# MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

## Vers une reconstruction de haut niveau des pipelines biologiques

NINDJEU NANGMO, Guyssel; SNICKERS, Florent

*Award date:*
2022

*Awarding institution:*
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

# Vers une reconstruction de haut niveau des pipelines biologiques

Florent Snickers
Guyssel Nindjeu

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

# Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de notre stage et qui nous ont aidés lors de la rédaction de ce mémoire.

Nous voudrions dans un premier temps remercier, notre promoteur de mémoire M. Jacquet, professeur à l'université de Namur, pour avoir proposé ce sujet, pour sa patience, sa disponibilité et surtout ses conseils judicieux, qui ont contribué à alimenter notre réflexion.

Nous souhaitons également remercier M. Zeippen, qui dans son rôle de co-promoteur, nous a conseillés, guidés lors de notre stage et de notre rédaction.

Un tout grand merci à Gaëtan, Édith, Jonathan et tout le reste de l'équipe du CHU pour leur chaleureux accueil tout au long de notre stage, pour leur esprit d'équipe, leur bonne humeur et leur climat positif.

Nous remercions également toute l'équipe pédagogique de l'université de Namur et les intervenants professionnels responsables de notre formation, pour nous avoir donné les connaissances et compétences nécessaires à aborder ce mémoire.

Nous tenons à témoigner toute notre reconnaissance aux personnes suivantes, pour leur aide dans la réalisation de ce mémoire :

Mademoiselle Rose Ogouliguende, qui nous a guidés dans la rédaction de la partie biologie.

Madame Paulette Vincx, pour avoir relu et corrigé notre mémoire.

Monsieur Frédéric Snickers, pour avoir relu et corrigé notre mémoire.

Un tout grand merci à nos familles qui nous ont soutenus et à tous ceux qui ont participé de près ou de loin à ce travail.

# Résumé

Le typage érythrocytaire consiste à identifier l'ensemble des antigènes des globules rouges d'un individu. Il représente un enjeu majeur de compatibilité lors des transfusions sanguines. Aujourd'hui, ce sont des méthodes antigéniques qui sont utilisées. Une nouvelle approche consiste à utiliser une méthode par séquençage ADN. Elle permet d'obtenir des informations plus précises et plus personnalisées pour identifier les antigènes des globules rouges. L'objectif de ce mémoire est d'adapter un pipeline existant d'analyse de données issus d'un séquençage ADN pour le typage erythrocytaire. La question est par conséquent : *"Comment adapter un pipeline de séquençage pour identifier le profil érythrocytaire d'individus ?"*. Celle-ci a été explorée à travers trois étapes différentes. La première est le développement d'un outil de conception d'amorces qui augmente la qualité des échantillons de séquençage grâce à un pré-traitement. Ensuite, il s'agit d'une modification de l'étape d'alignement suivi d'un ajout d'une étape d'analyse propre au typage érythrocytaire. Ces deux dernières étapes ont directement été intégrées dans le pipeline commencé par l'équipe du Laboratoire de Biologie Clinique du CHU UCL Namur. Dans un but d'optimisation de l'exécution et pour faciliter la reproductibilité des analyses, celui-ci s'est vu automatisé grâce à des outils de gestion de flux de travail et de conteneurisation. L'outil de conception d'amorces a été utilisé avec succès par les laborantins du CHU et peut être utilisé pour la construction d'amorces pour d'autres analyses par séquençage.

Mots clés : pipeline - séquençage - typage érythrocytaire - analyse - détection de variants

# Table des matières

# Introduction

Ce travail a été réalisé dans le cadre du projet *"ROBIN : Hight level reconstruction of biological pipelines"*, une collaboration entre le CHU UCL Namur et l'université de Namur.

La biologie moléculaire a évolué considérablement ces 50 dernières années, notamment grâce aux apports des sciences informatiques, mathématiques, ou encore statistiques. L'aboutissement de cette évolution a été mis en avant, par exemple lors de la pandémie de la Covid-19. Le virus SARS-COV 2, responsable de la COVID-19 et qui est un des plus gros virus à ARN, a nécessité une analyse rapide et précise de son génome entier pour mettre au point des vaccins. Cette analyse a été effectuée en séquençant son génome (qui mesure près de 30 000 bases azotés, réparties sur 15 gènes). À une échelle plus grande encore, le génome humain a une taille d'environ 3 milliards de bases répartis sur 30 000 gènes. Le traitement de telles quantités de données avec un pourcentage de précision et de temps raisonnable se fait aux travers de nombreux et divers outils informatiques conçus sur base d'algorithmes performants. La coordination de ces outils lors d'une analyse se fait généralement dans un pipeline. L'analyse bio-informatique basée principalement sur le séquençage, requiert aujourd'hui une forte puissance de calcul avec des algorithmes et des outils disponibles publiquement ou dans le commerce.

Ces outils, ainsi que les nouvelles techniques d'analyses utilisées, ou encore les programmes hautement performants vont permettre le développement de plusieurs autres projets de génétique moléculaire comme celui du *typage érythrocytaire* abordé dans ce travail. Le terme typage est le fait d'attribuer un type, tandis que le terme érythrocytaire précise la spécificité au niveau des globules rouges. Le typage érythrocytaire consiste donc à identifier l'ensemble des antigènes, génétiquement induits et déterminés, des globules rouges d'un individu. L'utilité principale de ce typage est d'éviter les problèmes de transfusion sanguine, essentielle à la survie de tous les patients devant subir une opération.

Aujourd'hui, les solutions existantes dans le laboratoire de biologie clinique du CHU UCL Namur pour ce typage érythrocytaire se font par sondes ADN, ce qui nécessite un long laps de temps, ou par méthodes antigéniques. La nouvelle réponse sera la méthode par *séquençage* en utilisant les technologies de nouvelle génération (NGS). Celle-ci permet l'analyse rapide d'un grand nombre de gènes chez une population de patients, dans

un délai relativement court, tout en facilitant l'analyse et améliorant l'interprétation clinique des variations génomiques.

L'objectif de ce travail est de mettre en place un pipeline (un ensemble d'outils mis bout à bout) d'analyse de données issues d'un séquençage par méthode Nanopore afin de déterminer le typage érythrocytaire d'un patient. La méthode utilisée sera découpée en trois phases de traitement. La première phase est la construction d'une librairie de données : ceci passe par la mise en place d'un outil efficace pour la conception des amorces (qui délimitent la portion d'ADN à amplifier lors du séquençage). La deuxième phase est la mise au point d'un aligneur permettant de générer une séquence d'ADN *consensus* (la séquence la plus fréquente à chaque position d'un alignement) à partir des résultats obtenus lors de l'étape séquençage proprement dite. La détection des variants, une étape clé du traitement données qui permet de décrire le variant selon plusieurs critères, intervient lors de cette phase. La troisème phase est l'identification du type érythrocytaire (carte d'identité du patient) à partir d'une base de données reprenant les différentes mutations.

Le manuscrit s'organise en deux parties. La première partie présente les bases nécessaires pour comprendre ce mémoire. Il comporte un premier chapitre sur la biologie moléculaire clinique et l'informatique, et un deuxième sur les algorithmes de bases utilisés en bio-informatique (avec le problème de correspondance, d'alignement global, alignement local et d'assemblage).

La deuxième partie présente nos contributions. Elle est organisée en deux chapitres. Le premier chapitre détaille la mise au point d'un outil de conception d'amorces à travers une rétro-ingénierie de deux outils existants (primalscheme et primer3), la mise en place d'un outil web pour la paramétrisation dynamique ainsi qu'une évaluation des amorces construites (dimers et hairpins). Le deuxième chapitre détaille d'abord l'automatisation et l'amélioration d'un pipeline existant au CHU pour le traitement et l'analyse des données de séquençage. Il contient les étapes nécessaires pour aligner les lectures sur une référence, le contrôle de qualité des séquences alignées, l'identification et la filtration des variants de manière fiable. Ensuite, une amélioration de l'étape d'alignement du pipeline a été réalisée grâce à un ajout d'informations dans le fichier vcf et à la génération d'une meilleure séquence consensus. Enfin, l'identification du typage érythrocytaire d'un patient a été effectuée sur base des mutations détectées lors du point précédent.

# Première partie

# État de l’art

# Chapitre 1

# Biologie moléculaire clinique et pipelines bio-informatiques

Chaque individu est unique du point de vue phénotypique à l'exception des vrais jumeaux. Mais qu'en est-il du point de vue génétique ? Tous les individus qui composent l'espèce humaine sont tous égaux génétiquement, mais ce qui marque la différence entre deux individus est l'expression des gènes. Celui-ci est lié à un code génétique, plus souvent connue sous le nom *ADN*. Dans ce code génétique humain, il peut apparaître des mutations (fruit du hasard par rapport à l'adaptation ou modification accidentelle) [58] qui sont très ponctuelles et qui représentent une très petite taille par rapport à celle du génome humain. Certaines mutations peuvent induire des maladies. Le fait qu'aucun individu n'exprime les mêmes gènes, excepté les vrais jumeaux, constitue un enjeu majeur pour adapter les traitements en fonction des dispositions génétiques de chacun. C'est ce qu'on appelle la *médecine de précision*. Elle permet principalement de détecter les pathologies ou de proposer des traitements adaptés et personnalisés. Ce chapitre couvrera toutes les notions de bases de la biologie pour bien comprendre la bio-informatique et ses enjeux.

## 1.1 ADN et ARN

### 1.1.1 Généralités

L'acide désoxyribonucléique (ADN) est le code génétique d'un individu. C'est la molécule qui constitue le support de l'information génétique héréditaire [4] .

*L'acide ribonucléique* ou *ARN* est un acide nucléique présent chez, pratiquement, tous les êtres vivants. L'ARN est très proche chimiquement de l'ADN et est en général synthétisé à partir d'un segment d'ADN matrice dont il est la transcription. Les cellules utilisent en particulier l'ARN comme support intermédiaire des gènes pour synthétiser les protéines dont elles ont besoin [3] .

### 1.1.2 Structures

L'ADN est une molécule très longue, composée d'une succession de nucléotides (voir Figure 1.1.1) attachés les uns aux autres par des liaisons phosphodiester [4]. Un nucléotide se compose d'un groupement phosphate, d'un sucre (dans ce cas un déoxyribose), et d'une base nucléotidique. Le code génétique est constitué de quatre bases nucléotidiques différentes :

- Adénine - A
- Cytosine - C
- Guanine - G
- Thymine - T

Chimiquement, l'ARN (voir Figure 1.1.1) est un polymère linéaire constitué d'un enchaînement de nucléotides [3]. On trouve quatre bases nucléiques dans l'ARN :

- Adénine - A
- Guanine - G
- Cytosine - C
- Uracile - U

La figure 1.1.1 illustre une représentation de l'ADN et l'ARN ainsi que les différentes bases qui les constituent. L'ADN est composé de deux hélices ou deux brins complémentaires de nucléotides. Ces deux chaînes sont complémentaires car il existe une complémentarité chimique qui engendre que la guanine fait toujours face à la cytosine et de la même manière, l'adénine fait toujours face à la thymine. Cela est permis grâce aux liaisons moléculaires appelées "liaisons hydrogènes" (3 pour le couple G-C et 2 pour le couple A-T). Selon les espèces, sa structure varie. Les êtres vivants peuvent être classifiés en *Procaryotes (bactéries)* qui ont une structure d'ADN circulaire et en *Eucaryotes* qui en ont une en fragments (sous forme de chromosomes).

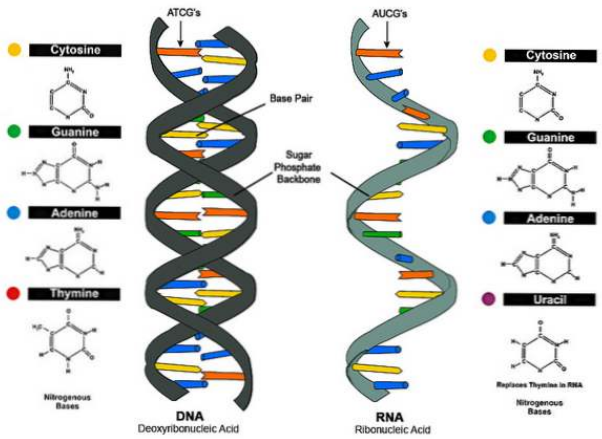



FIGURE 1.1.1 – Représentation d'une molécule d'ADN double brin (à gauche) et d'une molécule d'ADN simple brin (à droite) ainsi que de leurs bases nucléotidiques associées [20].

### 1.1.3 Rôle

L'ADN est la molécule qui porte le code génétique d'un individu, c'est-à-dire toutes les informations nécessaires à la fabrication et au développement d'un être vivant. L'ADN permet la production des protéines. L'élément responsable de la fabrication des protéines est le gène. Un gène comprend la séquence en nucléotides qui sera transcrite puis traduite en acides aminés, mais aussi des séquences permettant de réguler cette fabrication de protéine en fonction des conditions cellulaires. La longueur d'un gène peut varier de quelques centaines, à plus d'un million de nucléotides [66].

La figure 1.1.2 illustre la représentation de trois gènes sur un chromosome avec les phénomènes de transcription et de traduction soit d'ARNm (ARN messager), d'ARNt (ARN de transfert) ou d'un ARNr (ARN ribosomial). Le but est de montrer comment un gène fabrique une protéine.

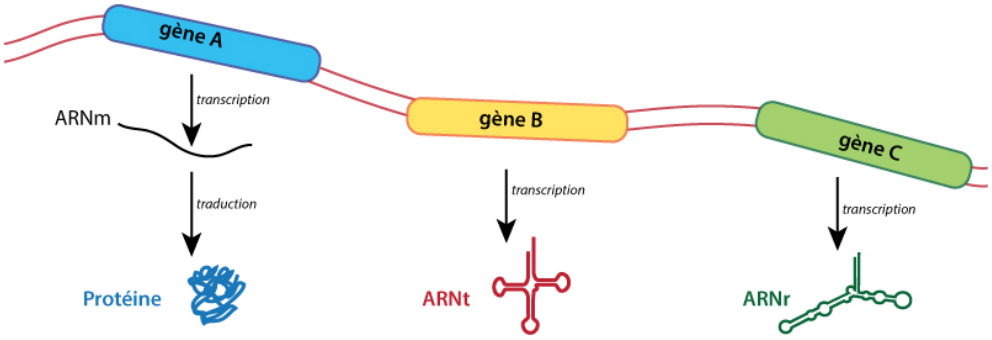



FIGURE 1.1.2 – Représentation d'un gène [50] : l'image représente un chromosome avec plusieurs gènes (qui sont une partie de cette information à un endroit précis). Le gène A est un exemple de transcription en ARNm (ARN messager : qui est comme une copie, une molécule chargée de transmettre l'information codée dans le génome [67]) suivi d'une traduction en protéine qui va être une unité fonctionnelle. Dans le cas du gène B, c'est simplement une transcription en ARNt (ARN de transfert : qui est une courte molécule d'ARN, qui va associer à chaque codon un acide aminé parmi 20 possibles [39]). Dans le cas du gène C, c'est une transcription ARNr (ARN ribosomiale : constituant principal des ribosomes qui se charge d'assurer l'interaction de l'ARNm avec l'ARNt [8]) .

### 1.1.4 Comparaison

| **Éléments différents** | **ADN** | **ARN** |
|---|---|---|
| Bases azotées | ATCG | AUCG |
| Sucre | Deoxyribose | Ribose |
| Nombre de brins | Double brin | Simple brin |
| Taille | Plus long | Plus court |
| Localisation intra-cellulaire | Noyau et Mitochondrie | Cytoplasme |
| Durée de vie | Longue | Courte |

TABLE 1.1 – Comparaison entre l'ADN et l'ARN [21]

## 1.2 Génome

Le génome est l'ensemble des informations reprises dans le matériel génétique d'un individu (Figure 1.1.2). Il contient des parties de séquences codantes (transcrites en ARN messagers, et traduites en protéines) et non-codantes (non transcrites, ou transcrites en ARN, mais non traduites) [87].

Le génome peut être comparé à une bibliothèque qui contient des étagères (les chromosomes). Sur celles-ci se trouvent des livres (les gènes). Dans ces livres il y a de l'information, des phrases écrites, dans un langage génétique représenté par quatre bases (adénine, guanine, cytosine et thymine) abrégées en AGCT [87]. Par suite, la science qui étudie le génome est appelé la génomique.

Chez les virus, le génome est contenu dans une (ou plusieurs) molécule(s) d'ADN ou d'ARN, à simple ou double brin. Chez les procaryotes (bactéries et archées), le génome est généralement contenu dans une molécule d'ADN circulaire.

Dans le cas de l'être humain, constitué de 22 paires de chromosomes + 1 paire, la $23^{eme}$ est la paire de chromosomes sexuels (XX chez la femme, XY chez l'homme). À noter qu'il y a une grande différence entre le génome (qui est l'ensemble de l'information génétique portée par l'ADN sur les 23 paires de chromosomes présentes dans le noyau plus l'ADN mitochondrial (hérité de la mère uniquement)) et le caryotype (qui est une représentation structurale, sous forme de photographie, de l'ensemble des chromosomes d'une cellule, classés par paire et selon la taille qui permet de dépister d'éventuelles anomalies chromosomiques) [27].

## 1.3 PCR

La PCR (Polymérase Chain Reaction ou réaction de polymérisation en chaîne) est une technique d'amplification d'ADN ou ARN in vitro décrite en 1985 (K. MULLIS et collaborateurs) [28]. Elle permet d'obtenir un très grand nombre de copies d'une séquence d'ADN choisie [65]. Avant de pouvoir effectuer une PCR il va falloir récolter le matériel génétique nécessaire, ce qui s'effectue par une méthode d'extraction d'ADN ou ARN.

#### Extraction d'ADN ou ARN

L'extraction d'ADN est un processus de libération de l'ADN chromosomique de la matrice cellulaire dans laquelle il est contenu. Elle nécessite une méthode de rupture robuste pour ouvrir les noyaux et les parois cellulaires, et implique le plus souvent l'ajout d'un détergent compatible ainsi qu'un cisaillement mécanique. L'échantillon d'ADN est obtenu après l'isolement dans lequel les protéines, les membranes cellulaires et d'autres matériaux cellulaires sont éliminés [29].

Une fois le matériel génétique prêt, l'étape de PCR peut débuter. L'image de la figure 1.3.1 met en évidence le principe de la PCR. Chaque cycle est constitué de trois étapes :

— une dénaturation de l'ADN : elle s'effectue à température élevée (95°C) et permet de séparer les deux brins qui composent l'ADN,

— ł'hybridation ou renaturation : permet l'appariement des deux amorces à leur région complémentaire sur les deux brins séparés (45 à 60°C)

— l'élongation : grâce à l'action d'une *ADN polymérase*

Ce cycle est répété un grand nombre de fois pour obtenir une multiplication exponentielle de la séquence d'ADN cible. Plus de détails se trouvent dans la section 3.1.2.

La PCR permet de détecter de très faibles quantités de génome exogène dans des prélèvements biologiques variés (sang, fluide biologique, cellules,...). Une autre technologie est en pleine expansion, le séquençage à haut débit qui permet d'obtenir encore plus d'informations.

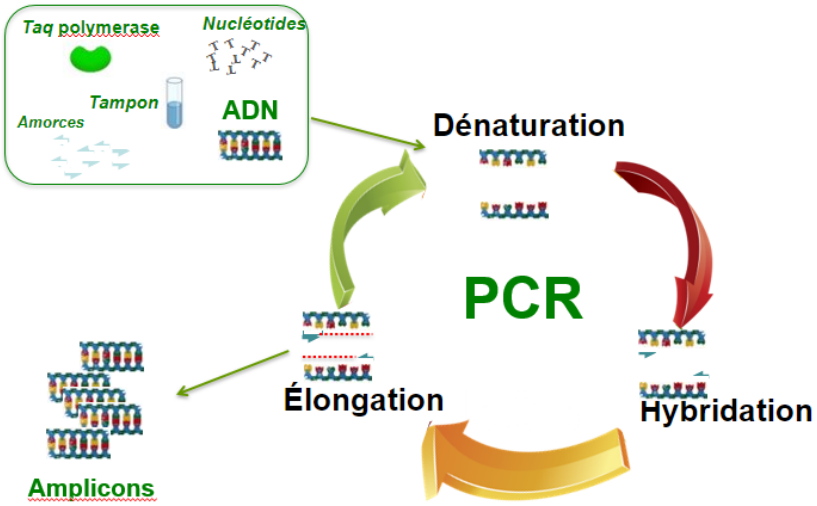



FIGURE 1.3.1 – Principe de la PCR : effectue un cycle passant par trois étapes principales. Chacun débute par l'étape de dénaturation lorsque les 2 brins de l'ADN se séparent par rupture des liaisons hydrogènes, pour obtenir des matrices simples brin. Ensuite dans l'étape d'hybridation, à 50°C-65°C, les amorces individuelles viennent s'accrocher sur ceux-ci. La dernière phase est l'élongation à 72°C, la taq polymérase se charge d'associer le nucléotide correspondant à son partenaire sur le brin guide. Courtesy of Prof. P. Bogaerts.

