.23	2.61	5.65
K. pneumoniae (INF042)	8.10	6.11	2.89	6.61
K. pneumoniae (KSB2 1B)	8.17	6.24	2.39	5.09
K. pneumoniae (NUH)	8.85	6.63	1.73	3.69
S. marcescens	8.50	6.00	1.04	1.94
S. sonnei	7.98	6.15	2.8	5.49
S. aureus	6.78	4.27	1.26	2.71
S. maltophilia	**8.87**	**6.83**	2.99	6.68
S. thermophilus (CNRZ1066)	**6.53**	**4.03**	**5.98**	**10.63**
S. thermophilus (LMD-9)	7.06	4.65	4.41	7.91
Mean	7.86	5.69	2.45	4.91
Median	8.06	6.06	2.5	5.29

To do so, we computed for each position the error rates for the three types of errors (substitutions, insertions and deletions) among all aligned reads. The results indicate that the errors were evenly distributed along the genome, and confirmed a prevalence of deletion errors [Dohm *et al.* 2020].

A deeper analysis of the electrical signal highlighted that long deletions (of at least 5 bases) were associated with a significant increase of translocation speed of the DNA through the sequencing pore: while the average expected translocation rate is around 450 bases per second, it reaches more than 1 000 bases per second in deleted regions (see Figure 2.3).

It is worth noting that a too slow translocation is also associated with an increased error rate. These results clearly indicate that the basecaller is calibrated for a certain range of translocation rates.

2.2.3 Bias in substitution errors

We took a closer look at substitution errors, by computing the frequency of each of the twelve substitution errors possible $X \to Y$ where X is the nucleotide in the reference

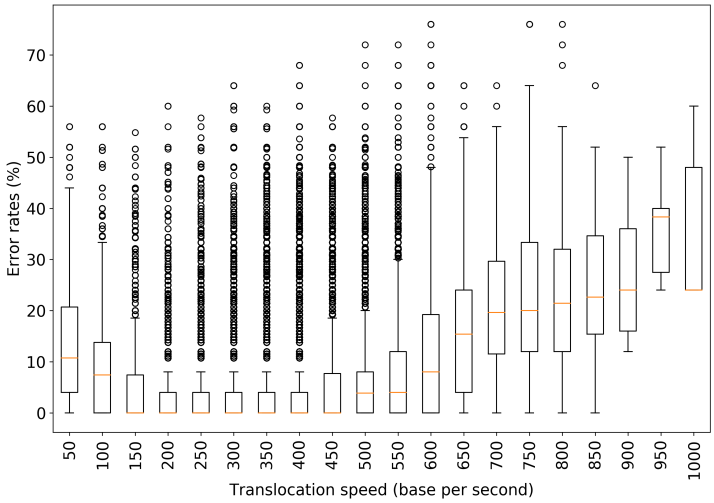


Figure 2.3 – **Higher deletion error rates are associated with an increased translocation speed.** We focus here on regions where long deletions (at least 5 bases) have been found in alignments. An increased translocation speed is linked to higher error rates.

genome and Y the corresponding nucleotide in the read (Figure 2.4). We found that, for bacteria as well as for the human dataset, substitutions between A and G occurred about three to five times more often than the transversion error. We also noted that substitution between C and T bases were also a bit more frequent than transversions. We performed two t-tests (one comparing A-G/G-A substitutions, the other comparing C-T/T-C substitutions against transversions) and found that transitions were significantly more frequent than transversions (p-value = 1.08e-78 and 1.48e-31 respectively).

This bias is likely due to the internal chemical similarities within the groups of purines (A and G), and pyrimidines (C and T), which are much higher than inter-group similarities. As a consequence, the electrical signals are harder to discriminate between two purines (two-ringed structures) or two pyrimidines (one ring).

2.2.4 Although being PCR-free, sequencing suffers for GC bias

PCR amplification is known for inducing sequencing biases depending on GC content of data [Benjamini and Speed 2012; Y.-C. Chen *et al.* 2013], and DNA sequences whose GC content deviates from 50% are likely to have lower depth of coverage. As ONT sequencing does not require PCR amplification step (except for certain cases where

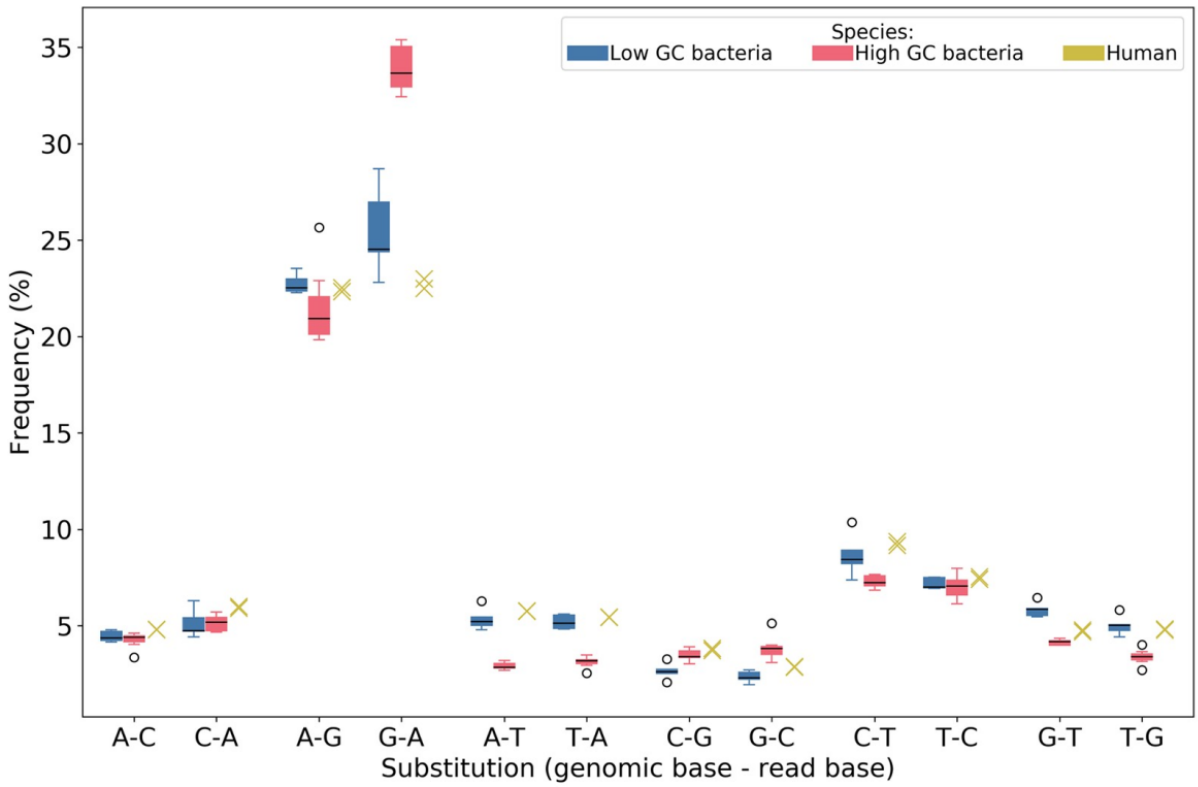


Figure 2.4 – **Substitution error profile for bacterial and human datasets, HAC basecalling mode.** Results for bacterial datasets are presented with boxplots, those for human are displayed with crosses. Bacterial species are grouped according to their GC content. Substitutions between A and G are prevalent, and substitutions between C and T are also significantly higher than the remaining ones.

input DNA is very low), one would expect these sequencing data not to be affected by such a bias, which has been confirmed in [Goldstein *et al.* 2019; Sevim *et al.* 2019; Browne *et al.* 2020]. However, these researches only focused on depth of coverage bias, and did not address the sequencing error rate.

We verified this by computing the relative coverage [Ross *et al.* 2013] of reads along the genome: for a sliding window of 100 bases, compute its coverage and divide it by the average coverage of all windows. We distinguished species whose reference genome's GC% is below 50% (called "low-GC species"), from those whose GC% is over 50% ("high-GC species"). Our results support the fact that ONT sequencing data are not subject to coverage bias due to GC content.

Nevertheless, we showed that AT-rich sequences were sequenced with more accu-

racy than GC-rich ones, with a difference of about 1.5% in the raw error rate (Figure 2.5).

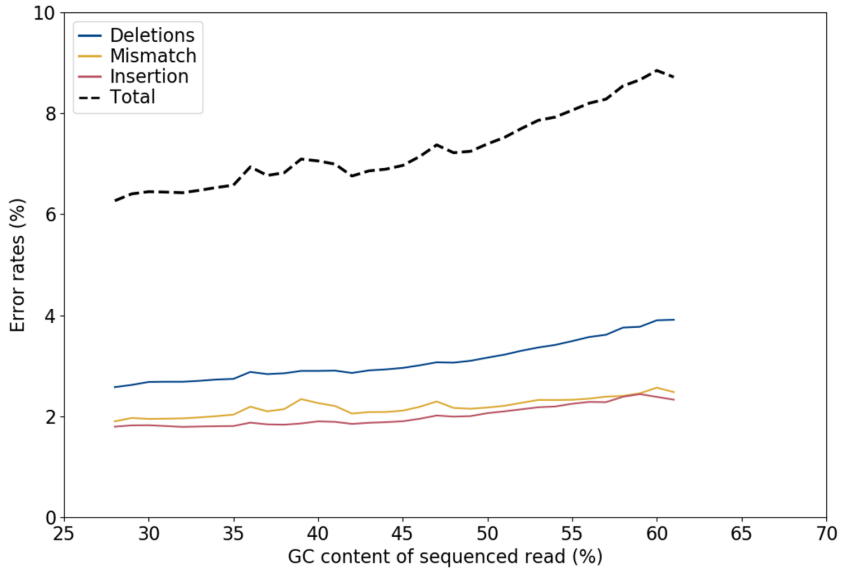


Figure 2.5 – **Error rate increases with %GC content of Nanopore sequenced reads.**

2.2.5 More insights on sequencing errors in low-complexity regions

ONT sequencing data are known to struggle when sequencing homopolymers, *i.e.* stretches of same bases in a sequence. Nanopore signal is mainly determined by five to six bases in the pore [Rang, Kloosterman, and J. d. Ridder 2018; Wick, Judd, and Holt 2019], thus as the homopolymer will be longer, probabilities that the basecaller mis-estimates the homopolymer length will increase.

We have focused on these errors in order to describe and quantify them more precisely. We found that about 47% of sequencing errors were linked to homopolymeric regions. We also analysed the sequencing errors in homopolymers, according to their expected length (Figure 2.6): short homopolymers were quite well sequenced (70% of them were errorless), but sequencing accuracy quickly drops, for example only one fourth of homopolymers of length 8 are sequenced without error.

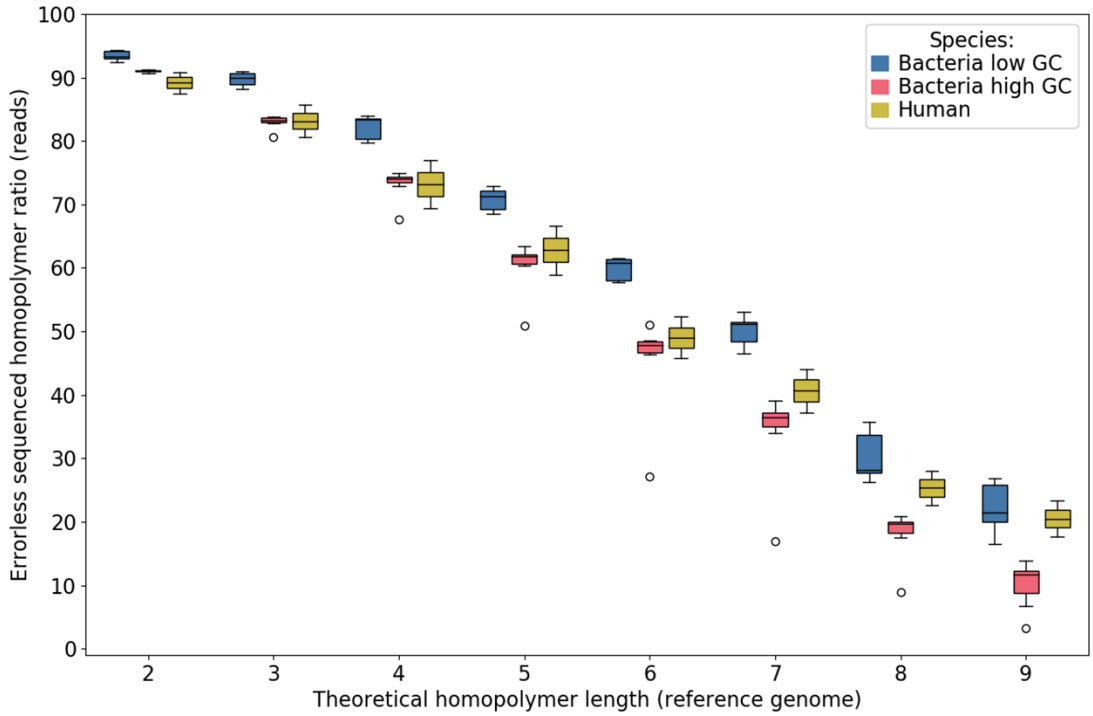


Figure 2.6 – **Accuracy in homopolymer drops with their length.**

2.2.6 The SeqFaiLR tool to evaluate errors in raw reads

As Nanopore technologies evolve and are updated on a regular basis, and no sequencing error profile of such data is available, we attempted to provide a simple tool to perform analyses of sequencing errors, to be able to witness changes between updates.

The SeqFaiLR (**Seq**uencing **Fai**lures of **L**ong **R**ead data) tool is available on Github (`https://github.com/cdelahaye/SeqFaiLR`). Users can apply it on any reads, but as SeqFaiLR has been primarily designed to analyze ONT sequencing errors, computed results may not be as relevant depending on the sequencing technology used.

SeqFaiLR has been designed to be run as simply as possible. There are a few requirements and they only concern classical python packages. The user only has to provide its sequencing reads (in fasta or fastq format) as well as a reference genome for each dataset, in dedicated folders, and to pay attention naming reads and reference accordingly (as the script will automatically associate a read file XYZ.fastq with the reference XYZ.fasta).

The first mandatory step is to "initialize", *i.e.* perform some preparation steps for fur-

ther analyses. The script offers to group species (group all, do not group at all, or create custom groups), so that results can be more readable if many datasets are present. The user can also specify the color associated to each dataset or group, to ease custom graph output.
Once initialized, SeqFaiLR computes alignments of reads against their references. The minimap2 aligner is used by default, but the users can bypass this and directly provide their own alignments with their favorite alignment tool.

Once the initialization and alignment steps are done, the user can choose which analyses to perform, among:

- quality analysis: compute relation between error rate and quality scores of reads compared to the expected Phred score (at both read and nucleotide level), as well as the quality score along reads;
- GC bias: compute the depth of coverage depending on the GC content of reads, and the error rates according to GC content of reads;
- analysis of low complexity regions:
  - homopolymers $(X)^n$: quantification of errors in homopolymer associated regions, error rates depending on homopolymer length, ratio of correctly sequenced homopolymer according to their length, difference between expected and sequenced homopolymer length;
  - heteropolymers $(XY)^n$: abundance in reference, associated error rates, difference between expected and sequenced length;
  - trinucleotides $(XYZ)^n$: abundance, sequencing accuracy.

To ease the running, the user can also simply run a single command to automatically execute all the steps described above.

SeqFaiLR automatically produces output results for each analyses: in raw plain text, as well as with plots (when relevant). An analytical reports including figures is also produced, to give a summary of all results computed.

2.3 How to deal with sequencing errors?

2.3.1 Core concepts for error correction

This part will present two major principles used to correct reads: building consensus sequence (Figure 2.7), and using k-mer frequency (Figure 2.8). Both methods are

based on the assumption that sequencing errors occur with a uniform distribution along the genome, and are expected to occur less frequently than correct nucleotides. These methods were first designed for short read data.

Consensus. The core idea is to exploit sequencing depth, *i.e.* the redundancy when a same genome region has been sequenced multiple times. First, reads have to be grouped according to the genome regions they stem from, which can be simply achieved by read alignments. For this task it is also possible to use UMI (Unique Molecular Identifiers) for NGS data, which are tags added before PCR amplification that identify reads in a same region. Once done, one can compute a multiple alignment of reads in each group. For a given position in the alignment, one expects that true nucleotides will be present more often than sequencing errors. Thus, a consensus is built by taking the most frequent nucleotide for each position. This approach becomes problematic when there is a tie between several bases. In such cases, there are several ways to proceed: rely on a reference sequence if available, exploit the quality score of each base, use the IUPAC alphabet [11] [Nomenclature 1970; Cornish-Bowden 1985] to represent all selected bases, randomly select one base, or do not select any base and label this position as unresolved.

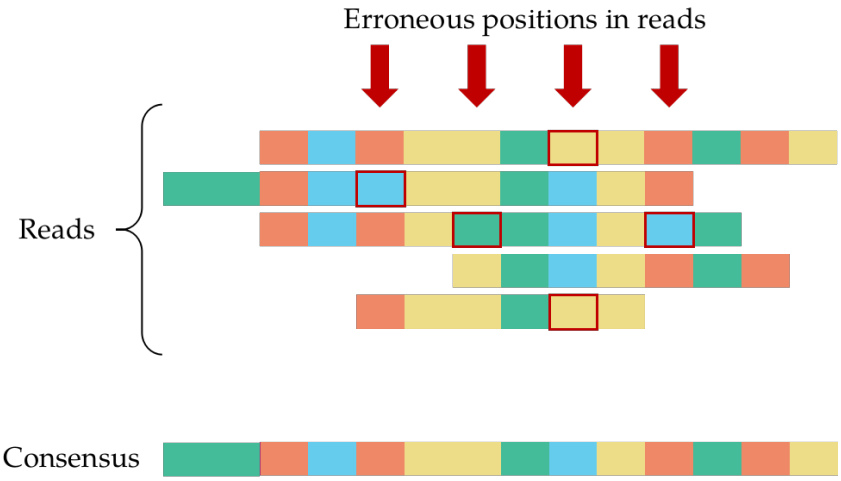


Figure 2.7 – **Principle of read correction by building a consensus sequence.** As errors (*in red*) are expected to be less frequent than correct bases, benches of reads corresponding to a same genomic region can be corrected by computing the majority base for each position of their alignment.

K-mer frequencies. Reads are first split into k-mers (*i.e.* words of length k). Here,

11. A degenerated alphabet that has been created to represent positions for which multiple bases can occur. For example "W" represent "A or T", and "D" represent "all but C"

one expects that k-mers containing a sequencing error will occur less frequently than an errorless k-mer. Hence, the idea is to set a threshold on the number of k-mer occurrences, above which a given k-mer will be considered reliable, otherwise it would be considered as likely to contain at least one error. Then, reliable k-mers are used to correct the erroneous ones.

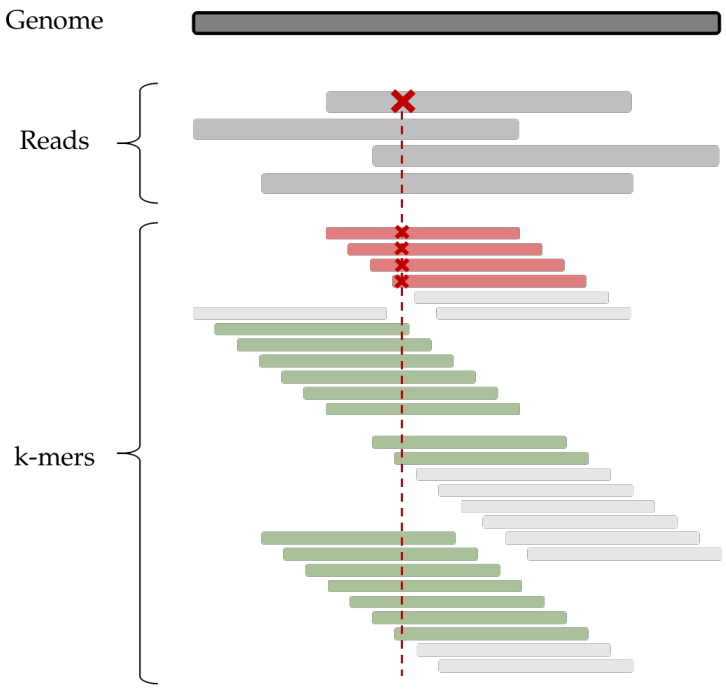


Figure 2.8 – **Principle of read correction based on k-mer distribution.** Erroneous k-mers (in red) are expected to be less frequent than true ones (in green). Based on k-mer occurrence distribution, erroneous k-mers can be detected, and corrected by replacing them with their most similar true k-mer.

Basecaller. Another way to improve sequencing accuracy is via the basecalling algorithm. This can be done either by implementing better machine learning algorithms, or by using a basecaller like Bonito to train the basecaller on custom datasets [Vereecke *et al.* 2020].

2.3.2 Raw read correction methods for long read data

Long read sequencing data have different error rate and profile from NGS short reads data: short read data mostly suffer from substitution errors and sequencing errors are mostly random; whereas long read data are mainly affected by insertion and

deletion errors, and systematic errors occur on homopolymer regions for ONT data. Therefore, dedicated correction methods have to be designed or adapted for long reads.

There are two main categories of error correction methods: *self-correction* and *hybrid correction*. The former uses long reads only, that are aligned against each other, and from which a consensus is derived and thus enables individual correction of reads. The latter uses both NGS and long read data, to take advantage of short read accuracy and long read larger spanning: shorts reads are used to create an assembly on which long reads are aligned and corrected. As the short reads have a lower error rate than long ones, the final accuracy of corrected reads is expected to be higher than for self-correction methods.

We will only present global strategies, and refer the reader to [Zhang, C. Jain, and Aluru 2020; Morisse, Lecroq, and Lefebvre 2020] for deeper comparisons between different correction tools.

Hybrid correction methods

As error rates of long read data were first really high, early correction methods required the use of accurate NGS reads.

The first proposed strategies were to align short reads against long ones, and then compute a consensus to correct them (implemented for example in Nanocorr [H. Lee *et al.* 2014]). Depending on the tool, some additional processing can be done to improve accuracy. For example, as long reads - especially ONT ones - can struggle in sequencing low complexity regions such as homopolymers, LSC [Au *et al.* 2012] introduced the use of homopolymer compression of reads. Any homopolymer made of n repetition of base X will be considered as a simple X, removing the difficulties to align reads with indels in such contexts. Finally, the alignment is updated according to the initial uncompressed sequence. Instead of computing a consensus from short read alignment on long ones, it is also possible to build an overlap graph from short reads. Then, look for optimal paths in the graph that minimize the differences with the long reads. This has been implemented in ColoRMap [Haghshenas *et al.* 2016]. However, this type of strategy heavily rely on which aligner is used, and can thus create bias or provide non consistent results depending on the chosen aligner.

The next generation of correction tools proposed to build an assembly from short reads and align long ones on it. The idea here is to compensate for the shortness of NGS reads, which can thus struggle to align on repetitive or highly erroneous regions

of long reads, by first creating contigs before performing the alignment step. Here again, different algorithmics choices are made depending on the tool. For example, ECTools [Goodwin *et al.* 2015] select alignments that best cover long reads. While some tools perform an assembly, others such as LoRDEC [Salmela and Rivals 2014] and FMLRC [J. R. Wang *et al.* 2018] are based on long read traversal of a short read dBG. Whereas LoRDEC looks for shared k-mers, FMLRC distinguishes itself by making 2 passes on the graph: a first one with small k value to correct most errors, and a second one with increased k value to correct repetitions in long reads.

A comparison of these correction tools has been done in [Fu, A. Wang, and Au 2019], based on several metrics: sensitivity, accuracy, time and memory efficiency, alignment rate and output rate. The graph-based algorithms had the best global results, with FMLRC showing the best performances.

Self-correction methods

Now that long read data have reached higher accuracy, particularly for PacBio, new correction strategies were set up, and do not rely anymore on short reads but only on long ones. This has the advantage of getting rid of the constraints (time, monetary, data storage, to name the main ones) to have samples of interest sequenced twice, with two different technologies.
Roughly, one can separate self-correction strategies into two main categories, both using graphs: based on read overlap, or using k-mers. Here again, for each strategy, existing tools may vary on some points, such as the aligner used, thresholds and parameter values, criteria to build the graph, but they rely on same core principle.

In correction methods using read overlap, the first step is to compute alignment between reads. This would be performed with aligners such as BLASR [Chaisson and Tesler 2012], BLAST+ [Camacho *et al.* 2009], MHAP (MinHash Alignment Process, [Berlin *et al.* 2015]) or minimap2 [H. Li 2018], by computing pairwise alignments between reads. The currently most used long read aligner is minimap2, which is based on a seed-and-extend strategy coupled with a seed filtering step. It has been designed to handle both the higher error rate of long reads, as well as their increased length which makes it impossible for other aligners to scale.
Computing alignments of long reads requires to allow non exact matches, which could lead to poor alignments. Some additional treatments can be done in order to filter out

spurious alignments. The correction module of Canu assembler [Koren *et al.* 2017] applies a *tf-idf* weight (term frequency inverse document frequency) during the mapping step, so that ubiquitous k-mers will have a lower relevance and k-mer being specific to one or few reads will have higher impact. MECAT tool [Xiao *et al.* 2017] uses a filtering of alignments in order to remove uninformative ones.
Once alignments are computed, a consensus will be derived from reads. This can be done by simply choose the most frequent base at each position as in Sprai [Miyamoto *et al.* 2014], or by finding highest scoring path with dynamic programming as for FalconSense [Chin, Peluso, *et al.* 2016]. While the further amounts to finding local solutions, using dynamic programming based enable to search the global optimal solution.

Correction methods based on building k-mer graph first create a dBG from long reads' k-mers. Then, long reads are mapped on this graph: those which can be anchored on the graph are considered as reliable. These reliable reads are then used to correct the remaining reads that could not be anchored.
The dBG can instead be built from reliable k-mers, based on an occurrence threshold, then anchor reads to the graph, and correct them by finding a path linking these anchor points, as implemented in LoRMA [Salmela, Walve, *et al.* 2017]. It is also possible to build a set of local dBG graphs to perform finer corrections, as in Daccord [Tischler and Eugene W Myers 2017].

It is worth noting that it is also possible to correct reads relying on the raw electrical current output by the sequencer. To the best of our knowledge, the only existing tool that implements this kind of strategy is Nanoreviser [L. Wang *et al.* 2020]. They propose a post-processing correction algorithm based on deep learning, relying on the raw signal to correct the errors made during the basecalling step. Yet, this approach is resource-intensive, so tools that are relying on already basecalled reads are preferred.

2.4 Conclusion

New sequencing technologies have been developed, aiming at producing significantly longer reads. Among the sequencers of third generation, two technologies can be distinguished: PacBio and Oxford Nanopore Technologies. The former has now reached really low error rates, especially with the HiFi reads, while the latter enables to produce longer reads and also offers a compact sequencer that widens the possible applications

of sequencing process. They both currently produce reads of several kilo base pairs, and their improvement coupled with correction algorithms contributes to reach error rates that are approaching those of NGS sequencers.
Having access to such long reads is a clear asset for problems such as genome assembly, to overcome issues related to repetitive regions for example, and it is also a valuable information in the context of haplotype phasing, as longer reads can cover more variant loci (see chapter 3).

Nevertheless, long reads still display a high error rate, which results in difficulties in the detection of similarity between reads, or in the distinction between true variants and sequencing errors. The PacBio and ONT technologies have different error profiles, the latter being more prone to systematic errors. Numerous correction tools have been developed for long read (a tool, ELECTOR, [Marchet *et al.* 2020] has even been designed in order to compare and evaluate their performances). Better results are obtained with hybrid corrections, but the need of double sequencing is a costly strategy, while self-correcting methods are improving and now enable to obtain satisfying results.
Further improvements of sequencing accuracy can be achieved through chemistry, for instance with the recent R10 flowcells of ONT, which promise a better sequencing of homopolymer regions and an overall reduction of systematic errors. Basecalling algorithms are also a key factor to enhance sequencing quality without having to improve the chemistry: basecalling the raw electrical signal of an old dataset with a more recent basecaller leads to an increase in read accuracy. Finally, it appears that exploiting the raw electrical signal of ONT reads would be a relevant approach for read correction, but this has barely been explored for the moment.

Chapter 3

Assembly and phasing of long reads

Contents

Preamble: In this chapter, we will focus on the haplotyping problem and existing methods to handle the associated challenges. We will start by a quick review on assembly methods, as the two problems share similarities, pointing out relevant principles that could be applied for haplotyping, as well as those to avoid. After an overview of experimental and computational existing ways of performing haplotyping, we will focus on those based on sequencing data. We will go through the state of the art of haplotyping algorithms, pointing out the complexity of the problem and challenges to overcome.

3.1 Assembling a genome from long read data

Genome assembly consists in building a genome sequence from reads. These contain varying amounts of sequencing errors which must be fixed. The goal of the assembly process is to organize reads in order to: 1) group reads belonging to a same region (enabling to correct them), 2) sort the reads by their position along the genome based on their overlaps and 3) build a single sequence from the analysis of the redundancy of sequences originating from the same region.

Methods for genome assembly are based on the same core principle, that is, to rely on a similarity graph over reads to recover the genome sequence. We present here the major approaches used for assembly, centering around the Overlap-Layout-Consensus (OLC) three steps method, while presenting specificities and minor improvements of several recent assemblers.

The OLC paradigm was introduced to process the read datasets of the first generation of sequencers. It has then been replaced by a simpler treatment with the emerging of NGS data, due to the difficulty to scale assembly to their high throughput. There are now hybrid methods that explicitly reintroduce an OLC component, because of their ability to take advantage of long reads.

For a more practical insight into the common assemblers available, we refer the reader to [Wick and Holt 2021] for a benchmark of recent assemblers, focused on bacterial dataset, and the work of [Vaser and Šikić 2021] providing an evaluation of major long read assemblers on human datasets for both ONT and PacBio datasets.

3.1.1 Define a graph of similarity on the set of reads (Overlap)

The first step of genome assembly is to compare pairs of reads and associate them on the basis of the observed similarity. This could be done by computing alignment, *i.e.* finding the exact differences between two sequences (Figure 3.1 A). Alignment has originally been used to retrace the evolution and mutations from an ancestral sequence to a derived one. This principle can also be applied in the context of genome sequencing, for which a read represents an initial sequence that has undergone some sequencing errors. In any case, the alignment is driven by the parsimony hypothesis. Performing a base-to-base alignment of two sequences of lengths n and m uses dynamic programming with a classical solution using $O(nm)$ complexity in time and space.
Though, it is not necessary to compute full alignments for assembling a genome: overlaps bring enough information, and require less computational resources (see below). Finding an overlap between two sequences means to look for the largest region on which they are locally matching. The rationale is that if the suffix of one read is similar enough to the prefix of another, they are likely coming from a larger sequence, so they can be merged.

In the context of assembling reads, the main overlap types of interest are those where a sequence's suffix is matching another sequence's prefix, as it basically enables to expand the assembled sequence. Still, other overlaps are also useful as they provide redundancy and can be used for correction (Figure 3.1 B).

The Overlap stage represents the main computational cost of the overall assembly method (it can take up to 95% of computation time in some assemblies [Berlin *et al.* 2015]), partially caused by the treatment of sequencing errors and repetitive regions.

Scaling for the quadratic search of overlaps

Considering a set of N elements, it requires $O(N^2)$ comparisons to establish all similarity links between them. Applying such strategy for genome assembly would make this step quadratic in number of reads, which is not suitable for the amount of reads (generally millions) generated in a sequencing project. There are two ways to reduce the computation time associated: narrowing the number of candidate pairs of reads, and efficiently determine each overlap.
Reduce the set of candidate pairs. Given a set of sequences, one can determine a set of credible candidate pairs of reads which are likely to overlap, as most of read pairs are not

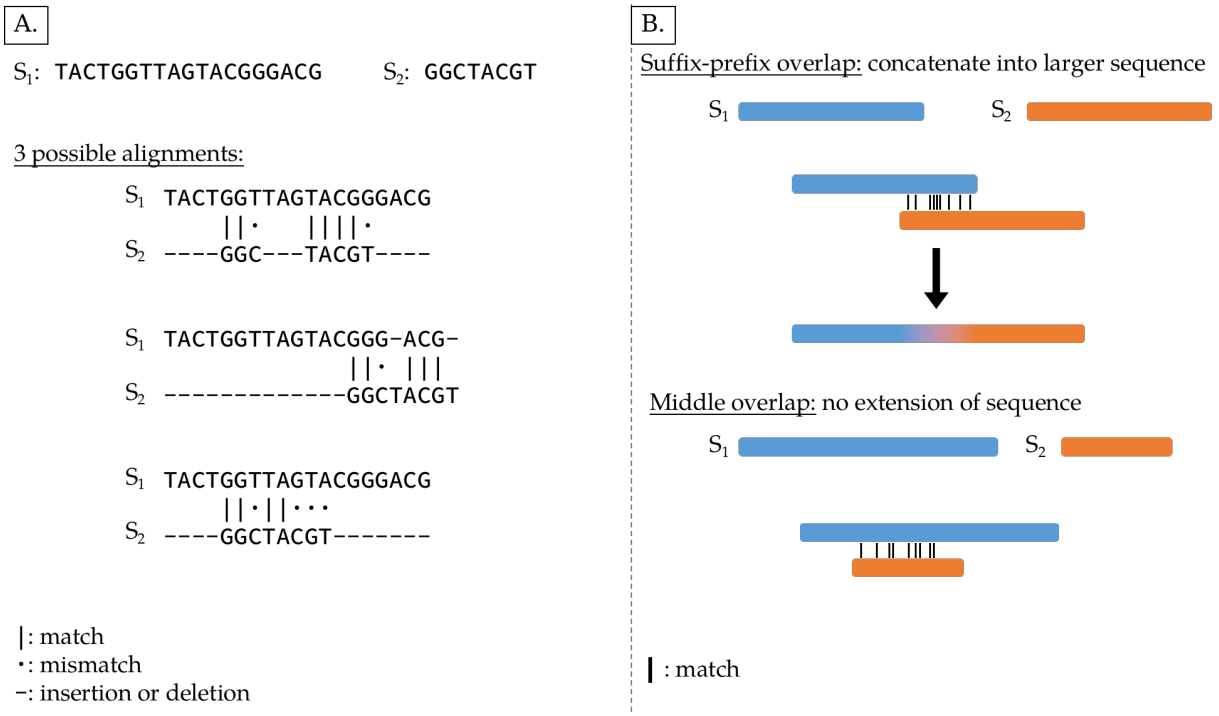


Figure 3.1 – **Alignment and overlap principles. A-** Aligning two sequences consists in finding their base-to-base differences. Depending on the parameters set, one can obtain different acceptable results. In these examples, the first one authorizes long insertions/deletions (indels) between blocs of matches, the second one favors short indels to keep the aligned part compact, and the last one prohibits any indel. **B-** Searching an overlap between two sequences limits to find the regions for which the sequences are locally similar. If the overlap is located at the end of one sequence and the beginning of the other one, then they are assumed to originate from a larger sequence and are concatenated. If the overlap is in the middle of one sequence, then it cannot directly be used for assembly by extending sequences, yet it provides redundant information which can be used for sequence correction. In overlaps, the matches are defined on a larger scale than just base-level (*e.g.* with seeds).

worth computing an overlap. It is based on the research of common short exact words, *e.g.* k-mers, called *seeds*, between sequences. Not all k-mers of a sequence are used as seeds, but only the most relevant ones (*e.g.* based on their frequency), as the time and space complexities of the search depend on the number of seeds kept. Assessing if a sequence of length m is an overlap candidate for a sequence of length n can be done in $O(n+m)$ in time and space: traverse one sequence and store (in $O(1)$ per k-mer) each seed found in a hashmap, then browse the second sequence likewise and for each of its k-mer test if it is present in the hashmap (in $O(1)$). The use of indexing data struc-

tures also reduces computation time, as the set of seeds has to be computed only once per sequence. If two sequences share a sufficient amount of seeds, one assume they overlap.

Efficient computation of an overlap. Once overlap candidates are established, one can estimate the length of the overlap between two sequences as well as its position, enabling to order the sequences. To do so, one looks for a subset of common seeds that are colinear (*i.e.* the order of the seeds is the same in the two sequences): their positions give an estimate of the overlapping region (Figure 3.2).

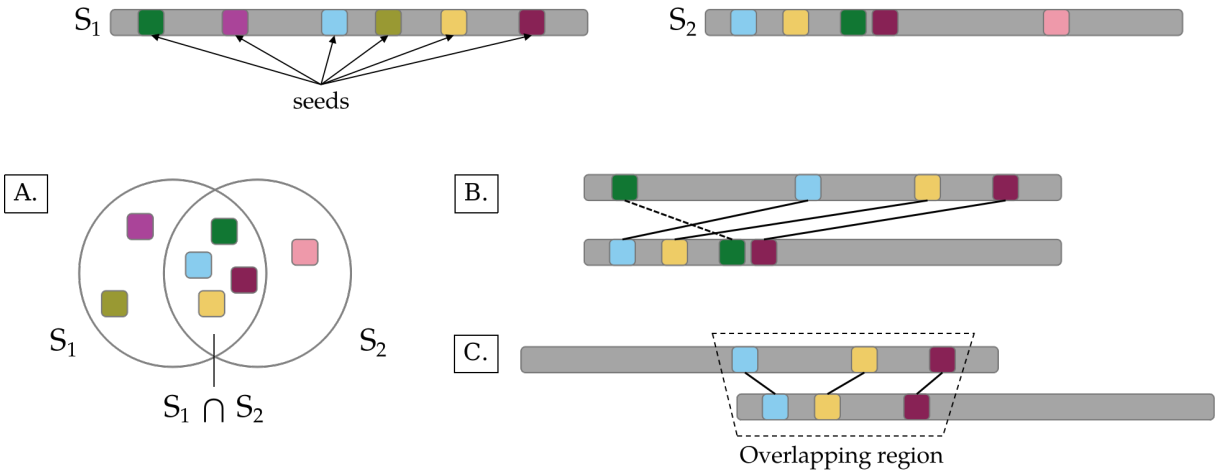


Figure 3.2 – **Estimation of the overlap between two sequences, using seeds. A-** The sets of seeds of both sequences are compared: if they share enough seeds (*i.e.* if $S_1 \cap S_2$ is over a defined threshold), then the overlap is computed. **B-** Considering only the set of shared seeds, one computes their position on each sequence, and evaluates their colinearity. In this example, three seeds occur colinearly in both sequences (plain line), but one does not, so it is ignored in the next step (dashed line). **C-** The positions of seeds give the length and position of the overlap, and the ordering of sequences. An estimation of their similarity can be computed based on the number of shared seeds.

Having a set of shared seeds between two sequences, one can then use a *seed and extend* strategy to compute a full alignment and obtain more precise information on reads similarity. Instead of computing a full base-to-base alignment, it uses the set of common seeds to establish anchor points where the sequences share exact short subsequences. This way, unmatching regions can be skipped from alignment computation. Then, the alignment is extended from these anchors. The seed-and-extend has been implemented in the well known BLAST aligner [Altschul *et al.* 1990], but also in more recent aligners such as Celera [Denisov *et al.* 2008] or Canu [Koren *et al.* 2017].

The principle of seeds has been later improved with *min-mers*: the set of used seeds is reduced to the set of their smallest [1] k-mers, forming a fingerprint of a given sequence. Then, reads similarity can be estimated by computing a distance between their fingerprints (which can be sped up by using hash functions), reducing the set of candidates. It has been described as the basis of the MinHash algorithm, then used in alignment algorithms like MHAP [Berlin *et al.* 2015] and minimap [H. Li 2016]. It is also possible to limit the number of selected seeds, as for the Raven assembler [Vaser and Šikić 2021] which bounds the number of seeds to $|read|/k$ (with k the seed length), which enables to reduce computation time and space while keeping enough accuracy.

Dealing with sequencing errors

As described previously, it is quite straightforward to find an overlap (if it exists) in the case of error-free sequences. However, in practice, reads may contain abundant sequencing errors. Consider a sequence of length L with an error rate of ϵ. In the worst case, these errors are evenly distributed along the sequence, and the maximal possible seed size so that not all seeds contain error is $\frac{L}{\epsilon+1} - 1$. The more errors in the sequence, the shorter the seeds have to be to ensure having at least some error-free seeds.
Another solution to deal with erroneous sequences is to release the constraint of exact matching (*e.g.* allow matches with at most one difference). This has led to the development of alternative seeds: the *spaced seeds* [Ma, Tromp, and M. Li 2002], allowing one or several mismatches on some fixed positions but no indel, making them more appropriate for conserved protein coding sequences, than other genomic regions; then the *indel seeds* [Mak, Gelfand, and Benson 2006], explicitly allowing insertions and deletions, improving the sensitivity on long read data, which are prone to such errors [Sović *et al.* 2016]. However, when using such seeds, the number of false positive matches is higher and the number of overlap candidates to examine increases, as one will also capture matches between more distant sequences.
When the sequencing error profile is known, aligners can adapt their strategies to better handle systematic errors. In the case of ONT reads, homopolymer regions are known to be a significant source of errors as their length is often misjudged (see chapter 2). A possible adaptation strategy is to compute alignments on an *homopolymer-compressed* version of reads, using the same principle as the run-length encoding: any homopolymer of a base X repeated $n \geq 2$ times is replaced by a single X base. This avoids missing

1. Several ordering are possible, such as the lexicographic order, or hash values.

an alignment in regions where errors are due to a wrong estimation of homopolymer length. Reads are decompressed afterwards to recover original sequences and the alignment is updated accordingly. Homopolymer compression is used in aligners such as minimap2 [H. Li 2018] and LSC [Au *et al.* 2012] as well as in the Shasta [Shafin *et al.* 2020] assembler.

Dealing with repeated regions

A final major challenge when assembling a genome is the treatment of repeated sequences. The presence of such sequences increases the number of matching reads in the alignment process, as two reads sharing a repeated subsequence can be considered as similar. With strategies using seeds to select candidates, it is quite straightforward to discard read pairs that are poorly similar. But when sequences share a repetition, it will be harder to distinguish truly similar sequences from those that only share the repeated part. Using larger k-mers for repeat discrimination conflicts with the need of short k-mers on erroneous sequences as mentioned previously, and does not enable to handle repetitions wider than the value of k.

A first radical way to handle repetitions in assembling process is simply to ignore them (the early Celera [Eugene W. Myers *et al.* 2000] assembler, now replaced by Canu [Koren *et al.* 2017], used a database of known repetitions to mask or to tag them). However, it amounts to deleting information, leading to incomplete assemblies that are fragmented at repetitions.

Finer strategies have been developed afterwards, where assemblers aim at including repeated regions in their computation. K-mers that are included in a repeat have a higher number of occurrences than other k-mers do, so they have more chances to be selected as seeds than random k-mers. Yet, a repeated subsequence carries less information for sequence discrimination as, by definition, it is present several times in different genomic region that are likely to be wrongly considered as close. To limit the influence of k-mers associated with a repeat, an adjusting weight on k-mers can be applied to guide seed selection, such as the *tf-idf* ("term frequency, inverse document frequency"). This weight has been originally used in text-mining to find subsets of similar texts, based on the following idea: in a corpus of texts, high-frequency words do not help to discriminate texts, whereas a word that is occurring several times in a subset of texts is a good indicator of their similarity. The tf-idf weight measures how often a k-mer is present in the whole set of sequences (term frequency), divided by the number of

sequences in which it appears (inverse document frequency), giving more importance to k-mers occurring in few sequences. This has been implemented in Canu [Koren *et al.* 2017], which applies the tf-idf weight on k-mers before seed selection.

Finally, overlaps and similarity relations between reads are integrated within a graph where nodes are reads and edges connect sufficiently similar reads, which will be the input of the remaining assembly steps.

Speeding up the Overlap step by working at the k-mer level

Whereas the de Bruijn Graph (dBG) and OLC approaches are often depicted as different, they actually share the same methodology (although due to their structure, some operations are specific to one or the other). Their core difference lies in the choice of the similarity graph: while OLC uses overlaps between reads, dBG is based on k-mer overlaps. But the following steps of the assembly are highly similar (*i.e.* cleaning the graph then finding a path, see next sections).

A dBG graph is built as follows (Figure 3.3 A): each read is split into k-mers using a sliding window, and the k-mers form the edges of the graph. This structure enables to considerably speed up the Overlap step, as finding a successor of a given node has a constant cost (proportional to the alphabet size, 4 in the case of genomic data), while with a read overlap graph it is quadratic in the number of reads.
The dBG graph also has a greatly reduced impact on memory, as the number of nodes is proportionate to the genome length, whereas in usual OLC paradigm it is function of the read number.

However, dBG comes with several limitations. The first one is due to sequencing errors. Each error generates k erroneous k-mers in the graph, and can thus have great impact on memory. This problem is generally easily solved by filtering out k-mers based on their number of occurrences, as one expects erroneous k-mers to be less frequent than correct ones. The second one, more critical, is related to repetitive regions. The general approach to retrieve the initial genome sequence from a dBG is to look for an Eulerian path in it, something that is not compatible with the existence of repetitions (one will have to go at least twice through the same edge or set of edges): either such path will not reflect the actual structure of the genome, or the graph will not have Eulerian path at all (Figure 3.3 B). Yet, it is possible to use a more flexible definition of the dBG to overcome this issue. For instance, a generalized eulerian path [Pevzner, H. Tang, and Waterman 2001], consisting in finding the shortest non cyclic path going at least once

through every edge.

The choice of k is crucial here, and embodies a trade-off between abundance and specificity of k-mers. Smaller values of k lead to k-mers with higher frequency, thus filtering out erroneous ones will be easier, but repetitive sequences will be harder to distinguish. Larger values of k lead to many unique k-mers, which eases handling of repetitive sequences (although still representing an issue as large values of k is only of the order of a few tens), but also causes less contiguity in the graph.

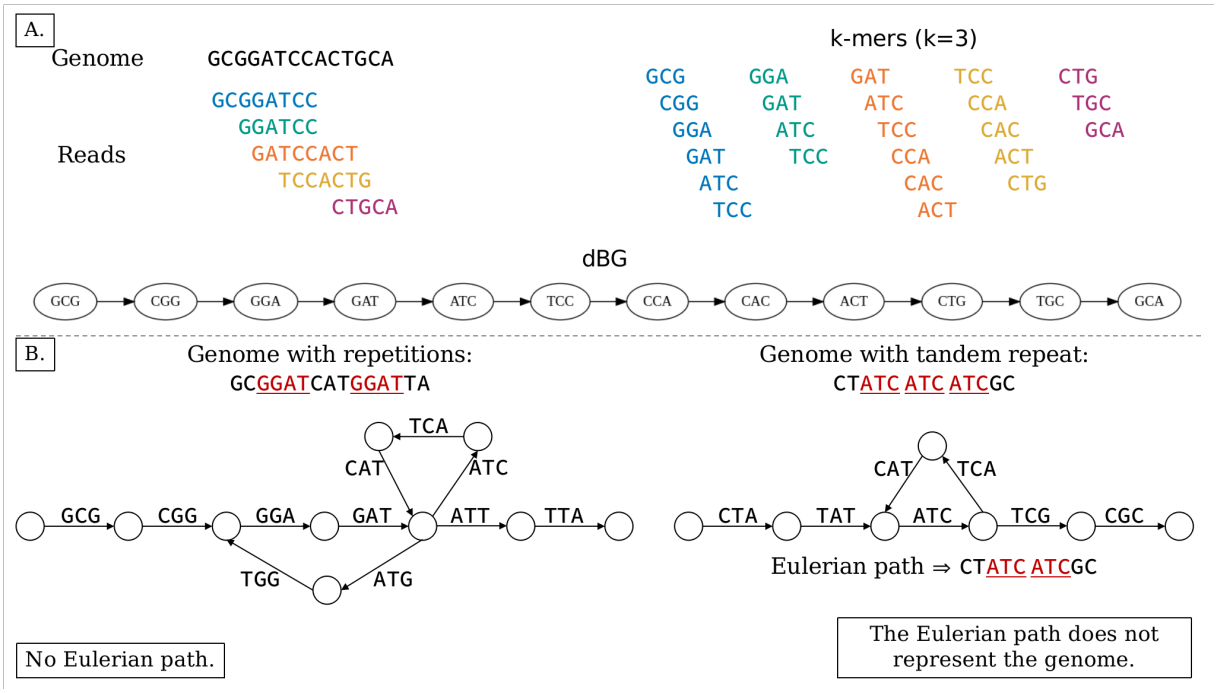


Figure 3.3 – **dBG principles for genome assembly. A-** From a set of reads, one extracts its set of k-mers (here we set k to a very low value for the sake of simplicity). Then, the dBG is built with k-mers as edges. Here, the Eulerian path on the graph gives the initial genome sequence. **B-** In the case of a genome containing repeated regions, finding an Eulerian path may not be suitable, as such path may not exist, or may not reflect the repetitive nature of the genome.

Assembly based on dBG structures has been implemented in tools like Redbean (formerly wtdbg2 [Ruan and H. Li 2020], using a fuzzy dBG which enables inexact matches between k-mers to handle erroneous reads), or Flye [Kolmogorov *et al.* 2019] (based on a repeat graph, a generalization of dBG for approximate matches, that enables to display and handle genomic repeats).

The dBG can be combined with compact structures that enable to further accelerate

the assembly, such as Bloom filters or other indexing structures to perform efficient queries on nodes successors. A Bloom filter is a bit array of fixed size, initially set to 0. To add an element to the filter, one uses h hash functions to compute h hash values and sets the corresponding positions to 1 in the array. To test if an element is present in the filter, one checks if all associated positions get 1 values. No false negative is possible using this structure, but there is a risk of collisions [2] causing false positives. When applied to assembly graphs, only node positions are stored. To know which edges are present in the graph, one queries for the presence of each potential successor node (4 in the case of DNA). The use of Bloom filters requires little memory resources as one does not directly store elements, and the array size is fixed. It is also efficient since adding an element and checking if one element is present is done in constant time $O(h)$, with h fixed, and can be parallelized as the h operations are independent.

3.1.2 Simplify the similarity graph (Layout)

The similarity graph built in the previous step cannot be used as is to recover the genome sequence since it generally contains redundant information or artifacts caused by sequencing errors and repetitive regions. This step aims at cleaning the similarity graph by removing the corresponding spurious nodes or edges in the graph. The objective is twofold: to simplify the graph, and to perform some corrections on it.

We describe here the three cleaning operations that are used in assemblers. The first one consists in removing useless redundant information to lighten the graph and reduce computation time and memory footprint. The two others aim at identifying incorrect patterns caused by errors.

Transitive reduction

In a similarity graph, transitive edges represent redundant information already carried by other edges, and can thus be safely removed from a graph without losing any information.

The goal here is to keep the smallest subset of edges so that the resulting graph still has the same reachability as initially: for each pair of nodes, if a path existed in the original graph, then a path should exist in its transitive reduction. For a graph with

2. The inherent structure of the filter implies a non-zero risk of error, no matter the amount of memory allocated.

n nodes and m edges, finding its transitive reduction has a time complexity of $O(nm)$ (and of $O(nm)$ in space in the general case, although recent works showed a $O(n^2)$ space complexity in practice [X. Tang *et al.* 2020]).

One can use overlapping length information to detect transitive edges: in case of transitive overlap between two nodes u and w, with existing edges (u, v) and (v, w), then it is likely the two latter will have stronger overlaps, whereas the first one will carry only a degraded information. In case of ties, one can use the identity rate.

Trimming tips

If a node (being a k-mer or a read) contains errors at one end, it will impact its similarity with other nodes, and may create an artifact (a *tip*) in the graph. More precisely, the presence of errors at the starting (resp. ending) extremity of a node impacts its predecessors (resp. successors).

One defines a *tip* as follows. Consider a node with an out-degree of at least 2 (the node is *branching*), and one of the branches contains only a few nodes, the last one being a dead end (*i.e.* a node with null out-degree). This short and dead-end branch is called a tip (see Figure 3.4).

Note that the length of a tip can be quite arbitrary, and that k-mers are more impacted by tips than reads, as only one error in the k-mer is enough, while the number of errors to generate a tip depends on the similarity defined between two reads.

Such structures clearly indicates an erroneous part of the graph, which can be detected and pruned quite easily.

Figure 3.4 – **A tip in a similarity graph.** Considering a set of overlapping reads, with one (c) having enough errors to miss an expected connection to the d, e and f reads. This configuration leads a tip in the similarity graph, that has to be removed.

Popping bubbles

If a node contains errors that are not located at an end, but inside the k-mer/read, it will create another type of structure in the graph: a *bubble*. However, bubbles are also created when a variant is involved.

A bubble is defined as follows. The graph contains a branching starting node N_s, of out-degree n, from which start n paths $P_1, ..., P_n$ of variable lengths, that are merged again in an ending node N_e of in-degree n (Figure 3.5). Popping a bubble consists in keeping only one of the n paths, so that this region of the graph becomes non branching.

Such structures are more complex to handle than tips, as one does not know *a priori* if the bubble is caused by an error (and should then be popped), or is due to a variant (thus carrying valuable information). Under the assumption of random errors occurring evenly along the genome, a criterion based on the coverage can be used to discriminate variants from errors: a variant is expected to be found in several reads at this position, while an error is not.
In the case of a diploid or polyploid organism, the presence of different haplotypes is reflected in the branches of a bubble. Popping a bubble then amounts to keep a single haplotype. Considering several bubbles, chances are high to select different haplotypes in each one, resulting in chimera made of fragments of haplotypes.
Therefore, a trade-off has to be set: popping every bubble increases graph contiguity but loses information, while keeping all of them causes fragmentation in the graph.

Figure 3.5 – **A bubble in a similarity graph.** Bubbles are caused either by errors or by true variant in reads. One can rely on the coverage information of such reads to assess whether it is a variant or an error (not illustrated in this figure). They cause alternative paths in the graph, and thus can be removed to gain contiguity, or kept to gain knowledge on variants.

Some additional cleaning steps may be performed depending on the assembler. The assembler Raven deletes long range edges due either to repeated sequences or sequencing errors. The assembler Shasta bounds the degree of the nodes (maximum number

of reads similar to it), a way of limiting long range edges caused by repetitive regions. Canu uses coverage information to remove regions with low support, and detect spurious regions that are likely adapters (used for the sequencing process) and chimera.

3.1.3 Obtain final sequences from the graph (Consensus)

Once the assembly graph has been created and cleaned, a simple and straightforward method to build the final sequences, and perform a correction step, is to take the consensus at each position (the most frequent base) of all reads that overlap on this position.

This step can be fully integrated to the assembler, as for Shasta, for example. It can also be performed by a dedicated tool, yet, some being only meant to be used on specific data: for instance Nextpolish [Hu *et al.* 2020] has been developed for NextDeNovo assembler, Quiver [Chin, Alexander, *et al.* 2013] and Nanopolish [Loman, Quick, and Simpson 2015] are designed respectively for PacBio and Nanopore sequencing data. To overcome this compatibility issue, the consensus tool Racon [Vaser, Sovic, *et al.* 2017] has been designed to be suitable to any assembler (as it uses standardized file formats as input), and has been optimized to perform efficient consensus. It is used for instance after the minimap/miniasm and Raven assemblers.

Computation of the consensus step can be optimized, for example using a Partial Order Alignment graph (POA, from a method described in [C. Lee, Grasso, and Sharlow 2002]). It was first used for proteins comparison. In a POA, each character forms a node, and two nodes are bound by an edge if they are consecutive in a sequence. Such representation enables to identify regions where consensus correction has to be performed, which are branching regions and bubbles. The POA representation also enables an efficient computation of multiple alignments: to compute alignment of two sequences of length N and M, dynamic programming has time complexity of $O(N.M)$; while aligning a sequence of length M on a POA graph has complexity of $O(\bar{n_p}.M.|V|)$, with $\bar{n_p}$ the average number of predecessors per node and $|V|$ the number of nodes in the graph [C. Lee, Grasso, and Sharlow 2002].
The consensus tool Racon, as well as the CONSENT tool [Morisse, Marchet, *et al.* 2021] tool (designed for long read correction in addition to assembly polishing), are based on the use of partial order graph for consensus computing.

3.1.4 Additional step: correction

While most assembly tools are relying on the three step OLC paradigm, they often perform an additional step, especially when working on long reads, which enables to improve assembly quality by correction. It can be done either before the Overlap step, then called *correction*, or after the final Consensus step, then referred to as *polishing*.

Not all assemblers handle errors in the same way: some (Falcon [Chin, Alexander, *et al.* 2013], Canu) perform read correction *prior to* overlap computation, while others are awaiting the *end of the assembly* process to polish the assembly (miniasm [H. Li 2016], Shasta [Shafin *et al.* 2020], Raven [Vaser and Šikić 2021], Flye [Kolmogorov *et al.* 2019], wtdbg2 [Ruan and H. Li 2020].
The former strategy eases read comparisons, and leads to more accurate and contiguous assemblies. However the correction step has a high computational cost, as the assembly information is absent and corrections cannot be restricted to small genomic regions. The later requires in contrast less resources but results in assemblies of lower qualities since it uses erroneous reads.

Different long read correction strategies exist, and either rely on additional short read data (*hybrid correction*), or produce a *self-correction* of the long reads. While self-correction tools are based on computing a consensus from an overlap or a de Bruijn graph, hybrid correction tools take advantage of the low error rate of short reads by aligning them on long reads (LSC [Au *et al.* 2012], Proovread [Hackl *et al.* 2014] or FCTools [Goodwin *et al.* 2015]), or by aligning long reads on a de Bruijn graph constructed from short reads (LoRDEC [Salmela and Rivals 2014], Jabba [Miclotte *et al.* 2016] or FMLRC [J. R. Wang *et al.* 2018]).

3.1.5 Metrics for assessing assembly quality

Several strategies exist in order to evaluate an assembly. Some can be computed directly from the assembly, while others need a reference sequence to compare with. The latter depicts more precisely the quality of the assembly, but requires to have a reference of sufficient accuracy - which is often not the case especially for non-model organisms - and poses the risk of biasing the evaluation towards the reference.
One can classify the metrics depending on whether they characterize the performance or the imperfection of assembly.
Criteria to maximize:

- total length of contig;
- length of the largest contig;
- ratio of bases of the assembly aligning on the reference;
- N50, representing assembly's contiguity: the largest contig length such that 50% of nucleotides are contained in contigs of length $\geq$ N50.

Criteria to minimize:

- number of contigs;
- $|\ \%GC_{reference}$ - $\%GC_{assembly}\ |$;
- number of missassemblies, mismatches and indels (can be computed with the dnadiff [3] tool) compared with the reference.

Such classical metrics are computed with tools like QUAST [Gurevich *et al.* 2013]. Other evaluation methods focus on more specific statistics on the assembly such as the gene content (*e.g.* Busco [Simão *et al.* 2015]) or k-mer content and distribution (Kat [Mapleson *et al.* 2017], Yak [4] and Merqury [Rhie *et al.* 2020]).

The authors of the Inspector software [Y. Chen *et al.* 2021] provide an up-to-date set of metrics for a detailed overview of a given long read assembly as well as a correction module to further improve the assembly.

3.1.6 Conclusion

Genome assembly strategies are built around a common core: create a similarity graph, then clean it and derive the assembly sequence or contigs. Existing methods now achieve satisfying assemblies from long read data only. However, the best assemblies are obtained from *hybrid* assemblers, that benefits both from the long range information of long reads and accuracy from short read data.

Yet, assemblies still suffer from a major issue, as they only build monoploid sequences and thus cannot represent variants nor haplotypes. Therefore, the resulting consensus genome sequence actually represents a chimera, and does not reflect the genetic reality of an individual or a species, in particular masking epistatic interactions within haplotype.

In the next section, we describe the main methods (both experimental and computational) that have been developed to have access on the haplotype information.

3. `https://github.com/marbl/MUMmer3`
4. `https://github.com/lh3/yak`

3.2 Overview of experimental and computational haplotyping approaches

As discussed previously as well as in the chapter 1, there is a need for assembled genomes taking into account haplotype information. There exist several techniques, either from wet- or dry-lab, to obtain to the haplotype information. We present here a quick overview of such methods, then propose a formalization of the haplotyping problem, as well as the associated challenges.