## 1.4 Technique de séquençage à haut débit

Le séquençage à haut débit est une technique qui réalise des centaines/milliers de réactions de séquençages en parallèle sur des fragments d'ADN ou ARN. Après une construction des données brutes, le traitement informatique se charge de reconstruire la séquence complète qui pourra ensuite être analysée pour identifier un organisme, un gène ou déceler d'éventuelles variations par rapport à la séquence de référence [5, 56, 41, 77]. Une vision simplifiée et globale est donnée à la figure 1.4.1.

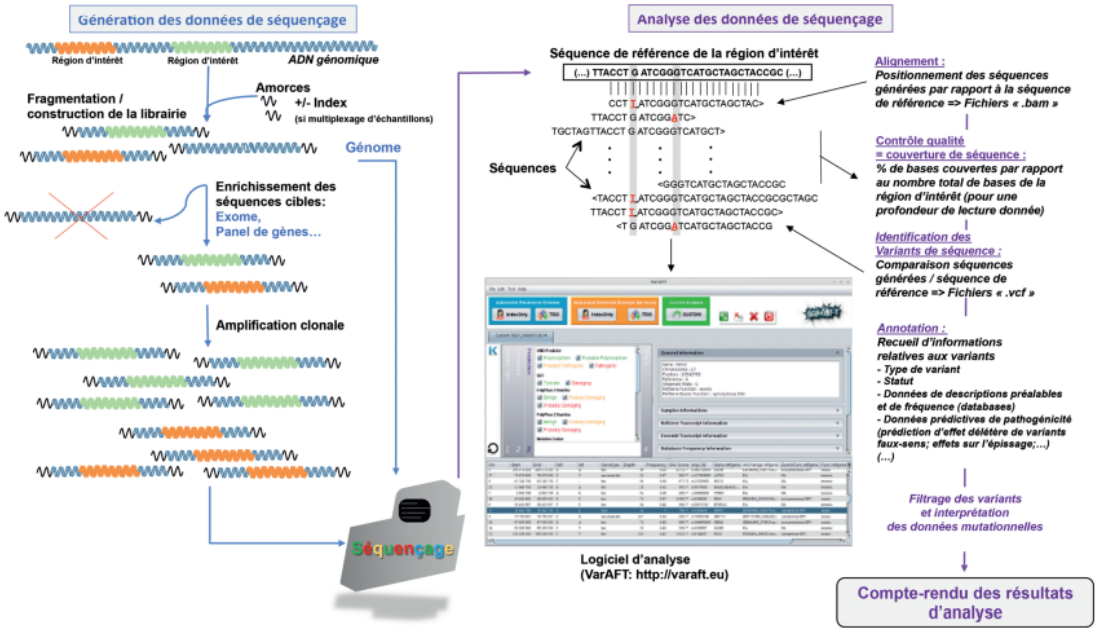



FIGURE 1.4.1 – Étapes du NGS qui se font principalement en trois phases. D'abord, la préparation des échantillons qui correspond en partie à la construction des librairies. Ensuite vient l'étape de séquençage proprement dite, effectuée par une technologie de NGS (Nanopore, illumina, etc). Enfin vient celle d'analyse ou les librairies séquencées sont alignées et annotées sur une séquence de référence pour obtenir un fichier .sam permettant d'effectuer d'autres processus comme l'illustre le schéma [44].

### 1.4.1 Génération des données de séquençage

Dans un premier temps, il y a la préparation des librairies, qui est un étiquetage des fragments d'ADN spécifiques à un patient qui peuvent ensuite être mélangés avec ceux d'autres patients avant le séquençage. Ensuite, vient l'étape d'enrichissement des séquences cibles avec les amorces. Enfin, vient l'étape d'amplification clonale qui consiste à produire des millions de fragments courts d'ADN (les librairies) afin de copier des régions d'intérêt qui doivent être analysées. Pour ce faire, plusieurs outils existent utilisant deux techniques différentes. La première est basée sur le multiplex PCR (amplification de nombreux fragments différents dans le même tube figure 3.1.1). La seconde est basée sur l'hybridation ou capture d'ADN : les régions d'intérêt, fragmentées par digestion enzymatique ou par traitement aux ultrasons, sont sélectionnées grâce à des sondes qui leur sont complémentaires [51]. Le but de cette partie est d'identifier les fragment d'un même patient ou échantillon.

### 1.4.2 Séquençage d'ADN

Le séquençage a été décrit, il y a un peu plus de 40 ans, et n'a cessé d'évoluer depuis cette période. Cette méthode est devenue une technique courante dans les laboratoires de biologie moléculaire. Les connaissances acquises grâce à cette méthode et à

la nécessité de séquencer des génomes de grandes tailles, tels que le génome humain, ont amené les chercheurs à développer des techniques de séquençage de plus en plus sophistiquées [48].

Au cours de ces 40 dernières années, plusieurs approches technologiques utilisant des techniques et algorithmes de plus en plus performants et moins coûteux ont été développées. La figure 1.4.2 décrit l'évolution des différentes techniques de séquençage d'ADN par génération.

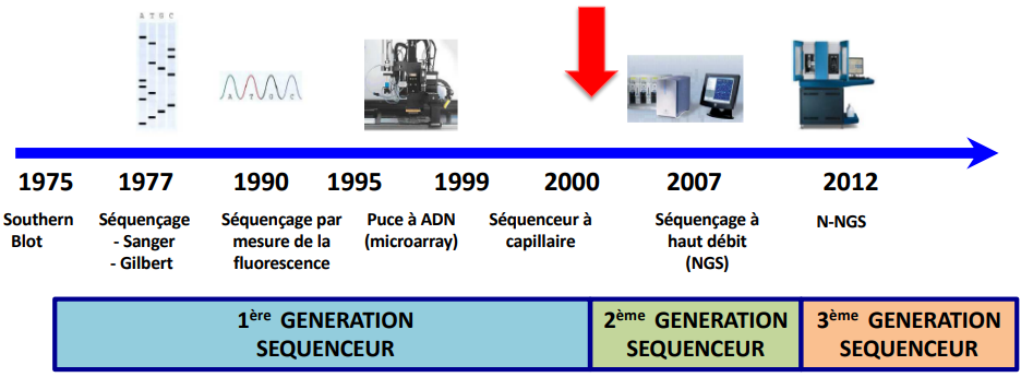



FIGURE 1.4.2 – Techniques de séquençage d'ADN par génération : la couleur bleue pour la *1ère génération de séquençage* qui est marquée par un début d'automatisation des séquençages avec la technique de Sanger qui est une méthode par synthèse enzymatique sélective. En vert, la *2ème génération de séquençage*, aussi appelée séquençage haut débit (HTS pour high-throughput sequencing) ou NGS pour next-génération, permettant l'augmentation du débit d'analyse de façon très importante passant de quelques milliers de paires de bases séquencées à plusieurs milliards. En orange, *la 3ème génération de séquençage* qui permet le séquençage direct de molécules uniques et rend ainsi superflu le marquage fluorescent ou d'autres étapes préparatoires spécifique [61].

Le séquençage d'ADN est devenu un outil essentiel en biologie moléculaire tant en clinique que dans la recherche fondamentale. Selon *Techno-Science.net*, en génétique, le séquençage concerne la détermination de la séquence des gènes, des chromosomes, ou encore du génome complet (ce qui techniquement revient à décoder l'ADN constituant ces gènes ou ces chromosomes) [88]. C'est donc une technique qui permet de lire l'enchaînement des nucléotides (bases) d'un fragment ou de la totalité de l'ADN/molécule constituant nos gènes. Selon les plateformes (illumina, nanopore), cette étape se fait par amplification sur support solide (*flow cells*) ou en émulsion sur billes isolées [51]. Pendant le processus de séquençage, l'automate recueille des données physiques (reads) en temps réel [56]. Celles-ci sont issues de différents procédés techniques utilisant la production d'un signal lumineux, d'un signal fluorescent ou d'une variation de potentiel électrique. Ceci dépend des technologies de séquençage utilisées (détaillées à la suite de cette section).

L'ère du séquençage d'ADN a connu un développement très rapide en termes de technologies de séquençage. Ses différentes analyses se basent, en premier lieu, sur des algorithmes d'alignement de chaînes de caractères (les lectures et le génome) qui utilisent un alphabet de quatre lettres ACTG.

Actuellement, la troisième génération vient compléter les limites des deux précédentes qui sont toujours utilisées dans les laboratoires. Les progrès réalisés dans ce domaine ont permis de faire baisser les coûts considérablement en appréciant le rendement (le nombre de nucléotides séquencés en une expérience ou « run »). Ces progrès ont été comparés à ceux de l'électronique (loi de Moore) : la période de doublement est de 10 mois pour les techniques de séquençage alors qu'elle est de 18 mois pour l'électronique.

— *Machine de première génération :* technologie Sanger (illustrée à la figure 1.4.3) qui permet de séquencer de façon fiable avec une précision élevée des petits fragments d'ADN (de 400 à 900 paires de bases), avec un faible débit. Le débit de cette technologie de séquençage et le coût élevé par base limitent son utilisation à la détection de variants dans des régions de petite taille (1 kb) ou à la validation de variants détectés par une autre technique. De plus, il est impossible de détecter des variants minoritaires qui ont un faible ratio allélique [1], comme cela peut être le cas pour les variants tumoraux, car le seuil de détection d'un variant par séquençage Sanger est d'environ 15-20% d'allèles mutés [70].

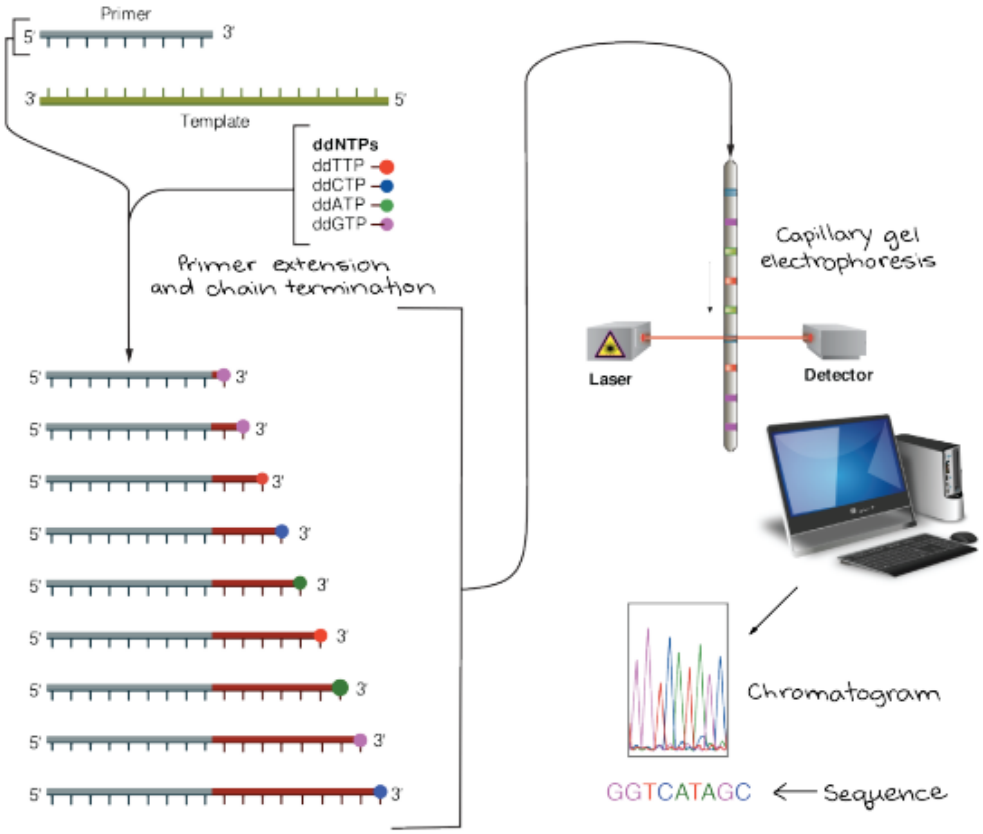



FIGURE 1.4.3 – Séquençage Sanger. Les fragments d'ADN synthétisés sont séparés en fonction de leur taille par électrophorèse sur gel et leur terminaison fluorescente est lue par le lecteur laser qui va identifier le ddNTP [2] correspondant [70].

1. fréquence à laquelle se trouve l'allèle d'un variant

— *Machine de seconde génération :* appelée NGS pour next-generation sequencing, elle regroupe l'ensemble des technologies ou plateformes de séquençage développées depuis 2005 produisant des millions de séquences en un run et à faibles coûts. Ces nouvelles technologies de séquençage massivement parallèles ont permis d'augmenter le débit d'analyse de façon très importante passant de quelques milliers de paires de bases séquencées à plusieurs milliards. Les principales plateformes de séquençage à haut débit de 2ème génération actuellement utilisées sont présentées par 3 sociétés : *Roche*, *Illumina* et *Life technologies*. Ces séquenceurs vont travailler sur support solide, Illumina présente une capacité de séquençage largement supérieure aux deux autres. Les séquenceurs SOLiD de Life technologies offrent une meilleure exactitude de séquençage et des délais plus courts, mais cela pour le même coût onéreux des équipements que les machines Illumina qui, elles, surpassent de six fois la capacité de séquençage de ses concurrents [13]. La figure 1.4.4 illustre les différentes technologies de séquençage de seconde génération ainsi que leurs caractéristiques.

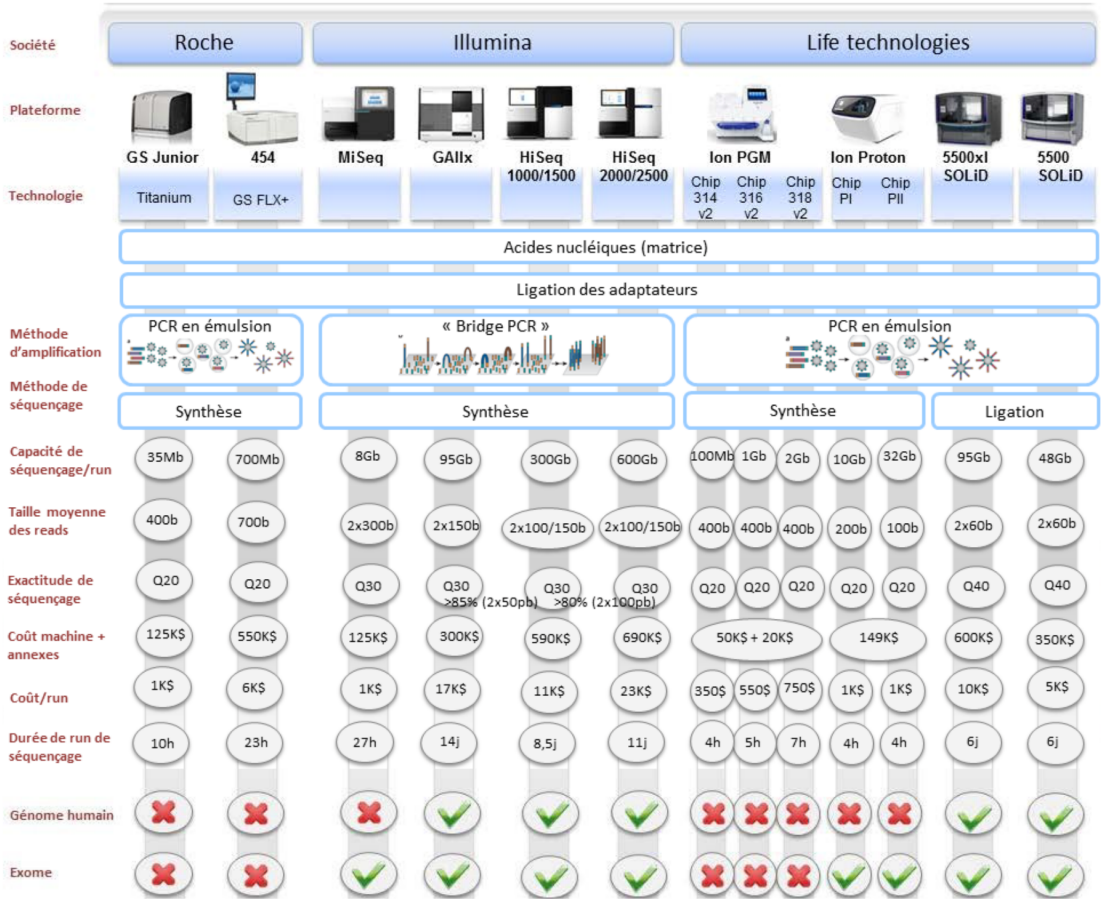



FIGURE 1.4.4 – Séquençage à haut débit de deuxième génération : principes et caractéristiques. (Image adaptée de Blervaque, R., août 2014) [13]

— *Machine de troisième génération :* une toute nouvelle technologie fait son apparition, elle peut être répartie en trois catégories : *technologie de séquençage par Nanopore, par Pacific biosciences ou par Helicos biosciences*. Elle permet le séquen-

çage direct de molécules uniques et rend ainsi superflu le marquage fluorescent ou d'autres étapes préalables au séquençage, étant donné que chaque nucléotide émet un signal électrique spécifique (dans le cas de Nanopore) [10]. Le génome comprenant beaucoup d'éléments complexes trop longs pour être résolus par les technologies de short-read sequencing, cette nouvelle génération a vu le jour avec pour objectif de séquencer des fragments de plusieurs milliers de paires de bases (jusqu'à 200 kb : long-read sequencing), permettant de mettre en évidence des variants structuraux de grande taille de l'ADN, ou encore des régions fortement répétées. Elles permettent également de reconstruire des haplotypes [1] complets. Ici aussi, deux types de méthodes se confrontent : le séquençage en temps réel de molécules uniques (single-molecule real-time sequencing= SMRT) et l'approche synthétique basée sur les technologies de short-read sequencing pour construire des reads longs in-silico [59].

Comme illustré précédemment, les technologies de séquençage de nouvelle génération constituent un progrès considérable dans la lecture de l'ADN, notamment en termes de coût et de rapidité. Il conviendra de choisir judicieusement la technologie employée en fonction de l'application. Les séquenceurs de fragments courts offrent une plus grande précision de lecture, permettant une détection plus sensible des variants de petite taille (de 1 à quelques dizaines de nucléotides), ce qui est un avantage précieux dans des domaines tels que l'oncologie pour la détection des variants somatiques. Les séquenceurs de fragments longs, malgré un taux d'erreurs plus important mais en constante amélioration, mettront plus facilement en évidence des réarrangements structuraux de grande taille, ou pourront lire des transcrits ARNm entiers [59].

| | Technologie de séquençage | Longueur des *reads* | Débit d'un *run* | Taux d'erreurs | Durée d'un *run* | Coût par Gb |
|---|---|---|---|---|---|---|
| **ABI 3730xl** | Séquençage par terminaison de chaîne | 400 à 900 pb | 2100 Kb | 0.001% | 1-3 hr | 2 400 000 $ |
| **Illumina MiSeq V3** | Séquençage par synthèse | 300 pb (PE) | 15 Gb | 0.1%, substitution | 21-56 hr | 100 $ |
| **Ion Proton** | Séquençage par synthèse | 300 pb (SE) | 10 Gb | 1%, InDel | 2-4 hr | 80 $ |
| **Oxford Nanopore MinION** | ONT | 200 Kb | 1.5 Gb | ~12%, InDel | 48 hr | 750 $ |
| **Pacific Biosciences RSII** | SMRT | 20 Kb | 1 Gb | 13%, en lecture unique | 4 hr | 1 000 $ |

FIGURE 1.4.5 – Comparaison de quelques séquenceurs de différentes générations [70]. En rose, un séquenceur Sanger. En orange, deux séquenceurs NGS de seconde génération. En mauve, deux séquenceurs de troisième génération. SE : Single-End. PE : Paired-End.SMRT : Single-Molecule Real-Time. Il est important de noter que les prix inscrits sur ce tableau ont probablement changer depuis la date de publication de l'article.

1. Un haplotype est un ensemble de gènes situés côte à côte sur un chromosome.

### 1.4.3 Analyse et traitement des données

Cette étape d'analyse et traitement de données est un chevauchement de plusieurs autres étapes afin d'arriver à des résultats concluants après le séquençage. Après le séquençage proprement dit, il est important d'évaluer la qualité des données de séquençage : ceci passe par deux paramètres qui sont *la profondeur de séquençage* (nombre de lectures ou reads obtenus indépendamment pour chaque base ciblée) et *la couverture* (pourcentage de bases effectivement séquencées par rapport au nombre de bases ciblées au départ). La prochaine étape est le traitement informatique des données brutes (*base calling*) où les données brutes subissent plusieurs pré-traitements suivant des algorithmes qui les convertissent en données de séquence (lectures). Ensuite, des millions de reads obtenus pour chaque échantillon sont alignés sur le génome de référence. C'est principalement à cette étape d'alignement que le biologie requiert un grand soutient de l'informatique. En effet, celle-ci présente deux grands problèmes connus sous le nom du problème d'alignement et d'assemblage :

1. **Problème d'alignement**

   En bio-informatique, un alignement de séquences est la façon d'organiser des séquences d'ADN, ARN afin d'identifier des régions de similarité entre celles-ci qui peuvent être une conséquence fonctionnelle, structurelle ou évolutive [46]. Résoudre un problème d'alignement, c'est comme résoudre un problème de puzzle qui contient d'une part les éléments à aligner (les pièces/séquences/lectures) et d'autre part, la référence à obtenir comme illustré sur l'image ci-dessous.