3.2.1 Experimental and computational approaches for haplotyping

Several techniques are employed in order to access haplotype information at different levels. Here, we will limit ourselves to the major ones, describing their principles and limitations, and we refer the reader to [Klau and Marschall 2017; Huang, Tu, and Lu 2017] for more detailed reviews.

Physically separate haplotypes

In cells, when DNA is condensed in shape of chromosomes, haplotypes of a same chromosome set are naturally separated, within physically distinct chromosomes. Hence, being able to segregate each chromosome would give a straightforward way to isolate each haplotype, before performing usual DNA extraction and sequencing.

Several methods enable to physically separate chromosomes, based on the natural condensed state of chromosomes in specific stage of cell division, then applying chemical treatments or laser pulses for example. Chromosomes (and thus haplotypes) are then sequenced separately. It is then possible to find which sequences constitute the haplotypes of the same chromosome by comparing sequences length and similarity. These methods require complex experiments and specific material, thus making them quite expensive.
With more details, such methods include, but are not limited to:

- *Pulse field electrophoresis* (*PFGE*), which separates chromosomes based on their molecular weight using a gel support in an electric field periodically changing direction (initially described in [Schwartz and Cantor 1984], and more recently reviewed in [Goering 2010]). It is though limited by the length of migrating fragments, which restricts its application to bacterial genomes. It is mainly used for

detection of pathogenic strains (such as for Listeria [Dalmasso and Jordan 2014]).

- *Microfluidics*: metaphase chromosomes are physically partitioned by passing through micro-scaled channels. Once separated, chromosomes can be sequenced separately [Fan *et al.* 2011]. A major limitation is that it requires a cell in metaphase, which excludes specialized cells such as skin cells, red blood cells, or neurons, that do not undergo mitosis anymore.
- *Microdissection*. Once performed with a glass needle [Scalenghe *et al.* 1981], and now with the help of lasers [Monajembashi *et al.* 1986], it consists in manual (eventually guided by computer) dissection to separate the chromosomes. This technique also requires metaphase chromosomes, and involves delicate manipulations.
- *Flow cytometry*. Chromosomes can be separated using flow cytometry techniques [Kuderna *et al.* 2020], which move cells through a laser beam to analyze the optical features of the sample. Yet this requires expensive equipments, and still requires metaphase chromosomes.

Use genotype information

Genotyping consists in obtaining, for a given sample, alleles present in certain genes of interest, but without knowing which one is related to which haplotype. Such information is quite easy and cheap to access, with the use of micro-arrays for example [Dasari and Alex 2014]. It is not possible, for a single individual, to infer the haplotype from genotype data, as such information only provides a mix of variant values, independently for each variant position (Figure 3.6). However, it is possible to exploit other data such as a pedigree genotype, or apply some statistical models on large genotype datasets, to obtain haplotypes.

Pedigree based. It relies on exploiting genotype information from relatives, the smallest configuration being having haplotype information of a trio mother-father-child. Having genotype information from individuals of known parentage enables to define haplotypes, based on Mendel's laws of segregation. These govern the transmission of alleles to the descendants: the two chromatids of a same chromosome are randomly segregated into gametes. Knowing parental genotypes adds constraints and restricts the scope of possibilities, as the child haplotypes will have to be consistent with them.

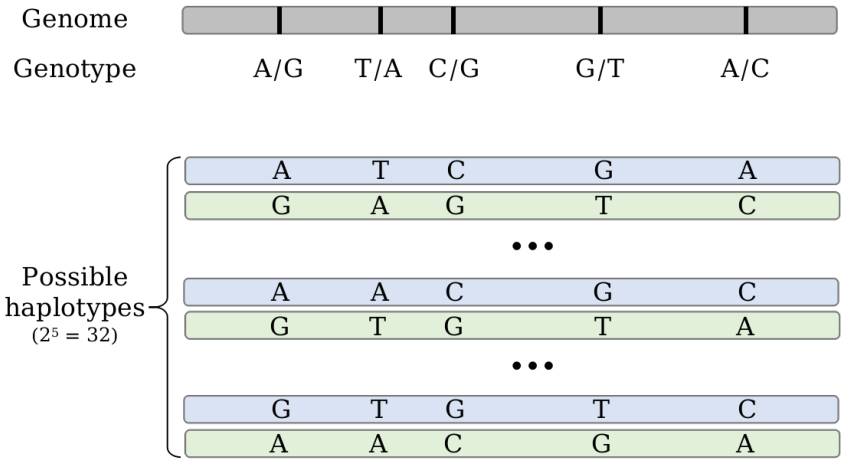


Figure 3.6 – **Genotype and haplotype.** Haplotype reconstruction from a single individual's genotype is not possible, as it does not provide correlations between variants at different positions.

Consider for instance two loci, for which a child has a $(A//G, T//A)$ genotype[5], his mother $(A//A, T//A)$ and his father $(A//G, T//C)$. Then, the only possible solution is that the child has one haplotype (A, A)[6] from the mother, and one (G, T) from the father (see Figure 3.7 for an illustration). However, such deductions are only possible if at least one parent's genotype is homozygous at the given loci, or if only one of its variant is expressed in child's genotype. Loci for which both parents and their child are heterozygous with same alleles, cannot be resolved, and the child's haplotype remains ambiguous at these positions.

Population based. If one does not have access to related individuals, it is still possible to exploit genotype information to retrieve haplotypes from a large set of individuals. Given an individual's genotype, one can list associated possible haplotypes. Then, from population's genotypes, one can estimate (*e.g.* with models such as [N. Li and Stephens 2003]) which are the most likely haplotypes that can explain the observed genotypes. Other statistical methods were developped to estimate haplotype frequencies in a population based on their genotypes, based for example on Expectation-Maximization algorithms [Excoffier and Slatkin 1995] or hidden Markov models [Stephens, Smith, and Donnelly 2001]. We refer the reader to [S. R. Browning and B. L. Browning 2011] for a

5. We use here the usual genotype writting. Genotypes are written into parenthesis, loci are separated by commas. For a given locus, the notation $x//y$ means that both alleles (or variants) x and y are present.
6. Following the same writing principle, haplotypes can be written within parenthesis, with comma-separated loci.

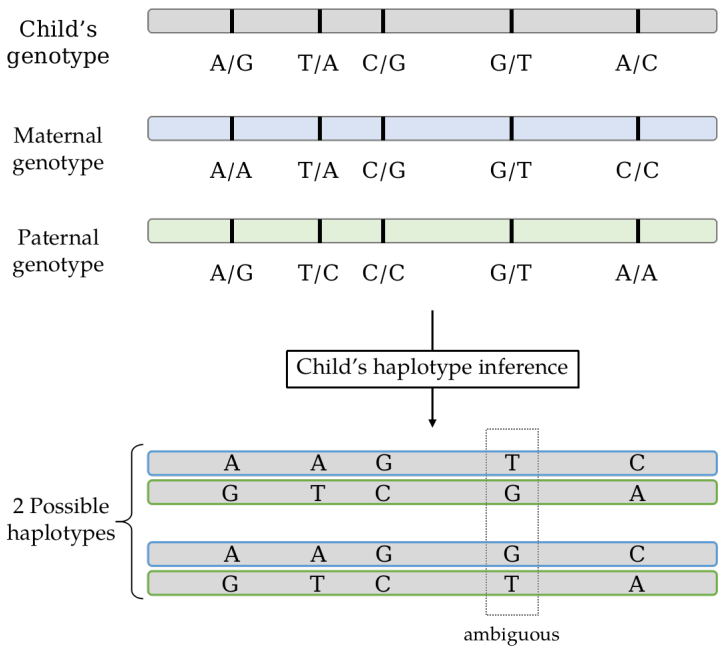


Figure 3.7 – **Haplotype inference from pedigree genotypes.** Having additional access to parental genotypes enables to constraint the set of possible haplotype solutions for their child. If at least one parent is homozygous for a given locus, then infering the child haplotype at this position is straightforward (*e.g.* first, third and last loci). By a similar reasoning, one can resolve haplotype for locus where both parents are heterozygous but at least one having an allele not appearing in the child's genotype (*e.g.* second locus). However, loci for which both parents are heterozygous with same genotype as the child are unsolvable (*e.g.* fourth locus).

detailed review of methods used for haplotype inference from population genotypes.

Exploit the 3D structure of DNA

This approach is somehow a combination of the two previous ones: the experimental separation of haplotypes takes advantage of the 3D chromatin conformation of DNA, and genotyping is based on the linear sequence of DNA. We sketch the principles of one of the most successful technique: the "3C" (chromosome conformation capturing) analysis. The idea is to immobilize the chromatin by creating bonds between two regions of a same chromosome that are close in 3D, but may be far away considering the linear structure of DNA. Then some enzymes cut the DNA into fragments: in bounded regions, one thus obtains two fragments of DNA linked together. The two fragments are ligated on one end to create a single continuous fragment, then the chemical bound is

removed. One obtains a single fragment that can be sequenced. The two initial regions can be distinguished as one knows the sequence of the ligation. This technique enables to link distant variants of an haplotype, even when using short reads.

However, the distribution of observable links is not uniform across the genome [Sati and Cavalli 2017] so some regions will not be covered, and only short range interactions of a few hundred kilobases [Simonis, Kooren, and Laat 2007] can be retrieved.

One of the derived method, the Hi-C, also require a tremendous amount of reads (several hundreds of million reads for a final resolution of less than 1 Mb [Servant *et al.* 2015]), which represents a substantial cost that cannot be afforded by all research labs.

Using sequencing data

While genotype data provide only punctual and independent information about variants, sequencing technologies give access to DNA fragments that each represents a small piece of one haplotype. The objective is to perform p simultaneous assemblies, representing the p haplotypes, by gathering reads stemming from a same haplotype based on their similarity. The read length is crucial here, as the longer the sequence is, the more variants it may carry. For this reason, NGS data are intrinsically limited by their short length of about hundred bases, covering most of the time a single variant. With the emergence of long read sequencing technologies, sequenced fragments are much wider an can include several variants, thus facilitating haplotyping.

Depending on the technology used, the length, accuracy and redundancy of fragments vary. Yet, it enables to access any organim's genome and is relatively affordable. Among TGS technologies, the HiFi sequencing now appears to provide the best results for haplotyping [Duan *et al.* 2022], in combination with HiC information that enables to generate chromosome scale results.

In this thesis, we focus on haplotype inference from long read sequencing data, without any other source of information. The next section elaborates on this in more details.

3.2.2 Formalization and challenges of the haplotyping problem

We outline here the haplotyping problem, precising under which conditions we will address it in this document, and what are its challenging aspects.

Definition (The long reads based haplotype reconstruction problem)**.**
Given *a set S of reads corresponding to one chromosome of a TGS-sequenced organism of fixed*

ploidy p,
***Find** an approximation $\hat{H}_i$ of the true haplotypes H_i, for all $i \in [1, p]$, which minimizes the difference $H_i - \hat{H}_i$ with respect to S.*

The difference $H_i - \hat{H}_i$ is still to be defined here. Several existing metrics can be used, such as the commonly used Hamming distance between the references and the reconstructed haplotypes, or the Switch Error Rate *i.e.* the number of variants that were erroneously associated to the wrong haplotype. In this manuscript, we use an other estimation of this distance, based on similarity and overlap between reads, which will be depicted in chapter 4.

In this document, we restrain certain parameters:
Ploidy. The ploidy level is assumed to be known, and $2 \leq p \leq 8$. Indeed, most common polyploid organisms are diploid to hexaploid [Song *et al.* 2012], and organisms displaying high levels of ploidy are plants, whose genome can easily undergo duplication, and are much harder to handle. Polyploidy is not well tolerated by higher vertebrates [7]
Long reads only. While numerous haplotyping methods rely either on short reads or on existing genome assembly to produce more accurate results, we chose to propose a *de novo* method that would be suitable for a wider range of dataset. The goal is to study how far one can go with this single source of data.
Any type of variants. Finally, instead on working on a set of SNP variants, we take into account any kind of variants, by working at read level. Usual haplotyping methods rely on variants called, which requires both a reference genome of sufficient quality and an accurate method for variant detection. Moreover, such methods only consider SNP - as they are easier to detect and analyze - and leave other variant aside.
Finally, it is assumed that even if two haplotype can be identical on some portions, every haplotype differs from others.

A major challenge here is to cope with the high combinatorics: our goal is to find a p-partition of N reads, which represent p^N potential solutions. As N is often at least of the order of a million, it is not an option to enumerate and evaluate each possibility, and it will be necessary to considerably narrow the solution space. It will also be necessary to define an estimation of the difference $H_i - \hat{H}_i$ according to S. It can be estimated by computing several metrics, such as the Hamming distance or the number of switch errors: this point will be addressed in the following parts.

7. Reptiles, birds and mammal.

3.3 Haplotyping from long read data

In this section, we go through major methods currently used for haplotyping from sequencing data - mostly for long reads, but also some algorithms firstly designed for shorts reads whose principle can be adapted to long ones. An overview of principle characteristics of mentioned haplotyping tools is summarized in Table 3.1. They are described below.

As discussed in the previous section, haplotyping is a highly combinatorial problem, that can hardly be solved without making additional assumptions. It has been resolved either by reducing the search space, and using heuristics that provide approximate solutions.

All haplotyping algorithms are guided by the same logic: on overlapping regions, a set of reads originating from a same haplotype will share more similarities than with reads from another haplotype. So, gathering reads according to their similarity or differences enables to find the initial haplotypes. It can be achieved using a descending or ascending strategy: either considering all reads as a set and dividing it into subsets, or considering reads as elements and aggregating them.

Table 3.1 – **Overview of major characteristics of presented haplotyping algorithms.**

Name	**Handles polyploids**	**Input reads**		**Relies on variant call**	**Phasing strategy**
		Short	**Long**		
Whatshap		×	×	×	Descending
HapCut		×		×	Descending
[Xu and Y. X. Li 2012]		×		×	Ascending
H-Pop	×	×		×	Descending
Whatshap Polyphase	×		×	×	Ascending
nPhase	×	×	×	×	Ascending
POLYTE	×	×			Ascending
flopp	×		×	×	Descending
hifiasm	×	×	×		Ascending

3.3.1 Haplotyping as a partitioning problem

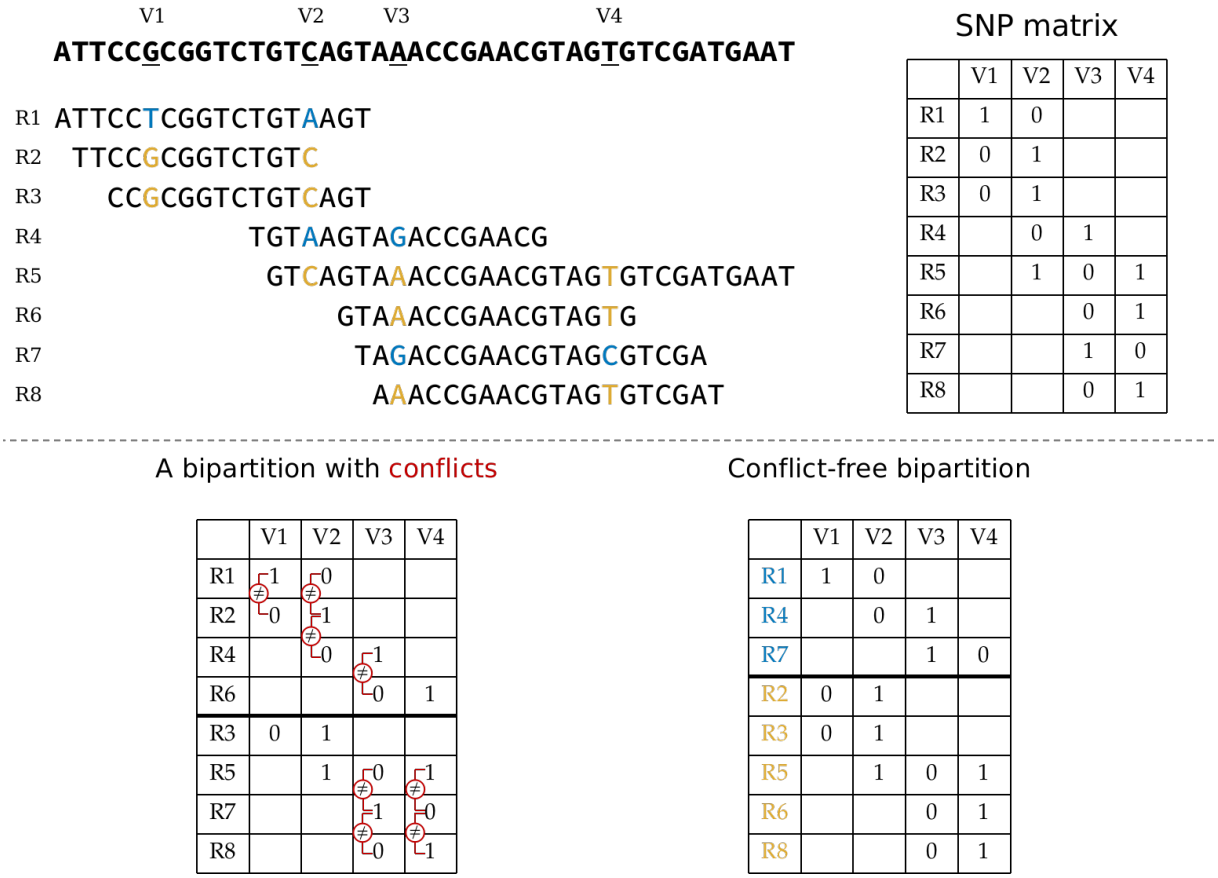
Principle

This method has been described in [Lippert 2002] as the SNP assembly problem. It was firstly designed for short read data, applied on diploid species, and considering only SNP variants. These choices have been made for simplification and with the human genome in mind, but they can be generalized for the treatment of polyploid organisms and other types of variants.

One builds a matrix, with reads in lines and variant positions in columns. In each cell, a special character (like a blank, a dash or a dot) means that the read does not cover this variant, value 1 denotes the presence of the variant in the read, and 0 the absence. The whole set of reads may have an identical value for some variants. Such variants are called *homozygous*, and can be left aside for haplotyping as they do not contribute at all to discriminate haplotypes. Only the *heterozygous* variant loci - for which at least two values occur in the set of reads - are used for haplotyping. The goal is to compute a p*-partition* of the matrix, representing the p haplotypes. Each part has to be *conflict-free*: within a given part, all variant values have to be consistent over all covering reads of this part (see 3.8).

In an ideal case where reads are totally error-free, computing the bi-partition would be straightforward: browse the columns of the matrix, and for each separate the reads at value 0 from the reads at value 1. However, reads actually contain varying degrees of sequencing errors, creating conflicts during splitting: an error at a variant position may either transform a reference value to a variant one, or the converse. The presence of errors troubles the search of a p-partition. Therefore, some optimization criteria have been designed and implemented in order to find the best estimation of the haplotypes, taking into account the errors.

Overall, enumerating all p-partitions of such matrix is a Stirling number of the second kind [Xie *et al.* 2016], so it is impossible to evaluate each and every possible solution to find the best one, especially when the number of rows (reads) is high, which is the case for sequencing data. Therefore, algorithms designed to partition reads for haplotype reconstruction rely on the use of heuristics.



	V1	V2	V3	V4
R1	1	0		
R2	0	1		
R3	0	1		
R4		0	1	
R5		1	0	1
R6			0	1
R7			1	0
R8			0	1

	V1	V2	V3	V4
R1	1	0		
R2	0	1		
R4		0	1	
R6			0	1
R3	0	1		
R5		1	0	1
R7			1	0
R8			0	1

	V1	V2	V3	V4
R1	1	0		
R4		0	1	
R7			1	0
R2	0	1		
R3	0	1		
R5		1	0	1
R6			0	1
R8			0	1

Figure 3.8 – **Haplotyping based on read partitioning.** This example depicts a diploid case, in the absence of error. A SNP/read matrix is built by reducing reads to their variant positions and values. If sufficient data is available, one can find a conflict-free partition that identifies the haplotypes.

Optimization criteria

The Minimum Error Correction (*MEC*). This criterion aims at finding the conflict-free partition that requires the least amount of flips (switching between 0 and 1) in the matrix. The underlying idea is to perform corrections in the matrix, following a parsimony principle, assuming that errors are less frequent than true variants. Solving haplotyping based on the MEC criterion is NP-hard [Lippert 2002].

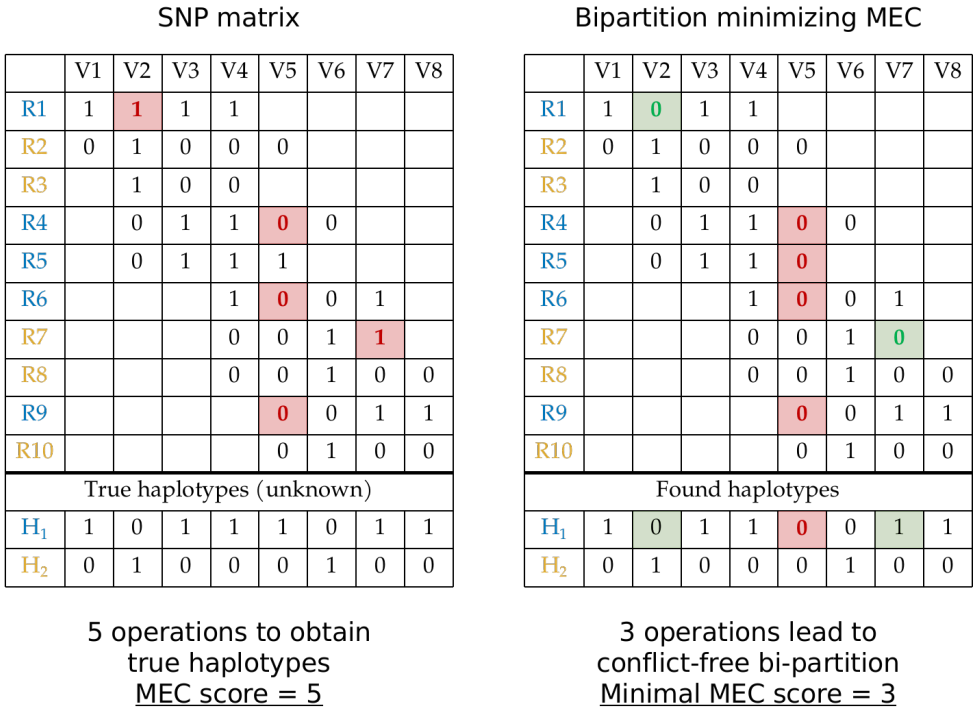
It is also possible to implement a weighted version of MEC (as done for Whatshap [Patterson *et al.* 2015]), integrating quality scores that reflect probability of errors in a read, giving priority to reads that are less likely to contain errors. An adaptation of MEC for polyploid data has been proposed by [Bonizzoni *et al.* 2016], who also demonstrated

that this variation is APX hard (meaning that it can be approximated by a polynomial-time algorithm with approximation ratio bounded by a constant). It consists in finding a k-partition such that there are no conflicting reads in a given part.
The MEC criterion has been declined into several alternatives (see [Lancia *et al.* 2001] for their formal definitions):

- minimum fragment removal (MFR): instead of switching cell values, remove lines (reads) containing conflicting values;
- minimum SNP removal (MSR): instead of switching cell values, remove columns (variant loci) containing conflicting values;
- longest haplotype reconstruction (LHR): maximize the number of variant loci in the conflict-free bipartition.

All these alternatives remain NP-hard. Note that these three criteria are governed by removing reads or variants, so their minimization or maximization may lead to highly reduced haplotypes. In particular, the MFR strategy is not suitable on TGS reads: given their substantial length and their high error rate, it is likely for reads to contain at least one error, so this criterion would remove a considerable amounts (if not all) of reads.
Numerous works have used the MEC criterion (or a variation of) to solve the haplotype determination problem. To speed up computations, Whatshap uses dynamic programming and bounds the read coverage: as the number of reads covering a given variant position is limited, the practical complexity becomes polynomial in the number of variant positions. The HapCut algorithm [Bansal and Bafna 2008] builds a graph from the SNP matrix where nodes are variant positions and edges link nodes covered by the same reads, weighted according to their consistency with the phasing. It performs MEC optimization based on a max-cut search on a graph. They demonstrate that finding a max-cut in such a graph amounts to find a solution minimizing the MEC criterion. Authors of [Xu and Y. X. Li 2012] proposed a clustering approach, (with a k-means algorithm) to group reads and recreate the haplotypes.
As MEC has been used in many haplotype solving approaches, [Majidian, Kahaei, and D. d. Ridder 2020] wondered whether this criterion was actually adapted to resolve this problem. They demonstrated that minimizing the MEC criterion does not guaranty to lead to the true haplotype. In fact, beyond a certain level of errors in the input data - more than half of erroneous values in a SNP column - there exists a solution, different from the true one, having a lower MEC value (Figure 3.9). The authors have taken the analysis a step further, and showed that while MEC criterion is quite suitable for short

read Illumina data (with low error rates), minimizing MEC on long read data is much more likely to lead to a false solution. To do so, they computed the probability that the minimal MEC value corresponds to the true haplotype, depending on the coverage and sequencing error probability. While the probability is around 0.97 for Illumina settings (error rate of 0.1%), it is only of 0.23 - 0.42 for PacBio and Nanopore (error rates of 2-5%) for a same coverage of 10X. To overcome this shortcoming, they suggest using a coverage of at least $25p$, with p the ploidy, to keep the ratio of errors low enough. However it may represent a serious barrier, especially for polyploid organisms, as it would require a sequencing depth of at least 100X to phase a tetraploid genome.



	V1	V2	V3	V4	V5	V6	V7	V8
R1	1	**1**	1	1				
R2	0	1	0	0	0			
R3		1	0	0				
R4		0	1	1	**0**	0		
R5		0	1	1	1			
R6				1	**0**	0	1	
R7				0	0	1	**1**	
R8				0	0	1	0	0
R9					**0**	0	1	1
R10					0	1	0	0
True haplotypes (unknown)								
H_1	1	0	1	1	1	0	1	1
H_2	0	1	0	0	0	1	0	0

	V1	V2	V3	V4	V5	V6	V7	V8
R1	1	**0**	1	1				
R2	0	1	0	0	0			
R3		1	0	0				
R4		0	1	1	**0**	0		
R5		0	1	1	**0**			
R6				1	**0**	0	1	
R7				0	0	1	**0**	
R8				0	0	1	0	0
R9					**0**	0	1	1
R10					0	1	0	0
Found haplotypes								
H_1	1	**0**	1	1	**0**	0	**1**	1
H_2	0	1	0	0	0	1	0	0

Figure 3.9 – **Minimizing the MEC criterion may not lead to the true haplotypes.** Consider a SNP matrix with some errors on variant positions V2, V5 and V7 (highlighted in red in left matrix). It would require five flipping operations to correct these errors and obtain the true haplotype. For erroneous variants in positions V2 (read R1) and V7 (read R7), performing one flip each enable to find the correct value for variants (highlighted in green). For the V5 variant, it requires 3 flips to find the correct haplotypes. However, performing only one operation on read R5 of variant position V5 is enough to obtain a conflict-free matrix. This amounts to a total of three operations, but leads to wrong haplotypes.

Min-sum max tree partition. This formulation is based on the search of maximal span-

ning trees in a read overlap graph with edges weighted according to a distance score between two overlapping reads. The underlying idea is to partition the graph into subgraphs that represent haplotypes. The maximum spanning tree of such subgraphs indicates the level of discordance of associated reads forming the haplotype. Therefore, minimizing the sum of maximum spanning tree leads to reconstruct haplotypes minimizing their internal differences. In order to prevent reads stemming from similar but different haplotypes to end in a same part, a score is used to enforce uniform coverage among haplotypes. This has been implemented in flopp (fast local polyploid phaser, [Shaw and Yu 2022]). Authors have also shown that this formulation is related to the MEC under certains conditions.

Intra- and interhaplotype variance. Instead of only minimizing differences between reads belonging to a same chromosome (*i.e.* the intrahaplotype variance), it is possible to design an optimization criterion that also maximizes differences between haplotype (*i.e.* interhaplotype variance). It has been implemented in H-Pop [Xie *et al.* 2016], which is based on the Polyploid Balanced Optimal Partition (PBOP) problem. It consists in finding a p-partition of a SNP matrix, aiming at both minimizing similarity within reads of a given haplotype, and maximizing differences between differently phased reads. As it is not guaranteed that minimizing the first optimization function would maximize the second one, several approaches can be implemented: looking for a Pareto front, giving priority to one function over the other, or set global optimization function with a weighted balance between the two functions (as used in H-Pop). Under certain conditions of weight, this amounts to the MEC criterion. As the PBOP problem is NP hard, H-Pop implements a dynamic programming approach, similar to seed-and-extend: first compute the best p-partitions over a small subset of rows, and then iteratively extend them to the whole matrix. Note that such approach is also used in Whatshap, which first looks for the best bi-partition on a subset of the SNP matrix, then extend it to the whole set of data.

A difficult issue for all current approaches: locally similar haplotypes

A crucial drawback of looking at a p-partition, with p fixed, is due to the fact that the distance between each pair of haplotypes is variable, and that such difference can vary itself for a same haplotype pair, depending on the positions considered. Two haplotypes, while being globally different, can be locally identical, or at least more similar

to each other than with the remaining haplotypes. In such case, forcing to partition reads into a fixed number of parts may lead to the following undesirable behavior. Two similar haplotype regions are merged into one of the haplotypes; and the remaining reads, those which were too erroneous to be associated to another haplotype, are gathered in a chimeric haplotype. This results in creating false haplotypes, but also a bias: while one expects all haplotypes to be (almost) evenly represented among reads, the merged haplotype will have twice (in this case) more reads than other haplotypes, and the chimeric one containing erroneous reads will only contain few reads (see Figure 3.10).

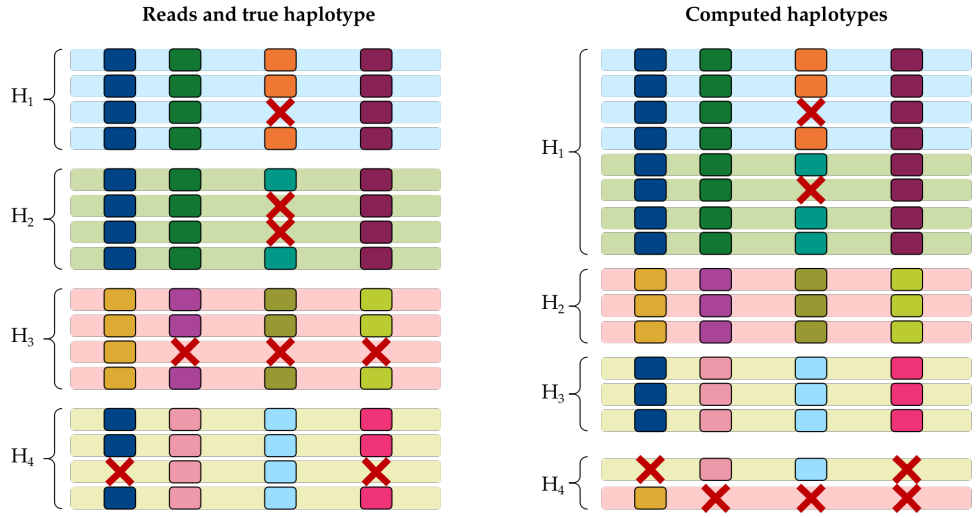


Figure 3.10 – **Collapsed haplotypes.** Consider a set of reads, each originating from a given haplotype (background color of the read), and a set of four variant positions. The coverage is assumed to be even between haplotypes. Reads contain different variant values (small colored boxes) and may have sequencing errors (red crosses). Computing haplotypes by forcing a 4-partition on these reads may result in some chimeric haplotypes: haplotypes H_1 and H_2 are quite similar and may be merged into a single part, while erroneous reads being gathered in a special haplotype. Moreover, the obtained coverage per computed haplotype is not even anymore. The expected coverage is around four reads per haplotype in this case: it is almost met for haplotypes H_2 and H_3, but the resulting H_1 has twice more reads, and H_4 has only half.

3.3.2 Haplotyping as a clustering problem

The previous approach was a top-down classification approach, in the sense that the classes (haplotypes) were fixed. The other possibility is to manage reads in a bottom-up

way, by grouping them gradually in different clusters according to their similarity. It is this thus typically an unsupervised classification task.

The obtained clusters represent haplotypes, or parts of them in the case of fragmentation. Such strategy has been implemented in POLYTE [Baaijens and Schönhuth 2019], Whatshap polyphase [Schrinner *et al.* 2020], and nPhase [Saada *et al.* 2021] algorithms, whose principles are described below.

Defining a similarity relation

As in the first approach, similarity between reads can be computed in different ways. It can be represented by overlap and identity scores, based on overlapping or aligning reads, as is done for POLYTE, yet it requires to be computed for each pair of reads. Another approach, implemented in nPhase, is to define the similarity between two reads (covering some common variant positions) as the ratio of their shared variant values, over the total number of covered loci. A similar idea is proposed in Whatshap polyphase: relying on a probabilistic model, it computes expected similarity in the case of reads belonging to the same haplotype or in the case of reads belonging to different haplotypes. Then, the actual similarity values are compared against these two references, to determine which case is the most likely.

Forming the clusters

Once similarity relations are established between reads, a similarity graph is built, with reads as nodes and edges linking similar reads. The creation of edges may be subject to some similarity threshold, or edges may be weighted according to similarity scores, depending on the implementation.

Then, clusters can be formed by iteratively grouping the closest related reads (nPhase), or by looking for cliques in the graph. This can be achieved either by using a heuristic cluster editing [8] (Whatshap polyphase) or by efficiently enumerating maximal cliques [9] (POLYTE).

At this point, for a given genomic region, the number of clusters can be bound by the expected ploidy or not, allowing to create more clusters if necessary. In any case, a

8. Roughly, cluster editing algorithms aim to find a way to form disjoint cliques (*i.e.* a subgraph for which each pair of node is connected by an edge) in a given graph.

9. Enumerating cliques in a graph belong to the Karp's list of 21 NP-complete problems, it is exponential in the number of nodes.

number of clusters below the expected ploidy reflect regions of locally identical haplotypes.

From clusters to haplotypes

Clusters of reads only represent local haplotype regions, and have to be connected together to form the actual haplotypes.

Clusters can be linked by *threading*, as proposed in Whatshap polyphase. It consists in selecting, at each variant position, exactly p clusters. As the previous clustering step was not ploidy-aware, there may be more than k clusters at a given position, then some will be left over and will not participate to the final haplotypes. Conversely, there may be less than k clusters at some positions, suggesting locally collapsed haplotypes. In this case, the coverage information of each cluster gives insight to detect those present in actually more than one haplotype.

Another approach is to *iteratively extend* the clusters. In this case, one has to define a similarity criterion to compare clusters, and a stopping condition. In nPhase, a consensus sequence is built, representing the set of reads of a cluster. To decide whether to add a sequence of another cluster to it, each base is compared to the consensus: it brings a positive score if it is consistent with the consensus, negative otherwise. If the overall score is over a certain threshold, then the cluster is extended. When there is no more cluster to be extended, the merging process stops. In POLYTE, a similar approach is used, this time relying on iterative merging of cliques until there is not any edge left in the graph.

3.3.3 Exploiting the OLC paradigm for haplotype phasing

It is possible to rely on the OLC approach and build an assembly graph from reads. In such graph, haplotype variants are represented in bubbles. Instead of collapsing them as usual assemblers do, an haplotype aware assembler can keep the information provided in bubbles and use them to reconstruct haplotypes.
Such method has been implemented in the hifiasm [Cheng *et al.* 2021] assembler, which outputs a usual consensus assembly completed with assembly fragments representing local differences on haplotypes. It relies essentially on the use of nearly error-free HiFi PacBio reads to discriminate between haplotypes. Hifiasm can also exploit some additional data such as Hi-C or parental short reads to enhance the haplotigs contiguity and

recreate full haplotype sequences.

3.4 Conclusion

Most haplotyping algorithms available are still designed for short read datasets, and diploid organisms. Yet one can notice that polyploid haplotyping from long read only is gaining attention, as sequence accuracy is improving. In almost every case, one has to provide either a set of variants, or a reference genome and short accurate data to perform variant calling, except for POLYTE for which variants are therefore not limited to SNP and which does not require an accurate reference genome. The best haplotyping results in the state-of-the-art are obtained either by hybrid methods, relying on both short and long read data, or by PacBio HiFi reads with the additional help of Hi-C data to improve contiguity with long range information.

Some other haplotyping methods were not detailed, yet are worth mentioning: GraphUnzip [Faure, Guiglielmoni, and Flot 2021] uses HiC datasets to reconstruct haplotypes from assembly graph of an haploid assembly by separating heterozygous regions and duplicating regions of collapsed haplotypes, Hap10 [Majidian, Kahaei, and D. d. Ridder 2020] and performs polyploid haplotyping from linked reads, and Falcon phase [Kronenberg *et al.* 2019] exploits HiC data to scaffold shorter phased assemblies.

A key limiting feature of existing assembling and haplotyping methods, is that they are designed to provide a unique solution. It seems never discussed but sounds scientifically poorly justified because of the complexity of the underlying problems, and the fact that methods use heuristics, and make some choices and approximations to reduce computations. Indeed, comparing the results of multiple algorithms on a same dataset would likely results in as many different answers as there are tools. Offering to produce and work on sets of solutions is a neat way to increase the robustness of haplotyping results and assessing the quality of the different regions. There is a lack of haplotyping methods from sequencing data proposing multiple equally satisfying solutions. Several haplotyping algorithms already rely on combinatorial serach: HAPLO-ASP using ASP [E. Erdem, O. Erdem, and Türe 2009], HybridIP using integer linear programming [Brown and Harrower 2006], SHIPS based on SAT [Lynce and Marques-silva 2006]. They are thus adapted to produce sets of solutions. Unfortunately, they rely only on genotype datasets and have not been applied so far on sequencing data.

In this thesis, we follow this track of research aiming at defining a combinatorial

search space and preference criteria to constrain admissible solutions in a flexible way.

Chapter 4

Haplotyping of polyploid genome from erroneous long read data with logic programming

---

Contents

Preamble: In this chapter we detail the different steps of the proposed haplotype phasing method: the principles and implementation, assumptions and choices made, and their limits. We have oriented our method towards the construction of an haplotig graph that highlights similar regions and specific ones that are proper to some haplotypes. We rely on the search of pseudo-cliques in an alignment graph to define groups of reads belonging to same genomic regions and haplotypes. Reads are then phased according to their similarities and inconsistencies.

4.1 Defining an haplotype graph

Our overall objective is to reconstruct sequences of a set of n haplotypes. Such sequences can be fragmented as in usual assemblies, and are then called *haplotigs* (from "haplotype contigs"): we will use the term haplotig also for whole sequences to remain as general as possible. They may be output either as a set of n separated linear sequences, or structured within an haplotig graph (Figure 4.1).
The latter will contain all information of the linear sequences, and enables in addition to highlight locally similar regions of the haplotigs as well as variations of the ploidy level. For the definition of such a graph, deciding of the meaning of nodes is a crucial choice. They could represent positions in the alignment of haplotigs, letters, strings or even patterns. We have shown in previous chapters the interest of considering k-mers for an efficient comparison of sequences. Apart from efficiency, k-mers offer a convenient way to define the level of abstraction at which haplotigs are considered. Although the focus so far has been on SNPs, we distinguish between *variation graphs*, which are relevant for population studies, and *haplotig graphs*, which are relevant for a same individual and

for which a coarser-grained representation of the identical or different areas is sufficient. We have thus adopted k-mers labels for nodes, where the parameter k can be set by the user to define the level of abstraction of haplotigs. Note, however, that the value of k cannot be too high in practice, because of sequencing errors that tend to increase the size of the differenciated areas. Edges will represent the succession of strings in the sequence, and we propose to label them with their associated haplotype, so that the phasing information is not lost at locally identical regions.

In this section, we focus on the construction of the haplotig graph, assuming the haplotype sequences are known. In next sections, we will aim at building such a graph from phased reads instead, but still relying on the principles outlined here.

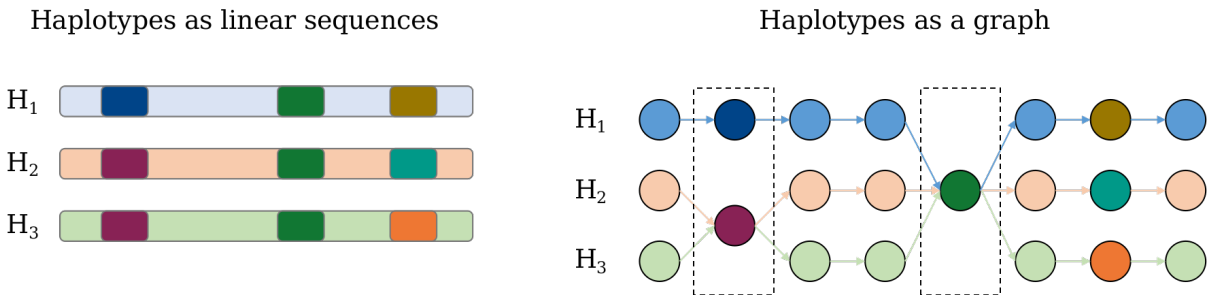


Figure 4.1 – **Advantage of graph *versus* linear representation of haplotypes.** A set of haplotype sequences can be represented in two different ways: as linear sequences and as a graph. The latter enables to explicitly identify regions where haplotypes are identical.

4.1.1 Formulation for the haplotig graph

We consider n haplotigs $H_i, i \in \{1, .., n\}$ where each H_i is a sequence of characters $H[j], j \in \{1, .., |H_i|\}$, and all H_i are different (although they can share some identical subsequences). We aim at building an alignment graph of these sequences, according to a similarity criterion based on a defined level of detail, k. In the case of $k = 1$, it amounts to perform a sequence alignment at the base level. We work in a more general framework with greater values of k, therefore the alignments will rely on identity search at word scale.

> **Haplotig graph.** A haplotig graph is a directed acyclic graph (DAG) $G = (V, E)$ representing the haplotigs of an individual. The nodes v of V are substrings of length $l_v \geq k$ that appear in at least two haplotigs. An edge e of E is labelled with an haplotig number i, and links two nodes v_1 and v_2 such that $v_1 v_2$ is a word in the sequence H_i. The graph contains exactly n paths $(e_{i,1}, .., e_{i,N_i})$ from a source node to a sink node, corresponding to the sequence of each haplotig. An example of such a graph is presented in Figure 4.2.

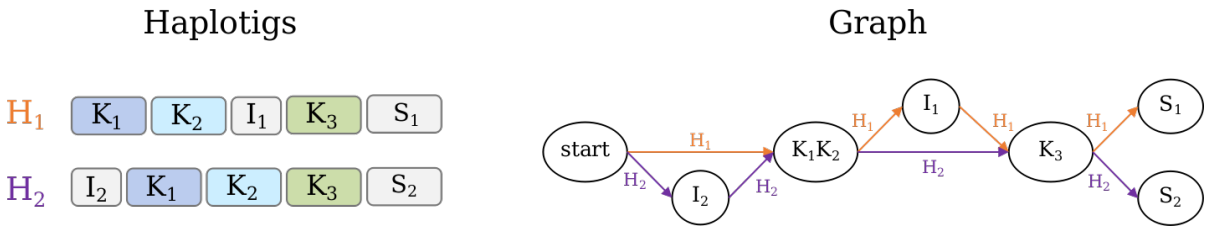


Figure 4.2 – **Example of an haplotig graph from k-mers of two haplotigs.** K-mers are used to anchor similar regions, then subsequences that are not involved in an identity relation are added to the graph to represent the full haplotig sequence. Based on the context in regions of differences, one can deduce the presence of insertions (I_i) or substitutions (S_i)

.

The principles of the haplotig graph are common with the multiple alignment graph described in the literature [C. Lee, Grasso, and Sharlow 2002; Blanchette *et al.* 2004; Ovcharenko *et al.* 2005; Angiuoli and Salzberg 2011; Berkemer, Höner zu Siederdissen, and Stadler 2021]. Sequence alignment is a key concept in bioinformatics, used for example to detect homology relations in phylogenetics, detect mutations or variants, or evaluate sequencing accuracy. It relies on the assumption that the aligned sequences are related. The problem of finding an alignment between two or more sequences has been widely addressed in researches. The multiple alignment of m sequences of length n has complexity $O(n^m)$. It is possible to reduce this complexity by producing an approximation, defining one of the sequence as a reference and aligning the others on this reference. The obtained results may vary depending on which sequence is chosen as a reference. For instance, if a region in present in a subset of the sequences to align, but not in the reference, then this region will not be identified in the alignment. The notions of *blocks* and *threaded blocks* have been defined in [Blanchette *et al.* 2004] to overcome this issue. A block is a local alignment of part or all sequences, and a blockset is a

set of blocks for which the aligned regions are colinear and have the same orientation among all sequences.
Here, we will not use blocks, but simpler entities (k-mers) to find strict identities. We use the principle of blockset that defines a reference sequence to identify regions of identity. By successively using all but one sequence as a reference, we ensure to take into account all relevant regions.

We base our graph on the Partial Order Alignment (POA) framework, introduced in [C. Lee, Grasso, and Sharlow 2002], and further used in variation graph tools such as vg [Garrison *et al.* 2018] (see next paragraph). A POA is a DAG whose nodes are characters at each position of the alignment, and edges represent the succession of characters in the aligned sequences. In our cases the "characters" will be k-mers. Recent researches have been oriented towards a way to represent multiple related sequences at once: *variation graphs* [Paten *et al.* 2017; Garrison *et al.* 2018; Eizenga *et al.* 2020; Qiu and Kingsford 2021]. They are used as a way to gather and compare sequences, highlighting shared subsequences. Variation graphs explicitly store paths for every sequence, in addition to nodes and edges, and storing all these paths is memory intensive. Here, we instead tag edges according to the associated haplotig(s), so that one has access to the explicit information of haplotigs at any part of the graph, without having to enumerate paths.

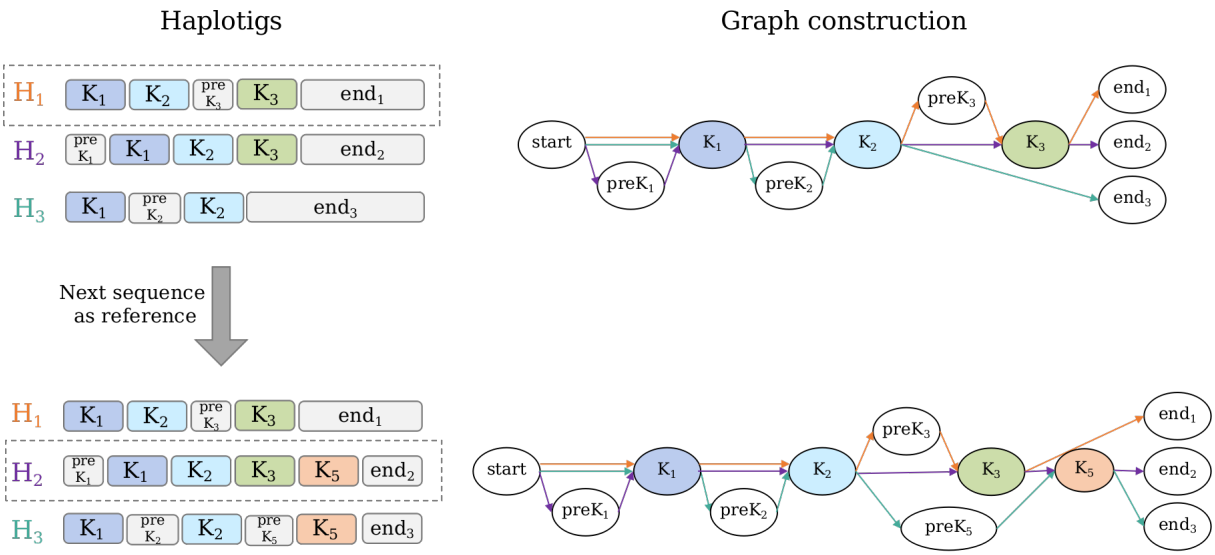


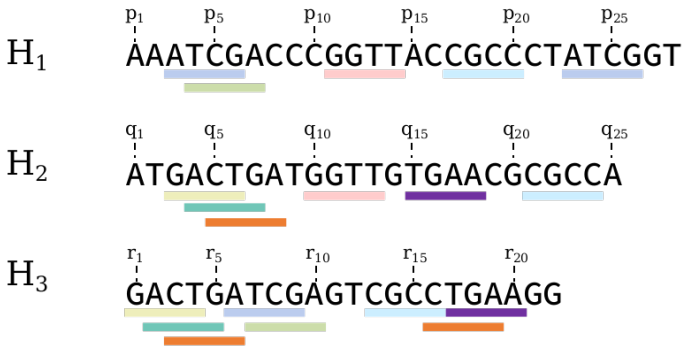
Figure 4.3 – **Use successively each haplotig sequence as reference to build the haplotig graph.** By doing so, it is guaranteed that all relevant k-mers are taken into account.

4.1.2 Local identity between haplotigs

To construct the haplotig graph, we need to define a matching relation between sequences. We rely here on the k-mer content to detect regions of strict matching between haplotigs. Dually, the other regions of the graph represent diverging parts of the sequences. The challenge of such representation lies in the fact that a given k-mer can be repeated several times inside an haplotig: matching has to find an optimal way of associating occurrences in all haplotigs.

In practice, k-mers are extracted from sequences and stored in a table with their occurrence positions for each haplotig (Figure 4.4). Each haplotig sequence is successively chosen as a reference, and scanned with a sliding window of length k, and step 1. A k-mer is kept only if it appears in at least one other haplotig, and if it has not been processed already. Its positions are stored for each concerned haplotigs.

Equivalence classes of k-mer occurrences.
One defines $Hap(w) = \{H \mid w \text{ has an occurrence in } H\}$. A triplet (H, p, L) represents an equivalence class of positions, where p is the first occurrence position for a k-mer $w = H[p : p+k-1]$, H is the smallest haplotig of $Hap(w)$, and L is a list of pairs (H_i, p_j) which specifiy the positions p_j of w's occurrences in the haplotigs H_i.



	ATCG	TCGA	GGTT	CGCC	GACT	ACTG	CTGA	TGAA
H_1	p_3,p_{23}	p_4	p_{11}	p_{17}				
H_2			q_{10}	q_{21}	q_3	q_4	q_5	q_{15}
H_3	r_6	r_7		r_{13}	r_1	r_2	r_3,r_{15}	r_{17}

Figure 4.4 – **Store k-mer identities between sequences in an occurrence table.** ($k = 4$)

However, the set of defined classes is too fragmented. We adopt the notion of *unitig*

in a graph and apply it on the k-mers: it consists in concatenating linear portions of a graph. Indeed, if the identity concerns a region of length $k+2$ for instance, three triplets (H, p, L), $(H, p+1, L)$, and $(H, p+2, L)$ will be generated, although they correspond to a same identity.
More formally, if the identity is an equivalence class related to words of length $k+x$, $x \geq 0$, then it will be described with $k+x+1$ triplets: $(H, p, L), \ldots, (H, p+k+x, L)$. In this case, one can gather such triplets into a quadruplet that takes into account the length $k' = k+x$: (H, k', p, L), where p is the smallest position over the set of consecutive triplets.

This problem is related to the maximal repeat search.

> **Maximal repeat search in a single sequence.** A repeat r of a string S is a substring having at least two occurrences in S. A repeat is said *maximal* if it cannot be extended in either direction without losing at least one of its occurrences.
> *Example:* considering the string $S = "GTACTAACTCTACTC"$, $"AC"$ is a repeat but not a maximal repeat as it can be extended to $"ACT"$ (without losing any occurrence of AC) which is a maximal repeat (extending it to $"ACTA"$ or $"ACTC"$ would decrease its number of occurrences).

Here, we do not search repeats inside a same sequence. Instead, we look for maximal repeats with respect to its list of occurrences in sequences other than the reference. In other words, for a given position in the current reference sequence, we look for the longest word starting at this position such that its number of occurrences in the other sequences stays the same as for the initial k-mer. It amounts to look for unitigs in a graph, and stop word elongation each time a new occurrence appears, or an old reference disappears (Figure 4.5).
Note that the retained words (extended k-mers) may overlap along sequences: the next step will aim at improving the representation of sequences with non-overlapping words.

4.1.3 Find optimal matching between haplotigs

Each quadruplet gathers a set of positions that are as many possibilities of inter-haplotig matching. An admissible matching must fulfill the constraint that two different

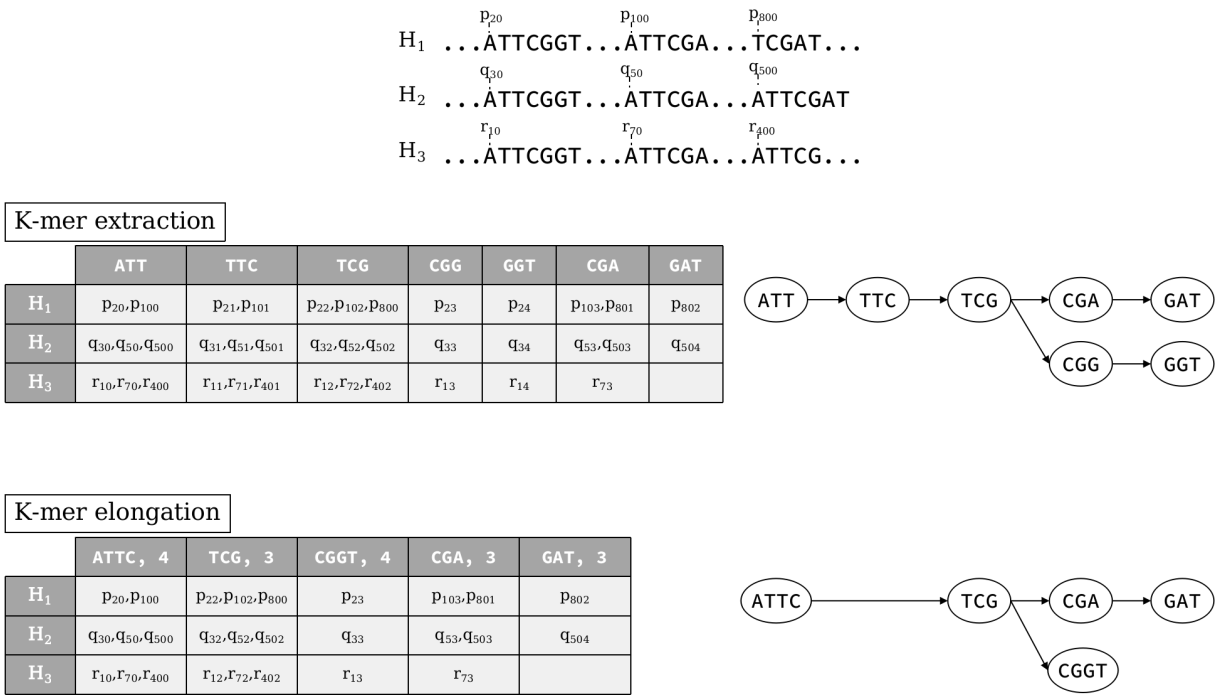


Figure 4.5 – **K-mer extraction and elongation for a set of haplotigs.**

positions of a reference haplotig cannot be matched with a same element.
In the case where there is only one position for a given word $(H_1, t, p_1, [H_1 : \{p_1\}, H_2 : \{p_2\}, \ldots, H_m : \{p_m\})$, which is quite frequent if t is high enough, then only one match is possible: $(H_1, p_1) \xrightarrow{t} (H_2, p_2), \ldots, (H_1, p_1) \xrightarrow{t} (H_m, p_m)$.
In the general case, the set of matching choices of a position from the reference haplotig to the other haplotigs is the Cartesian product of the sets of positions for the other haplotigs.

To differentiate the possible matches, we set a colinearity condition between haplotigs, and take into account the potential overlapping between the identities. In other words, if $(H_1, p_1) \xrightarrow{t_1} (H_2, p_2)$ and $(H_1, q_1) \xrightarrow{t_2} (H_2, q_2)$ for $p_1 < q_1$, then $p_2 < q_2$.
Moreover, if $q_1 < p_1 + t_1$ (overlap), then $q_2 < p_2 + t_2$. Note that these constraints may be unsatisfiable even when only one match is possible and the quadruplet should not be used in the solution in such case.

The construction of the haplotig graph is done in two steps: choosing the best matches among identities, then build the graph and complete it with the haplotig-specific regions.

The optimal choice of matching can be seen as searching a set of compatible matches and maximizing the sum of associated repeat lengths among haplotigs. It can be solved as a multiple alignment problem, the alphabet being based on repeats and not on single characters. We have divided the matching problem in two subproblems in order to reduce its complexity. The first one tries to match anchor points and the second to match unitig occurrences in the remaining quadruplets. The *anchor points* are defined as repeats occurring exactly once in each haplotig, and with a sufficient length (defined with a threshold parameter).
Continuing with the example of Figure 4.4, assume the length of anchors to be greater than 3. The word $"CGCC"$ is present exactly once in each haplotig, therefore, it constitutes an anchor point and one has to decide if it is part of the solution or not.
Once the optimal decision is achieved for anchors, one compute the optimal matching for the remaining unitigs.

Finally, one retrieves the non-matched sequences and builds the haplotig graph.

We described here the haplotig graph framework to represent the different haplotig sequences, by highlighting the regions of local similarity. For simplicity, we treated the case where the full haplotig sequences are available, to focus on the haplotig graph construction and its characteristics. Our objective in this chapter is to manage to build such a graph, only relying on sequenced reads and phasing results, through the method that is described in the following parts, relying on what has been described in this section.

We will now introduce the Answer Set Programming language. The algorithm for determining the haplotig graph will then be presented in a later section, as a first example of ASP-based resolution.

4.2 An introduction to Answer Set Programming

We present here the basis of the *Answer Set Programming* (ASP) language, as well as the useful features provided for haplotype phasing application. This section heavily relies on a course given by Torsten Schaub [1] from Potsdam University, leader of the team that produced one of the best ASP solvers available. In this work we use their environment, the Potassco system (Potsdam Answer Set Solving Collection) [Gebser *et al.* 2011].

1. Available at `https://www.cs.uni-potsdam.de/~torsten/Potassco/Slides/asp.pdf`.

We used the Answer Set Programming (ASP, detailed in next section) language for our haplotyper implementation, as it fulfills several valuable criteria (detailed in next section).

First, ASP has been designed to address combinatorial problems, and is therefore well suited in the haplotyping application. It has already been used for the genotype-to-haplotype reconstruction problem ([E. Erdem, O. Erdem, and Türe 2009; Brown and Harrower 2006; Lynce and Marques-silva 2006]), although it has not been pursued with the spread of sequencing data.

In addition, for a given problem, it provides an exhaustive enumeration of all possible solutions satisfying the specifications, enabling to work on a set of solutions instead of a unique answer. Last but not least, the ASP formalism is compatible with adjusting and incrementally refining models of the problem: it is *elaboration tolerant*. This concept, introduced in 1988 by John McCarthy [2] refers to the ability to integrate new elements in a conceptualization without the need to start again from scratch. It allows the user to express additional conditions, preferences or constraints, and it will eases the interaction with biology experts or bioinformaticians.

4.2.1 ASP principles

When dealing with a problem with usual imperative programming, one has to design an algorithm that describes, step by step, how to solve it. Other programming paradigms exist, among which declarative programming. Instead of providing directions on *how to solve* the problem, one only specifies it by describing *what the problem is*.

For this thesis, we chose to use declarative programming, and more precisely a language combining aspects of logic programming and constraint programming: *Answer Set Programming* (ASP). On the one hand, logic programming describes problems with *facts* and *rules* that represent the problem instance and consequences that can be derived from it. On the other hand, constraint programming enables to solve efficiently highly combinatorial problems. It is based on performing an intelligent, conflict-driven exploration of the solution space, especially by using *learned constraints* (nogoods), which dynamically restrain possible solutions. It guarantees, provided that the problem is correctly stated, to find a solution (or set of solutions) it if exists, and to prove the absence

2. Available at `http://www-formal.stanford.edu/jmc/`

of solution (*unsatisfiability*) otherwise.