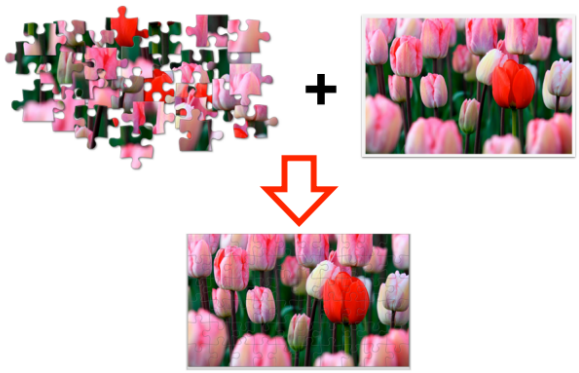


FIGURE 1.4.6 – Illustration du problème d'alignement sous la forme d'un puzzle avec d'un côté les différentes pièces et d'un autre côté l'image à obtenir.

   De nombreuses méthodes et algorithmes ont été conçus pour fournir des solutions adaptées afin de pouvoir répondre à cette problématique. La section 2.1 illustre le fonctionnement des algorithmes basiques utilisés derrière le problème d'alignement.

2. **Problème d'assemblage**

L'assemblage peut être défini comme un procédé qui consiste à reconstruire le génome de base d'un organisme à partir des milliards de séquences générées par un séquenceur à haut débit. Par analogie avec un puzzle, il s'agit d'assembler les pièces sans l'image modèle sur laquelle se baser (Figure 1.4.7). Ici, il ne s'agit pas de tester toutes les combinaisons en comparant les résultats deux par deux. En effet, même le plus puissant des ordinateurs prendrait énormément de temps pour effectuer le travail.

Les progrès des technologies de séquençage et l'accès accru au séquençage ont récemment suscité un regain d'intérêt pour les algorithmes et les outils d'assemblage de séquences. De nombreux algorithmes (dont certains sont illustrés dans la section 2.4) existent afin de traiter le problème d'assemblage de séquence d'ADN. Cependant, la propension pour l'erreur dans l'assemblage due à la répétition des bases de nucléotide dans le génome augmente la difficulté d'assembler les courtes séquences correctement. Lorsque le génome est répétitif, les tentatives d'assembler échoueront d'une manière ou d'une autre car les algorithmes ne savent pas les estimer ou les prédire. La façon d'échouer dépendra de l'algorithme utilisé.

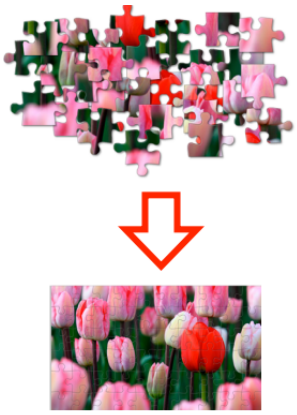


FIGURE 1.4.7 – Illustration du problème d'assemblage : sous forme d'un puzzle avec juste les pièces du puzzle sans aucune référence, dont l'objectif est de trouver des matchs afin d'obtenir la meilleure représentation de la pièce finale.

Lorsque le génome d'origine est connu (projet génome humain), il suffit d'*aligner* les lectures et de regarder où ça matche le mieux (là où il y a le plus de bases communes). Si le génome d'origine n'est pas connu, il faut tenter d'*assembler* les lectures. Le problème d'alignement de lectures est un problème complexe dans le sens que le séquenceurs les plus modernes fournissent des milliards de lectures et pour chacune d'elle, il faut les comparer avec un génome d'origine (le génome humain est de +/- 3 milliards de nucléotides) [49]. Ces deux problèmes seront l'objet du chapitre 2.

La phase d'alignement est suivie de celle de la détection des variants qui relève les coordonnées génomiques des différentes variations, identifiées entre les données expérimentales et la séquence de références, en les compilant et annotant dans un fichier au

format .VCF (variant call format). Ensuite vient la dernière phase qui consiste à interpréter biologiquement les données obtenues.

### 1.4.4 Pipeline

Dans le monde de la bio-informatique, le pipeline est devenu un terme conventionnel pour désigner l'ensemble des étapes conduisant à l'interprétation des résultats bruts de séquençage. En général, un pipeline n'est pas une structure en tant que telle, mais plutôt un ensemble malléable d'éléments qui peuvent être arrangés, configurés et adaptés en de nouvelles structures selon les besoins. Ces éléments peuvent être d'une part des moyens humains avec des domaines de tâches assignés et d'autre part des systèmes matériels /logiciels [30]. Comme supposé dans la définition ci-dessus, il n'existe pas de pipeline universel, chacun est conçu et adapté à une tâche et une utilisation bien précise.

Une exemple de flux de travail complet pouvant être appliqué est la détection des variants qui est réalisée à l'étape d'analyse de données. Ce flux de travail comprend les étapes suivantes :

- le contrôle de qualité (BaseCalling)
  - évaluation de la qualité de données FastQ
  - ajustement des lectures de mauvais qualités, etc.
- l'alignement de séquences
- l'annotation du génome
- l'identification des variants d'intérêts

La figure 1.4.8 illustre une exemple de pipeline bio-informatique pour la détection de variant de l'étape d'analyse et de traitement de données lors d'un séquençage à haut débit avec les différents types de fichiers d'entrée et de sortie (qui sont détaillés à la section 1.4.5).

La figure 1.4.9 illustre un autre exemple de flux de travail de lecture vers des variants mettant en évidence l'entrée et la sortie, une brève description et les outils qui peuvent être utilisés à chaque étape. Bien que cette figure présente les outils utilisés, il est important de rappeler qu'il existe une grande variété d'outils/algorithmes qui peuvent être utilisée pour chaque processus [74].

Ces deux-ci montrent que pour un même objectif de travail les pipelines ne sont pas identiques, ni du point de vue du cheminement, ni des outils, ni même de la logique.

À l'annexe B, se trouve un exemple de pipelines d'analyse des lectures longues pour les données des séquenceurs de troisième génération (présenté dans l'article "Opportunities and challenges in long-read sequencing data analysis") avec pour chaque étape à l'intérieur, plusieurs outils qui peuvent être utilisés [6]. Les cases verte représentent les processus communs aux analyses à lectures courtes et longues. Les cases orange représentent les processus propres aux analyses à lectures longues.

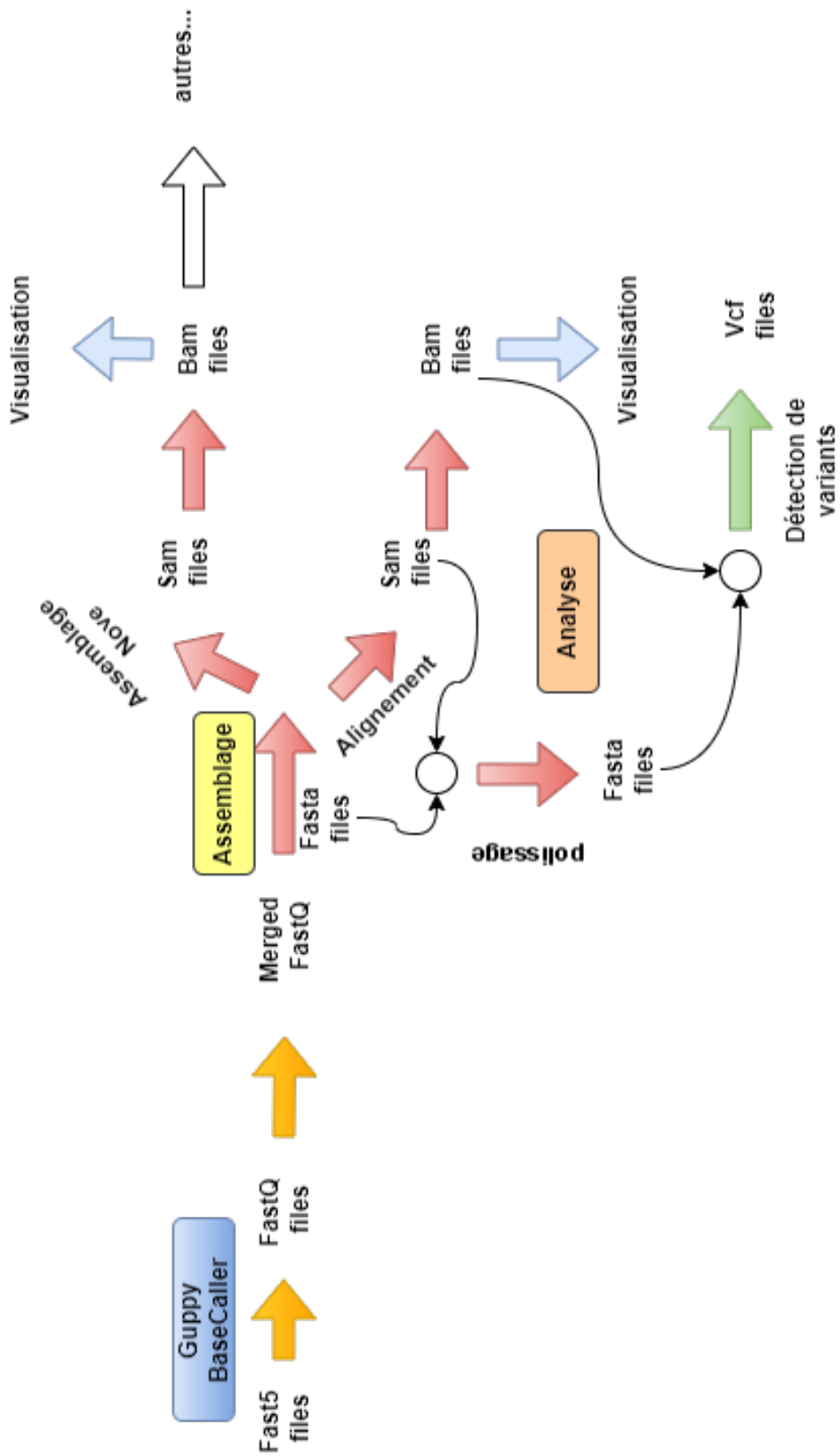



FIGURE 1.4.8 – Exemple de pipeline bio-informatique (étape d'analyse de données) pour la détection de variant. Il débute avec des données (fichier fast5) provenant d'un séquenceur qui sont transformées, traitées pendant l'étape de basecalling pour obtenir un fichier fastQ. Celui-ci servira d'entrée pour le phase d'alignement qui produire un fichier Sam qui sera converti en fichier Bam. Celui-ci permet d'avoir une visualisation de l'alignement effectué par rapport à une référence (généralement le génome humain), mais également de détecter les variants qui seront présentés dans une fichier au format vcf.

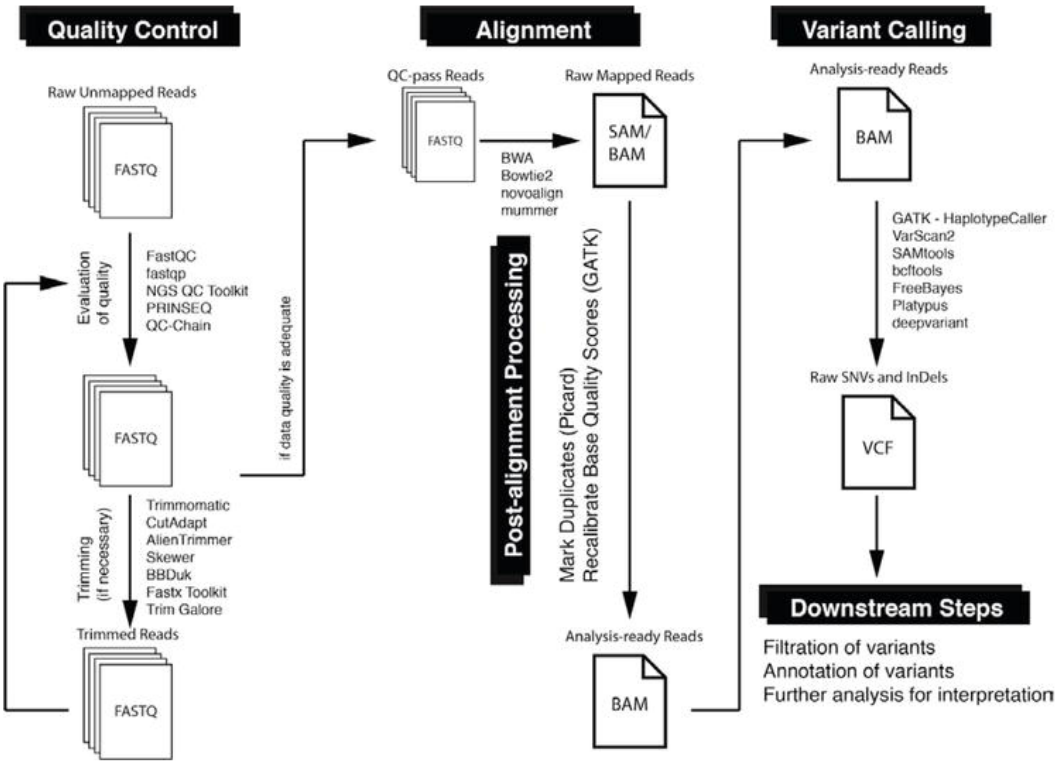



FIGURE 1.4.9 – Un exemple de flux de travail de découverte de variants à échantillon unique. Chaque étape est étiquetée dans les rectangles noirs. Les outils les plus utilisés pour chaque opération sont également présentés [74].

### 1.4.5 Formats de fichiers utilisés lors d'une analyse complète de séquençage à haut débit

La plupart des technologies de séquençage génèrent des données sous la forme de reads (séquences) rassemblées dans un fichier de type particulier (exemple : Fasta, FastQ, etc). Ces données brutes sont inutilisables telles quelles, jusqu'à ce qu'elles soient traitées par différents algorithmes générant des données "analysables". Dans la pratique les séquences d'ADN sont stockées dans des fichiers bien particuliers dont les plus récurrents :

— FASTQ (.fastq) : un fichier FASTQ est un fichier de texte reprenant un ensemble de lecture, chaque lecture est composée d'un ensemble de 4 lignes : la première indique le nom de lecture, la deuxième représente la séquence de bases, la troisième est inutile, et la quatrième est la séquence de qualité de base de la lecture. Ces fichiers peuvent être caractérisés comme suit :

- *Type de fichier* : séquence de lecture
- *Signification du nom* : comme FASTA, mais avec la qualité (Q)
- *Qui le génère* : le séquenceur
- *Qui le lit* : les outils de mapping, les visualisateurs, l'utilisateur
- *Exemple* : ci-dessous, 4 lignes d'un fichier fastq


FIGURE 1.4.10 – Représentation du contenu d'un fichier FASTQ : chaque séquence est codée sur 4 lignes : 1. un @, suivi du nom de la séquence ; 2. la séquence elle-même ; 3. un + (avec éventuellement le nom de la séquence, encore une fois) ; 4. les indices de qualité de la séquence [25].

— La première ligne commence par un caractère "@" suivi de l'identifiant de la séquence et éventuellement d'une description (de la même façon qu'un fichier au format FASTA, où le "@" serait remplacé par ">").

— La deuxième ligne contient la séquence nucléique brute.

— La troisième ligne commence par un caractère "+", parfois suivi par la répétition de l'identifiant de la séquence et de sa description si celle-ci est présente.

— La ligne 4 contient les scores de qualité associés à chacune des bases de la séquence de deuxième ligne et doit avoir exactement le même nombre de symboles qu'à celle-ci [24].

La qualité de base Q peut être calculée de plusieurs façons. Une formule est la suivante :

$$\text{Q'} = -10 * log_{10}p$$

avec p la probabilité que la base soit incorrecte.

Le $-10 * log_{10}$ permet une meilleure interprétation. Par exemple si Q' = 20, cela signifie qu'il y a une chance sur 100 que la base soit incorrecte (le nombre de la dizaine correspond à l'indice x dans $10^x$). Ensuite pour obtenir Q, 33 est ajoutée à la valeur de Q' avant de regarder la correspondance dans un tableau ASCII. Le caractère ASCII trouvé sera la valeur de Q [49].

— FASTA (.fa ou .fasta) : un fichier FASTA est un fichier de texte reprenant un ensemble de lectures, chacune composée de deux lignes : la première donne des informations d'identification, la deuxième donne la séquence du génome (la lecture). Ces fichiers peuvent être caractérisés comme suit :

- *Type de fichier* : séquence
- *Signification du nom* : format utilisé par l'outil FastA (fast alignment)
- *Qui le génère* : le séquenceur
- *Qui le lit* : presque tous les outils de bio-informatique, l'utilisateur
- *Exemple* :

```
>sequence1
CGATGTACGCTAGAT
```

FIGURE 1.4.11 – Fichier FastA : Chaque séquence commence par un chevron (>), suivi du nom de la séquence. Bien que cela ne soit pas obligatoire, il est recommandé que le nom de la séquence soit unique dans le fichier. La séquence elle-même suit [25].

— plusieurs autre formats qui serviront pour la suite du processus :

- SAM (simple alignment map), détaillé dans la section 4.2.2
- BAM (binary alignment map), la version binaire d'un fichier SAM
- VCF (viariant call format), détaillé dans la section 4.2.1
- etc.

## 1.5 Médecine de précision

Pour mieux soigner les maladies, il est nécessaire d'avoir une idée concrète sur leurs causes. L'analyse d'ADN, en trouvant les gènes qui en sont responsables, permet des diagnostics et pronostics plus sûrs. Au fil du temps, la médecine se rend compte des limites à donner le même traitement à différents patients montrant les mêmes symptômes. Le taux de réponse aux traitements traditionnels varie entre 20 et 80%. *Les différences génétiques individuelles peuvent être plus ou moins responsables de l'efficacité des traitements [57].*

La recherche et la médecine génèrent aujourd'hui des quantités de données colossales et les mécanismes biologiques étudiés sont souvent d'une complexité extrême. La «bio-informatique » et le «big data» sont des termes largement employés dans l'actualité scientifique et médicale [11]. De quoi s'agit-il ? *La bio-informatique* consiste en un développement d'algorithmes efficaces permettant de générer, stocker, analyser et résoudre un problème biologique spécifique. Elle peut ainsi être définie comme la science du traitement et de l'analyse informatique de données biologiques. La bio-informatique met en jeu plusieurs champs disciplinaires tels que décrits dans l'image 1.5.1 [32].

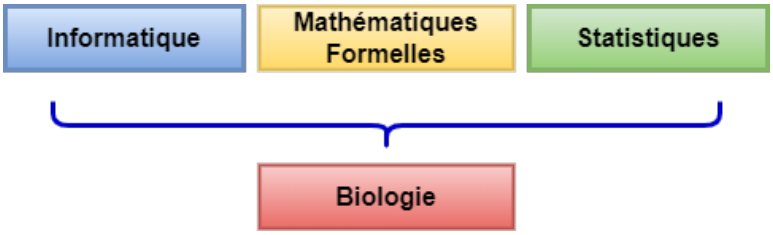



FIGURE 1.5.1 – Bio-informatique en temps que pluridisciplinarité scientifique

Aussi appelé *Biologie in Silico*, la bio-informatique est principalement axée autour de trois activités :

— L'acquisition et organisation des données biologiques
— La conception de logiciels pour l'analyse, la comparaison et la modélisation des données.
— L'analyse des résultats produits par les logiciels.

Cependant, l'un des défis est de décoder l'information contenue dans les séquences d'ADN c'est-à-dire :

— Trouver les gènes
— Prédire la séquence d'acides aminés produite par un gène
— Identifier les régions régulatrices du génome
— Reconstruire la séquence cible à partir des fragments obtenus.
— Identifier la présence d'erreurs, régions répétées

Les perfectionnements apportés par la bio-informatique vont de l'analyse du génome à la modélisation de l'évolution d'une population animale dans un environnement donné, en passant par la modélisation moléculaire, l'analyse d'images, le séquençage de génomes, la reconstruction d'arbres phylogéniques [34].

## 1.6 Particularité des globules rouges

Dans le cadre du projet de typage érythrocytaire, lancé par le CHU UCL Namur, il est important de dire quelques mots sur les globules rouges. Le rôle de ceux-ci est de faire office de sac à hémoglobine et de permettre le transport de l'oxygène des poumons vers les cellules et à l'inverse, l'évacuation du $CO_2$ des cellules aux poumons. Le globule rouge est une cellule ayant perdu son noyau et qui est donc dépourvue d'ADN. Dans le cadre de typage érythrocytaire, c'est l'ADN des globules blancs qui va être séquencé.

Le groupe sanguin est déterminé par les antigènes des globules rouges, des protéines à la surface des cellules, que le système immunitaire reconnaît (type érythrocytaire) [15]. Ceux-ci dépendent du matériel génétique de la personne. Les patients récemment transfusés présentent dans leur sang les globules rouges du patient, mélangées avec les globules rouges transfusés. Quand des globules rouges se retrouvent en présence d'anticorps qui reconnaissent les antigènes qu'ils expriment à leur surface, ils vont s'agréger. C'est ça qui va créer des caillots. C'est cela qu'il faut éviter lors de la transfusion. Pour cela, avant de la faire, il faut s'assurer que le sang qui va être donné ne contient pas des antigènes qui vont être reconnus par les anticorps du receveur. Il existe 36 systèmes pour un total de 308 antigènes. Le système le plus connu est le système ABO.