4.2.2 Syntax and properties

In practice, ASP offers a predicative language that is automatically converted to a propositional one by a solver called *grounder*. We only give global indications here, and provide further details concerning the syntax in the following sections when necessary.

The syntax of ASP is reduced and use notations coming either from set theory or logic programming.

> **Logic program.** A logic program over a set $\mathcal{A}$ of atoms is a finite set of rules of the form: $h : -b_1, ..., b_i, \neg b_{i+1}, ..., \neg b_n$, $n \geq 0$, with:
>
> - h and $b_i \ \forall i \in \{1, .., n\}$ are **atoms** ($\{b_1, .., b_i\}$ are *positive* and $\{b_{i+1}, ..b_n\}$ are *negative*);
> - h is the **head**;
> - all $b_i \ \forall i \in \{1, .., n\}$ form the **body**.
>
> Atoms can be constants (starting with a lowercase letter) or variables (starting with an uppercase letter) over a finite domains.
> If the body holds (*i.e.* if positive atoms are *true* and negative atoms are *false*) then the head is *true*.

> **Fact.** A body-less rule, *i.e.* a rule containing only a head. A fact is always *true*.

> **Constraint.** A head-less rule, *i.e.* a rule containing only a body (or equivalently, a rule with head equal to the special atom *false*). The body of a constraint must always be *false*.

4.2.3 Semantics

The semantics of the ASP language is inspired by works on common sense reasoning and non monotonic logic. Although its syntax shares strong similarities with the Prolog language, its semantics is quite different.

Easy representation of a problem. Its semantics is built under the closed word assumption,

which means that we only consider predicates described in the problem specification, and anything that is not described is considered as not happening (false).
Elaboration tolerant. Unlike usual imperative languages, or Prolog for instance, the ordering of statements in ASP does not matter. It is then handy to modify only parts of the program (*e.g.* adjusting a condition, or adding a constraint) without having to propagate this modification to the rest of the program.
Provide models as solutions. Logical models are (minimal, for ASP) sets of atoms that are true. From a given problem instance and a problem encoding, ASP explores the set of possible solutions and provides an answer set which is a stable model (see below). It also enables to apply operations on these models (like intersection or union).

> **Stable model.** A minimal set of facts satisfying a logic program, such that all atoms are justified by some rule. In details, each program is reduced to a program without negation, by guessing the value of negated predicates. Stability means that the guess is retrieved in the model. Positive programs are evaluated by simply propagating the truth value of facts within rules. Positive programs have a single model, while programs with negation may have 0, 1 or multiple models.
>
> *Example:* the logic formula $\mathcal{P} = q \wedge ((q \wedge \neg r) \rightarrow p)$ has three possible models: $\{p, q\}$, $\{q, r\}$, and $\{p, q, r\}$ but only one stable model $\{p, q\}$, because r is not justified by any rule in $\mathcal{P}$.

The ASP language is designed to resolve combinatorial problems in the realm of NP and with large datasets, such as planning, graph coloring, or traveling salesperson problems.

4.2.4 Obtain a stable model from a problem specification

Grounder: from predicates to proposition

In a problem instance, the set of variable-free atoms (*i.e.* the facts) are called **ground**. The *grounding* step consists in deriving in all possible ways the atoms containing variables according to the ground. It also aims at reducing the set of ground instantiations, based on the closed world assumption: every instance that is not justified by a rule is removed.

Solver: explore the solution space

After grounding, the ASP solver searches for stable models. It can output either one solution or a fixed number of solutions, but it can also be asked for all solutions. Advanced solving algorithms has been developed to avoid enumerating all possible models to check them.

The solver combines the traversal of a binary search tree, with conflict detection and backjumping. When a solution is explored and a conflict is found, the solver backjumps just before the source of conflict. Then, it can continue the traversal, pruning branches leading to conflicts and thus speeding the solution search.

Solution and reasoning modes

If no stable model satisfying the input facts and problem encoding is found, then the problem is said *unsatisfiable*.

In addition to the satisfiability status, ASP offers to compute operations on the stable models found, such as enumeration, optimization (according to defined criteria), intersection or union.

In the context of haplotype phasing, the intersection of stable models in particular is highly valuable, as it shows the regions of the genome that can be phased without ambiguity.

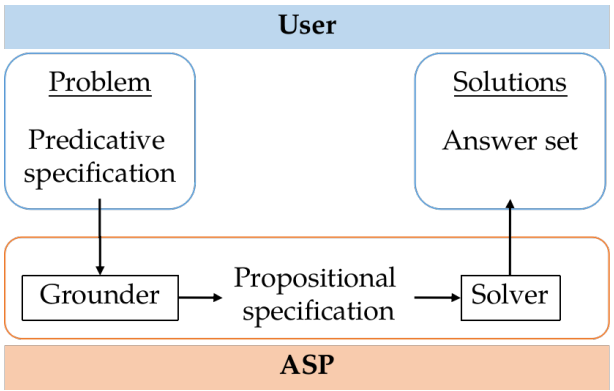


Figure 4.6 – **Overview of ASP based solving.**

4.2.5 Building an haplotype graph from a set of haplotig: a first ASP contribution

The ASP implementation of the haplotig graph construction is provided in Listing 4.1.

Listing 4.1 – **ASP implementation basis of haplotig graph construction.** The input is a list of atoms `unitig(U,H,P)`, with `U` an identifier of the unitig, `H` an haplotig identifier in which the unitig appears at position `P`.

```
% For a given unitig, match pairs of haplotigs in which it occurs
% First compute these matches for anchor unitigs (linkA) then for the remaining unitigs (linkNA)
%    chose one match among its occurrences in other haplotigs
{linkA(U,H1,P1,H2,P2): unitig(U,H2,P2), H1<H2}1:-
    unitig(U,H1,P1), anchor(U).
selectedA(U,H,P):- linkA(U,H,P,_,_).
selectedA(U,H,P):- linkA(U,_,_,H,P).
%     do the same for non anchor unitigs
{linkNA(U,H1,P1,H2,P2): unitig(U,H2,P2), H1<H2}1:-
    unitig(U,H1,P1), not anchor(U).
selectedNA(U,H,P):- linkNA(U,H,P,_,_).
selectedNA(U,H,P):- linkNA(U,_,_,H,P).

% Once the matching are chosen, consider all links together
link(U,H1,P1,H2,P2):- linkA(U,H1,P1,H2,P2).
link(U,H1,P1,H2,P2):- linkNA(U,H1,P1,H2,P2).

% Constraints.
% Two occurrences of a unitig in the same haplotype H1 must not be link to a same occurrence in another haplotype H2
:- link(U,H1,P11,H2,P2), link(U,H1,P12,H2,P2), P11!=P12.
% Keep colinearity among selected unitigs
:- link(K1,H1,P11,H2,P12), link(K2,H1,P21,H2,P22), P11<P21, P12>P22.

% Minimize the number of non selected unitigs
#minimize{1@50,U,H,P: unitig(U,H,P), anchor(U), not selectedA(U,H,P)}.
#minimize{1@10,U,H,P: unitig(U,H,P), not anchor(U), not selectedNA(U,H,P)}.

% Maximize the sum of selected unitig sizes
scope(U,S):-  link(U,_,_,_,_), size(U,S).
#maximize{S@1,U: scope(U,S)}.
```

4.3 Rationale of the phasing method

4.3.1 Core principles

As depicted in previous chapters, haplotype reconstruction from sequencing data is a highly combinatorial problem. It requires adapted methods to be handled, and to scale to the amount of input data (often 10^5 to 10^6 reads). A commonly applied strategy is to *divide and conquer*: define smaller instances of the problem that can be solved independently.

Most existing - either assembling or haplotyping - methods rely on an external information source, such as a reference genome, a set of short reads, or Hi-C data, to name only the main ones. Such additional data enable to split the whole problem into subproblems, by defining independent regions of the genome. Yet relying on these resources comes with additional constraints, both financial and technical. We base our method on using only one source of information, *i.e.* long read data. To create independent instances of the phasing problem, we use the longest reads from the set of reads as anchors points for the remaining ones. This enables to build groups of reads originating from common genomic regions.

A central aspect that we attempt to address in our contribution concerns the result provided to the user: while usual methods output only one solution - assumed to be the best possible solution according to the underlying algorithm - we believe that the final decision falls to the user, and that providing a set of equally optimal solutions is more useful in an applied context.

4.3.2 Overview of the method

We divided our haplotyping method into several steps (see Figure 4.7). At each step of the process, we allow ourselves not to make a decision (*i.e.* phase a read) if it appears that there is no sufficient information on which to rely.

Longest reads phasing. The set of longest reads is isolated using a minimal length parameter. These reads will be used to delimit subproblem instances associated to comparable regions in each haplotype. To do so, an all-*vs*-all alignment is performed, with the goal to cluster similar reads. This has a quadratic complexity so we have to ensure the set of reads is not too big. Then a similarity graph is built from the alignment results. Its connected components - representing groups of *seed* reads - are identified. Within

each group, seeds are phased based on their similarity, provided sufficient information is available. This amounts to partitioning reads into p parts, relying on stringent constraints to reduce the search space.

Phase remaining reads. Remaining reads are affected to different seed groups, based on their alignments (limited to the top best ones to reduce computation resources). They are then phased according to seeds phasing, exploiting both similarity and consistency among seeds.

Build global phasing. Once all reads have been phased, the overall haplotyping result is created by linking results from each seed group. We output the reconstructed haplotigs with a graph of k-mers highlighting similar regions and specific ones in each haplotype.

The haplotype phasing method we developed is further detailed from section 4.4 onwards.

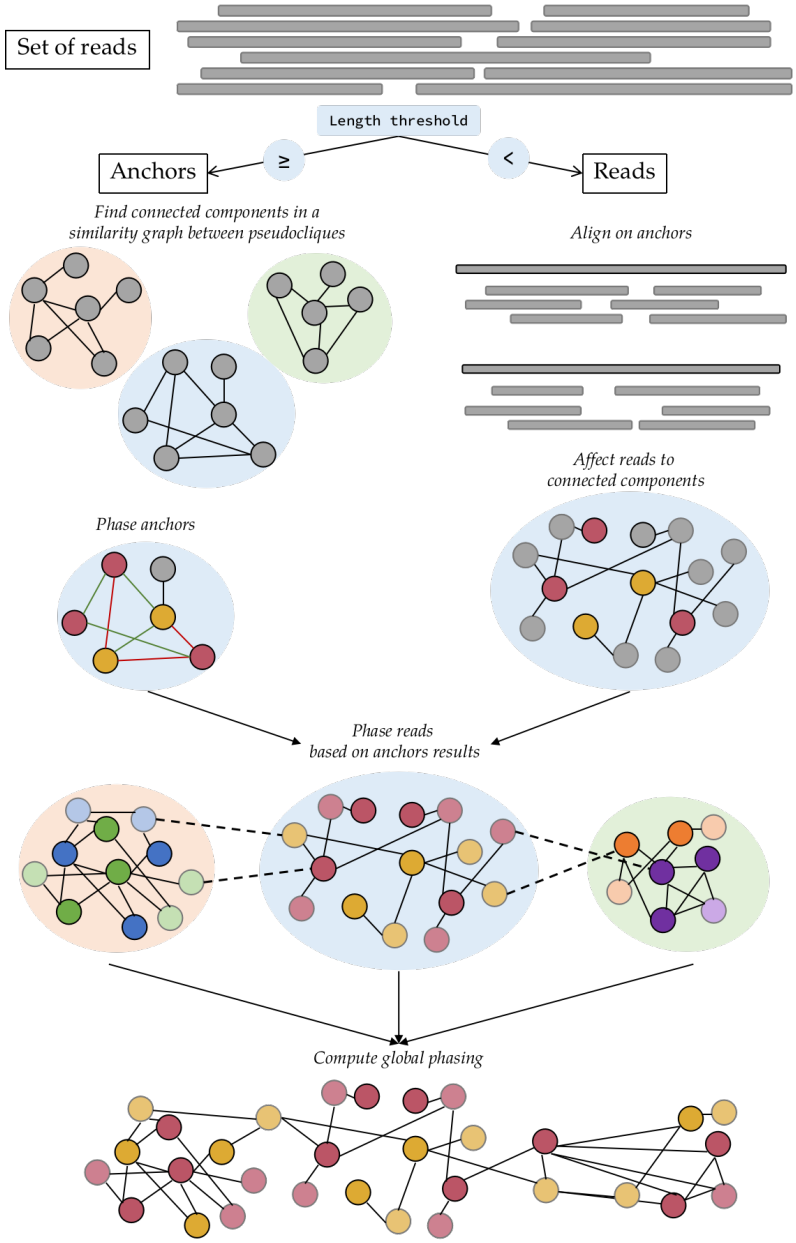


Figure 4.7 – **Overview of the phasing process.** From a set of reads, we select the longest ones to be used as seeds. They are then aligned on each other in order to create a similarity graph, in which we look for connected components, representing groups of highly similar seeds (likely to stem from the same haplotype region). Within each connected component, seeds are phased based on their similarity relations. Remaining reads are then affected to connected components, and phased based on their similarity with seeds. Finally, haplotigs are built from local phasing results of connected components.

4.4 Longest reads phasing

In order to reduce the complexity of this problem, we aim in this section at delineating subproblem instances that can be further resolved independently, and defining for each a pseudo-reference to ease phasing of the whole set of reads.

To do so, the idea is to select the longest reads, that will thus cover a high number of variant positions. They will serve as pseudo-references, or *seeds*, for the phasing task. The longest reads are compared in order to build a similarity graph between them. They are then separated into independent subsets based on quasi-clique patterns. The underlying assumption is that a quasi-clique in such graph gathers reads that are likely to stem from the same unique haplotype. The ploidy is based on the pairing of chromosomes (with multiple related pairs in the polyploid case), therefore our phasing method will attempt to build groups based on pairs of close quasi-cliques. Then, within each subset, seeds are phased according to each others' similarity and overlap.

4.4.1 Select candidate phasing seeds

Seed reads will serve two functions: to define groups of similar regions in the different haplotypes of a genome, and to provide a pseudo-reference for further phasing of the whole read set.

We select seeds based on a tradeoff between keeping a small fraction of the longest ones in order to reduce phasing complexity, while still having enough of them so that most of the genome regions are covered. Repeated sequences and regions of local identity between haplotypes cause trouble during the phasing process: if such regions are longer than read length, then it would be hard to unambiguously assign these reads to a specific haplotype or region. Selecting the longest reads as seeds decreases their chances to be contained in such problematic regions. The number of kept reads depends on the dataset's characteristics such as the technology used, the depth of sequencing, or the length of the genome.

4.4.2 Build a similarity graph between longest reads

Instead of reasoning at a pairwise level, we work on the overall similarity between the selected reads: while two reads stemming from different haplotypes may share identities, it is likely that such similarity will not hold when considering larger sets

of reads. We assume that within a set of reads originating from a common region of a same haplotype, many similarity relations are present: in an extreme and ideal situation, every read in this set should be similar to any other read in the set. Conversely, reads stemming from different haplotypes or from different regions of a same haplotype, would share few isolated similarity relations.
Our method builds a similarity graph, and gathers sets of reads originating from a same haplotype region based on the assumption that they generate patterns of highly interconnected nodes in the graph, that is, close to cliques (complete subgraphs).

Longest reads have to be aligned to determine their similarity relations. The quadratic complexity of computing alignments is an additional motivation for selecting a reduced set of seeds. In practice, we have chosen to perform these alignments using the minimap2 [H. Li 2018] aligner, that provides a special mode for all-vs-all alignments, using default parameters for Nanopore or PacBio data (command: `minimap2 -a -x map-hifi/map-ont -X -secondary=yes -sam-hit-only`).

We then build a similarity graph, with seeds as nodes and edges between two seeds considered highly similar.

The condition of searching highly interconnected nodes in the similarity graph is quite stringent and we have to carefully design their similarity relation. We define the similarity based on two parameters: their identity and overlap ratio. We do not require strict identities: at this point, our goal is to gather seeds that are likely to come from the same haplotype region, but also seeds originating from similar regions of different haplotypes.
The identity percent between two sequences is defined as follow (the *soft clipped*[3] regions are ignored):

$$Identity = \frac{\#match}{\#match + \#mismatch + \#indels} = \frac{\#match}{|\text{aligned bases}|}$$

We also take into account the overlapping ratio between two seeds as an indicator of identity's reliability: two seeds sharing 90% of identity but aligned on only 5% of their sequences cannot be considered as a solid similarity. The overlap percent between two

3. In an alignment, a set of bases at a sequence end that does no sufficiently match the other sequence may be considered as not belonging to the alignment, and be tagged as a *soft clip*.

sequences is defined as:

$$Overlap = \frac{|\text{aligned bases}|}{|\text{whole alignment}|}$$

4.4.3 Find connected components in the alignment graph

We look for complete subgraphs in the graph of similarity, that is, cliques. Yet, enumerating cliques in a graph is a NP-complete problem. In addition, requiring exact cliques would lead to reject credible candidates.
Therefore, we relax the constraints and look for quasi-cliques instead of cliques. Quasi-cliques are defined as dense subgraphs that are close to cliques, but lack some edges to be a clique. In practice, they are detected by estimating the density of edges, that is, dividing the number of effective edges by the number of edges in a clique with the same number of nodes: if the obtained value is over a defined threshold, it is considered as a quasi-clique.

As mentioned before, we proceed as follows we proceed in two steps: first detecting local quasi-cliques, then extending them with pairs of highly connected quasi-cliques.

Define quasi-cliques.
For each seed, consider its adjacent nodes (*neighbors*). Then, compute *local clustering coefficient* (CC_{local}) (Equation 4.1) for each node having enough similar neighbors (similarity $\geq min\_similarity\_clique$) , that is, compute the number of cliques of size 3 between this node and its neighbors (Figure 4.8).

Using such coefficient enables to perform a rough search of dense regions in the graph, without having to search for cliques.

> **Local clustering coefficient.** The value of the local clustering coefficient for a given node of a non-directed graph represents how close this node and its neighbors are close to form a clique. It is the ratio of effective edges between these neighbors, over the total possible number of edges. A value of 1 is obtained for a clique, a 0 for a star.
>
> $$CC_{local}(x) = \frac{S}{N(N-1)/2} \tag{4.1}$$
>
> with S the number of edges between the neighbors of the node x, and N the number of neighbors involved.

Seeds having a sufficient CC_{local} value are retained for quasi-cliques.

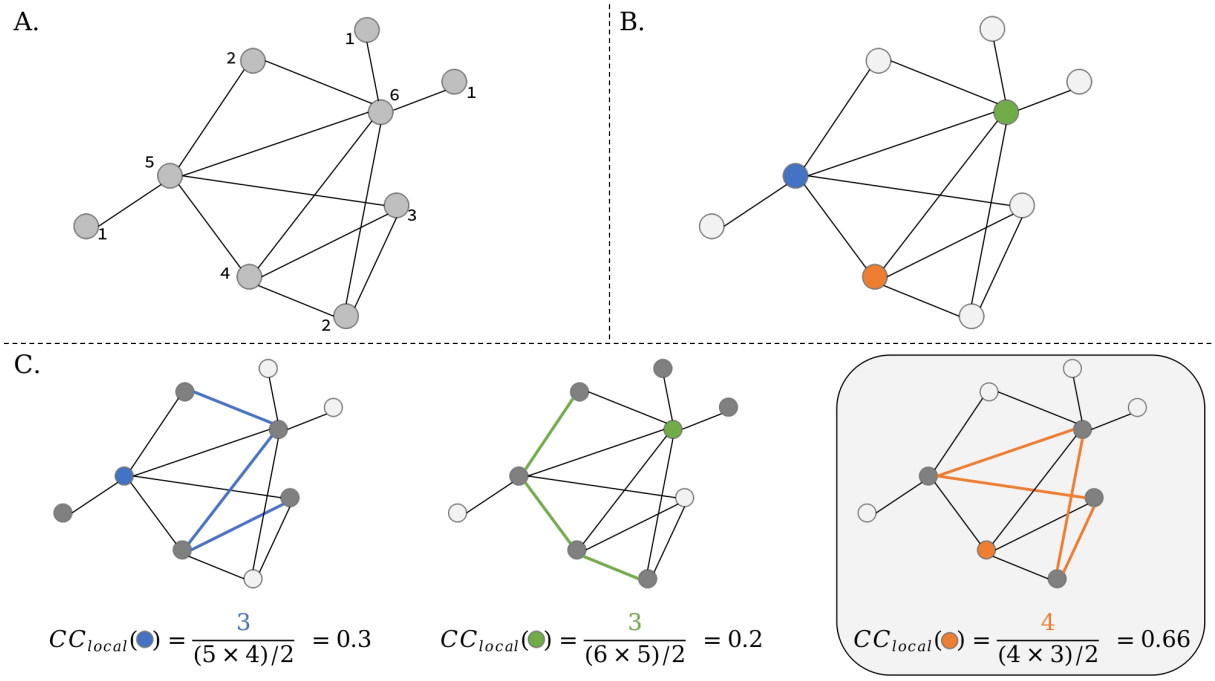


Figure 4.8 – **Finding quasi-cliques in a graph.** A) A similarity graph. Nodes are labeled with their number of neighbors. Only edges with a sufficient similarity score are considered. B) Only nodes having at least three neighbors with strong similarity scores are selected for quasi-clique search. C) For each candidate node, the local clustering coefficient is computed. Neighbors of the current candidate node are highlighted in dark gray. The node having the highest value is chosen as the representing of the quasi-clique.

In the ASP code (see Listing 4.2), each quasi-clique is represented by the node having the highest CC_{local} value.

Listing 4.2 – **ASP basis for quasi-clique search.** The first rule compute the local clustering coefficient: it reduce the set of candidate reads to the ones having enough neighbors, then enumerates the number of 3-cliques between these nodes and their neighbors. The second rule defines a representative node for each quasi-clique as the one having the highest local clustering coefficient.

```
1 % Select reads having a sufficient local clustering coefficient, with
    enough similar neighbors.
2 cc_local(R,CC):- read(R), nb_neighbours(R,N), N>=min_neighbors,
3                  S=#sum{1,R1,R2: edge(R,R1,S1), S1>=min_similarity_clique,
4                                  edge(R,R2,S2), S2>=min_similarity_clique,
```

```
5                                   edge(R1,R2,S3), S3>=min_similarity_clique,
6                                   R1<R2}>0,
7                 CC=(S*2)/(N*(N-1)), CC>=min_cc_local.
8
9  % Each quasi-clique is represented by the node having the highest local
       clustering coefficient
10 cliquerep(R):- cc_local(R,CCR); CCR>= CCX: edge(R,X), cc_local(X,CCX).
```

Extend the quasi-cliques to connected components.

The CC_{local} characterizes a quasi-clique with respect to the neighborhood of one representative node. We extend here this notion to define larger sets of nodes, using a *global clustering coefficient* (CC_{global}). It amounts here to searching cliques, not in the whole graph, but close to the pre-defined dense regions of the quasi-cliques. In contrast to the local clustering coefficient, this coefficient enables to define precisely the quasi-cliques.

We compute the local clustering coefficient (Equation 4.2) on larger regions of the graph: we define a "2-quasi-cliques" as two highly connected quasi-cliques (Figure 4.9). The associated ASP code is provided in Listings 4.3.

> **Global clustering coefficient.** This measure is based on the computation of triangles, *i.e.* cliques made of three nodes. In the usual global clustering coefficient, this value is divided by the number of triplets, *i.e.* sets of three nodes connected by at least two edges. We use here a variant, and divide the number of closed triplets by the total number of potential closed triplets.
>
> $$CC_{global} = \frac{S'}{N(N-1)(N-2)/6} \tag{4.2}$$
>
> with S' the sum of closed triplets of nodes, and N the number of nodes involved.

Listing 4.3 – **ASP basis for quasi-clique extension to 2-quasi-cliques.** The global coefficient clustering is computed for each 2-quasi-cliques, by enumerating their number of triangles.

```
1 % Compute global clustering coefficient for each 2-quasi-cliques
2 cc_global(Rep,CCRep):- clique2rep(Rep);  deg(Rep,N);
3     S=#sum{1,X: inclique2(Rep,X), inclique2(Rep,Y), inclique2(Rep,Z),
4                 X<Y, Y<Z,
```

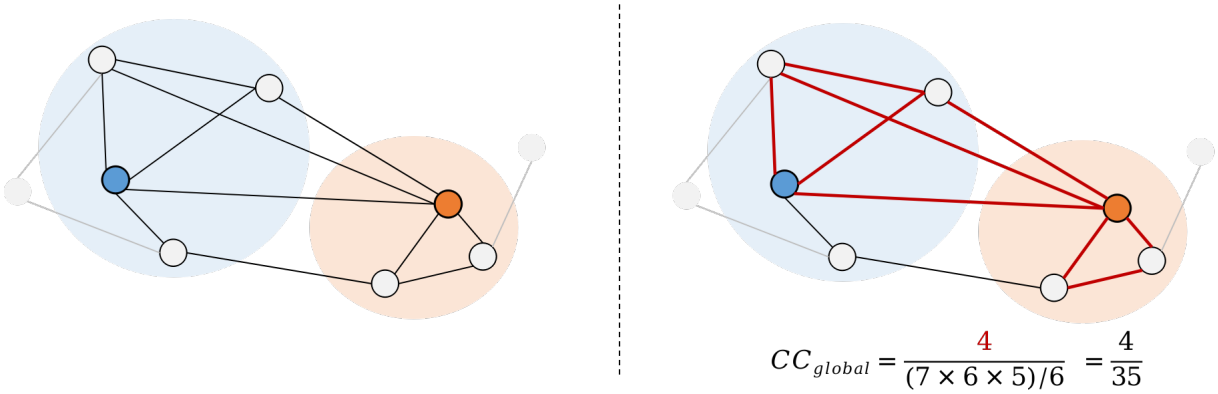


Figure 4.9 – **Example of computation of global clustering coefficient in a 2-quasi-clique.** Computation of the global clustering coefficient on two quasi-cliques: count the number of closed triplets (triangles) within the two quasi-cliques. Nodes connected but that do not belong to the quasi-cliques (in light gray) are not taken into account.

```
5                   edge(X,Y,S1), S1>=min_similarity_clique,
6                   edge(Y,Z,S2), S2>=min_similarity_clique,
7                   edge(X,Z,S3), S3>=min_similarity_clique
8               };
9       CCRep=(S*6)/(N*(N-1)*(N-2)).
```

At this point, we build an abstract graph: the 2-quasi-cliques are set as nodes, and edges are defined based on a *biclustering coefficient* (Equation 4.3) between two nodes. Using this graph representation, edges will represent two highly connected sets of reads. Then, two patterns can be identified in the graph: singletons (reads with in- and out- degree of 0), and connected components (subsets of nodes that are reachable from each other with a path).

> **Biclustering coefficient.** This coefficient measures the connectivity between two quasi-cliques by dividing the number of edges between them divided by the total number of edges in a complete bipartite graph.
>
> $$BCC = \frac{1}{nm} \sum_{x,y} E(x,y) \tag{4.3}$$
>
> with x a node in the first quasi-clique (of n nodes) and y in the other quasi-clique (of m nodes).

To do so, we define the entity of a pseudoclique, representing a 2-quasi-clique (see Listings 4.4). In the case of several overlapping 2-quasi-cliques, the pseudoclique is identified by the 2-quasi-clique having the highest global clustering coefficient. Edges are set between pseudocliques sharing a sufficient biclustering coefficient.
Finally, we gather reads belonging to pseudocliques that are reachable with a path in the graph.

Listing 4.4 – **ASP basis for pseudoclique identification.** Define a pseudoclique as a set of connected 2-quasi-cliques, and set its representative as the 2-quasi-clique having the highest global clustering coefficient. Pseudocliques will be used as nodes to abstract the graph. A biclustering coefficient is computed for each pair of pseudocliques, and is used to connect those that have a sufficient biclustering coefficient.

```
1  % A pseudoclique is represented by the 2-quasi-cliques having the maximal
       global clustering coefficient among the other 2-quasi-cliques connected
       to it
2  pseudoclique(Rep):- cc_global(Rep,CCRep);
3      CCRep>=CCX: overlapping_clique2(Rep,RepX), cc_global(RepX,CCX).
4
5  % Build an abstracted graph with pseudocliques as nodes
6  % Edges are defined as the biclustering coefficient between nodes
7  %  = the ratio of edges with respect to a complete bipartite graph
8  %  edgeP = edge between two pseudocliques
9  %  inP = element belonging to a pseudoclique
10 %  nb_inP = number of elements belonging to a pseudoclique
11 edgeP(P1,P2):- pseudoclique(P1) ; nb_inP(P1,K1) ;
12     pseudoclique(P2) ; nb_inP(P2,K2) ; P1<P2 ; K1<=K2;
13     N=#sum{1,X,Y: inP(P1,X), inP(P2,Y), edge(X,Y,_)}>0;
14     W=N*100/(K1*K1), W>=min_biclustering.
15 edgeP(P1,P2):- pseudoclique(P1) ; nb_inP(P1,K1) ;
16     pseudoclique(P2) ; nb_inP(P2,K2) ; P1<P2 ; K1>K2;
17     N=#sum{1,X,Y: inP(P1,X), inP(P2,Y), edge(X,Y,_)}>0;
18     W=N*100/(K2*K2), W>=min_biclustering.
```

4.4.4 Phase seeds

Within a given connected component, we then group the seed reads to provide some guidance for the future phasing of remaining reads. We aim here at phasing seeds based on their similarity. At this point we do not impose the ploidy level, only setting a large

upper bound to maintain the search within likely values.

To do so, we define four types of relation between seeds, based on their identity and overlap scores. The identity defines similarity relation between two seeds, and this relation is weighted according to the size of their overlap:

- an identity $\geq min_{id}$ represents *similar* seeds, otherwise they are considered *different*;
- over min_{ovlp} the alignment is considered as *highly* reliable, otherwise is it only said as *likely*.