Une des solutions actuelles pour identifier le typage érythrocytaire est d'utiliser des sondes ADN ou encore d'utiliser un test d'agglutination cependant. Cependant, le génotypage par séquençage pourrait également être utilisé, il s'agirait alors de regarder les variants identifiées pour connaître le profil érythrocytaire (plus de détails se trouvent dans la section 4.4).

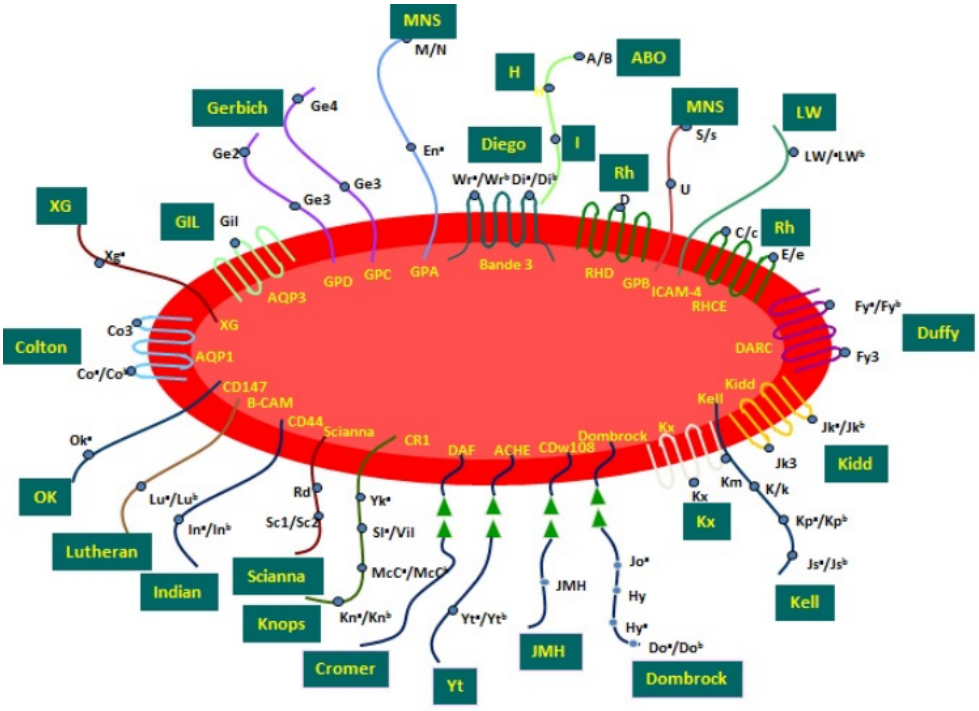



FIGURE 1.6.1 – Illustration des antigènes dans un globule rouge

## 1.7 Conclusion

Dans ce chapitre, une brève introduction sur quelques notions de la biologie moléculaire, utiles pour comprendre les bases de la bio-informatique, a montré que l'avènement des technologies de séquençage de nouvelle génération a grandement favorisé le progrès dans l'étude des maladies humaines aux niveaux génomiques. Ce progrès passe par l'extraction de l'ADN ou ARN d'un organisme, le séquençage au niveau profond et l'analyse des données afin de pouvoir efficacement détecter des variants pathogènes et découvrir des gènes cibles pour des thérapies. D'autre part, les enjeux de l'identification des profils érythrocytaires de patients ont été exposés. Le chapitre suivant introduira les différents algorithmes de base utilisés par les traitements de données issues du séquençage.

# Chapitre 2

# Algorithmes de base

Le but de ce chapitre est d'expliquer les algorithmes de base de la bio-informatique afin de bien comprendre le contexte de ce mémoire. Son contenu vient du cours en ligne de Coursera "Algorithms for DNA Sequencing" [49]. Les images qui figureront dans ce chapitre sont également issues de ce cours en ligne, car elles illustrent bien les concepts et sont donc pertinentes pour une bonne compréhension globale.

Comme expliqué dans le chapitre précédente, l'ADN est constitué de bases A, C, G, T. Un séquenceur d'ADN va fonctionner de la façon suivante : il va prendre en entrée un ADN sur lequel il va lire des sous-chaînes sélectionnées, dans de nombreux cas, aléatoirement. Ces sous-chaînes sont appelées les *lectures*. Bien qu'elles soient plus petites par rapport à la longueur de l'ADN en entrée, elles sont en grandes quantités, en nombre suffisant pour couvrir l'ADN d'entrée. D'un point de vue informatique, cet ADN et sous-chaînes seront considérés comme un *type string*. Du point de vue de la syntaxe, il s'agira d'un alphabet $\epsilon$ = {'A', 'C', 'G', 'T'}.

L'ADN pouvant être considéré comme un string, les fonctions élémentaires de ce type sont applicables et pertinentes pour l'analyse ADN. Comme illustré dans l'annexe A, il peut facilement en être déduit des fonctions permettant par exemple de faire le complément inverse d'un brin ADN.

## 2.1 Problèmes de correspondance

Comme expliqué dans la sous-section 1.4.3, le premier problème à résoudre est le problème d'alignement. Pour pouvoir aligner deux séquences, celles-ci doivent matcher. Par simplicité, notre lecture correspond à un mot et notre génome à une phrase. Il faut donc trouver tous les endroits dans la phrase où le mot matche parfaitement.

### 2.1.1 Correspondance exacte

Une idée d'algorithme est de faire glisser une fenêtre, de la longueur du mot, sur la phrase. Pour chaque position de cette fenêtre sur la phrase, regarder les caractères un

à un du mot et déterminer s'ils matchent celui correspondant dans la phrase dans cette fenêtre.

La fonction a comme *paramètres le mot et la phrase* et comme *valeur de retour une liste reprenant toutes les positions* pour lesquelles le mot, placé à partir de celle-ci, matche parfaitement avec le morceau du texte. Un code relativement simple et intuitif serait :

```
"""
Renvoie une liste de positions pour lesquelles, le mot, placé
à partir de celle-ci sur la phrase, matche parfaitement le morceau de
la phrase commencant à cette position.

Param:
- mot (str) : le mot dont on cherche les occurrences
- phrase (str) : la phrase dans la quelle on cherche les occurrences

Return:
- occurences : la liste de positions pour lesquelles, le mot,
placé à partir de celle-ci, matche parfaitement le morceau du
texte commencant à cette position.
"""
def naive(mot, phrase):
    occurences = []
    # boucle sur les alignements
    for i in range(len(phrase) - len(mot) + 1):
        match = True
        # boucle sur les caractères du mot
        for j in range(len(mot)):
            # compare les caractères du mots et de la phrase
            if not phrase[i+j] == mot[j]: # si ils ne matchent pas
                # on stop l'alignement
                match = False
                break  # break of this loop
        # si tous les caractères matchent, on note la position du
        # début du mot dans la phrase
        if match:
            occurences.append(i)
    return occurences
```

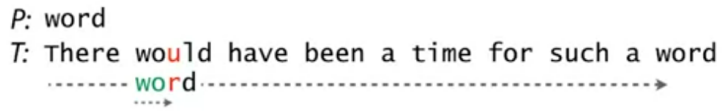
Par hypothèse, ce mot a une longueur $x = |mot|$ et ce texte une valeur y = $|texte|$ . Alors le nombre d'alignements possibles est :

$$y - x + 1$$

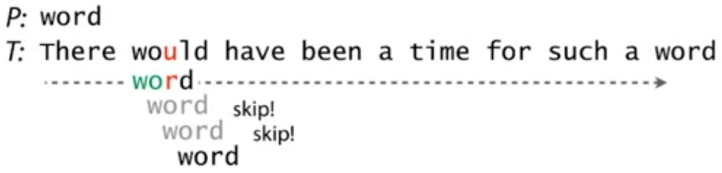
avec un nombre maximal et minimal de comparaisons, respectivement de :

$$x\ (y - x + 1)\ \textit{et}\ y - x + 1$$

Cependant, de nombreux alignements sont inutiles. Supposons un mot P et une phrase T tels qu'illustrés ci-dessous :




À la position d'alignement telle que représentée, le mot "word" rencontre une différence avec la phrase à la position de la lettre "r". La solution naïve voudrait que le mot soit décalé d'une position. Or, il paraît évident qu'à la prochaine position, le mot ne matchera pas la phrase étant donné l'alignement de la lettre *u et o*. Il serait donc intéressant de déplacer le mot en fonction de l'erreur rencontrée. Autrement dit, il faudrait casser la boucle lorsque les lettres ne matchent pas et avancer jusqu'à dépasser la lettre fautive.




#### 2.1.1.1 Boyer-Moore

L'algorithme de Boyer-Moore est une bonne alternative à la solution naïve, car il ignore les alignements dont il n'a pas besoin. La particularité de celui-ci est qu'il boucle d'une part *de gauche à droite pour les alignements* (comme dans la solution naïve) et ensuite de *droite à gauche pour les comparaisons* de lettres. Boyer-Moore travaille selon *deux règles* : celle du *mauvais caractère* et celle du *bon suffixe*.

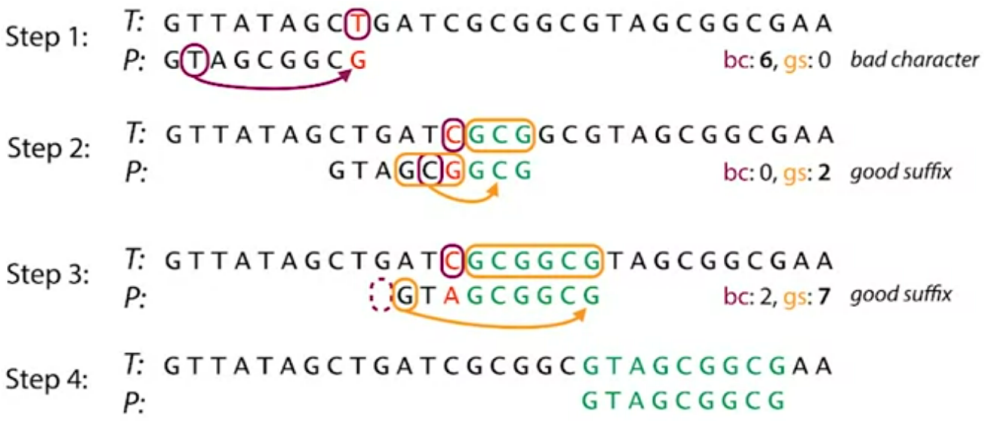
— *Règle du mauvais caractère :* arrivé à un mauvais caractère, il faut passer les alignements jusqu'à (1) arriver à un match, ou (2) le mot en entier dépasse le mauvais caractère.


- Dans l'*étape 1*, les 3 premiers caractères comparés matchent. Le 4ème ne matche pas. Les alignements sont passés jusqu'à rencontrer la lettre "C" dans le mot P (1).
- Dans l'*étape 2*, le premier caractère matche. Le second ne matche pas, les alignements sont ensuite passés jusqu'à ce que le mot dépasse le mauvais caractère (2).

- Dans l'*étape 3*, toute la séquence matche.

— *Règle du bon suffixe :* soit t le sous-string du *mot P* qui matche dans la *phrase T*. Les alignements sont passés jusqu'à (1) ce qu'il n'y ait plus d'erreur entre le mot et *t* ou (2) le mot P dépasse t.


En pratique, les deux règles sont utilisées en même temps. Celle qui aura le plus grand nombre de caractères à passer sera utilisée. Voici une illustration :




Afin de savoir exactement combien d'alignements sont à sauter, Boyer-Moore construit en preprocessing une table de recherche. Donnons un exemple pour la règle du mauvais caractère pour un mot P = "TCGC (pour rappel, l'aphabet $\Sigma = \{"A", "C", "G", "T"\}$) :

$P$

| $\Sigma$ | T | C | G | C |
|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 |
| C | 0 | - | 0 | - |
| G | 0 | 1 | - | 0 |
| T | - | 0 | 1 | 2 |


La consultation du tableau en fonction de la lettre T de la phrase et la lettre G du mot donne 1, ce qui correspond au nombre d'alignements à sauter.

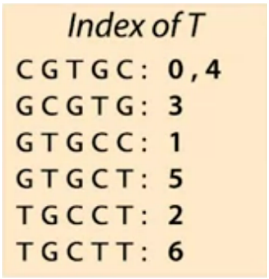
#### 2.1.1.2 Préprocessing et indexation

Dans les algorithmes présentés, tant que le mot n'est pas connu, aucune action n'est effectuée sur le texte. Il est intéressant de déjà travailler le texte dans ce qui s'appelle le préprocessing. Un algorithme qui prend en entrée un mot P et un texte T est dit :

— *offline* : si T est préprocessé (ex : moteur de recherche, alignement des lectures)

— *online* : sinon (ex : solution naïve dans la correspondance exacte, Boyer-Moore

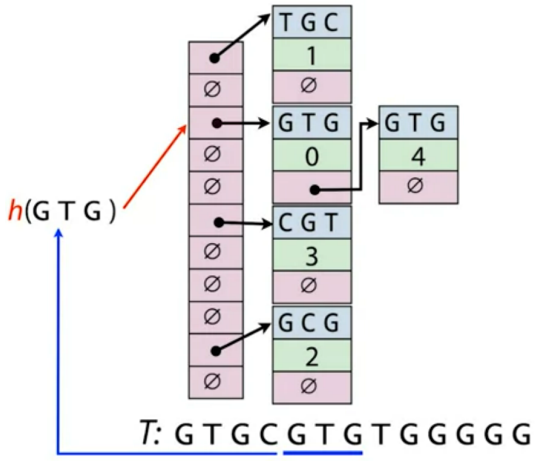
Dans le cas d'un moteur de recherche, tout le world wide web (www) est indexé et déjà organisé de sorte que lorsqu'un utilisateur effectue une recherche sur internet, il tombe rapidement sur des résultats. Cette technique est analogue avec l'index d'un livre qui indique, pour certains mots, à quelles pages les trouver. Pour les algorithmes d'alignement de lecture, c'est le même principe qui va être appliqué. Le génome de référence va être indexé.
*L'indexation k-mer* consiste à prendre chaque sous-séquence (mot) de taille *k* de l'ADN (texte) et d'indiquer la position à laquelle elle commence dans celui-ci. Ci-dessous un exemple de table d'indexation 5-mer pour un texte T = 'CGTGCGTGCTT' :




Ici il s'agit d'un multimap car un 5-mer peut être associé à plusieurs endroits dans le texte T.
Afin de représenter un multimap, il existe plusieurs structures de données pour le représenter :

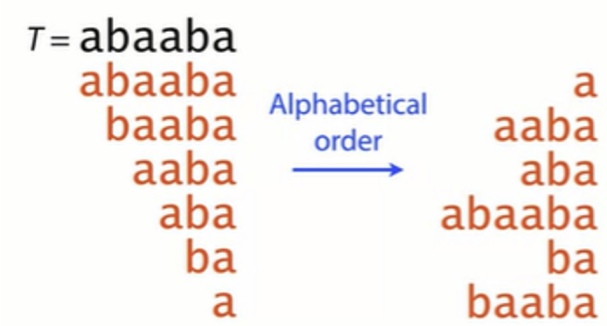
1. *Clé-valeur* : il s'agit d'une sorte de dictionnaire ordonné où chaque clé est associée à une valeur. La clé sera un k-mer tandis que la valeur sera l'indice du texte où ce k-mer matche parfaitement. Pour interroger une telle structure ordonnée, une *recherche binaire* peut-être utilisée. Le principe consiste à prendre l'élément au milieu de l'index et de le comparer avec ce qu'on cherche. S'il est plus grand (ou plus petit), on considère uniquement la première (ou deuxième) moitié dans laquelle l'élément se trouvant au milieu sera comparé avec ce qui est cherché. Cette boucle se répète jusqu'à trouver l'élément.

2. *Hash table* : il s'agit d'une table associée à une fonction de hashage.



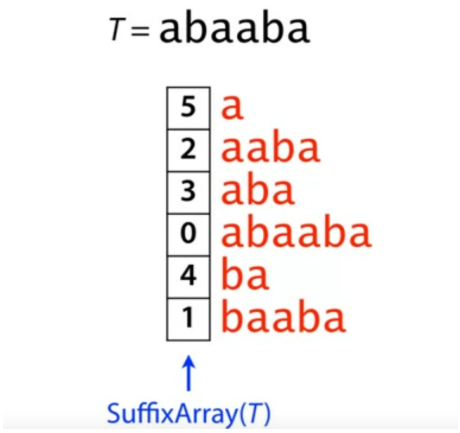

La fonction de hashage est appliquée sur le 3-mer pour l'associer à une case dans la table. Ensuite, la case pointe vers un tuple de 3 valeurs. La première valeur est le 3-mer, la deuxième l'indice du texte T où il commence et la 3ème un pointeur au cas où un autre 3-mer identique existerait ailleurs dans le texte. En python le type de structure dictionnaire est une implémentation de la hashtable.

3. *Indexation des suffixes* : il s'agit ici d'une méthode qui utilise tous les suffixes d'un génome.




L'idée est de prendre tous les suffixes du génome, puis de les mettre dans l'ordre alphabétique afin d'obtenir une liste des suffixes. Comme les suffixes sont dans l'ordre, tous ceux qui partagent un préfixe, qui ont le même préfixe seront consécutifs dans cette liste. Ensuite, il faut effectuer une recherche binaire comme dans les méthodes précédentes. Sauf que le nombre d'index va être trop grand pour une génome avec des milliards de bases. Pour contourner ce problème, il va falloir construire ce qu'on appelle le *Tableau d'index (index Array en anglais)*.

4. *Tableau de suffixes* : au lieu de représenter juste une liste de suffixes, ceux-ci vont être représentés comme un seul entier qui indique le décalage par rapport au début du génome.




Se contenter de ce tableau d'indices de décalage n'est pas suffisant, le génome ou le texte doit être stocké. Le liste indique l'ordre alphabétique des suffixes, mais il faut savoir quelle est la séquence d'un suffixe particulier. Pour cela, il suffit d'aller regarder à l'intérieur de T grâce à l'indice de décalage.

Il existe beaucoup d'autres méthodes permettant d'indexer un génome afin de faciliter le problème d'alignement de séquences. Une autre méthode identique à celui du tableau d'index est appelée *Arbre de suffixe (en anglais Suffix tree)* et *FM index*.

### 2.1.2 Correspondance approximative

Dans la pratique, la correspondance exacte n'est pas la solution. Des différences peuvent exister à cause d'erreurs de séquencement ou suite au fait que le génome étudié n'est pas le même que le génome du modèle. Il faut donc des algorithmes qui autorisent des erreurs entre le mot et son occurrence dans un texte.

Un premier algorithme utilise la *distance de Hamming* définie comme suit : pour X et Y tel que $|X| = |Y|$, la distance de Hamming est le nombre minimum de substitutions nécessaires à transformer X en Y. Par exemple, ci-dessous, le nombre minimum de substitutions est 3 :


Une autre façon de transformer X en Y est d'utiliser la *distance d'édition (ou distance Levenshtein)*. Il s'agit du nombre minimum d'éditions (substitutions, suppressions, insertions) nécessaire pour transformer l'un en l'autre.

La fonction naïve de correspondance exacte illustrée dans la section 2.1.1 peut être facilement adaptée pour vérifier qu'une certaine distance de hamming donnée n'est pas dépassée. Lorsque le caractère du mot ne matche pas celui dans la phrase, une variable distance est incrémentée de 1 et ce, tant qu'elle ne dépasse pas une certaine valeur. Une version adaptée du code serait :

```python
"""
Renvoie une liste de positions pour lesquelles, le mot, placé
à partir de celle-ci sur la phrase, matcherait parfaitement le
morceau de la phrase commencant à cette position après un certain
nombre d'opérations d'édition inférieur à maxDistance.

Param:
- mot (str) : le 1er mot
- phrase (str) : le 2eme mot
- maxDistance (int) : la distance d'édition à ne pas dépasser
"""
def naiveHamming(mot, phrase, maxDistance):
    occurences = []
    # boucle sur les alignements
    for i in range(len(phrase) - len(mot) + 1):
      distance = 0
        # boucle sur les caracteres du mot
        for j in range(len(mot)):
            # compare les caracteres du mots et de la phrase
            if not phrase[i+j] == mot[j]: # si ils ne matchent pas
              distance += 1
                if distance > maxDistance:
                  # on stop l'alignement
                  break  # break of this loop
        # si tous les caracteres matchent, on note la position du
        # debut du mot dans la phrase seulement si on depasse pas
        # la distance de H.
        if distance <= maxDistance:
          occurences.append(i)
    return occurences
```

De manière comparative, il peut être affirmé que :

si $|X| = |Y|$ :

$$editDist(X, Y) \leq hammingDistance(X, Y)$$

si $|X| \neq |Y|$ alors :

$$\text{editDistance(X, Y)} \geq ||X| - |Y||$$

Soient deux chaînes de caractères $X = \alpha C$ et $Y = \beta A$ dont la distance d'édition des préfixes $\alpha$ et $\beta$ est connue, alors la distance d'édition entre les deux chaînes est :

$$\text{editDist}(\alpha C\ ,\ \beta A) = \min \begin{cases} \text{editDist}\ (\alpha, \beta) + 1 \\ \text{editDist}\ (\alpha C, \beta) + 1 \\ \text{editDist}\ (\alpha, \beta A) + 1 \end{cases}$$

où :

- $editDist(\alpha, \beta) + 1$ transforme C en A (substitution)
- $editDist(\alpha C, \beta) + 1$ ajoute A en $\alpha$ (insertion)
- $editDist(\alpha, \beta A) + 1$ supprime C (suppression)

De manière plus générale, pour toute base quelconque x, y tel que le préfixe est connu dans les expressions $\alpha x$ et $\beta y$ la distance d'édition est :

$$\text{editDist}(\alpha x\ ,\ \beta y) = \min \begin{cases} \text{editDist}\ (\alpha, \beta) + \delta(x, y) \\ \text{editDist}\ (\alpha x, \beta) + 1 \\ \text{editDist}\ (\alpha, \beta y) + 1 \end{cases}$$

avec $\delta(x, y)$=1 si x==y, 0 sinon

Une implémentation intuitive serait de dire que si X est vide alors la distance d'édition est la longueur de Y. Si Y est vide, la distance d'édition est la longueur de X. Et si aucune n'est vide, comparer les caractères à la même position, ensuite appeler la fonction récursive sur les trois cas : *substitution, insertion et suppression* et en retenir que le minimum. Dans le cas de la substitution, il s'agirait d'incrémenter de 1 si les caractères sont différents et de 0 s'ils sont les mêmes :

```
"""
Retourne le nombre minimum d'éditions pour faire matcher deux mots a
et b

Param:
- a (str) : le premier mot
- b (str) : le deuxième mot
"""
def naiveedDistRecursive(a, b):
    if len(a) == 0:
        return len(b)
    if len(b) == 0:
        return len(a)
    delt = 1 if a[-1] != b[-1] else 0

    return min(naiveedDistRecursive(a[:-1], b[:-1]) + delt,
               naiveedDistRecursive(a[:-1], b) + 1,
               naiveedDistRecursive(a, b[:-1]) + 1)
```

Le problème de ce code est que la fonction effectue de nombreux appels avec les mêmes arguments. Autrement dit, la fonction calcule plusieurs fois la même chose. Afin d'éviter ces appels inutiles, une solution serait l'utilisation d'un algorithme de programmation dynamique. C'est-à-dire qui vérifierait dans un tableau si la valeur pour tels paramètres est déjà connue. Le code ci-dessus adapté selon une approche dynamique devient :

```
"""
Retourne le nombre minimum d'éditions pour faire matcher deux mots a
et b

Param:
- a (str) : le premier mot
- b (str) : le deuxième mot
"""
def edDistRecursive(a, b):
    if len(a) == 0:
        return len(b)
    if len(b) == 0:
        return len(a)
    delt = 1 if a[-1] != b[-1] else 0

    i1 = (a[:-1], b[:-1])
    if not i1 in memo:
```

```
        memo[i1] = edDistRecursive(*i1)
    i2 = (a[:-1], b)
    if not i2 in memo:
        memo[i2] = edDistRecursive(*i2)
    i3 = (a, b[:-1])
    if not i3 in memo:
        memo[i3] = edDistRecursive(*i3)

    return min(memo[i1] + delt,
               memo[i2] + 1,
               memo[i3] + 1)
```

Afin de retrouver le cheminement des opérations d'éditions effectuées sur les mots, il est important de garder une trace du parcours du tableau jusqu'à la plus petite distance d'édition :

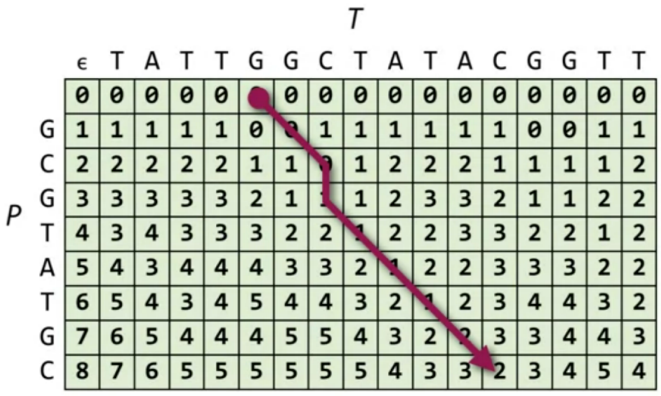



FIGURE 2.1.1 – Trace de la distance d'édition

## 2.2 Alignement global

Le problème de cette première distance d'édition est qu'elle pénalise les substitutions de la même façon que des suppressions ou insertions. Hors dans la réalité, certaines transformations de nucléotides ont une plus grande probabilité d'avoir lieu. Par exemple, les mutations d'ADN sont de deux types[55] :

- *les transversions* qui sont des échanges de purines (A $\leftrightarrow$ G)
- *les transitions* qui sont des échanges de pyrimidines (C $\leftrightarrow$ T)

La figure 2.2.1 illustre ces deux types de mutations et montre qu'il y a plus de transversions.

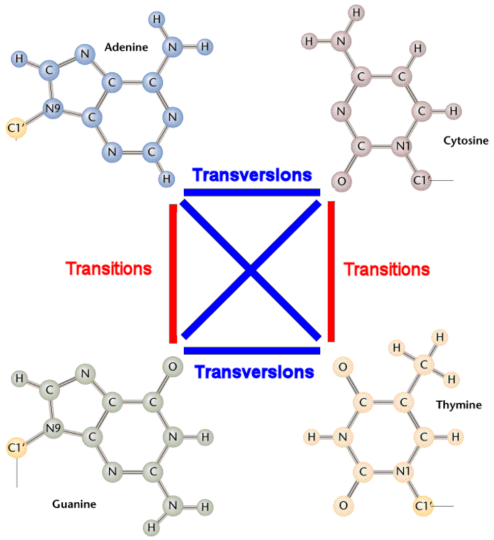



FIGURE 2.2.1 – Transversions vs transitions

Bien qu'il y ait 2 fois plus de transversions que de transitions, la transition est deux fois plus fréquente que la transversion entre 2 génomes de deux humains non-liés. Pour deux génomes humains indépendants, voici le ratio entre le taux de substitution et le taux d'indels (suppression ou insertion) [16] :

- substitution : 1/1000 bases
- indel : 1/4000 bases

Les indels étant moins fréquents, ils doivent donc être plus pénalisés que les mutations. De la même façon, dans les mutations, les transversions étant deux fois moins fréquentes que les transitions, celles-ci doivent être plus pénalisées. Cela donne, dans le cas du génome humain, la matrice de pénalité suivante :

| | A | C | G | T | - |
|---|---|---|---|---|---|
| A | 0 | 4 | 2 | 4 | 8 |
| C | 4 | 0 | 4 | 2 | 8 |
| G | 2 | 4 | 0 | 4 | 8 |
| T | 4 | 2 | 4 | 0 | 8 |
| - | 8 | 8 | 8 | 8 | |

2 *Transitions* (A↔G, C↔T)
4 *Transversions*
8 *Gaps*

FIGURE 2.2.2 – Matrice de pénalité

De l'équation de la distance d'édition peut-être dérivée celle l'alignement global :

$$\text{galign}(\alpha x\ ,\ \beta y) = \min \begin{cases} \text{galign}\ (\alpha, \beta) + p(x, y) \\ \text{galign}\ (\alpha x, \beta) + p(x, -) \\ \text{galign}\ (\alpha, \beta y) + p(-, y) \end{cases}$$

avec p(x, y), la valeur de la matrice de pénalité correspondant à (x, y).

## 2.3 Alignement local

Le problème de l'alignement local part du principe qu'il ne s'agit pas de trouver la distance entre deux chaînes X et Y, ni les occurrences d'une dans une autre, mais d'identifier les sous-chaînes de X et de Y qui sont les plus semblables l'une à l'autre. Voici un exemple :


FIGURE 2.3.1 – Sous-chaîne semblable

Dans ce cas, une matrice de score va être utilisée. Ici, un score positif (+) est donné à un match tandis qu'une pénalité (-) sera donnée à une différence entre deux caractères.

| | A | C | G | T | - |
|---|---|---|---|---|---|
| A | 2 | -4 | -4 | -4 | -6 |
| C | -4 | 2 | -4 | -4 | -6 |
| G | -4 | -4 | 2 | -4 | -6 |
| T | -4 | -4 | -4 | 2 | -6 |
| - | -6 | -6 | -6 | -6 | |

FIGURE 2.3.2 – Matrice de score

L'équation de l'alignement local est :

$$\text{lalign}(\alpha x\ ,\ \beta y) = \max \begin{cases} \text{lalign}\ (\alpha, \beta) + s(x, y) \\ \text{lalign}\ (\alpha x, \beta) + s(x, -) \\ \text{lalign}\ (\alpha, \beta y) + s(-, y) \\ 0 \end{cases}$$

avec s(x, y), la valeur de la matrice de score correspondant à (x, y)

Dans la pratique, c'est une combinaison de la programmation dynamique et de l'indexation qui sera utilisée. Ces deux outils travaillent parfaitement ensemble par leur complémentarité : l'indexation permettra de trouver rapidement un ensemble de candidats dans le génome qui pourrait avoir une bonne correspondance avec ce qui est analysé. L'index évite de devoir faire une recherche dans l'intégralité d'un génome. Si aucun d'index ou de filtre n'existe, la programmation pour les occurrences approximatives à un modèle sera utilisée. Cependant, les matrices utilisées seront bien plus grandes que celles présentées ci-dessus : les lectures font entre 100 et 300 bases tandis que le génome, par exemple celui de l'homme, fait environ 3 milliards de bases de long. Voilà pourquoi, afin d'éviter de prendre des années à analyser un tel génome, un index est nécessaire.

La programmation dynamique [1] est nécessaire pour compenser l'index qui ne gère pas bien les incohérences et lacunes. En effet, il va trouver des correspondances exactes. Il faut donc vérifier si la lecture analysée a une correspondance approximative avec le voisinage de cet index trouvé.

## 2.4 Assemblage

Cette première section présentait le problème d'alignement de lectures. De manière analogue avec des objets plus familiers, il s'agit d'un puzzle dont les pièces (lectures) doivent être alignées afin de reconstituer le modèle (le génome). Par exemple l'étude d'un génome d'un individu quelconque sur base du modèle qui est le génome retranscrit lors du Human Genome Project (HGP). La section qui suit présente les outils et notions de bases pour comprendre comment, dans le cas où le modèle de référence n'est pas connu, les lectures sont réassemblées afin d'obtenir ce qui pourrait l'être. C'est le problème d'assemblage de lectures dans génome de base qui est abordé.

L'assemblage de novo (à partir de zéro) a été utilisé lors du projet du génome humain. À titre de compréhension et pour bien comprendre les notions de bases, la figure 2.4.1 illustre, en bleu des lectures et en rouge le génome reconstitué par alignement des lectures..

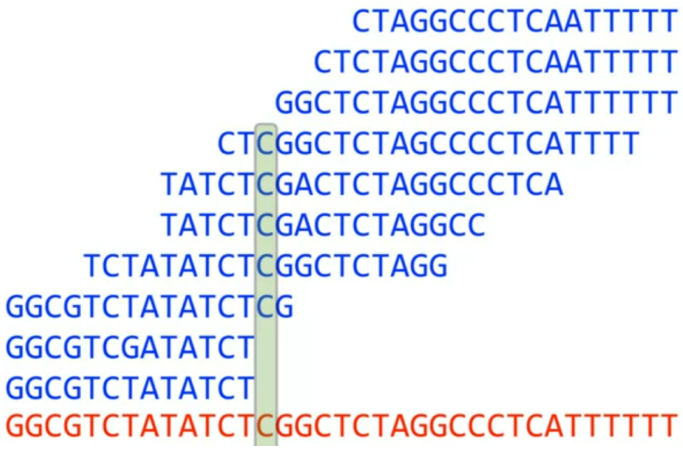



FIGURE 2.4.1 – Visualisation d'assemblage

Une notion importante est la *couverture* qui, ici sur le ruban vert, est de 5. Ce nombre indique la quantité de bases redondantes qui a permis de conclure à celle du génome en cette position. Cette base dans le génome (en rouge) reconstitué a donc une couverture de 5. Dans l'exemple, l'ensemble des lectures ont une longueur totale de 177 bases tandis que le génome de référence a une longueur de 35 bases. La couverture moyenne est donc de *5-fold*. Comment un tel alignement des lectures est-il obtenu ? En regardant si un suffixe d'une lecture est similaire au préfixe d'une autre lecture. Cela mène à deux lois d'assemblage :

1. https ://fr.wikipedia.org/wiki/Programmation_dynamique

— *Première loi* : si le suffixe d'une lecture A est similaire au préfixe d'une lecture B, alors A et B pourraient se chevaucher dans le génome.

— *Deuxième loi* : plus il y a de couverture, plus le chevauchement entre les têtes de lecture sera de plus en plus grand.

### 2.4.1 Graphe de chevauchement

Une façon de représenter le chevauchement de manière structurée est d'utiliser un graphe de chevauchement. Il s'agit d'un graphe orienté dont chaque noeud représente une lecture et chaque lien lie le noeud qui a le suffixe au noeud qui a le préfixe.
Soit un texte GTACGTACGAT et tous ses 6-mers. Pour une longueur de chevauchement $\geq 4$, le graphe de chevauchement peut être représenté par la figure 2.4.2.

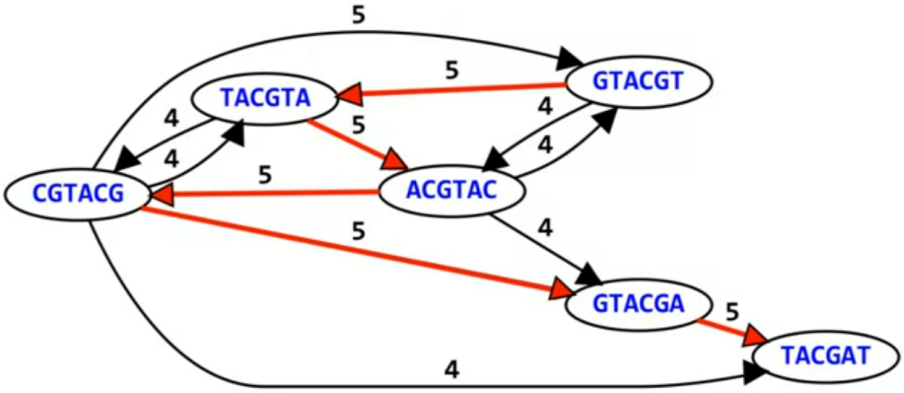



FIGURE 2.4.2 – Graphe de chevauchement

Dans le graphe ci-dessus, le chiffre associé à chaque arête représente la longueur du chevauchement entre le noeud de départ (6-mers qui possède le suffixe) et le noeud d'arrivée (6-mers qui possède le préfixe). En parcourant le graphe (lignes rouges), le texte de base peut être reconstitué.

### 2.4.2 Problème de la super-chaîne la plus courte

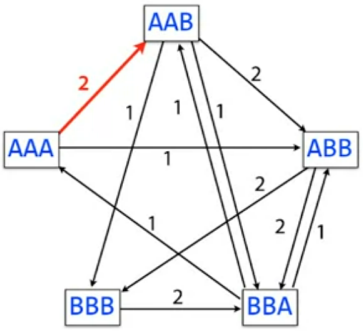
Lors d'un assemblage de novo, mettre bout à bout les lectures afin de trouver un génome reconstitué serait inutilement long et erroné. L'idée est donc de construire un génome "optimal". C'est dans ce contexte que le problème de la super-chaîne la plus courte apparaît.
Soit S un ensemble de strings, le problème de la super-chaîne la plus courte appliqué sur S doit renvoyer la chaîne la plus courte qui contient tous les strings de l'ensemble S.
Par exemple, soit :

$$S = \{'BAA','AAB','BBA','ABA','ABB','BBB','AAA','BAB'\}$$

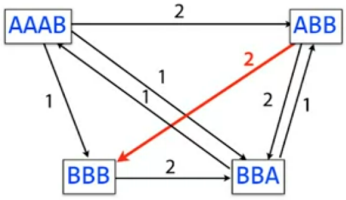
Le problème de la super-chaîne la plus courte est :

$$SCS(S) = \text{'}AAABBBABAA\text{'}$$

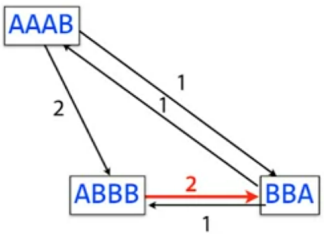
L'idée derrière un tel algorithme consiste à prendre l'ensemble S et à effectuer toutes les permutations possibles dedans. Pour chaque permutation des éléments de l'ensemble S, le plus long chevauchement est calculé pour chaque paire de strings (à l'intérieur de cet ensemble) avant d'en dériver un nouveau string. À la fin l'ensemble ne possède plus qu'un string. Ensuite, tous ces strings obtenus (un par permutation) sont comparés afin d'en retenir que le plus court. Cette approche semble idéale pour résoudre le problème d'assemblage de génome.
Cependant, ce problème est NP-Complet. Autrement dit, il n'existe *pas de solution efficace*. Si l'ensemble S possède n strings, alors il existe $n!$ permutations.
Une alternative est l'algorithme glouton. L'idée consiste, à partir du graphe de chevauchement introduit précédemment, à le parcourir selon une heuristique : à chaque passage, le lien possédant le plus grand nombre (de chevauchements) est sélectionné. Les deux noeuds à ses extrémités sont fusionnés sur base du chevauchement (suffixe-préfixe). L'itération sur le graphe se fait jusqu'à ce qu'il n'y ait plus qu'un noeud : la superchaîne la plus courte.



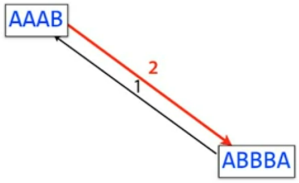

(a) 1er tour




(b) 2ème tour




(c) 3ème tour




(d) 4ème tour

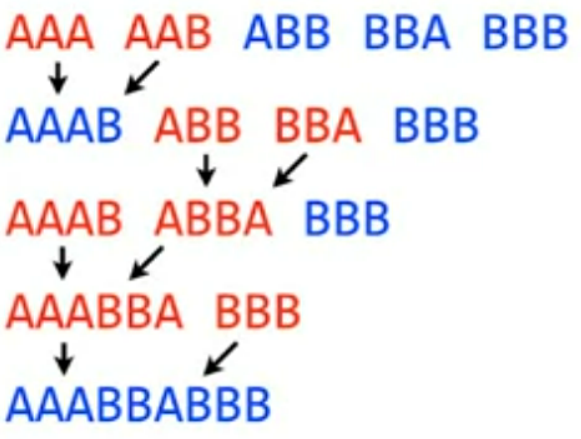
AAABBBA

(e) 5ème tour

FIGURE 2.4.3 – Itérations de l'algorithme glouton

Bien que cet algorithme soit plus court, il ne trouve pas toujours la solution optimale. En effet, quand le génome (ou la chaîne de caractère) est répétitif, la super chaîne

commune la plus courte n'est pas toujours la bonne. Par exemple :




(a) Solution non-optimale


(b) Solution optimale

La raison en est que la solution aura tendance à perdre les parties répétitives des génomes. Son but est de fournir le moins de copies nécessaires pour expliquer les lectures. Il s'agit d'un des problèmes d'assemblage, mais aussi de la troisième loi d'assemblage :

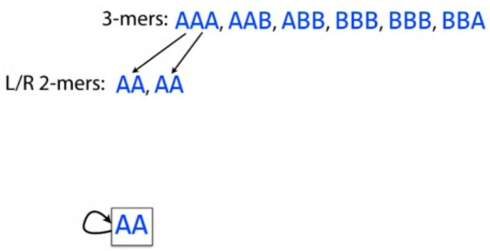
— *Troisième loi* : les répétitions rendent l'assemblage difficile.

Dans le cas du génome humain, cela est problématique vu qu'environ la moitié du génome est couverte par des séquences d'ADN répétitives.

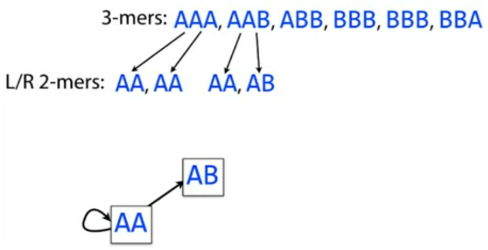
### 2.4.3 Algorithme basé sur le graph de De Bruijn

Une alternative au graphe de chevauchement est le graphe De Bruijn. Soit un génome $AAABBBBA$ et ses 3-mers : $AAA$, $AAB$, $ABB$, $BBB$, $BBB$, $BBA$. Le graphe sera construit comme suit : les 3-mers sont pris un à un afin d'en dériver le left et right 2-mers (L/R 2-mers). Pour chacun de ces 2-mers, un noeud est crée s'il n'existe pas déjà. Ensuite, un lien est établi du left 2-mers vers le right 2-mers. La figure 2.4.4 illustre cette construction, chaque itération correspond à l'analyse d'un 3-mers.