These relations are associated to non-symmetrical situations.
The alignment of two seeds stemming from a same haplotype region will display strong overlap and similarity. But the converse is not true: two seeds may be highly identical due to local similarity but stem from two different haplotypes. However, regions of local similarity should be highlighted in phasing results. Plus, while such seeds are indistinguishable at this stage of the phasing process, the following addition of other reads may provide sufficient information to differentiate the two haplotypes.
The alignment of two seeds stemming from different haplotypes (or different regions of a same haplotype) will not or poorly overlap, or will overlap with lower identity. Yet, the converse is not true. Two seeds originating from a same haplotype region may not produce an alignment with high similarity, for several reasons. First, they may originate from regions that are a bit distant, which impact the overlap level, then the presence of sequencing errors in seeds would also blur the identity.
Taking this into consideration, we rely on both overlap and identity to make phasing decisions.

The specification of a seed is established in two parts: mandatory properties and properties to be optimized.
Constraints.

- We set an upper bound to ploidy (12 in practice) to prevent the algorithm looking for too complex solutions. We do not constrain precisely the ploidy level, as there will be some regions of local identity between haplotypes, which must be allowed to be expressed. Conversely, detected differences between seeds must be taken into account, as they may be due to errors in reads.
- Two *highly similar* seeds must be phased in the same haplotype. Two *different* seeds must not be phased in the same haplotype. In case of conflicts that would render

the problem unsatisfiable, some of such seeds can be left aside as "unphased". Such seeds are likely to contain too much errors and can be considered for further treatment by the user.

Optimization criteria (in decreasing order of priority).

- As much seeds as possible should be phased (when sufficient information is available). Seeds for which no reliable similarity information is available can be affected to a special state instead of being phased. Such seeds can be phased latter based on additional information provided by other reads or by the user;
- as few haplotypes as possible must be used to phase the seeds, following a parsimony principle ;
- two *likely similar* (resp. *different*) seeds should (resp. should not) be phased in a same haplotype.

Listing 4.5 – ASP basis for seed phasing

```
% Associate 0 or 1 haplotype to each anchor with reliable alignment
    information
{phased_anchor(A,H): hap_id(H)}1:- highly_rel_anchor(A).
% When no strong information is available, associate anchor to the "
    uncertain" haplotype
phased_anchor(A,uncertain):- anchor(A), not highly_rel_anchor(A).

% Constraints.
%   highly similar anchors must end up in the same haplotype
:- phased_anchor(A1,H1), phased_anchor(A2,H2),
      sim_rel(high(sim),A1,A2), H1!=H2, A1<A2.
% highly different anchors must NOT end up in the same haplotype
:-  phased_anchor(A1,H), phased_anchor(A2,H),
      sim_rel(high(diff),A1,A2).

% Optimization criteria.
%   1- phase as much anchors as possible
#minimize{1@3,A: anchor(A), not phased_anchor(A,_)}.
%   2- use as few haplotypes as possible ("uncertain" not counted)
#minimize{1@2,H: phased_anchor(_,H), H!=uncertain}.
%   3- minimize inconsistencies between alignment and haplotype assignment
% likely similar anchors should end up in the same haplotype
likely_similar_not_same_hap(A1,A2,H1,H2):- phased_anchor(A1,H1),
    phased_anchor(A2,H2), sim_rel(likely(sim),A1,A2), H1!=H2, H1!=uncertain,
```

```
    H2!=uncertain.
#minimize{1@1,A1,A2,H1,H2: likely_similar_not_same_hap(A1,A2,H1,H2)}.
% likely different anchors should NOT end up in the same haplotype
likely_diff_same_hap(A1,A2,H):- phased_anchor(A1,H), phased_anchor(A2,H),
    sim_rel(likely(diff),A1,A2).
#minimize{1@1,A1,A2,H: likely_diff_same_hap(A1,A2,H)}.
```

4.4.5 Evaluation of seed clustering and phasing

The validation of a method results in bioinformatics is often a problem in its own right. It is the case for the haplotype phasing issue. We propose here a score to evaluate the relevance of the seed clustering process. The clustering consists in partitioning the set of seeds into a certain number of groups that we expect to represent regions of a same haplotype (or at least, chromosome). A way to assess the accuracy of such a partition is to compare it with the true partition, *i.e.* here, the partition based on the built ground truth.

There are several scores to compare two partitions, among which the Jaccard index [Hubert and Arabie 1985] (Equation 4.4) and the Rand index [Rand 1971] (Equation 4.5) are the most famous used.

They are both based on the same principle: two partitions $\mathcal{T}$ and $\mathcal{P}$ of a same set of elements S are identical if, for each pair of elements, these are either in a same class both in $\mathcal{T}$ and in $\mathcal{P}$, or separated in different classes both in $\mathcal{T}$ and $\mathcal{P}$ (such pairs are said to *agree*). For each pair (x, y) in the set of elements, we use the following notations to build a contingency table using par membership in $\mathcal{T}$ and $\mathcal{P}$ (illustrated in Figure 4.10):

- a: x and y belong to the same class in both $\mathcal{T}$ and $\mathcal{P}$;
- b: x and y are in different classes in both $\mathcal{T}$ and $\mathcal{P}$;
- c: x and y belong to same class in $\mathcal{T}$ and are in different classes in $\mathcal{P}$;
- d: x and y are in different classes in $\mathcal{T}$ and belong to same class in $\mathcal{P}$.

The Rand index consists in computing the ratio of pairs that agree. The Jaccard index computes the same ratio, except that it totally ignores the pairs that are simultaneously separated in the partitions.

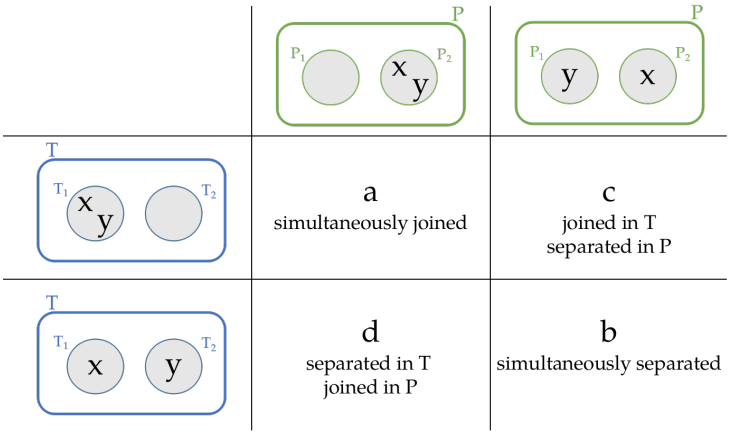


Figure 4.10 – **Notations used by Rand and Jaccard indices for partition comparison.** These indices are based on the comparison of all pairs of elements, determining whether they are joined or separated in the true partition $\mathcal{T}$ and the predicted one $\mathcal{P}$. Partitions can have more than two classes.

$$Jaccard(T,P) = \sum_{(x,y)} \frac{a}{a+c+d}, (x,y) \in S^2, x \neq y \tag{4.4}$$

$$Rand(T,P) = \sum_{(x,y)} \frac{a+b}{n(n-1)/2}, (x,y) \in S^2, x \neq y, |S| = n \tag{4.5}$$

We chose to not rely on these metrics, as their are based on computations over the set of all possible pairs, which is not suitable for very large datasets. The size of partitions are not prohibitive here, as in our experiments, the connected components contained most of the time less than 10 elements (seeds). Yet, we also use this metric to evaluate the accuracy of the remaining reads (see next section), and the number of reads assigned to a connected component most of the time (about 85% in our experiments) exceeded a 500 elements and often (more than one third) over a thousand elements.

Instead, we adapted another distance proposed in [Goldberg, Hayvanovych, and Magdon-Ismail 2010], its complexity only depending on the number of classes in each partition. It is based on finding, for each class of $\mathcal{T}$, its *Best match* class in $\mathcal{P}$, and conversely.

Similarity between two partitions. We have as input two partitions $\mathcal{T}$ and $\mathcal{P}$ of the same set, respectively made of $|C_i|, i \in \{1, .., n\}$ and $|D_j|, j \in \{1, .., m\}$ classes. We define a similarity measure between two classes $sim(C_i, D_j)$ as the number of elements that are both in C_i and D_j.
One defines a mapping between two partitions $\mathcal{A}$ and $\mathcal{B}$ as:

$$\begin{aligned} &\mathcal{A} \to \mathcal{B} \\ &x \mapsto rep_{AB}(x) = argmin_{y \in \mathbf{B}} sim(x, y) \end{aligned}$$

Finally, we can define the similarity between two partitions as the symmetric distance:

$$\begin{aligned} sim(\mathcal{T}, \mathcal{P}) &= \sum_{i=1}^{n} \min_{j=1,..,m} sim(C_i, rep_{\mathcal{TP}}(C_i)) + \sum_{j=1}^{m} \min_{i=1,..,n} sim(D_j, rep_{\mathcal{PT}}(L_j)) \\ &= dist_{\mathcal{T} \to \mathcal{P}} + dist_{\mathcal{P} \to \mathcal{T}} \end{aligned}$$

This measure can be normalized by the total number of distinct elements:

$$score_{acc} = \frac{sim(\mathcal{T}, \mathcal{P})}{2 * |\mathcal{S}|}$$

. It amounts to a symmetric measure of the ratio of elements that were mis-assigned in partitions. It takes its values between 0 and 1, with 1 meaning that every element has been correctly assigned, and 0 meaning that no element has been correctly assigned.

We also depict the fragmentation of the constructed partition, by setting three categories:

- underestimated ploidy if $|P| < |T|$;
- overestimated ploidy if $|P| > |T|$;
- correct ploidy if $|P| = |T|$.

4.5 Phase remaining reads relying on seeds

Once a pseudo-reference for phasing has been established for every connected component, the set of remaining reads (non-seed reads) can be phased, relying on this

guidance.

4.5.1 Assign reads to connected components

Reads are aligned on seeds, so that they can further be assigned to a (or several) connected component(s). To do so, we use the minimap2 aligner, with default parameters. The first m alignments are kept in the aligner's output (minimap2 default $m = 6$). If an alignment has an overlap or identity score below 70%, it is removed (note that here, the soft-clips are taken into account in the percent identity computation to require stronger identity and overlap). Each read is associated this way to several seeds. It is then assigned to every corresponding connected components.

We also compute the ranking of the alignment results as information for the read phasing step: they are filtered first on their overlap, then according to identity.

4.5.2 Phase reads

Within each connected component, a read inherits of only one haplotype among those of the seeds it has been mapped on. If all of its seeds are phased in the same group, then there is only one possible choice for phasing the read.

There are not only a single way to resolve the other cases. We have chosen to prioritize global consistency between seed and read phasing. To do so, we rely on several optimization criteria to describe such consistencies, at different levels (in decreasing level of priority):

Seed level. Seeds cover large regions of an haplotype, and therefore are expected to gather a large number of reads aligned to it. This represents the highest level of consistency required: in the ideal case, for a given seed, the set of reads aligned to it are phased in a single same group, the one of the seed. For each seed, we minimize the number of its aligned reads that are assigned to a different haplotype.

Read level. A read can be aligned on several seeds. Ideally, these seeds would stem from close regions of the same haplotype, but they also may represent local identical regions of different haplotypes. For a given read, we aim at minimizing the number of conflicts between its assigned haplotype and the number of seeds on which it aligns that were assigned to another haplotype.

Read-seed alignment level. Finally, we consider the finest level of consistency for phasing. The different alignments between a read and a set of seeds are ranked according to

their reliability. Therefore, if there is still a tie, we can rely on this information to chose a phasing for the read: the better the alignment, the higher priority it has.

4.6 Reconstruct haplotypes

Finally, we produce haplotigs from the phased reads (including seeds) from each connected component. In each connected component, each phased read has been associated to a local number of haplotype , representing the local phasing between the subset of reads. By exploiting the fact that some reads belong to multiple connected component, we can extend the phased sets of reads in linked connected components. To do so, we affect a global haplotype number for each read, and propagate the information to linked connected components while ensuring to not generate conflicts.

Once this is done, a consensus sequence can be built a consensus sequence for each haplotig, using minimap2 and miniasm.

Reads that could not be phased are also relevant and are reported to the user.

Chapter 5

Evaluation on a real dataset

Contents

Preamble: In this final chapter, we present preliminary evaluation of our phasing method on a real complex dataset: *Adineta vaga*. It is a diploid organism with ancient tetraploidy, for which a recent and reliable phased reference was available. Our challenge was to retrieve most of this phasing while using only PacBio Hifi sequencing data. Thanks to the haplotype aware reference, we were able to evaluate the overall accuracy of our method. Our method resulted in the creation of multiple small local phasing results, covering major parts of haplotypes, and with good phasing results. Our results also revealed the overall similarity or divergence of haplotypes.

5.1 Description of the dataset

Adineta vaga is a bdelloid rotifer. It is a diploid species that shows evidences of ancient tetraploidy [Simion *et al.* 2021]. Its genome is organised in 6 chromosomes, with an organization in closer pairs (they are called *homeologue* chromosomes): 1-4, 2-5 and 3-6, all pairs showing a high level of synteny, which reveals the paleotetraploidy (Figure 5.1). It is thus a hard case study with many similar regions across the genome. This represents a challenge for haplotype phasing as the associated haplotypes are likely to be highly identical, and therefore, harder to distinguish.

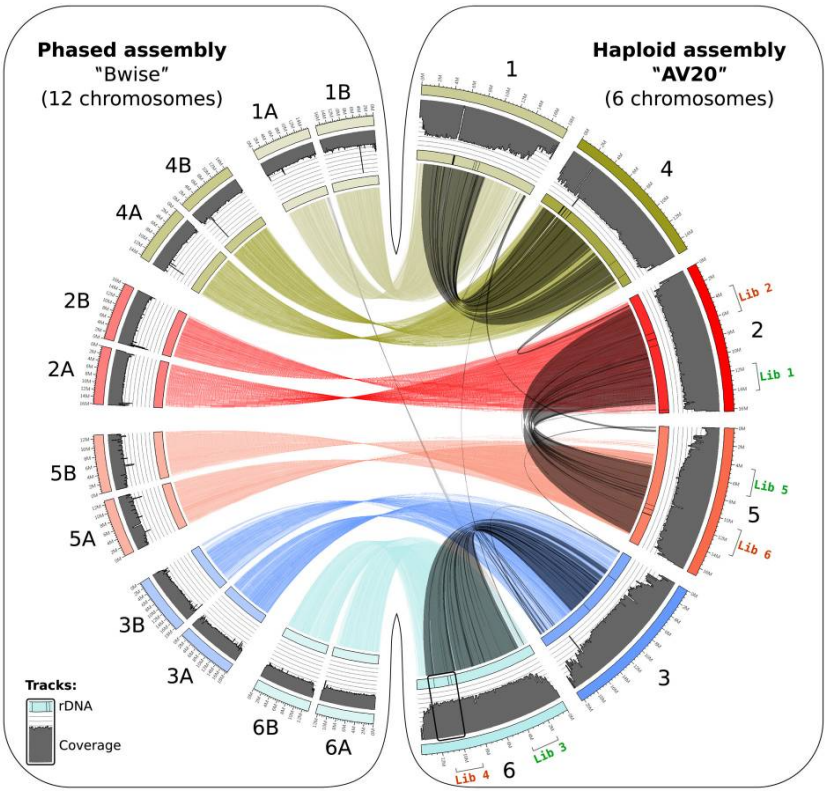


Figure 5.1 – **Structure of *A. vaga*'s genome.** Circos plot of chromosomes and haplotypes of *A. vaga*. The chromosome pairs (1-4, 2-5 and 3-6) indicates an ancient tetraploïdy of the genome. From [Simion *et al.* 2021].

For this dataset, an accurate phased reference assembly is available [1] built from multiple sequencing technologies: Illumina, PacBio, ONT and Hi-C, and phased with Bwise [2].

1. on Zenodo: `https://doi.org/10.5281/zenodo.5126643`
2. `https://github.com/Malfoy/BWISE`

We base our experiments on a set of PacBio Hifi reads, available under the project accession number PRJNA680543. The reference genome as well as the reads have been generated and analyzed in [Simion *et al.* 2021].

The chromosomes are of similar sizes, even considering their haplotype (except for the chromosome 1, for which one haplotype is significantly longer) (see Table 5.1). The overall genome has a GC% of about 30.5%.

Table 5.1 – **Chromosome and haplotype length for the *A. vaga*genome.** These lengths have been computed on the phased assembly of *A. vaga*.

	chr. 1	**chr. 2**	**chr. 3**	**chr. 4**	**chr. 5**	**chr. 6**
hap. A	15694880	16153339	18525074	14476117	12832358	12456688
hap. B	17274718	16173584	18652167	14769778	12833466	13407955

At the time of writing, only the results for this dataset are available. Yet, we are currently evaluating our method on a tetraploid dataset of *Solanum tuberosum* sequenced with ONT technology. We chose to prioritize the study of the *A. vaga* dataset due to the availability of an accurate reference genome, which eases the evaluation of the results.

5.2 Ground truth

A first challenge is to establish a ground truth, representing the objective to be pursued, and on which evaluate our method and results. To do so, we rely on the phased assembly of *A. vaga*. We proceed as follows. All reads of the dataset are aligned against the phased assembly, using minimap2 (default parameters), with at most 6 alignments are kept (minimap2's default). Relying on the overlap and identity scores, we keep only the best alignments, that is, setting the primary alignment as the best one, only those that differ from it by less than 2% in accuracy and less than 5% in coverage. In addition, we require stringent conditions on alignments, with overlap over at least 50% of their length, and identity greater than 80%. Alignments of identity between 80% and 90% are tagged as *uncertain*.

We therefore obtain for each read the set of likely corresponding haplotype identifier(s). If the set is empty, we exclude the corresponding read from the evaluation, as we assume we do not have reliable evidence.

Using this protocol, almost every reads (77.2%) could be assigned at least at chromosome level, and many (60.9%) were identified at haplotype level. Overall, almost all

chromosomes and haplotypes display similar amount of reads that are assigned. This indicates that all haplotypes have approximately the same depth of coverage. Yet, the chromosome 5 stands out: concerned reads cannot be associated unambiguously to a single haplotype. It is not caused by sequencing errors nor an by erroneous assembly, but to the inherent structure of this chromosome. Indeed, in [Simion *et al.* 2021], authors showed that the fifth chromosome displayed a high proportion of regions with loss of heterozygosity (LOH) events: while being diploid, several regions of chromosome five do not show heterozygosity between its haplotypes, and such regions are nearly identical.
Only a negligible part (0.1%) had an ambiguous assignment to multiple chromosomes. Yet, a non-negligible part (22.7%) could not be assigned with sufficient confidence, due to poor alignment results. The divergence between these reads and the built assembly could be due to high sequencing error rate, or remaining problems in the assembly.

Table 5.2 – **Distribution of reads and the haplotypes their were assigned to.** The table display ratio (%, rounded to the nearest tenth) of reads that were aligned on each chromosome and haplotypes (those with identity over 90% + those of identity between 80% and 90%). Reads that were assigned to 0 or multiple chromosomes are not displayed here.

Chromosome	Both haplotypes	Hap. A only	Hap. B only	Total
1	0.6	5.5 + 0.2	6.8 + 0.2	13.3
2	1.1	6.0 + 0.2	6.4 + 0.2	13.9
3	0.8	5.1 + 0.1	6.6 + 0.2	12.8
4	1.0	6.6 + 0.2	5.9 + 0.2	13.9
5	11.6	0.0	0.0	11.6
6	1.2	4.8 + 0.1	5.4 + 0.2	11.7
Total	16.3	28.8	32.1	77.2

5.3 Read size distribution and seed selection

We select the longest reads of dataset as seed candidates. Based on the read length distribution, we define a threshold so that the selected reads provide sufficient coverage and depth of sequencing of the genome.

In the case of *A. vaga*, we set the threshold at 25kb. It represents 45 420 retained reads as seeds (near 2% of the 1 703 204 reads) and a coverage of about 6X, for a genome length of 200Mb (Figure 5.2). As the reads are HiFi PacBio reads, they are expected to be quite accurate: for a given chromosome region, the aligned reads will not diverge much from

this sequence, and therefore are likely to be quite similar to each other. In this context, the seeds will cluster easily, and it is not necessary to select a huge amount of seeds.

If a more erroneous dataset had been used (for example with ONT reads), the depth of coverage threshold would have been higher. As erroneous reads will show more differences with the actual genome, there would have been fewer reads to be highly connected to each other, and therefore fewer (and likely smaller) connected components. So the input set of the longest reads should have been larger to compensate for this.

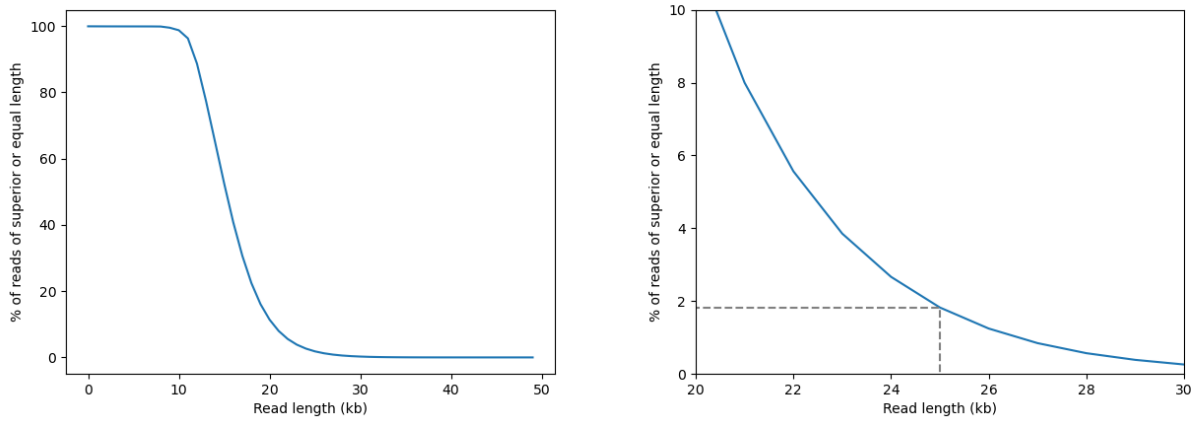


Figure 5.2 – **Read length distribution for *A. vaga*.**

5.4 Seed all-*vs*-all alignment and similarity graph

We built a similarity graph relying on alignments over the set of seeds. We do not require high levels of similarity: seed pairs sharing at least 50% of identity are considered *similar*. Likewise, we consider that seed alignments of overlapping ratio over 70% are *reliable*.

Over the set of 45,420 selected seeds, the majority (42,924 *i.e.* 94.5%) could be aligned to at least one other seed with sufficient overlap and identity.

The associated similarity graph contains 124,149 edges, indicating a rather sparse graph. This is consistent with a graph where similarities are local: the Figure 5.3, displaying a region of the graph, reveals connected components. The overall graph contains numerous connected components, that we analyze in the following sections.

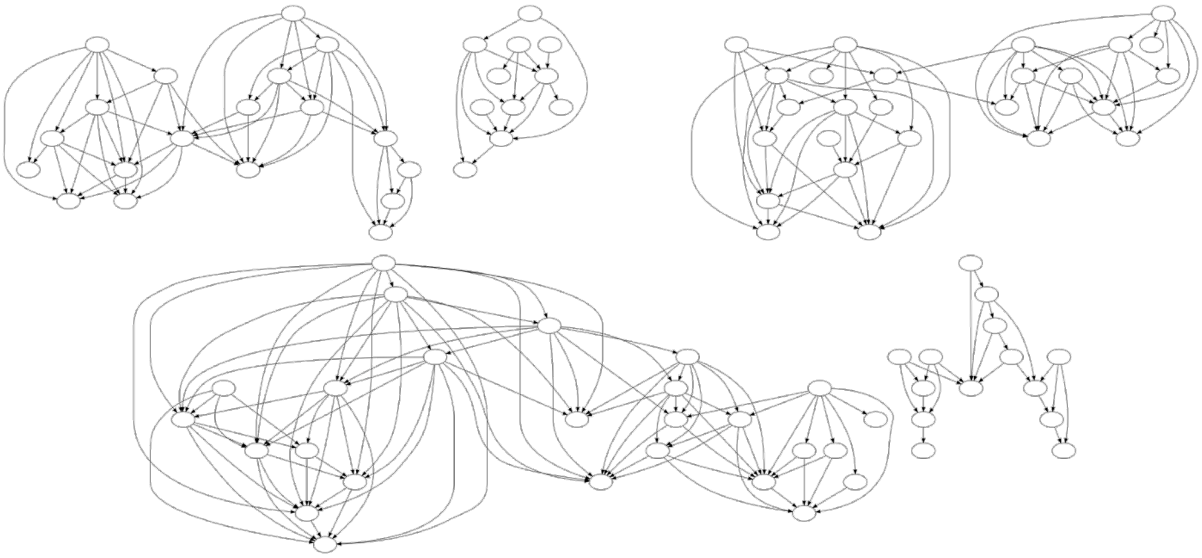


Figure 5.3 – **Seed similarity graph.** Only a small region of the similarity graph is displayed.

5.5 Looking for connected component

Following the method described in chapter 4, we look for groups of highly interconnected nodes, forming connected components, in the similarity graph. These connected components will be used to delineate independent subsets of the graph, on which we can apply the phasing algorithm, instead of applying it on the whole graph.

We used the following thresholds:

- minimum for local clustering coefficient $min\_cc\_local = 0.5$ (which means that at least half of the edges to form a clique must be present in the graph);
- minimum biclustering coefficient $min\_biclustering = 0.15$.

This resulted to the identification of 3,122 connected components. Most of the time, these connected components grouped small sets of seeds: 75% contain at most 11 seeds, with a median of 6 seeds per connected components, and the largest one containing 115 seeds (Figure 5.4).

By using the ground truth information of each seed, we assessed the ability of the formed connected components to gather seeds within same chromosomes and same haplotypes. These connected components achieved it with a rather good accuracy (Table 5.3): almost all (2,903 *i.e.* 93%) gather seeds originating from a same chromosome, and about one third (1,096 *i.e.* 35%) identified groups of same haplotype.

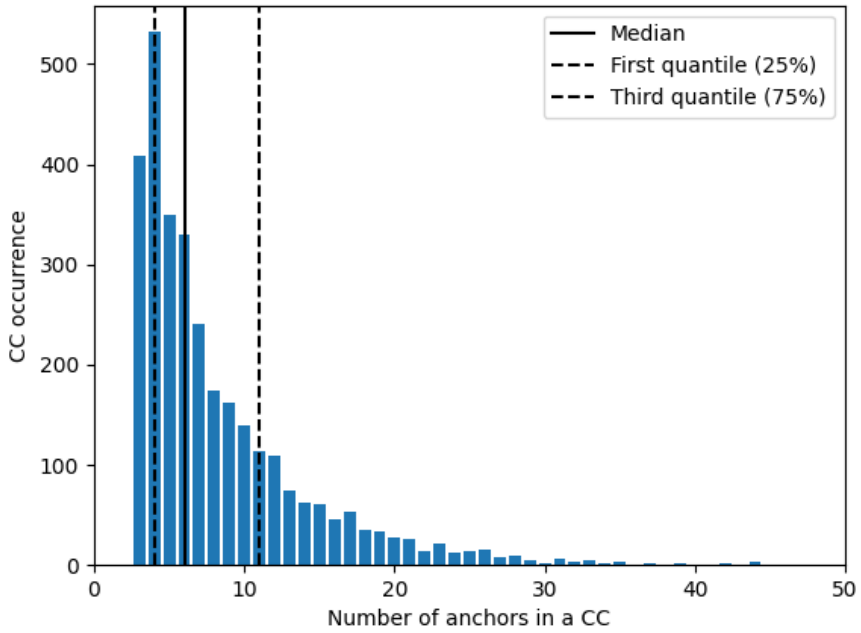


Figure 5.4 – **Distribution of connected components sizes (*A. vaga*).**

These results are consistent with the expected distribution of reads presented earlier: the number of connected components is almost even among haplotypes and chromosomes, except for chromosome 5 for which the results indicate again a high identity between the two haplotypes.
Among the remaining 219 connected components, 217 contained only seeds for which the ground truth could not be established with enough reliability. The two others gathered seeds from chromosomes 1 and 4, and 1 and 6, and are likely to be due to a local identical region between these chromosomes.

Table 5.3 – **Number of connected components per chromosome and haplotype.**

Chromosome	Both haplotypes	Hap. A only	Hap. B only	Total
1	261	107	172	540 (17.3%)
2	330	85	99	514 (16.5%)
3	267	114	175	556 (17.8%)
4	294	82	77	453 (14.5%)
5	412	1	0	413 (13.2%)
6	243	82	102	427 (13.7%)

5.6 Seed phasing

Once the connected components are defined, seeds are phased according to their similarity. We display here some relevant examples (Figures 5.5, 5.6 and 5.7) to depict the general cases and problematic cases, then we provide an overview of the overall phasing results (Table 5.4).

5.6.1 Examples of phasing instances and challenges

Several configuration are encountered within connected components, depending on the expected ploidy and the actual similarity relations between the seeds.

A first simple situation is depicted in Figure 5.5. In this case, there are four seeds to be phased. Their ground truth information indicates that they are all expected to be phased in the second haplotype of the chromosome 1. In addition, they form a dense graph of similarity: for every seed, strong similarity relations has been previously found. This is a very simple configuration, with no ambiguity in the similarity information, so the phasing algorithm groups all these seeds together.

The second example (Figure 5.6) depicts the case where the seeds originate from two different chromosomes. In addition, one of the seeds (7) has an ambiguous ground truth: we could not determine whether it belong to the first or the second haplotype of the chromosome 3. The phasing algorithms succeed at separating the seeds, based on their similarity relations. In addition, it provides a resolution for the seed 7: according to its similarities and differences with other seeds, it seems that this seeds is actually more likely to originate from the first haplotype than to the second one.

The final example illustrates a more complex situation: four seeds are expected to belong to a same haplotype, yet their alignment information revealed inconsistencies between them. The phasing algorithm therefore resulted in two groups of haplotypes, which could be interpreted as a wrong phasing result. However, in this case, it is more likely that the seeds 3 and 4 (that are highly different from their other overlapping seeds) are actually erroneous seeds. In such a case, this can be confirmed with the next steps of phasing, with the additional information brought by reads: if these two seeds are truly erroneous, then it is likely that (almost) no read could be aligned on it.