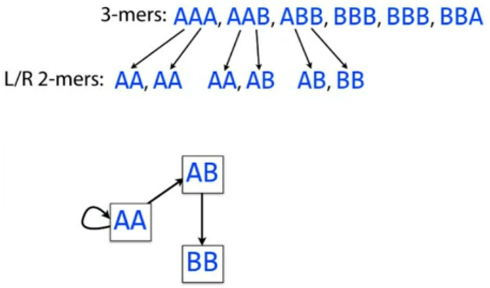
Dans ce type de graphe (voir figure 2.4.5), chaque lien correspond à un k-mer (3-mers dans ce cas) et chaque noeud à un k-1-mer distinct (2-mers dans ce cas). À partir de cela, le génome original peut facilement être reconstruit : il suffit de parcourir le graphe dans son entièreté en passant qu'une seule fois sur chaque bord. Il s'agit d'un *parcours Eulérien*. Cependant, cela peut arriver qu'il soit possible de parcourir un graphe dans son entièreté de plusieurs manières, autrement dit, il existe plusieurs chemins possibles.



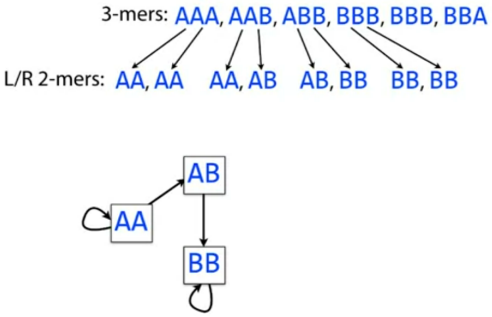

(c) 1ère itération



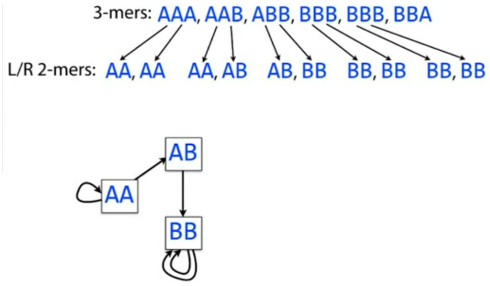

(d) 2ème itération



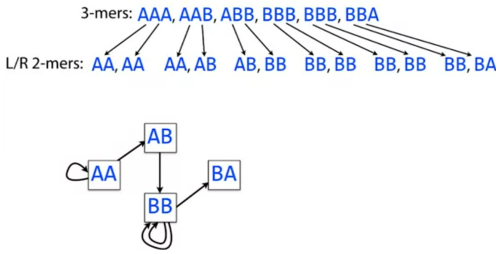

(e) 3ème itération




(f) 4ème itération




(g) 5ème itération




(h) 6ème itération

FIGURE 2.4.4 – Illustration des étapes lors de la construction d'un graph de De Bruijn

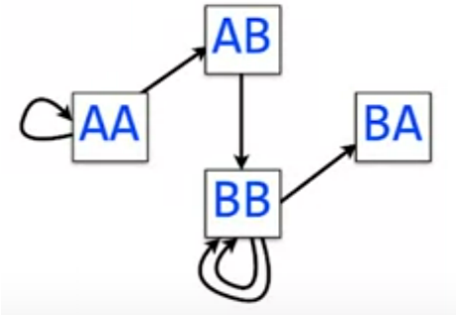



FIGURE 2.4.5 – Graphe de De Bruijn final après sa construction

## 2.5 Conclusion

Au terme de ce chapitre, il était question de prendre connaissance des algorithmes de base utilisés en bio-informatique, afin de pouvoir avoir une idée du fonctionnement des outils bio-informatiques. La vitesse du traitement, la performance de calcul sont très importantes pour l'analyse et le partage des ressources de calcul. Il en ressort qu'en bio-informatique lors d'une analyse de données de séquençage, deux principaux problèmes suscitent l'intérêt des informaticiens, il s'agit du problème de correspondance approximative et le problème d'alignement global. Le but n'étant pas de maîtriser ou de réinventer un algorithme, mais de comprendre le fonctionnement pour pouvoir adapter les outils et les utiliser.

# Deuxième partie

# Contributions

# Chapitre 3

# Mise au point d'un outil efficace pour la conception d'amorces

## 3.1 Analyse détaillée de la conception d'amorces

Ce chapitre se concentre sur la génération des données de séquençage. C'est une approche pré-traitement qui est en fait abordée. L'idée est d'améliorer la qualité des échantillons qui seront séquencés, grâce à la conception d'amorces qui délimitent la portion d'ADN à amplifier (qui sera le point d'entrée du séquençage), et donc la qualité des données qui seront produites par le pipeline. Ceci implique la mise en place d'un outil efficace, configurable grâce une interface web, de conception d'amorces qui inclut un traitement simultané de plusieurs fichiers, avec une vérification des amorces construites (dimers et hairpins).

### 3.1.1 Notion d'amorces

Souvent le séquençage ne peut être dissocié de la PCR (voir section 1.3) qui permet d'amplifier des séquences d'ADN de manière spécifique et d'augmenter de manière considérable la quantité d'ADN dont on dispose initialement. Autrement dit, c'est le processus qui crée un grand nombre de copies d'un fragment d'ADN. Cette réaction nécessite de connaître la séquence des régions qui délimitent l'ADN à amplifier. Ces séquences serviront à synthétiser des amorces (qui sont un brin court d'ADN ou d'ARN servant de point de départ à la synthèse de l'ADN) oligonucléotidiques complémentaires (de longueur de 18 à 30 nucléotides en général). Ces oligonucléotides serviront à délimiter la portion d'ADN à amplifier et ainsi à mieux cibler certaines zones lors du séquençage afin d'avoir une meilleure qualité pour celles-ci. L'ADN polymérase les utilisera comme amorces [28].

*Les amorces* sont donc des courtes séquences d'ADN simple brin utilisées dans la technique de réaction en chaîne par polymérase (PCR). Dans la méthode PCR, une paire d'amorces est utilisée pour s'hybrider avec l'échantillon d'ADN et définir la région de l'ADN qui sera amplifiée (l'amplicon) [54]. Deux types d'amorces sont utilisés dans la PCR et sont appelés amorces directes et inverses.

### 3.1.2 Fonctionnement des amorces dans la PCR

L'ADN a deux brins qui sont maintenus ensemble. Chaque paire de bases est complémentaire dans les deux brins. Chaque brin a sa propre direction. Par convention, un brin a une direction de 5' à 3' et est connu sous le nom de brin sens, tandis que l'autre a une direction de 3' à 5' et est connu sous le nom de brin antisens. Chaque brin doit être synthétisé individuellement lors de la PCR. Les trois étapes de la PCR sont :

— la dénaturation thermique de l'ADN à 95°C,

— l'hybridation des amorces à 50°C - 65°C,

— l'élongation à 72°C

La figure 3.1.1 illustre le rôle des amorces dans la PCR. Lors de la dénaturation (1), les deux brins d'ADN sont séparés en rompant les liaisons hydrogène en les chauffant à 95°C. L'amorce directe se lie au brin sens tandis que l'amorce inverse se lie au brin antisens. À l'étape d'hybridation (2), le mélange réactionnel est refroidi rapidement (température de 45 à 60°C), ce qui permet l'appariement des amorces à leurs régions complémentaires dans les brins d'ADN. Dans l'étape d'élongation, la taq polymérase (ADN polymérase) catalyse la réplication à partir des ADN mono-caténaires amorcés de façon sélective (sélectivité qui découle du choix des amorces). Après l'élongation (3), le cycle recommence, à la différence que les fragments précédemment synthétisés servent de matrice pour la synthèse de nouveaux fragments d'ADN [86]. Au final l'ADN se retrouve multiplié. La PCR prendre donc l'ADN et les amorces en entrée et en ressortira un ADN multiplié.

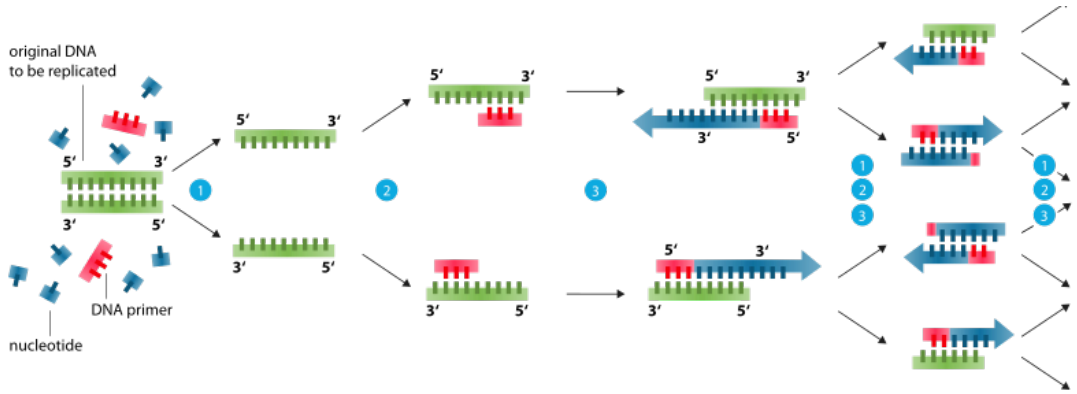



FIGURE 3.1.1 – Cycle de PCR et rôle des amorces [17] avec la représentation des phases de dénaturation (1) qui permet de séparer les brins d'ADN, d'hybridation (2) où l'amorce vient s'accrocher sur le brin d'ADN complémentaire, d'élongation (3) où l'ADN se multiplie

### 3.1.3 Quelques outils existants de conception d'amorces

La qualité des primers conçus est très importante pour l'étape de séquençage d'ADN ou ARN. Et pour obtenir des primers de bonne qualité, il faudra miser sur des outils de qualité, fiables, précis et performants. Aujourd'hui, avec l'évolution de la médecine génomique, il existe beaucoup d'outils de conception d'amorces tels que :

— *Primer3* : qui est un programme (codé en C) largement utilisé pour la conception d'amorces. Aujourd'hui, primer3 fait partie d'une library *BioStars* qui est une library python pour la Bio-informatique. Primer3 détient également une version web ( *Primer3Web et Primer3Plus*).

— *PrimalScheme* : qui est un outil de conception de panels d'amorces pour la PCR multiplex [63]. PrimalScheme utilise le logiciel *Primer3* pour générer des paires d'amorces candidates (cinq, par défaut) [68]. Il sélectionne les amorces en fonction de la modélisation thermodynamique, qui prend en compte la longueur, la température de recuit, le %GC, la stabilité en 3', la structure secondaire estimée et la probabilité de formation d'amorces-dimères, maximisant ainsi les chances de réussite de la réaction PCR [60].

— *PrimerBLAST* : qui est une combinaison d'un programme appelé Primer3 qui aide à la conception d'amorces avec des propriétés spécifiques et BLAST [1].

— *ThermoFisher* : outils Primer Designer pour PCR et séquençage Sanger [64].

### 3.1.4 Outil de conception d'amorces propre au CHU UCL Namur

Dans le cadre d'un projet de typage érythrocytaire, il serait pertinent pour le CHU UCL Namur de mettre au point un outil efficace pour la conception d'amorces spécifiques aux cibles sélectionnées. Les gros problèmes, que de tels outils peuvent rencontrer, sont la taille des cibles qui peuvent être fort différentes. Il y a également l'approche par "PCR tilling sequencing" qui implique la génération d'amplicons de plus petites tailles et partiellement recouvrants. Enfin l'interaction entre amorces telle que la dimérisation qui engendre une perte de rendement d'amplification ou bien de l'aspécificité. Les outils en ligne existants offrent parfois trop peu ou pas assez de paramètres. De plus les laborantins doivent se contenter des résultats obtenus, sans possibilité d'ajouter de nouvelles informations pertinentes dans ceux-ci.
Pour réaliser ce premier outil, l'exigence du CHU est de combiner deux outils de bio-informatiques de conception d'amorces existants :

— primalscheme [1], qui a pour fonctionnalités :
  - le chaînage automatique des amplicons pour couvrir entièrement le gène ciblé
  - le choix de la longueur maximum et minimum de recouvrement entre deux amplicons

— primer3 [2], qui a pour paramètres :
  - select species(=human genome), l'espèce d'origine (il est important pour identifier sur le gène cible les régions qui sont uniques ou peu représentées dans le reste du génome)
  - la taille de amorces (minimale, optimale, maximale)
  - la température d'annealing des amorces (minimum, optimal, maximum)
  - l'intervalle de la taille des produits
  - le nombre de paires d'amorces à indiquer

1. https ://github.com/aresti/primalscheme
2. https ://github.com/primer3-org

## 3.2 Description détaillée de l'outil de conception d'amorces développé

### 3.2.1 Rétro-ingénierie de primer3 et primalscheme

Commençons par primalscheme qui est choisi comme premier outil à analyser. Une fois celui-ci téléchargé de Github, le programme peut être ouvert avec Pycharm. Cet IDE permet de générer automatiquement le diagramme de classe du projet (disponible à l'annexe C). Celui-ci est utile pour une compréhension des différentes classes ainsi que de la structure globale et permet d'avoir une vue rapide des différentes méthodes et attributs contenus dans chacune.

Primalscheme possède quatres classes pincipales :

1. *MultiplexScheme* qui offre un schéma global d'amorce multiplex avec pour principales fonctions et attributs :
   - *design_scheme(self)* : instancie des régions avant d'en calculer des amorces
   - *primers(self)* : renvoie la liste de toutes les amorces
   - *amplicon_size_min* et *amplicon_size_max* : la taille minimale et maximale des amplicons [1]
   - *target_overlap* : le nombre de bases qui doivent être superposées entre deux amplicons
2. *MultiplexReporter* qui étend la classe *MultiplexScheme* en offrant la gestion des fichiers de report de sortie. Ces méthodes et attributs principaux sont :
   - *write_primer_tsv(self)* : écrit le fichier .tsv
   - *write_run_report_json(self)* : écrit le fichier json des logs de l'exécution
   - *outpath* : le chemin du répertoire où stocker les sorties du programme
3. *Primer* qui représente une amorce avec pour fonctions et attributs principaux :
   - *design_primers(self)* : modélise les amorces
   - *_mismatch_penalties_for_ref(self, ref)* : calcule les pénalités d'inadéquation pour chaque position par rapport à la référence (ref)
   - *seq(self, seq)* qui définit la séquence et calcule ses caractéristiques grâce notamment aux fonctions suivantes (qui sont hors de la classe) :
     - *calc_gc(seq)* : calcule le pourcentage de G et C de la séquence (seq).
     - *calc_tm(seq)* : calcule la température de fusion pour une séquence (seq)
   - *seq* : la séquence de bases de l'amorce
   - *start* : la position de la référence où la première base de l'amorce s'accrochera (en théorie)
   - *direction* : si l'amorce sera sur le brin sens ou anti-sens
4. *Region* qui représente la région d'un amplicon. Elle a pour méthode princaple :

1. un amplicon est un fragment d'ADN amplifier par PCR

— *find_primers(self)* : trouve les amorces pour la région

Il est possible de lancer une exécution de primalscheme à l'aide de l'instruction terminal suivante :

```
~/PycharmProjects/primalscheme$ primalscheme multiplex ./tests/inputs/nCov-2019.fasta --force
```

Le programme primalscheme démarre sur une instruction "multiplex". Il peut, à présent, être intéressant de se plonger dans le code de ce programme. Le fichier cli.py (pour commande-line interface) est le point d'entrée du programme. En voici un extrait :

```
@cli.command()
@click.argument("fasta", type=click.Path(exists=True, dir_okay=False)
    )
@click.option(
    "--amplicon-size",
    "-a",
    multiple=True,
    type=click.IntRange(90),
    help=(
        "Amplicon size target. Pass twice to set an exact range,
    otherwise expect "
        f"+/- {config.SIZE_RANGE_AUTO/2 * 100}%."
    ),
    metavar="<int>",
    default=[config.AMPLICON_SIZE_MIN, config.AMPLICON_SIZE_MAX],
    show_default=True,
)
@click.option(
...
```

Ces quelques lignes illustrent la gestion de l'instruction tapée en ligne de commande, avec le fichier fasta comme argument principal ainsi que d'autres arguments optionnels. La fonction "multiplex()" sert de point de départ pour les appels de fonctions à travers le programme.

En suivant les appels successifs de fonctions à partir du fichier cli.py ainsi qu'en exécutant le programme en mode "débogage", une exécution normale en ligne de commande pour un fichier fasta peut être représentée par le schéma 3.2.1.

Dans ce schéma, l'exécution, déclenchée en terminal de commande, est gérée par le fichier cli.py (commande-line interface). Pour toute flèche, partant de A vers B, portant comme étiquette l'appel d'une fonction myfunction() : A est le fichier qui contient la méthode qui invoque la fonction myfunction() et B est le fichier qui la contient. De manière plus détaillée :

(1) La méthode principale, multiplex(), de ce fichier va instancier un *objet MultiplexReporter dans une variable scheme*

(2) Cette classe MultiplexReporter étant une sous-classe de Multiplex contenue dans multiplex.py va instancier celle-ci

(3) Multiplex() va alors invoquer designscheme() de MultiplexReporter() sur son instance créee précédemment

(4) Cette méthode designschem() contenue dans multiplex.py va instancier la classe Region() contenue dans region.py afin de calculer les regions de la séquence passée en paramètre de l'instruction en ligne de commande

(5) Elle va ensuite invoquer findprimers() de l'objet crée qui va calculer les primers des différentes régions

(6) Enfin, multiplex() va appeler write_default_outputs() de l'objet MultiplexReporter() afin de générer les différents rapports utilisateurs

(7) Cette méthode va appeler les différentes fonctions propres à chaque type de fichiers d'output

Durant l'exécution du programme, le fichier *config.py* est consulté régulièrement pour récupérer des constantes.

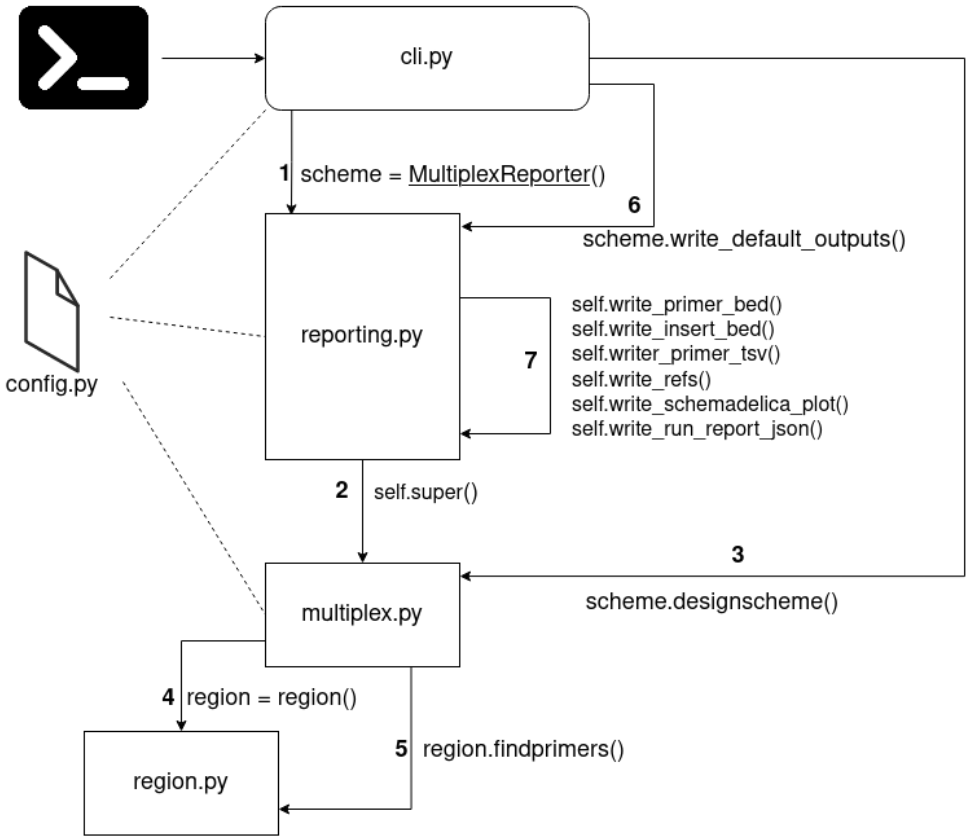



FIGURE 3.2.1 – Schéma représentant les fichiers et classes principales agissant lors d'une exécution de primalscheme

Lorsque les dépendances du projet sont explorées, primer3 apparaît dans la liste. Il semble que primalscheme soit construit sur base de primer3.

Si de premier abord, primalscheme semble être compliqué à comprendre, après une analyse rigoureuse, l'outil devient facilement compréhensible. L'imbrication des différentes classes et fonctions devient logique. De plus, étant donné que l'outil primalscheme est basé sur primer3, de nombreuses fonctionnalités offertes par primer3 sont déjà intégrées dans celui-ci. De nombreuses nouvelles fonctionnalités, telles que une paramétrisation dynamique (détaillée au point 3.2.2), peuvent donc directement être apportées à primalscheme et semblent être suffisantes pour répondre aux exigences du CHU.