This step of the algorithm enables to identify problematic seeds: they are unphased and therefore are left aside (as we consider that they contains too much errors to be correctly phased). It is also possible that some seeds could not be phased at this point,

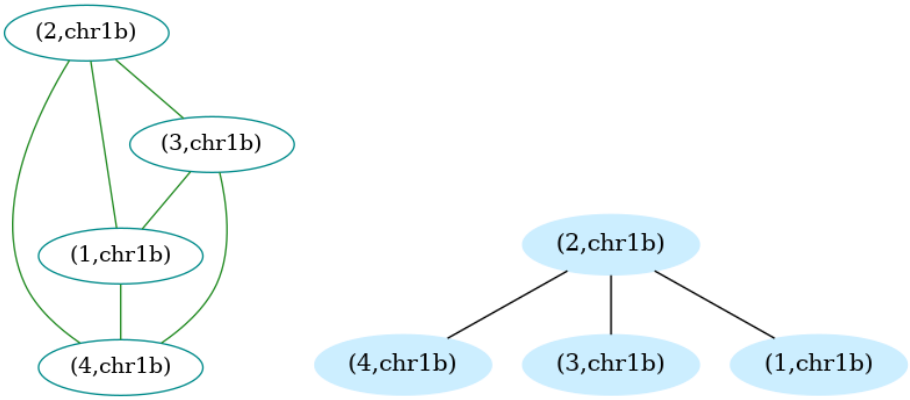


Figure 5.5 – **Example 1. One expected haplotype.** (**Left**) Similarity relations between seeds. Nodes are labeled by a number and their ground truth haplotype, and are colored according the latter information. Edges represent similarity relations between seeds, with a color code (here, only green edges are present, *i.e.* all edges represent high similarity). (**Right**) Result of ASP phasing algorithm. Nodes are labeled as in the similarity graph. They are colored according to the phasing results: two nodes phased in the same group have the same color, and two nodes phased in different groups have different colors (the color itself does not have any meaning). Edges represent the reliability of the phasing results, as well as their consistency with the ground truth phasing. A black edge between two nodes means that these nodes are phased together both in the ground truth and in the computed phasing. A solid edge means that the phasing result is unambiguous.

because of a lack of information. In such cases, these seeds are identified, and the next steps of phasing will try to phase them relying on additional information.

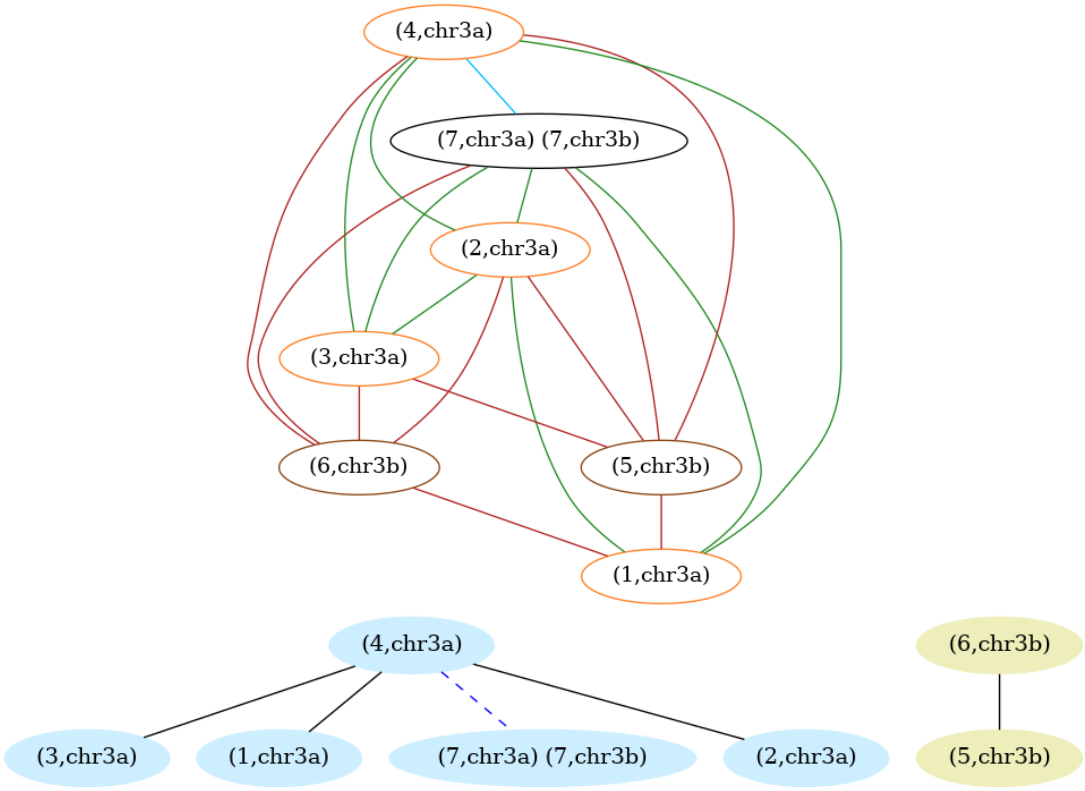


Figure 5.6 – **Example 2. Two expected haplotypes.** (**Top**) Red edges indicates that the two connected nodes are highly different, and blue ones show weak similarities. (**Bottom**) The phasing algorithm recreated the two haplotype groups, and suggests that the ambiguous node 7 is more likely to belong to the haplotype 3A. The blue dotted edge indicates that the phasing result was not sufficiently supported to be certain.

5.6.2 Evaluate accuracy of seed clustering

We evaluate the phasing results for the seeds (see Figure 5.4 for detailed results at haplotype level, and Figure 5.5 for results at chromosome level), relying on the score $score_{acc}$ defined in the previous chapter, with the following thresholds:

- $score_{acc} < 0.7$: many assignment errors;
- $0.7 \leq score_{acc} < 0.9$: some assignment errors;
- $score_{acc} \geq 0.9$: few assignment errors.

.

In these tables, each line gather the result of associated haplotype/chromosome connected components. Each cell represent the fraction of involved connected components for the given category of result.

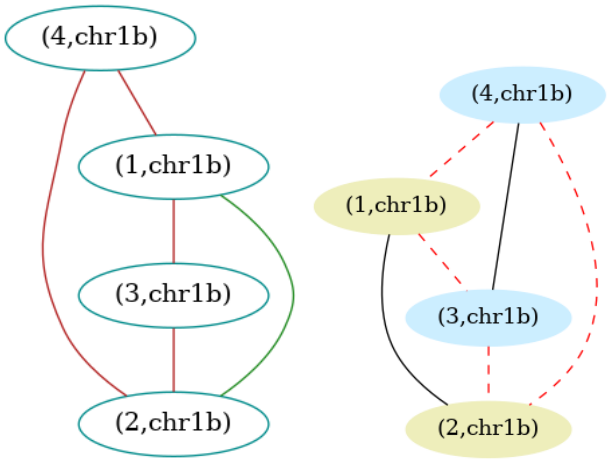


Figure 5.7 – **Example 3. One expected haplotype.** (**Left**) While belonging two the same haplotype, some seeds are conflicting with the others. (**Right**) The dotted red edges indicates that the nodes have not been phased as expected: the conflicts resulted in the incorrect solution of forming two groups of seeds.

Overall, the phasing of the seeds gives good results: for many connected components, the estimation of the ploidy was correct and few assignation errors were made. It also appears that for almost every haplotype and chromosome, the phasing tends to be too much fragmented, which increases the level of assignation errors.

As depicted in the previous section, a fragmented phasing estimation is not necessarily wrong, and can be caused by some erroneous seeds that are left aside in a dedicated group, which can be counterbalanced in the following phasing steps.

Table 5.4 – **Evaluation of phasing results for seeds (haplotype level).** For each chromosome and haplotype, shows the level of fragmentation and accuracy of phasing of associated connected components: each cell contains the fraction of connected component for a given haplotype, with the fragmentation and accuracy categories (the sum of each line equals 100). For each row, the two highest values are highlighted in green, and other cells of value over 10 are highlighted in blue.

Seed phasing ground truth		Underestimated ploidy			Correct ploidy			Overestimated ploidy			Total (%)
		Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	
1	A	1,05	3,16	0,00	31,58	2,11	3,16	9,47	47,37	2,11	100
	B	2,61	13,07	0,00	38,56	7,19	1,31	8,50	25,49	3,27	100
	Both	7,41	13,58	4,53	20,58	18,93	10,70	5,76	11,52	7,00	100
2	A	0,00	3,75	0,00	16,25	5,00	3,75	10,00	57,50	3,75	100
	B	0,00	0,00	0,00	24,18	3,30	0,00	5,49	64,84	2,20	100
	Both	4,09	12,58	7,55	18,24	22,01	9,75	1,89	14,47	9,43	100
3	A	0,00	7,00	0,00	28,00	7,00	0,00	6,00	50,00	2,00	100
	B	1,86	8,70	0,00	39,13	4,35	0,62	8,70	32,92	3,73	100
	Both	3,28	14,34	4,51	22,54	20,49	10,25	5,74	12,70	6,15	100
4	A	2,82	2,82	0,00	21,13	5,63	1,41	4,23	53,52	8,45	100
	B	1,43	4,29	0,00	34,29	4,29	1,43	5,71	40,00	8,57	100
	Both	3,17	14,44	6,69	16,20	20,42	10,56	2,46	15,49	10,56	100
5	A	-	-	-	-	-	-	-	-	-	-
	B	-	-	-	-	-	-	-	-	-	-
	Both	0,26	3,91	0,26	37,24	7,03	3,39	12,24	33,33	2,34	100
6	A	1,30	7,79	0,00	20,78	2,60	2,60	5,19	55,84	3,90	100
	B	2,17	8,70	0,00	27,17	5,43	1,09	9,78	40,22	5,43	100
	Both	4,37	15,72	8,30	17,03	15,28	11,79	3,06	9,61	14,85	100

5.7 Haplotype reconstruction

Once the seeds are phased for each connected components, remaining reads are affected to their closest connected component, according to their similarity with seeds. Then, within each connected components, reads are phased according to the seed phasing results (Table 5.6).

These overview of the results indicate a quite good ploidy estimation, and overall

Table 5.5 – **Evaluation of phasing results for seeds (chromosome level).** For each chromosome, shows the level of fragmentation and accuracy of phasing of associated connected components: each cell contains the fraction of connected component for a given chromosome, with the fragmentation and accuracy categories (the sum of each line equals 100). For each row, the two highest values are highlighted in green, and other cells of value over 10 are highlighted in blue.

Seed phasing ground truth	Underestimated ploidy			Good fragmentation level			Overestimated ploidy			Total (%)
	Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	
1	4,68	11,41	2,24	28,31	12,02	6,31	7,33	22,81	4,89	100
2	2,66	8,38	5,11	19,02	15,54	7,16	3,89	31,08	7,16	100
3	2,18	11,09	2,18	28,91	12,67	5,15	6,73	26,53	4,55	100
4	2,82	10,82	4,47	20,00	15,29	7,53	3,29	25,88	9,88	100
5	0,78	1,04	0,00	39,22	4,68	1,30	13,51	37,66	1,82	100
6	3,27	12,56	4,77	20,10	10,55	7,54	5,03	25,63	10,55	100

few haplotype assignation errors.

Overall, 1107 haplotigs were obtained for the 12 different haplotypes. These built connected components are overall well distributed along the haplotypes (Figures 5.8).

The haplotypes of chromosome 1 seem to have diverged a lot: they have a size difference of about 2Mb. In addition, there seems to have been either sequencing troubles, or issues during sequencing: the end of haplotype A and the start of haplotype B are poorly covered by connected components. It can be due to an issue during sequencing, and these parts of the chromosomes were not sequenced at all. Or, these parts have been sequenced but the resulting reads were too error prone to form pseudocliques.
Along the chromosome, we can observe a slight change in the slope of links: the haplotype B seems to accumulate small insertions compared to the haplotype A. We can also notice that the connected components are colinear in the two haplotypes, which suggests that the absence of large rearrangement events.

In contrast, the distribution of the connected components along the haplotypes of the chromosome 5 are in agreement with their known loss of heterozygosity: they have equal lengths, and all the connected components of one haplotype is (colinearly) associated with another connected component in the other haplotype.

Table 5.6 – **Evaluation of phasing results for reads (chromosome level).** For each chromosome, shows the level of fragmentation and accuracy of phasing of associated connected components: each cell contains the fraction of connected component for a given chromosome, with the fragmentation and accuracy categories (the sum of each line equals 100). For each row, the two highest values are highlighted in green, and other cells of value over 10 are highlighted in blue.

Seed phasing ground truth	Underestimated ploidy			Correct ploidy			Overestimated ploidy			Total (%)
	Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	Few errors	Some errors	Many errors	
1	2,78	3,18	0,00	51,39	11,14	2,55	4,08	21,22	3,66	100
2	1,12	2,23	0,00	48,66	13,88	2,93	10,01	17,97	3,21	100
3	4,01	3,37	0,00	57,16	11,90	1,65	7,06	12,48	2,37	100
4	1,90	1,65	0,00	53,28	12,13	2,39	18,25	6,93	3,47	100
5	0,00	0,00	0,00	14,99	0,00	0,00	70,64	14,29	0,09	100
6	2,22	1,94	0,00	55,63	12,13	2,69	8,66	12,47	4,26	100

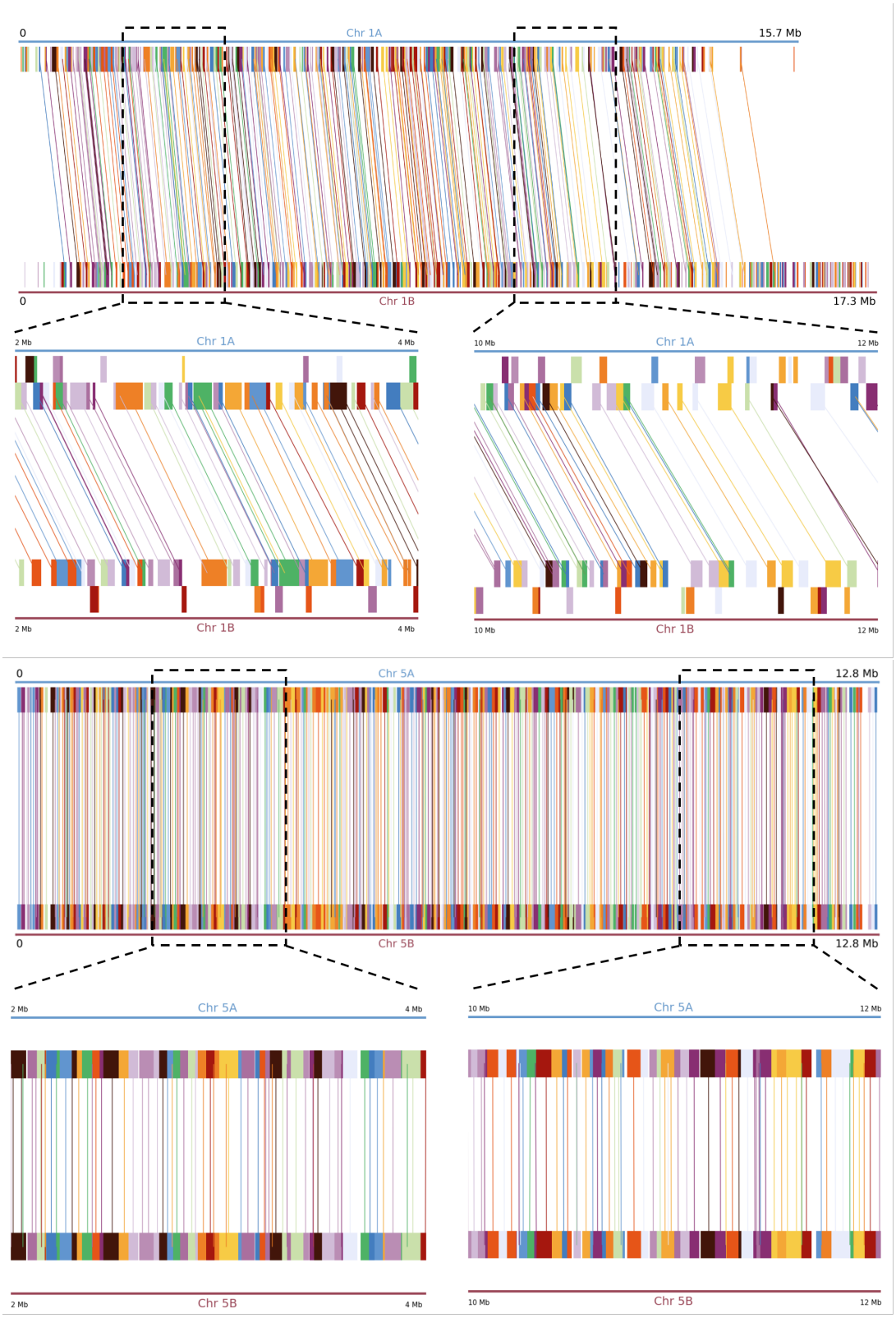


Figure 5.8 – **Distribution of connected components along chromosomes 1 and 5 of *A. vaga*.** Each colored box represents a connected components. They can be either alone (then displayed on first and last lines), or related to a similar region of the other haplotype (then, two boxes of the same color are displayed and linked).

Conclusion and perspectives

Error profile of ONT sequencer

This thesis has been oriented towards the use of sequencing data generated with ONT sequencers. As this technology has gaining interest, and is subject to frequent updates, our first contribution aims at compensating for a lack of knowledge concerning the error profile of ONT sequencers. We analyzed several aspects of Nanopore-generated reads and highlighted major sources of errors. Our evaluation has been conducted based on a pool of multiple bacteria (of different species, as well as identical species with different strains) and human datasets, of varying length and GC content. We relied on sequenced read sequences coupled with available reference genomes, as well as the associated raw electrical signal that delivers valuable indications on the underlying source of mistakes.

We start our analyses by evaluating and comparing the global performances of the two basecalling modes available (at the time of this thesis), HAC and FAST. Both modes shows similar error profiles, yet the former mode enables to obtain greater results (about 2% less error rates), which is crucial in fine analyses of genomes. The FAST mode reduces computation time only by half. We therefore would recommend to rather use the HAC mode, as the gain of accuracy is worth the additional computation time.
We also study more precisely the type of errors. We show that the ONT sequencers are subject to a significant bias concerning substitution errors: purines are highly likely to be erroneously switched (A-to-G and G-to-A substitutions), as well as pyrimidines (although to a lesser degree). Concerning indel errors, the analysis of the electrical signal revealed that a faster DNA translocation leads to more errors (mainly deletions), but also that a too slow translocation is a source of errors too (mainly insertions).
Furthermore, as the ONT sequencing do not necessarily require PCR amplification, the generated reads are expected not showing GC bias. We could verify that the sequenced reads indeed did not suffer from GC bias affecting their depth of sequencing. However, we found that the GC content of reads impact the sequencing quality, with lower GC% better sequenced.

We also provided some insights concerning the systematic errors of Nanopore sequencing, quantifying them in their major sources: homopolymers, but also short repeated pattern (of size 2 or 3). In such contexts, longer words are incorrectly sequenced as they are wider than the reading head of the sequencing device, and as the electrical signal shows fewer variations, thus being harder to correctly segment. Such errors are expected to be less abundant with the new chemistry (R10 flowcells) aiming at better handling homopolymeric regions.
Moreover, the signal exploitation enabled us to identify the patterns of base succession that are harder to correctly sequence, as their associated electrical values are close.

Finally, we developed the SeqFaiLR tool that enables to monitor the evolution in sequencing error profile of Nanopore data over their frequent updates. Overall, we expect that this work will be further used as basis for future read correction tools, and for a better understanding of ONT sequencing.

Polyploid phasing from long reads based on ASP

As a second contribution, we designed a new approach for haplotype phasing.

A reference-free haplotype phasing method for polyploid data

While existing approaches rely on the use of a reference genome (or reference set of Illumina reads) and known variants, we propose a method relying on long reads only (Nanopore or PacBio reads). This represents a great challenge, as it makes haplotyping more difficult, yet this could enable to produce more accurate assemblies, as a reference genome may be misleading. We also aim as providing a method that is suited for the phasing of polyploid organisms, which is rarely done for long read datasets. In addition, we rely on the use of the Answer Set Programming (ASP) to define and solve this haplotyping problem.

In order to compensate for the absence of reference genome, we based our method on a divide and conquer strategy, by delimiting sub-problem instances: from a restrained set of the longest reads, we build a similarity graph in which we search connected components. In this way, we are able to create multiple instances to be solved independently, and for which a pseudo-reference can be used to guide phasing. The longest reads are phased within each connected component, and are then used as anchors to associate re-

maining reads to connected components. Reads are phased according to their similarity with the anchors, maximizing the consistency with the phased anchors.

We applied our haplotype phasing algorithm on a diploid (showing ancient tetraploid state) dataset for which a confident phased reference genome is available to evaluate the results. We were able to divide the overall problem of phasing into numerous smaller instances, that were resolved independently. Our algorithms showed good phasing results, but its accuracy can still be improved.

For the moment, the evaluation has been made only on the diploid dataset of *A. vaga*. Yet, we designed our algorithm to be able to phase polyploid datasets as well. We are now assessing its performance exploiting a tetraploid dataset of *Solanum tuberosum*, sequenced with ONT technology. This experiment will provide us insights on the applicability of our phasing algorithm on read polyploid datasets. The challenges are manifold: ensure it will scale, as the complexity of the phasing problem is increased in the polyploid case; verify that it is able to separate the haplotypes (as locally identical haplotypes may happen); and correctly assess the ploidy level. In addition, as ONT sequencing data have a higher error rate, and more systematical errors, than PacBio HiFi data, the phasing algorithm may have more troubles to distinguish true haplotypes.
This dataset has been used to evaluate the two polyploid phasing algorithms Whatshap Polyphase [Schrinner *et al.* 2020] and nPhase [Saada *et al.* 2021], which will enable us to compare our results to these state of the art algorithms.

We also provided some guidance for a final representation of haplotypes, with an haplotig graph, which enable to explicitly point the regions of identity between haplotypes, as well as their differences (we are still working on its implementation). As our method does not rely on the use of known variants, this representation could be a base of work for the search of variants, located in the haplotype-specific regions. In such context, a bi-directional representation of k-mers (or unitigs) in the haplotig graph could be suitable to reveal events of inversions between haplotypes.
It could also be used for other purposes: for visualization of the haplotigs (size, length, number, coverage of expected haplotypes), for their exploration (for instance in regions in which the phasing did not performed well), and finally for evaluation (by comparing the haplotig graph built from a reference, with one made from phasing results).

We are planning to write an article of this method, as well as its evaluation on real datasets and compared to other polyploid phasing tools.

Interaction with the user

A major goal of our proposed method is to provide relevant information to the user, therefore we do not output a single solution, but an answer based on the whole set of optimal solutions found. To do so, we relied on the Answer Set Programming (ASP). Overall, we formalized the haplotype phasing problem from long read data, and build a framework with defined optimization criteria that are easily adjustable depending on the user needs or preferences.
We are also able to provide the user confidence levels about the different parts of the results: by explicitly outputting reads that could not be aligned, clustered or phased, or by pointing the cases of inconsistencies in the data. In such cases, this information would enable the user to reconsider previous choices made in the algorithm, and to have insights on the confident or fragile parts of the results.

A possible improvement would be to integrate the *asprin* tool [Brewka *et al.* 2015] to ease the formulation of personalized constraints and optimizations.

Decrease computation time

We computed the similarity relations between reads with alignments, which a highly time consuming. It would be interesting to investigate whether the alignment process - that are computationally costly - could be replaced by overlaps (maybe coupled with optimized indexation structure for example). It would represent a huge reduction of time and memory used. For example, an estimation of overlap (size, length, position, and identity) could be computed based on sets of small perfect seeds matches, without the need to compute the full alignment.

Yet, it remains to know if the estimated similarity would be sufficient to still be able to discriminate haplotypes. A work on the overall optimization of the algorithm would be highly valuable, making it possible to be applied on larger and more complex genomes (in particular, plant genomes with many repetitions and complex variants).

Handle additional data

In our first formulation of the phasing algorithm, we only rely on long read sequencing data. Yet, a further implementation could be to use additional data, such as genotype information, which could be used either to build new constraints, or used as seeds.

Bibliography

Abed, Amina and François Belzile (2019), « Comparing single-SNP, multi-SNP, and haplotype-based approaches in association studies for major traits in barley », in: The Plant Genome, DOI: 10.3835/plantgenome2019.05.0036 (cited on pages 14, 30).

Altschul, S. F. *et al.* (1990), « Basic local alignment search tool », in: J. Mol. Biol., DOI: 10.1016/S0022-2836(05)80360-2 (cited on page 73).

Amarasinghe, Shanika L. *et al.* (2020), « Opportunities and challenges in long-read sequencing data analysis », in: Genome Biology, DOI: 10.1186/s13059-020-1935-5 (cited on page 52).

Angiuoli, Samuel V. and Steven L. Salzberg (2011), « Mugsy: fast multiple alignment of closely related whole genomes », in: Bioinformatics, DOI: 10.1093/bioinformatics/btq665 (cited on page 104).

Au, Kin Fai *et al.* (2012), « Improving PacBio long read accuracy by short read alignment », in: PLoS One, DOI: 10.1371/journal.pone.0046679 (cited on pages 65, 75, 82).

Baaijens, Jasmijn A. and Alexander Schönhuth (2019), « Overlap graph-based generation of haplotigs for diploids and polyploids », in: Bioinformatics, DOI: 10.1093/bioinformatics/btz255 (cited on page 97).

Bansal, V. and V. Bafna (2008), « HapCUT: an efficient and accurate algorithm for the haplotype assembly problem », in: Bioinformatics, DOI: 10.1093/bioinformatics/btn298 (cited on page 93).

Benjamini, Yuval and Terence P. Speed (2012), « Summarizing and correcting the GC content bias in high-throughput sequencing », in: Nucleic Acids Research, DOI: 10.1093/nar/gks001 (cited on pages 32, 58).

Berkemer, Sarah J., Christian Höner zu Siederdissen, and Peter F. Stadler (2021), « Compositional Properties of Alignments », in: Math.Comput.Sci., DOI: 10.1007/s11786-020-00496-8 (cited on page 104).

Berlin, Konstantin *et al.* (2015), « Assembling large genomes with single-molecule sequencing and locality-sensitive hashing », en, in: Nat. Biotechnol., DOI: https://doi.org/10.1038/nbt.3238 (cited on pages 66, 71, 74).

Bhat, Javaid Akhter *et al.* (2021), « Features and applications of haplotypes in crop breeding », in: Commun. Biol., DOI: 10.1038/s42003-021-02782-y (cited on page 30).

Blanchette, Mathieu *et al.* (2004), « Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner », in: Genome Res., DOI: 10.1101/gr.1933104 (cited on page 104).

Bonizzoni, Paola *et al.* (2016), « On the Minimum Error Correction problem for haplotype assembly in diploid and polyploid Genomes », in: Journal of Computational Biology, DOI: 10.1089/cmb.2015.0220 (cited on page 92).

Borel, Franck *et al.* (2002), « Multiple centrosomes arise from tetraploidy checkpoint failure and mitotic centrosome clusters in p53 and RB pocket protein-compromised cells », in: Proc. Natl. Acad. Sci. U.S.A., DOI: 10.1073/pnas.152205299 (cited on page 29).

Boža, Vladimír, Broňa Brejová, and Tomáš Vinař (2017), « DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads », in: PLoS One, DOI: 10.1371/journal.pone.0178751 (cited on page 49).

Brewka, Gerhard *et al.* (2015), « asprin: customizing answer set preferences without a headache », in:
AAAI'15: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press, DOI: 10.5555/2886521.2886524 (cited on page 152).

Brown, Daniel G. and Ian M. Harrower (2006), « Integer programming approaches to haplotype inference by pure parsimony », in: IEEE/ACM Trans. Comput. Biol. Bioinf., DOI: 10.1109/TCBB.2006.24 (cited on pages 99, 110).

Browne, Patrick Denis *et al.* (2020), « GC bias affects genomic and metagenomic reconstructions, underrepresenting GC-poor organisms », in: GigaScience, DOI: 10.1093/gigascience/giaa008 (cited on page 59).

Browning, Sharon R. and Brian L. Browning (2011), « Haplotype phasing: existing methods and new developments », in: Nat. Rev. Genet., DOI: 10.1038/nrg3054 (cited on page 86).

Camacho, Christiam *et al.* (2009), « BLAST+: architecture and applications », en, in: BMC Bioinformatics, DOI: 10.1186/1471-2105-10-421 (cited on page 66).

Carradec, Quentin *et al.* (2020), « A framework for *in situ* molecular characterization of coral holobionts using nanopore sequencing », in: Sci. Rep., DOI: 10.1038/s41598-020-72589-0 (cited on page 33).

Castro-Wallace, Sarah L. *et al.* (2017), « Nanopore DNA sequencing and genome assembly on the International Space Station », in: Sci. Rep., DOI: 10.1038/s41598-017-18364-0 (cited on page 33).

Chaisson, Mark J and Glenn Tesler (2012), « Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory », en, in: BMC Bioinformatics, DOI: 10.1186/1471-2105-13-238 (cited on page 66).

Chen, Yen-Chun *et al.* (2013), « Effects of GC bias in Next-Generation-Sequencing data on *de novo* genome assembly », in: PLoS ONE, ed. by Ying Xu, DOI: 10.1371/journal.pone.0062856 (cited on pages 32, 58).

Chen, Yu *et al.* (2021), « Accurate long-read *de novo* assembly evaluation with Inspector », in: Genome Biol., DOI: 10.1186/s13059-021-02527-4 (cited on page 83).

Cheng, Haoyu *et al.* (2021), « Haplotype-resolved *de novo* assembly using phased assembly graphs with hifiasm », in: Nat. Methods, DOI: 10.1038/s41592-020-01056-5 (cited on page 98).

Chikhi, Rayan and Paul Medvedev (2014), « Informed and automated k-mer size selection for genome assembly », in: Bioinformatics, DOI: 10.1093/bioinformatics/btt310 (cited on page 37).