### 3.2.2 Paramétrisation dynamique

Une des exigences pour ce premier outil à développer est d'avoir une flexibilité au niveau des paramètres pour la conception d'amorces. Des paramètres intéressants à pouvoir entrer par le CHU lors de son utilsation sont :

— la taille minimale et maximale des amplicons

— la taille minimal, maximale et optimale des primers

— la température de fusion (Tm) minimale, maximale et optimale des primers

— la taille du chevauchement

À ce stade, la seule façon d'exécuter primalscheme est d'utiliser une instruction en ligne de commande telle que celle ci-dessous avec une possibilité d'entrer quelques paramètres optionnels : "amplicon-size", "outpath", "name", "debug", etc. Ceux-ci restent limités par rapport aux exigences attendues du nouvel outil.

```
~/PycharmProjects/primalscheme$ primalscheme multiplex ./tests/inputs/nCov-2019.fasta --force
```

FIGURE 3.2.2 – Instruction en ligne de commande pour exécuter primalscheme

Afin d'intégrer les paramètres, il pourrait être intéressant d'ajouter des arguments dans cette ligne de commande. Cependant, vu que ceux listés ci-dessus sont déjà de 8 et que, potentiellement, de nouveaux pourraient s'ajouter, développer une interface utilisateur semble plus pertinent. D'autant plus que les utilisateurs amenés à l'utiliser sont des scientifiques qui peuvent provenir de différents domaines et ne pas nécessairement avoir de grandes notions d'informatique. L'interface semble plus adaptée pour ces profils du fait qu'elle offre une abstraction du terminal et un meilleur confort visuel.

Dans la figure 3.2.3, les modifications peuvent être apportées sur les éléments suivants (en vert) :

— un remplacement du terminal par une interface utilisateur

— une modification du fichier cli.py qui gérait l'instruction ligne de commande afin de considérer les paramètres utilisateurs de l'interface

— une modification du fichier config.py en fonction des paramètres modifiés.

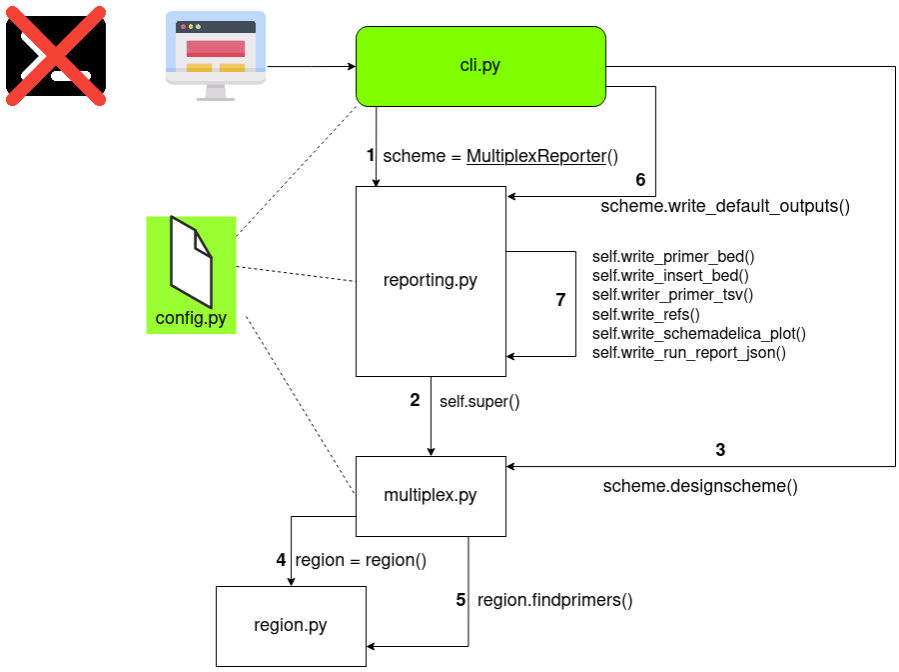



FIGURE 3.2.3 – Schéma représentant les fichiers et classes principales agissant lors d'une exécution de primalscheme et ceux où apporter des modifications pour une paramétrisation dynamique

De manière plus détaillée, chaque élément modifié peut être vu un à un. Pour commencer par l'interface utilisateur, les paramètres requis par le CHU ont été pris en compte. Un premier prototype (figure 3.2.4) a été réalisé grâce à l'outil Draw.io et a été soumis à l'équipe du CHU afin d'obtenir un retour sur leur expérience utilisateur et les modifications à apporter.

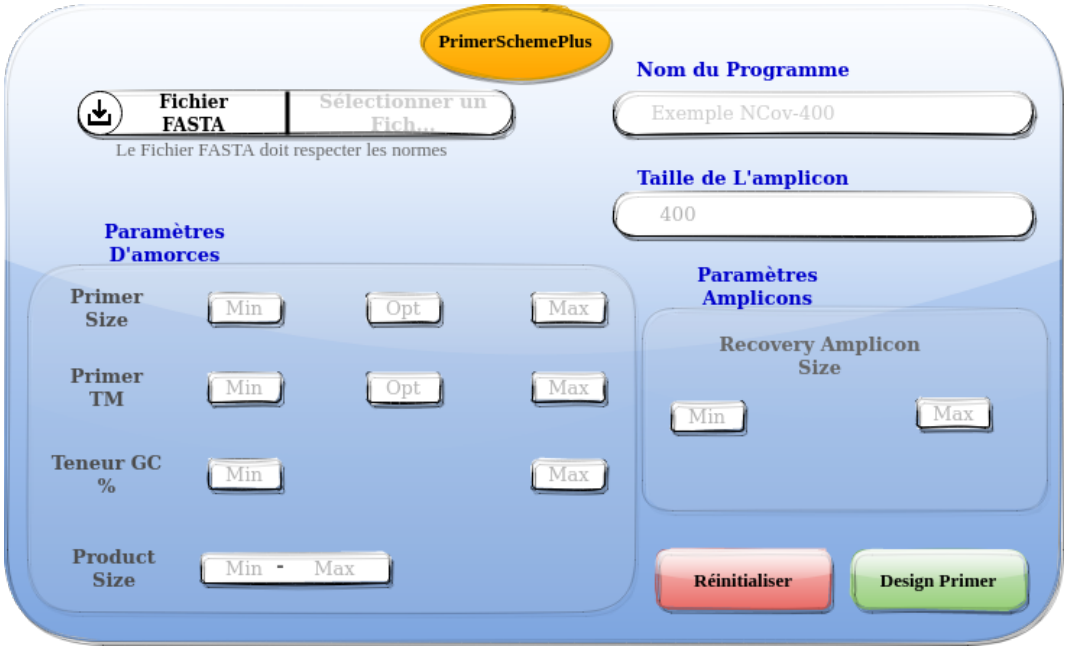



FIGURE 3.2.4 – Prototype de l'interface de l'outil de conception d'amorces

Une fois cette expérience utilisateur menée sur le prototype, et corrections apportées, l'interface finale a été développée sur base des technologies suivantes :

— HTML5 : celle-ci apporte un niveau structurel amélioré par rapport au simple HTML et une possibilité d'unification de ce qui peut aujourd'hui être utilisé pour réaliser des sites web, à savoir le HTML, les feuilles de style (CSS) pour la partie graphique, et le JavaScript pour l'interactivité.

— CSS pour le style

— Bootstrap5 : d'abord pour éviter un perte de temps au niveau de la conception d'un objet, pour retrouver une structure HTML standardisée tout en permettant d'éviter certaines erreurs.Ensuite, pour apporter un visuel plus attrayant aux utilisateurs.

— JavaScript et jQuery pour l'interactivité.

La figure 3.2.5 représente l'interface finale obtenue. Elle comporte 3 zones principales : la première pour la sélection des références comme fichiers d'entrée, une liste déroulante pour le choix du fichier consensus, un choix sur le type d'analyse (normal ou high GC) et le nom du programme (qui est le nom que porte le dossier qui va contenir les résultats). La seconde zone est destinée à la paramétrisation des amorces tandis que la troisième zone est pour la paramétrisation des amplicons.

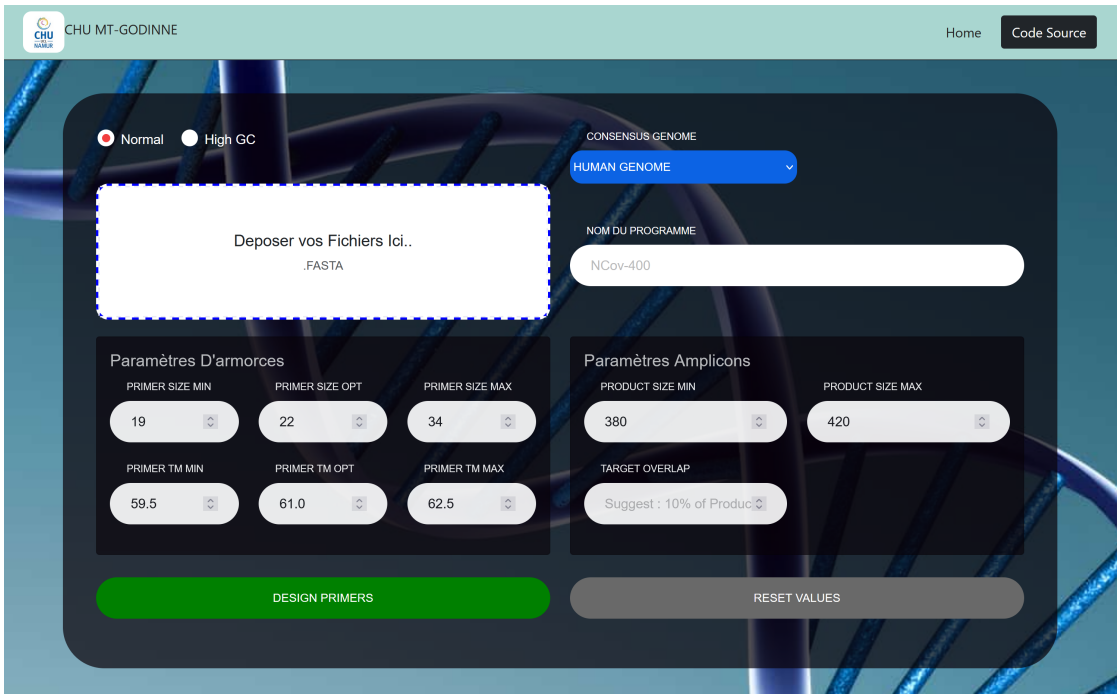



FIGURE 3.2.5 – Interface prenant en compte de nombreux paramètres pour la conception d'amorces

En plus de prendre les paramètres listés au début de la section, l'interface considère le fichier fasta pour lequel les amorces vont être désignées ainsi qu'un génome consensus (humain, bactérien, etc). L'utilité de ce dernier sera détaillée dans la section 3.2.3 pour le calcul de k-mers. Afin de pouvoir intégrer ces paramètres dans l'exécution de primalscheme, il faut modifier le fichier config.py dont en voici un extrait :

```
PrimerSizeRange = namedtuple("PrimerSizeRange", "min max opt")
PrimerGCRange = namedtuple("GCRange", "min max opt")

PRIMER_SIZE_RANGES = {
    "DEFAULT": PrimerSizeRange(19, 34, 22),
    "HIGH_GC": PrimerSizeRange(19, 34, 22),
}

PRIMER_GC_RANGES = {
    "DEFAULT": PrimerSizeRange(30, 55, 50),
    "HIGH_GC": PrimerSizeRange(40, 65, 60),
}
..
AMPLICON_SIZE_MIN = 3000
AMPLICON_SIZE_MAX = 3200
..
```

Afin de coordonner le tout, une implémentation selon une approche client-serveur a été optée. L'interface utilisateur n'est pas directement reliée au fichier cli.py. C'est le serveur qui communiquera directement avec celui-ci et qui ira modifier le fichier config.

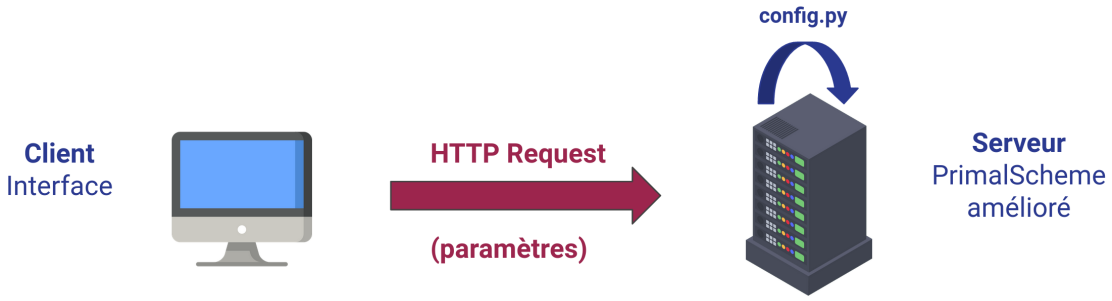



FIGURE 3.2.6 – Modèle client-serveur adapté au système de primalscheme pour récupérer les paramètres utilisateurs

Le client est l'interface WEB illustrée à la figure 3.2.5 . Lorsque l'utilisateur valide le formulaire, les paramètres sont envoyés à travers une requête HTTP de type POST. Le serveur les reçoit, modifie le fichier config.py avant de lancer l'exécution de primalscheme en invoquant la fonction multiplex() du fichier cli.py. Les outputs sont directement stockés dans un dossier local sur la machine sur laquelle est déployé le serveur.

### 3.2.3 Calcul de k-mers

Lorsque primalscheme est exécuté sur un fichier .fasta, il génère un ensemble de fichiers de sortie dont un ayant une extension .tsv tels que celui illustré dans le tableau 3.1. Celui-ci reprend chaque amorce par ligne avec son nom, son pool, sa séquence, sa taille, son taux de bases G et C, ainsi que sa température de fusion (tm).

Comme il avait déjà été abordé plus haut, un brin ADN est composé de deux hélices : le brin sens et le brin anti-sens (qui n'est autre que le complément inverse du brin sens).

| name | pool | seq | size | %gc | tm (use 65) |
|---|---|---|---|---|---|
| FUT1_seq_1_LEFT | 1 | GATGTAGTGGTTCTGGGAGTTCAG | 24 | 50.00 | 60.94 |
| FUT1_seq_1_RIGHT | 1 | GAGTGATTCTAGTTGGGAGAGTTGG | 25 | 48.00 | 61.01 |
| FUT1_seq_2_LEFT | 2 | CGATGAAACCCCGTCTCTACTAAAA | 25 | 44.00 | 61.00 |
| FUT1_seq_2_RIGHT | 2 | CCTTTAACTTCCTCTTTCCCTGGG | 24 | 50.00 | 61.14 |
| FUT1_seq_3_LEFT | 1 | CTTCAGGAAAGGATCTCTCAAGTCC | 25 | 48.00 | 61.01 |
| FUT1_seq_3_RIGHT | 1 | GACCCTAGAATTCCCCAAAAGGAG | 24 | 50.00 | 60.95 |
| FUT1_seq_4_LEFT | 2 | GTAAGTTCCGCTGATACTGGACTC | 24 | 50.00 | 61.05 |
| FUT1_seq_4_RIGHT | 2 | GGGACATTGTGAGAAGTGACCTAG | 24 | 50.00 | 60.94 |

TABLE 3.1 – Contenu du fichier CHU_FUT1_seq.primer.tsv produit par une exécution de primalscheme sur un fichier CHU_FUT1_seq.fasta avec des paramètres quelconques

Pour chaque amorce générée, il serait donc intéressant de connaître le nombre de fois qu'elle est présente dans le génome consensus (elle s'attachera sur l'autre brin, le brin "anti-sens") ainsi que le nombre de fois où son complément inverse y est présent (elle s'attachera au brin "sens"). L'intérêt d'obtenir ce nombre permet d'avoir une idée du risque que l'amorce s'attache à un autre endroit que la zone ciblée.

L'idée est donc d'afficher ce chiffre dans le fichier .tsv et donc d'y créer une ou plusieurs nouvelles colonnes (une par k-mers pour chaque amorces). Cela implique la création d'un nouveau module propre au calcul d'occurrences des k-mers d'amorces dans un génome consensus. Celui-ci pourra être appelé directement dans la fonction write_primer_tsv() présente dans le fichier reporting.py (et donc avant la création de ce fichier .tsv). La figure 3.2.7 illustre les éléments qui devront être modifiés (en vert).

#### 3.2.3.1 Pré-traitement du génome consensus

**A. Solution intuitive**

Afin de trouver ce nombre d'occurrences, une première idée fut de construire un dictionnaire où chaque clé est un k-mer (la valeur k est la taille de l'amorce la plus petite) du génome consensus et sa valeur le nombre de fois où il apparaît dans celui-ci. Ensuite pour chaque amorce et son complément inverse, il aurait fallu regarder s'ils se trouvaient dans le dictionnaire et si tel aurait été le cas, retourner la valeur correspondante. En supposant le génome suivant de longueur 11 :

{ACGTACGTCCT}

et ses 4-mers :

['ACGT', 'CGTA', 'GTAC', 'TACG', 'ACGT', 'CGTC', 'GTCC', 'TCCT']

le dictionnaire correspondant serait alors celui illustré dans le tableau 3.2.

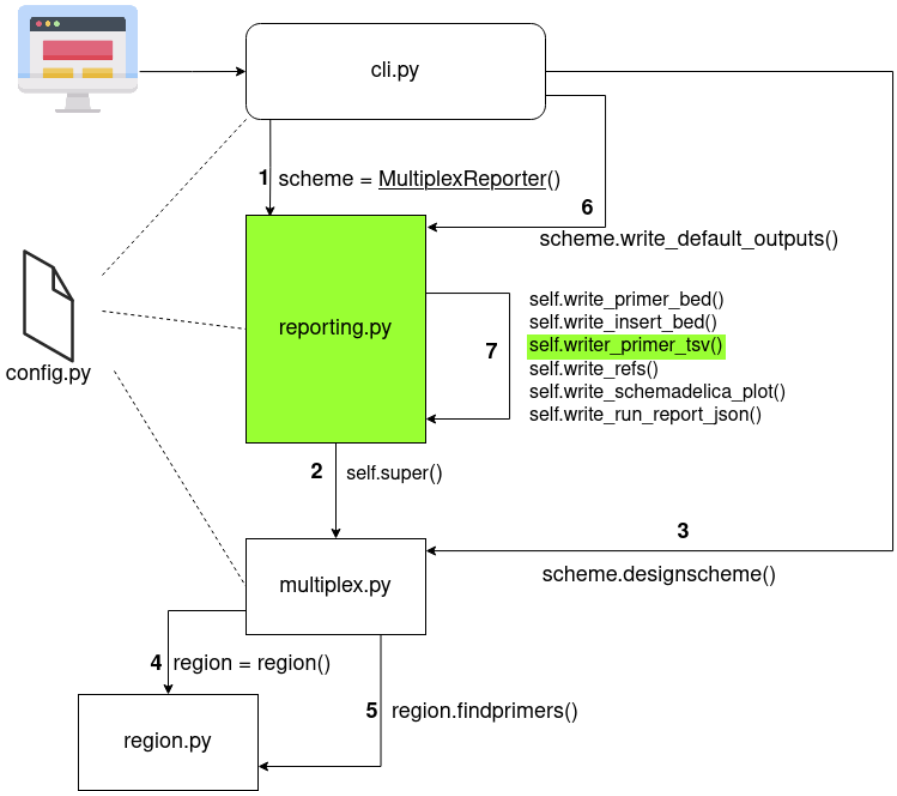



FIGURE 3.2.7 – Schéma représentant les fichiers et classes principales agissant lors d'une exécution de primalscheme et les endroits où apporter les modifications afin d'intégrer une colonne de k-mers dans le fichier .tsv

| 4-mers | nombre d'occurence |
|---|---|
| 'ACGT' | 2 |
| 'CGTA' | 1 |
| 'GTAC' | 1 |
| 'TACG' | 1 |
| 'CGTC' | 1 |
| 'GTCC' | 1 |
| 'TCCT' | 1 |

TABLE 3.2 – Dictionnaire des 4-mers du génome ACGTACGTCC

Sachant que l'alphabet considéré (celui de l'ADN) est de taille 4 (A,C,G,T) et connaissant la taille 11 du génome, il peut être déduit que pour ce génome :

— le nombre possible de 4-mers différents est de $4^4$

— le nombre de 4-mers présents dans le génome est de $11 - 4 + 1$

De manière plus générale pour tous *k*-mers sur un génome de longueur *l* dont l'alphabet est ACGC :

— le nombre possible de k-mers différents est de $4^k$

— le nombre de k-mers présents dans le génome de longueur l est de $l - k + 1$

Ces deux informations représentent un problème de complexité en mémoire. Tout d'abord, le génome humain (qui mesure plus de 3 200 000 000 de bases) implique qu'il y aurait $\approx$ *3 200 000 000 de 22-mers à regarder* (22 est la taille normale pour une amorce dans la pratique). En fonction de la taille, cette information croît de manière linéaire. Cependant, en supposant un 22-mers, il y aurait pas loin de $4^{22}$ 22-mers différents soit $\approx$ *17 500 000 000 000 de 22-mers uniques*. Ici, il y a une relation exponentielle entre k et le nombre de k-mers différents possibles. Cela signifie que tous les 22-mers lus dans le génome humain peuvent possiblement être différents. Construire un dictionnaire de cette taille serait pratiquement impossible sans un superordinateur. Pour preuve, un ordinateur de 134 GB de RAM sur lequel les déploiements et tests de cette solution naïve étaient effectués a vu sa mémoire saturer au bout de quelques minutes avec un tel algorithme :

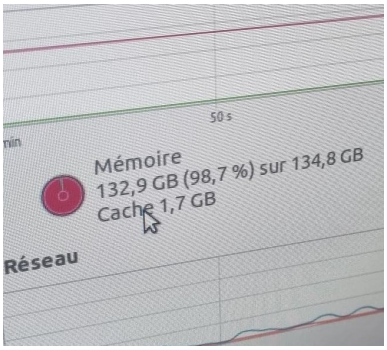



FIGURE 3.2.8 – Mémoire saturée par un algorithme stockant tous les k-mers du génome humain

### B. Opérations des expressions régulières

Construire un dictionnaire dans lequel chaque clé serait un k-mers unique prend beaucoup de temps et surtout une grande ressource en termes de mémoire. Une méthode bien plus rapide est d'utiliser une fonction existante du module re (Regular Expression Operations) : *findall()*. Celle-ci renvoie toutes les correspondances sans chevauchement du motif dans la chaîne, sous la forme d'une liste de chaînes ou de tuples.