Chin, Chen-Shan, David H. Alexander, *et al.* (2013), « Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data », in: Nat. Methods, DOI: 10.1038/nmeth.2474 (cited on pages 81, 82).

Chin, Chen-Shan, Paul Peluso, *et al.* (2016), « Phased diploid genome assembly with single-molecule real-time sequencing », en, in: Nat. Methods, DOI: 10.1038/nmeth.4035 (cited on page 67).

Compeau, Phillip E. C., Pavel A. Pevzner, and Glenn Tesler (2011), « How to apply de Bruijn graphs to genome assembly », in: Nat. Biotechnol., DOI: 10.1038/nbt.2023 (cited on page 36).

Cornish-Bowden, Athel (1985), « Nomenclature for incompletely specified bases in nucleic acid sequences: rcommendations 1984 », in: Nucleic Acids Res., DOI: 10.1093/nar/13.9.3021 (cited on page 63).

Dalmasso, Marion and Kieran Jordan (2014), « Pulsed-Field Gel Electrophoresis (PFGE) Analysis of Listeria monocytogenes », in: Listeria monocytogenes: Methods and Protocols, New York, NY, USA: Springer, DOI: 10.1007/978-1-4939-0703-8_5 (cited on page 85).

Dasari, Shivraj and Livy Alex (2014), « Microarray based genotyping: a review », in: Journal of Cancer Sciences (cited on page 85).

David, Matei *et al.* (2017), « Nanocall: an open source basecaller for Oxford Nanopore sequencing data », in: Bioinformatics, DOI: 10.1093/bioinformatics/btw569 (cited on page 49).

Deamer, David, Mark Akeson, and Daniel Branton (2016), « Three decades of nanopore sequencing », in: Nature Biotechnology, DOI: 10.1038/nbt.3423 (cited on page 46).

Dehal, Paramvir and Jeffrey L. Boore (2005), « Two rounds of whole genome duplication in the ancestral vertebrate », in: PLoS Biol., DOI: 10.1371/journal.pbio.0030314 (cited on page 27).

Delahaye, Clara and Jacques Nicolas (2021), « Sequencing DNA with nanopores: Troubles and biases », in: PLOS ONE, DOI: 10.1371/journal.pone.0257521 (cited on pages 18, 54).

Denisov, Gennady *et al.* (2008), « Consensus generation and variant detection by Celera Assembler », in: Bioinformatics, DOI: 10.1093/bioinformatics/btn074 (cited on page 73).

Dohm, Juliane C. *et al.* (2020), « Benchmarking of long-read correction methods », in: NAR Genomics Bioinf., DOI: 10.1093/nargab/lqaa037 (cited on pages 15, 33, 53, 57).

Drysdale, Connie M. *et al.* (2000), « Complex promoter and coding region β2-adrenergic receptor haplotypes alter receptor expression and predict *in vivo* responsiveness », in: Proc. Natl. Acad. Sci. U.S.A., DOI: 10.1073/pnas.97.19.10483 (cited on page 30).

Duan, Hongyu *et al.* (2022), « Physical separation of haplotypes in dikaryons allows benchmarking of phasing accuracy in Nanopore and HiFi assemblies with Hi-C data », in: Genome Biol., DOI: 10.1186/s13059-022-02658-2 (cited on page 88).

Eberlein, Chris *et al.* (2017), « The rapid evolution of an ohnolog contributes to the ecological specialization of incipient yeast species », in: Mol. Biol. Evol., DOI: 10.1093/molbev/msx153 (cited on page 29).

Eizenga, Jordan M. *et al.* (2020), « Pangenome Graphs », in: Annu. Rev. Genomics Hum. Genet., DOI: 10.1146/annurev-genom-120219-080406 (cited on page 105).

Erdem, Esra, Ozan Erdem, and Ferhan Türe (2009), « HAPLO-ASP: Haplotype inference using answer set programming », in: Lect. Notes Comput. Sci., DOI: 10.1007/978-3-642-04238-6_60 (cited on pages 99, 110).

Excoffier, L. and M. Slatkin (1995), « Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. », in: Mol. Biol. Evol., DOI: 10.1093/oxfordjournals.molbev.a040269 (cited on page 86).

Fan, H. Christina *et al.* (2011), « Whole-genome molecular haplotyping of single cells », in: Nat. Biotechnol., DOI: 10.1038/nbt.1739 (cited on page 85).

Faure, Roland, Nadège Guiglielmoni, and Jean-François Flot (2021), « GraphUnzip: unzipping assembly graphs with long reads and Hi-C », in: bioRxiv, DOI: https://doi.org/10.1101/2021.01.29.428779 (cited on page 99).

Fu, Shuhua, Anqi Wang, and Kin Fai Au (2019), « A comparative evaluation of hybrid error correction methods for error-prone long reads », in: Genome Biol., DOI: 10.1186/s13059-018-1605-z (cited on page 66).

Garrison, Erik *et al.* (2018), « Variation graph toolkit improves read mapping by representing genetic variation in the reference », in: Nat. Biotechnol., DOI: 10.1038/nbt.4227 (cited on page 105).

Gebser, M. *et al.* (2011), « Potassco: The Potsdam Answer Set Solving Collection », in: undefined, DOI: 10.3233/AIC-2011-0491 (cited on page 109).

Glenn, Travis C. (2011), « Field guide to next-generation DNA sequencers », in: Molecular Ecology Resources, DOI: https://doi.org/10.1111/j.1755-0998.2011.03024.x (cited on page 32).

Goering, Richard V. (2010), « Pulsed field gel electrophoresis: A review of application and interpretation in the molecular epidemiology of infectious disease », in: Infect. Genet. Evol., DOI: 10.1016/j.meegid.2010.07.023 (cited on page 84).

Goldberg, Mark K., Mykola Hayvanovych, and Malik Magdon-Ismail (2010), « Measuring Similarity between Sets of Overlapping Clusters », in: 2010 IEEE Second International Conference on Social Computing, IEEE, DOI: 10.1109/SocialCom.2010.50 (cited on page 128).

Goldstein, Sarah *et al.* (2019), « Evaluation of strategies for the assembly of diverse bacterial genomes using MinION long-read sequencing », in: BMC Genomics, DOI: 10.1186/s12864-018-5381-7 (cited on page 59).

Goodwin, Sara *et al.* (2015), « Oxford Nanopore sequencing, hybrid error correction, and *de novo* assembly of a eukaryotic genome », in: Genome Res., DOI: 10.1101/gr.191395.115 (cited on pages 66, 82).

Gurevich, Alexey *et al.* (2013), « QUAST: quality assessment tool for genome assemblies », in: Bioinformatics, DOI: 10.1093/bioinformatics/btt086 (cited on pages 37, 83).

Hackl, Thomas *et al.* (2014), « proovread: large-scale high-accuracy PacBio correction through iterative short read consensus », in: Bioinformatics, DOI: 10.1093/bioinformatics/btu392 (cited on page 82).

Haghshenas, Ehsan *et al.* (2016), « CoLoRMap: Correcting Long Reads by Mapping short reads », en, in: Bioinformatics, DOI: 10.1093/bioinformatics/btw463 (cited on page 65).

Hu, Jiang *et al.* (2020), « NextPolish: a fast and efficient genome polishing tool for long-read assembly », in: Bioinformatics, DOI: 10.1093/bioinformatics/btz891 (cited on page 81).

Huang, Mengting, Jing Tu, and Zuhong Lu (2017), « Recent advances in experimental whole genome haplotyping methods », in: Int. J. Mol. Sci., DOI: 10.3390/ijms18091944 (cited on page 84).

Hubert, L. and P. Arabie (1985), « Comparing partitions », in: undefined, DOI: 10.1007/BF01908075 (cited on page 127).

International HapMap Consortium (2005), « A haplotype map of the human genome », in: Nature, DOI: 10.1038/nature04226 (cited on page 30).

Jain, Miten *et al.* (2016), « The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community », in: Genome Biology, DOI: 10.1186/s13059-016-1103-0 (cited on page 46).

Joosten, P. H. *et al.* (2001), « Promoter haplotype combinations of the platelet-derived growth factor alpha-receptor gene predispose to human neural tube defects », in: Nat. Genet., DOI: 10.1038/84867 (cited on page 30).

Kerem, B. *et al.* (1989), « Identification of the cystic fibrosis gene: genetic analysis », in: Science, DOI: 10.1126/science.2570460 (cited on pages 14, 30).

Kircher, Martin and Janet Kelso (2010), « High-throughput DNA sequencing – concepts and limitations », in: BioEssays, DOI: https://doi.org/10.1002/bies.200900181 (cited on page 31).

Klau, Gunnar W and Tobias Marschall (2017), « A guided tour to computational haplotyping », in: Lecture notes in computer science, DOI: https://doi.org/10.1007/978-3-319-58741-7_6 (cited on page 84).

Kolmogorov, Mikhail *et al.* (2019), « Assembly of long, error-prone reads using repeat graphs », in: Nat. Biotechnol., DOI: 10.1038/s41587-019-0072-8 (cited on pages 36, 77, 82).

Koren, Sergey *et al.* (2017), « Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation », en, in: Genome Res., DOI: 10.1101/gr.215087.116 (cited on pages 67, 73, 75, 76).

Kronenberg, Zev N. *et al.* (2019), « Extended haplotype phasing of *de novo* genome assemblies with FALCON-Phase », in: bioRxiv, DOI: https://doi.org/10.1038/s41467-020-20536-y (cited on page 99).

Kuderna, Lukas F. K. *et al.* (2020), « Flow sorting enrichment and Nanopore sequencing of chromosome 1 from a Chinese individual », in: Front. Genet., DOI: 10.3389/fgene.2019.01315 (cited on page 85).

Lancia, Giuseppe *et al.* (2001), « SNPs problems, complexity, and algorithms », in: Algorithms — ESA 2001, Berlin, Germany: Springer, DOI: 10.1007/3-540-44676-1_15 (cited on page 93).

Laver, T. *et al.* (2015), « Assessing the performance of the Oxford Nanopore Technologies MinION », in: Biomolecular Detection and Quantification, DOI: 10.1016/j.bdq.2015.02.001 (cited on pages 33, 53).

Lawson, Daniel John *et al.* (2012), « Inference of population structure using dense haplotype data », in: PLos Genet., DOI: 10.1371/journal.pgen.1002453 (cited on pages 14, 31).

Lee, Christopher, Catherine Grasso, and Mark F. Sharlow (2002), « Multiple sequence alignment using partial order graphs », in: Bioinformatics, DOI: 10.1093/bioinformatics/18.3.452 (cited on pages 81, 104, 105).

Lee, Hayan *et al.* (2014), « Error correction and assembly complexity of single molecule sequencing reads », DOI: https://doi.org/10.1101/006395 (cited on page 65).

Leggett, Richard M. *et al.* (2016), « NanoOK: multi-reference alignment analysis of nanopore sequencing data, quality and error profiles », in: Bioinformatics, DOI: 10.1093/bioinformatics/btv540 (cited on page 54).

Leitwein, Maeva *et al.* (2020), « Using haplotype information for conservation genomics », in: Trends Ecol. Evol., DOI: 10.1016/j.tree.2019.10.012 (cited on pages 14, 31).

Li, Heng (2016), « Minimap and miniasm: fast mapping and *de novo* assembly for noisy long sequences », in: Bioinformatics, DOI: 10.1093/bioinformatics/btw152 (cited on pages 74, 82).

— (2018), « Minimap2: pairwise alignment for nucleotide sequences », in: Bioinformatics, ed. by Inanc Birol, DOI: 10.1093/bioinformatics/bty191 (cited on pages 55, 66, 75, 119).

Li, Na and Matthew Stephens (2003), « Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. », in: Genetics, DOI: 10.1093/genetics/165.4.2213 (cited on page 86).

Li, Zhenyu *et al.* (2012), « Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph », in: Brief. Funct. Genomics, DOI: 10.1093/bfgp/elr035 (cited on page 36).

Lippert, R. (2002), « Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem », in: Briefings in Bioinformatics, DOI: 10.1093/bib/3.1.23 (cited on pages 91, 92).

Loman, Nicholas J., Joshua Quick, and Jared T. Simpson (2015), « A complete bacterial genome assembled *de novo* using only nanopore sequencing data », in: Nat. Methods, DOI: 10.1038/nmeth.3444 (cited on page 81).

Luo, R. *et al.* (2012), « SOAPdenovo2: an empirically improved memory-efficient short-read *de novo* assembler », in: GigaScience (cited on page 35).

Lynce, Inês and João Marques-silva (2006), « Efficient haplotype inference with boolean satisfiability », in: Proceedings of the National Conference on Artificial Intelligence (cited on pages 99, 110).

Ma, Bin, John Tromp, and Ming Li (2002), « PatternHunter: faster and more sensitive homology search », in: Bioinformatics, DOI: 10.1093/bioinformatics/18.3.440 (cited on page 74).

Majidian, Sina, Mohammad Hossein Kahaei, and Dick de Ridder (2020), « Minimum error correction-based haplotype assembly: Considerations for long read data », in: PLOS ONE, ed. by Tiratha Raj Singh, DOI: 10.1371/journal.pone.0234470 (cited on pages 93, 99).

Mak, Denise, Yevgeniy Gelfand, and Gary Benson (2006), « Indel seeds for homology search », in: Bioinformatics, DOI: 10.1093/bioinformatics/btl263 (cited on page 74).

Mapleson, Daniel *et al.* (2017), « KAT: a k-mer analysis toolkit to quality control NGS datasets and genome assemblies », in: Bioinformatics, DOI: 10.1093/bioinformatics/btw663 (cited on pages 38, 83).

Marçais, Guillaume and Carl Kingsford (2011), « A fast, lock-free approach for efficient parallel counting of occurrences of k-mers », in: Bioinformatics, DOI: 10.1093/bioinformatics/btr011 (cited on page 37).

Marchet, Camille *et al.* (2020), « ELECTOR: evaluator for long reads correction methods », en, in: NAR Genom Bioinform, DOI: https://doi.org/10.1093/nargab/lqz015 (cited on page 68).

Miclotte, Giles *et al.* (2016), « Jabba: hybrid error correction for long sequencing reads », in: Algorithms for Molecular Biology : AMB, DOI: 10.1186/s13015-016-0075-7 (cited on page 82).

Miyamoto, Mari *et al.* (2014), « Performance comparison of second- and third-generation sequencers using a bacterial genome with two chromosomes », en, in: BMC Genomics, DOI: https://doi.org/10.1186/1471-2164-15-699 (cited on page 67).

Monajembashi, S. *et al.* (1986), « Microdissection of human chromosomes by a laser microbeam », in: Exp. Cell Res., DOI: 10.1016/0014-4827(86)90223-5 (cited on page 85).

Morisse, Pierre, Thierry Lecroq, and Arnaud Lefebvre (2020), « Long-read error correction: a survey and qualitative comparison », DOI: 10.1101/2020.03.06.977975 (cited on page 65).

Morisse, Pierre, Camille Marchet, *et al.* (2021), « Scalable long read self-correction and assembly polishing with multiple sequence alignment », in: Sci. Rep., DOI: 10.1038/s41598-020-80757-5 (cited on page 81).

Myers, Eugene W. *et al.* (2000), « A whole-genome assembly of drosophila », in: Science, DOI: 10.1126/science.287.5461.2196 (cited on page 75).

Nomenclature, Iupac-Iub Comm. On Biochem. (1970), « IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for the description of the conformation of polypeptide chains. Tentative rules (1969) », in: Biochemistry, DOI: 10.1021/bi00820a001 (cited on page 63).

Ovcharenko, Ivan *et al.* (2005), « Mulan: multiple-sequence local alignment and visualization for studying function and evolution », in: Genome Res., DOI: 10.1101/gr.3007205 (cited on page 104).

Paten, Benedict *et al.* (2017), « Genome graphs and the evolution of genome inference », in: Genome Res., DOI: 10.1101/gr.214155.116 (cited on page 105).

Patterson, Murray *et al.* (2015), « WhatsHap: weighted haplotype assembly for future-generation sequencing reads », in: Journal of Computational Biology, DOI: 10.1089/cmb.2014.0157 (cited on page 92).

Payne, Alexander *et al.* (2018), « BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files », in: Bioinformatics, DOI: 10.1093/bioinformatics/bty841 (cited on page 49).

Pevzner, Pavel A., Haixu Tang, and Michael S. Waterman (2001), « An Eulerian path approach to DNA fragment assembly », in: Proc. Natl. Acad. Sci. U.S.A., DOI: 10.1073/pnas.171285098 (cited on page 76).

Pruitt, Kim D. *et al.* (2012), « NCBI Reference Sequences (RefSeq): current status, new features and genome annotation policy », in: Nucleic Acids Res., DOI: 10.1093/nar/gkr1079 (cited on page 55).

Qiu, Yutong and Carl Kingsford (2021), « Constructing small genome graphs via string compression », in: Bioinformatics, DOI: 10.1093/bioinformatics/btab281 (cited on page 105).

Ramsey, Justin and Douglas W. Schemske (1998), « Pathways, mechanisms, and rates of polyploid formation in flowering plants », in: Annu. Rev. Ecol. Syst., DOI: 10.1146/annurev.ecolsys.29.1.467 (cited on page 28).

Rand, William M. (1971), « Objective Criteria for the Evaluation of Clustering Methods », in: J. Am. Stat. Assoc., DOI: 10.1080/01621459.1971.10482356 (cited on page 127).

Rang, Franka J., Wigard P. Kloosterman, and Jeroen de Ridder (2018), « From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy », in: Genome Biology, DOI: 10.1186/s13059-018-1462-9 (cited on page 60).

Rennie, B. C. and A. J. Dobson (1969), « On stirling numbers of the second kind », in: Journal of Combinatorial Theory, DOI: 10.1016/S0021-9800(69)80045-1 (cited on pages 17, 42).

Rhie, Arang *et al.* (2020), « Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies », in: Genome Biol., DOI: 10.1186/s13059-020-02134-9 (cited on page 83).

Ross, Michael G. *et al.* (2013), « Characterizing and measuring bias in sequence data », in: Genome Biol., DOI: 10.1186/gb-2013-14-5-r51 (cited on page 59).

Ruan, Jue and Heng Li (2020), « Fast and accurate long-read assembly with wtdbg2 », in: Nat. Methods, DOI: 10.1038/s41592-019-0669-3 (cited on pages 36, 77, 82).

Saada, Omar Abou *et al.* (2021), « nPhase: an accurate and contiguous phasing method for polyploids », in: Genome Biol., DOI: 10.1186/s13059-021-02342-x (cited on pages 97, 151).

Salmela, Leena and Eric Rivals (2014), « LoRDEC: accurate and efficient long read error correction », en, in: Bioinformatics, DOI: http://doi.org/10.1093/bioinformatics/btu538 (cited on pages 66, 82).

Salmela, Leena, Riku Walve, *et al.* (2017), « Accurate self-correction of errors in long reads using de Bruijn graphs », en, in: Bioinformatics, DOI: 10.1093/bioinformatics/btw321 (cited on page 67).

Sanderson, Nicholas *et al.* (2022), « Comparison of R9.4.1/Kit10 and R10/Kit12 Oxford Nanopore flowcells and chemistries in bacterial genome reconstruction », in: DOI: 10.1101/2022.04.29.490057 (cited on pages 51, 53).

Sanger, F. and A. R. Coulson (1975), « A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase », in: J. Mol. Biol., DOI: 10.1016/0022-2836(75)90213-2 (cited on page 31).

Sati, Satish and Giacomo Cavalli (2017), « Chromosome conformation capture technologies and their impact in understanding genome function », in: Chromosoma, DOI: 10.1007/s00412-016-0593-6 (cited on page 88).

Sayers, Eric W. *et al.* (2022), « Database resources of the national center for biotechnology information », in: Nucleic Acids Res., DOI: 10.1093/nar/gkab1112 (cited on page 31).

Scalenghe, F. *et al.* (1981), « Microdissection and cloning of DNA from a specific region of Drosophila melanogaster polytene chromosomes », in: Chromosoma, DOI: 10.1007/BF00286105 (cited on page 85).

Schrinner, Sven D. *et al.* (2020), « Haplotype threading: accurate polyploid phasing from long reads », in: Genome Biology, DOI: 10.1186/s13059-020-02158-1 (cited on pages 97, 151).

Schwartz, D. C. and C. R. Cantor (1984), « Separation of yeast chromosome-sized DNAs by pulsed field gradient gel electrophoresis », in: Cell, DOI: 10.1016/0092-8674(84)90301-5 (cited on page 84).

Sears, E. R. (1976), « Genetic control of chromosome pairing in wheat », in: Annu. Rev. Genet., DOI: 10.1146/annurev.ge.10.120176.000335 (cited on page 29).

Sereika, Mantas *et al.* (2022), « Oxford Nanopore R10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing », in: Nat. Methods, DOI: 10.1038/s41592-022-01539-7 (cited on page 53).

Servant, Nicolas *et al.* (2015), « HiC-Pro: an optimized and flexible pipeline for Hi-C data processing », in: Genome Biol., DOI: 10.1186/s13059-015-0831-x (cited on page 88).

Sevim, Volkan *et al.* (2019), « Shotgun metagenome data of a defined mock community using Oxford Nanopore, PacBio and Illumina technologies », in: Scientific Data, DOI: 10.1038/s41597-019-0287-z (cited on page 59).

Shafin, Kishwar *et al.* (2020), « Nanopore sequencing and the Shasta toolkit enable efficient *de novo* assembly of eleven human genomes », in: Nature Biotechnology, DOI: 10.1038/s41587-020-0503-6 (cited on pages 54, 55, 75, 82).

Shaw, Jim and Yun William Yu (2022), « flopp: extremely fast long-read polyploid haplotype phasing by uniform tree partitioning », in: J. Comput. Biol., DOI: 10.1089/cmb.2021.0436 (cited on page 95).

Simão, Felipe A. *et al.* (2015), « BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs », in: Bioinformatics, DOI: 10.1093/bioinformatics/btv351 (cited on pages 38, 83).

Simion, Paul *et al.* (2021), « Chromosome-level genome assembly reveals homologous chromosomes and recombination in asexual rotifer *Adineta vaga* », in: Sci. Adv., DOI: 10.1126/sciadv.abg4216 (cited on pages 134–136).

Simonis, Marieke, Jurgen Kooren, and Wouter de Laat (2007), « An evaluation of 3C-based methods to capture DNA interactions », in: Nat. Methods, DOI: 10.1038/nmeth1114 (cited on page 88).

Song, Can *et al.* (2012), « Polyploid organisms », in: Sci. China Life Sci., DOI: 10.1007/s11427-012-4310-2 (cited on page 89).

Sović, Ivan *et al.* (2016), « Fast and sensitive mapping of nanopore sequencing reads with GraphMap », in: Nat. Commun., DOI: 10.1038/ncomms11307 (cited on page 74).

Squitieri, F. *et al.* (1994), « DNA haplotype analysis of Huntington disease reveals clues to the origins and mechanisms of CAG expansion and reasons for geographic variations of prevalence », in: Hum. Mol. Genet., DOI: 10.1093/hmg/3.12.2103 (cited on pages 14, 30).

Stephens, Matthew, Nicholas J. Smith, and Peter Donnelly (2001), « A new statistical method for haplotype reconstruction from population data », in: Am. J. Hum. Genet., DOI: 10.1086/319501 (cited on page 86).

Tang, Xian *et al.* (2020), « One pdge at a time: a novel approach towards efficient transitive reduction computation on DAGs », in: IEEE Access, DOI: 10.1109/ACCESS.2020.2975650 (cited on page 79).

Teng, Haotian *et al.* (2018), « Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning », in: GigaScience, DOI: 10.1093/gigascience/giy037 (cited on page 49).

The International HapMap Consortium (2007), « A second generation human haplotype map of over 3.1 million SNPs », in: Nature, DOI: 10.1038/nature06258 (cited on page 30).

Thomason, Andrew (1989), « A simple linear expected time algorithm for finding a hamilton path », in: Discrete Math., DOI: 10.1016/0012-365X(89)90100-3 (cited on page 35).

Tischler, German and Eugene W Myers (2017), « Non hybrid long read consensus using local DE Bruijn graph assembly », DOI: https://doi.org/10.1101/106252 (cited on page 67).

Vaser, Robert and Mile Šikić (2021), « Time- and memory-efficient genome assembly with Raven », in: Nat. Comput. Sci., DOI: 10.1038/s43588-021-00073-4 (cited on pages 70, 74, 82).

Vaser, Robert, Ivan Sovic, *et al.* (2017), « Fast and accurate *de novo* genome assembly from long uncorrected reads », in: Genome Res., DOI: 10.1101/gr.214270.116 (cited on page 81).

Vereecke, Nick *et al.* (2020), « High quality genome assemblies of Mycoplasma bovis using a taxon-specific Bonito basecaller for MinION and Flongle long-read nanopore sequencing », in: BMC Bioinf., DOI: 10.1186/s12859-020-03856-0 (cited on page 64).

Wall, Jeffrey D. and Jonathan K. Pritchard (2003), « Haplotype blocks and linkage disequilibrium in the human genome », in: Nat. Rev. Genet., DOI: 10.1038/nrg1123 (cited on page 31).

Wang, Jeremy R. *et al.* (2018), « FMLRC: Hybrid long read error correction using an FM-index », in: BMC Bioinf., DOI: 10.1186/s12859-018-2051-3 (cited on pages 66, 82).

Wang, Luotong *et al.* (2020), « NanoReviser: an error-correction tool for nanopore sequencing based on a deep learning algorithm », in: Front. Genet., DOI: 10.3389/fgene.2020.00900 (cited on page 67).

Warby, Simon C. *et al.* (2011), « HTT haplotypes contribute to differences in Huntington disease prevalence between Europe and East Asia », in: Eur. J. Hum. Genet., DOI: 10.1038/ejhg.2010.229 (cited on pages 14, 30).

Wenger, Aaron M. *et al.* (2019), « Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome », in: Nature Biotechnology, DOI: 10.1038/s41587-019-0217-9 (cited on page 53).

Wick, Ryan R. and Kathryn E. Holt (2021), « Benchmarking of long-read assemblers for prokaryote whole genome sequencing », in: F1000Research, DOI: 10.12688/f1000research.21782.4 (cited on page 70).

Wick, Ryan R., Louise M. Judd, and Kathryn E. Holt (2019), « Performance of neural network basecalling tools for Oxford Nanopore sequencing », in: Genome Biology, DOI: 10.1186/s13059-019-1727-y (cited on pages 54, 55, 60).

Xiao, Chuan-Le *et al.* (2017), « MECAT: fast mapping, error correction, and *de novo* assembly for single-molecule sequencing reads », en, in: Nat. Methods, DOI: https://doi.org/10.1038/nmeth.4432 (cited on page 67).

Xie, Minzhu *et al.* (2016), « H-PoP and H-PoPG: heuristic partitioning algorithms for single individual haplotyping of polyploids », in: Bioinformatics, DOI: 10.1093/bioinformatics/btw537 (cited on pages 91, 95).

Xu, Xin Shun and Ying Xin Li (2012), « Semi-supervised clustering algorithm for haplotype assembly problem based on MEC model », in:

International Journal of Data Mining and Bioinformatics, DOI: 10.1504/ijdmb.2012.049279 (cited on pages 90, 93).

Yang, Jun *et al.* (2017), « Haplotype-resolved sweet potato genome traces back its hexaploidization history », in: Nature Plants, DOI: 10.1038/s41477-017-0002-z (cited on pages 14, 31).

Zerbino, Daniel R. and Ewan Birney (2008), « Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs », in: Genome Res., DOI: 10.1101/gr.074492.107 (cited on page 35).

Zhang, Haowen, Chirag Jain, and Srinivas Aluru (2020), « A comprehensive evaluation of long read error correction methods », en, in: BMC Genomics, DOI: 10.1186/s12864-020-07227-0 (cited on page 65).

UNIVERSITÉ DE RENNES 1

Titre : Phasage d'haplotypes par ASP à partir de longues lectures : une approche d'optimisation flexible

Mot clés : Bioinformatique, phasage d'haplotype, polyploïde, séquençage longues lectures, clustering, optimisation, Answer Set Programming

Résumé : Chaque chromosome d'organisme di- ou polyploïde présente plusieurs haplotypes, qui sont fortement similaires mais divergent sur un certain nombre de positions. Cependant, la majorité des génomes de référence ne renseignent qu'une seule séquence pour chaque chromosome, et ne reflètent donc pas la réalité biologique. Or, il est crucial d'avoir accès à ces informations, qui sont utiles en médecine, en agronomie ou encore dans l'étude des populations. Le récent développement des technologies de troisième génération, notamment des séquenceurs PacBio et Oxford Nanopore Technologies, a permis la production de lectures longues facilitant la reconstruction des séquences d'haplotypes. Il existe pour cela des méthodes bioinformatiques, mais elles ne fournissent qu'une unique solution. Cette thèse propose une méthode de phasage d'haplotype basée sur la recherche de composantes connexes dans un graph de similarité des lectures pour identifier les haplotypes. Cette méthode utilise l'*Answer Set Programming* pour travailler sur l'ensemble des solutions optimales. L'algorithme de phasage a permis de reconstruire les haplotypes du rotifère diploïde *Adineta vaga*.

Title: Haplotype phasing from long reads with ASP: a flexible optimization approach

Keywords: Bioinformatics, haplotype phasing, polyploid, long read sequencing, clustering, optimization, Answer set programming

Abstract: Each chromosome of a di- or polyploid organism has several haplotypes, which are highly similar but diverge on a certain number of positions. However, most of the reference genomes only provide a single sequence for each chromosome, and therefore do not reflect the biological reality. Yet, it is crucial to have access to this information, which is useful in medicine, agronomy and population studies. The recent development of third generation technologies, especially PacBio and Oxford Nanopore Technologies sequencers, has allowed for the production of long reads that facilitate haplotype sequence reconstruction. Bioinformatics methods exist for this task, but they provide only a single solution. This thesis introduces an approach for haplotype phasing based on the search of connected components in a read similarity graph to identify haplotypes. This method uses *Answer Set Programming* to work on the set of optimal solutions. This phasing algorithm has been used to reconstruct haplotypes of the diploid rotifer *Adineta vaga*.