Le seul inconvénient de cette fonction est qu'elle considère les occurrences sans chevauchement d'un pattern dans un string. Une façon de contourner ce souci est de l'invoquer de la manière suivante :

re.findall(f'(?={$k\_mer$})', consensus_content))

où *k-mer* est le k-mer dont l'occurrence est cherchée et *consensus_content* le contenu du génome consensus. Il est donc plus intéressant de parcourir à chaque fois l'intégralité du génome consensus pour toutes les amorces plutôt que de construire un dictionnaire de k-mers du génome consensus. C'est la taille de l'amorce la plus petite qui devient le k du k-mers. Plus le k-mers est de petite taille, plus il y a de chance d'en trouver des occurrences dans le génome consensus. Il est donc intéressant de calculer les occurrences de k-mers pour plusieurs valeurs de k qui diminuent. En supposant que les 22-mers, 19-mers,

16-mers, 13-mers et 10-mers soient calculés de manière séquentielle, cela implique le workflow suivant :


FIGURE 3.2.9 – Calculs de multiples k-mers de manière séquentielle

Pour obtenir le temps total d'exécution, il suffit de faire la somme des temps des différentes exécutions. Pour un calcul de 22-mers, 19-mers, 16-mers, 13-mers, 10-mers sur un génome bactérien, le temps est de :

$$11{,}14\ s + 4{,}26\ s + 7{,}18\ s + 10{,}6\ s + 12{,}54\ s \approx 36{,}2\ s$$

et sur un génome humain, ce temps est de :

$$14\ m\ 35\ s + 61\ m\ 48\ s + 101\ m\ 11\ s + 144\ m\ 45\ s + 188\ m\ 36\ s \approx 8\ heures\ 29\ min\ 75\ s$$

En supposant maintenant une exécution concurrente de ces calculs de k-mers, le workflow deviendrait :

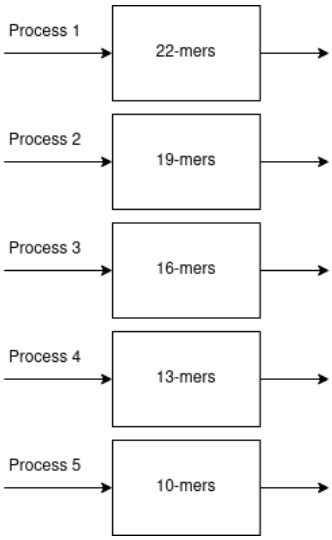



FIGURE 3.2.10 – Calcul de multiples k-mers de manière concurrente

Pour obtenir le temps total d'exécution, il suffit de prendre le maximum des temps des différentes exécutions. Pour un calcul de 22-mers, 19-mers, 16-mers, 13-mers, 10-mers sur un génome bactérien, le temps sera de :

$$max(1.4\ s, 4.55\ s, 7.79\ s, 12.2, 15.52\ s) \approx 18\ s$$

et sur un génome humain, ce temps sera de :

$$max(14m\ 22s, 63m\ 97s, 104m\ 9s, 149m\ 79s, 194m\ 33s) \approx 3\ heures\ 18\ min$$

Pour le génome humain, le gain de temps est de plus de 5 heures ce qui est non négligeable.

De manière plus visuelle, la figure 3.2.11 montre la comparaison du temps d'exécution total entre le singleprocessing et multiprocessing pour les exécutions de 22-mers, 19-mers, 16-mers, 13-mers et 10-mers sur le génome humain.

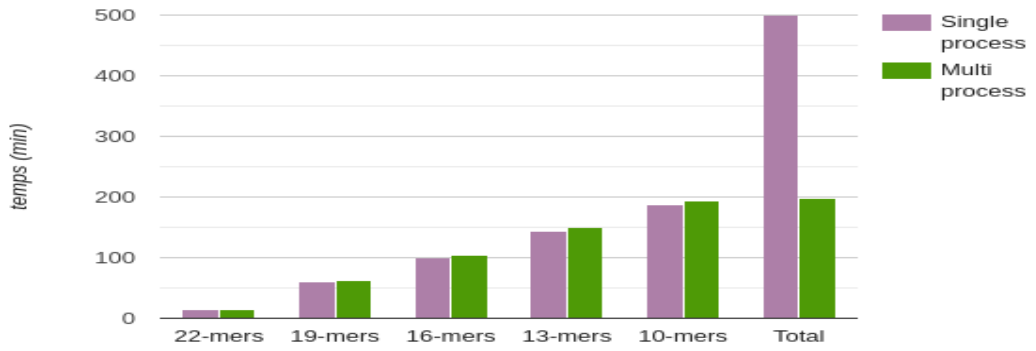



FIGURE 3.2.11 – Comparaison du temps d'exécution total entre le single processing et multiprocessing pour le calcul des 22-mers, 19-mers, 16-mers, 13-mers et 10-mers du génome humain

Bien évidemment dans la pratique le calcul ne se fera pas pour autant de k-mers. Le plus pertinent est de calculer cette valeur d'occurrence pour des 3 k-mers avec k qui commence avec pour valeur la taille de l'amorce la plus petite et qui diminue de 1. Un fichier .tsv généré sera alors :

| ... | seq | size | %gc | tm(use 65) | 24-mers | 23-mers | 22-mers |
|---|---|---|---|---|---|---|---|
| ... | GATGTAGTGGTTCTGGGAGTTCAG | 24 | 50.00 | 60.94 | 1 | 2 | 3 |
| ... | GAGTGATTCTAGTTGGGAGAGTTGG | 25 | 48.00 | 61.01 | 2 | 3 | 4 |
| ... | CGATGAAACCCCGTCTCTACTAAAA | 25 | 44.00 | 61.00 | 0 | 0 | 0 |
| ... | CCTTTAACTTCCTCTTTCCCTGGG | 24 | 50.00 | 61.14 | 1 | 2 | 3 |
| ... | CTTCAGGAAAGGATCTCTCAAGTCC | 25 | 48.00 | 61.01 | 2 | 3 | 4 |
| ... | GACCCTAGAATTCCCCAAAAGGAG | 24 | 50.00 | 60.95 | 1 | 2 | 3 |
| ... | GTAAGTTCCGCTGATACTGGACTC | 24 | 50.00 | 61.05 | 1 | 2 | 3 |
| ... | GGGACATTGTGAGAAGTGACCTAG | 24 | 50.00 | 60.94 | 0 | 0 | 0 |

TABLE 3.3 – Contenu du fichier CHU_FUT1_seq.primer.tsv produit par une exécution de primalscheme sur un fichier CHU_FUT1_seq.fasta avec des paramètres quelconques et calculant les occurrences de chacune des amorces pour différents k-mers du génome consensus

### 3.2.4 Multiprocessing

La nouvelle interface offre la possibilité à l'utilisateur de pouvoir sélectionner directement plusieurs fichiers fasta en entrée afin de lancer primalscheme sur chacun d'eux. Une première implémentation séquentielle ayant été testée pour les exemples de fichiers .fasta (fournit par le CHU) : GYPA.fasta, GYPE.fasta, RHD.fasta implique le workflow suivant :

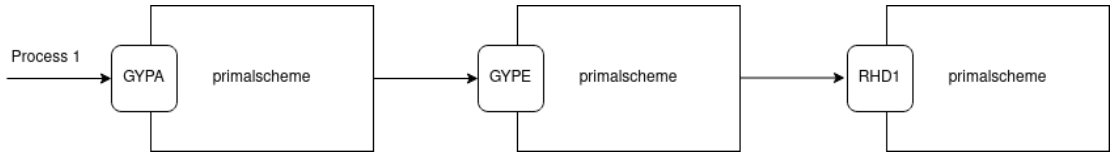



FIGURE 3.2.12 – Exécution de primalscheme pour les gènes GYPA, GYPE, RHD en séquentielle

Primalscheme est donc d'abord exécuté sur le fichier fasta Gypa, ensuite sur le fasta Gype, enfin sur le fasta RHD. Le temps d'exécution total est donc la somme des durées des trois exécutions. Lorsque le génome consensus est celui d'une bactérie, les temps des trois exécutions ainsi que le total sont respectivement :

$$7\ sec + 8\ sec + 15\ sec \approx 30\ sec$$

Lorsque le génome consensus considéré est humain, ces temps deviennent, pour les mêmes fichiers en entrée et les mêmes paramètres, respectivement :

$$30min\ 42s + 38min\ 20s + 1h\ 19min\ 54s \approx 2h\ 28min$$

Cette grande différence de temps d'exécution peut s'expliquer par la différence de taille entre les génomes consensus (celui de l'humain est 1500x plus grand que celui de la bactérie). Les données peuvent être les gènes ou bien le génome consensus. Sur base du résultats des exécutions précédentes, il peut être intéressant de se pencher sur l'évolution de ce temps d'exécution en fonction de la taille de ces différents fichiers.

### A. Croissance de la taille des gènes

La figure 3.2.13 représente le temps d'exécution de primalscheme en fonction de la taille des gènes (repris dans l'annexe D) pour le même génome consensus d'une bactérie et les mêmes paramètres. D'après le graphique, il s'agit d'une relation linéaire entre le temps d'exécution et la taille du gène.

### B. Croissance de la taille des génomes de référence

La figure 3.2.14 représente le temps d'exécution de primalscheme en fonction de la taille des génomes de références (repris dans l'annexe E) pour le même gène RHD et les mêmes paramètres. D'après le graphique, le temps de calcul du programme primalscheme tend à croître de manière linéaire en fonction de la taille du génome consensus.

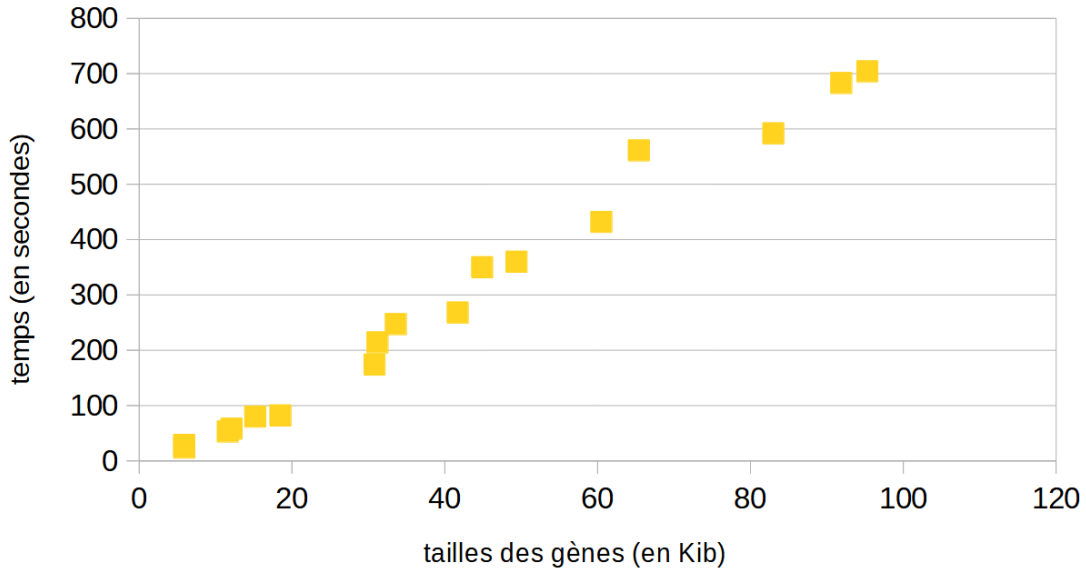



FIGURE 3.2.13 – Évolution du temps de calcul de primalscheme en fonction de la taille des gènes

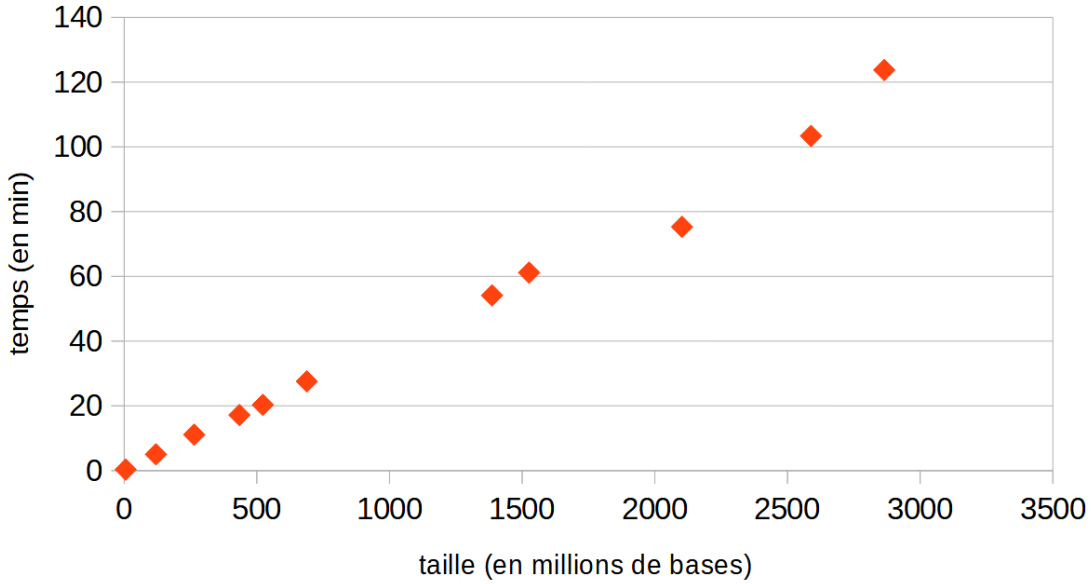



FIGURE 3.2.14 – Évolution du temps de calcul de primalscheme en fonction de la taille du génome consensus

Plus la taille des données manipulées augmente, plus l'exécution de primalscheme prendra du temps. Cela sera d'autant plus vrai pour le temps d'exécution total lorsque l'utilisateur sélectionnera plusieurs grands fichiers en entrée et que l'exécution se fera en séquentielle. Une façon de gagner en temps de traitement est d'introduire du multiprocessing afin de permettre la programmation concurrente. Une deuxième implémentation utilisant du multiprocessing sur les mêmes fichiers d'inputs (GYPA.fasta, GYPE.fasta, RHD.fasta) implique le workflow suivant :

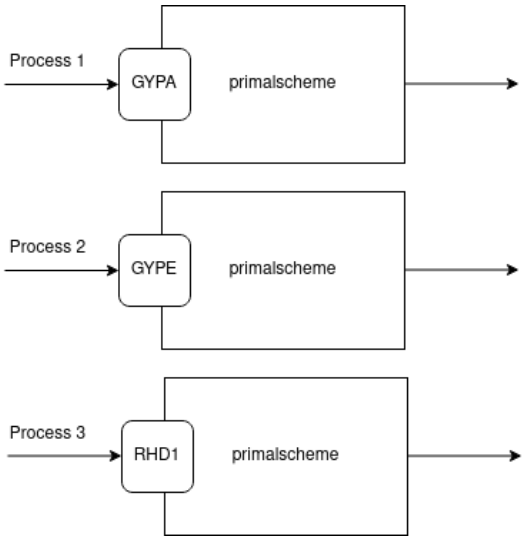



FIGURE 3.2.15 – Exécution de primalscheme sur les gènes de tests GYPA, RHD, FUT1 en utilisant des processus concurrents

Ici l'exécution de primalscheme est faite en parallèle sur les différents fichiers d'entrée. D'après les observations faites, le temps total d'exécution n'est plus la somme des trois temps d'exécution, mais correspond approximativement au maximum de ces trois temps (il faut cependant compter un temps de coordination du multiprocessing). Lorsque le génome consensus est celui d'une bactérie, le temps total d'exécution devient :

$$max(7\ sec,\ 8\ sec,\ 15\ sec) \approx 15\ sec$$

Lorsque le génome consensus considéré est humain, le temps total d'exécution devient, pour les mêmes fichiers en entrée et les mêmes paramètres :

$$max(30min\ 42s,\ 38min\ 20s,\ 1h\ 19min\ 54s) \approx 1h\ 20min$$

Le temps total est approximativement le maximum des temps d'exécution (il y a en effet un temps de coordination à prendre en compte qui est minime). De manière plus visuelle, le schéma 3.2.16 montre la comparaison du temps d'exécution total entre le singleprocessing et multiprocessing pour les 3 fichiers tests : Gypa.fasta, Gype.fasta, RHD.fasta.

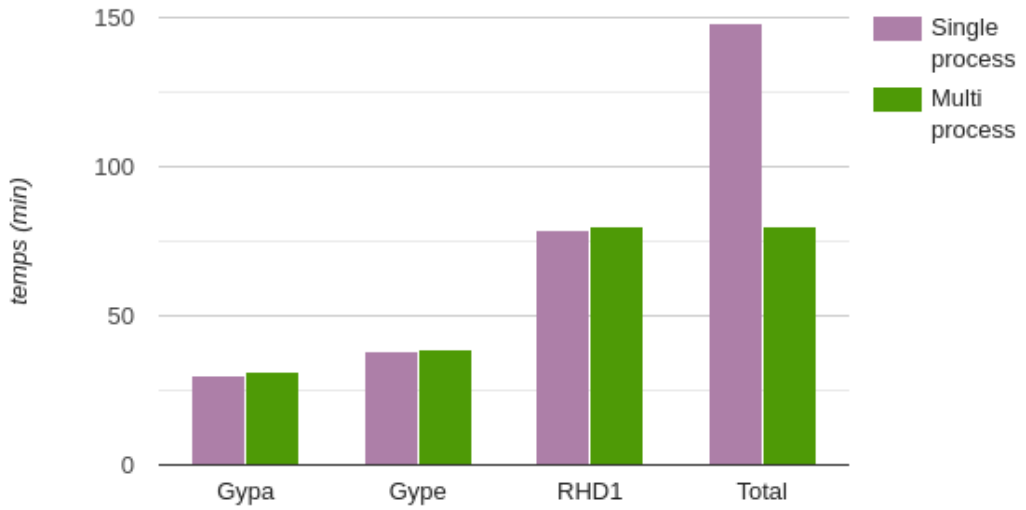



FIGURE 3.2.16 – Comparaison du temps d'exécution total entre le singleprocessing et multiprocessing de primalscheme exécuté sur Gypa, Gype, RHD pour le génome humain

Bien évidemment, cette optimisation par multiprocessing nécessite une rélfexion. S'il n'y a qu'un processus, l'exécution serait en réalité séquentielle. S'il y a plusieurs processus, alors une optimisation du temps d'exécution total serait évidente et il s'agirait bien de parallélisme. Idéalement, il faudrait :

$$\text{nombre de processus} \geq \text{nombre de fichiers d'entrée}$$

Néanmoins, cette situation n'est pas toujours vraie, par exemple dans le cas où il y a une contrainte technique liée aux limitations de la machine comme, par exemple, un nombre de coeurs limité. Une idée à implémenter pourrait être de prioritiser l'allocation des processus en fonction du temps d'exécution du programme liée à la taille des fichiers d'entrée. Un exemple serrai d'attribuer un processus par fichier considéré comme lourd et de distribuer le reste des fichiers aux processus restants. Bien évidemment, la taille d'un fichier considéré comme lourd pourrait être sujet à discussion, notamment en fonction de l'intervalle des tailles des fichiers pris en compte.

### 3.2.5 Évaluation des dimers d'amorces

Lors de la conception d'amorces ou primers pour des applications multiplex (qui prennent en compte plusieurs cibles) telles que la PCR, l'extension d'amorce, l'hybridation spécifique ou les événements, il est utile de disposer de moyens pour comparer efficacement les amorces/sondes [82]. Le programme de sélection d'amorces PCR basé sur Web Primer3 et PrimerScheme les choisit pour générer un seul produit PCR. Les séquences d'amorces directes et inverses sont sélectionnées afin d'éviter la formation potentielle de dimers d'amorces (amorce susceptible de s'attacher à une autre amorce) et d'épingles à cheveux intramoléculaires ou hairpins (amorce susceptible de se lier à elle-même), mais aucune indication (sur les caractéristiques des dimers et des hairpins)

n'est laissée au biologiste sur celles-ci. L'utilisateur obtient juste le résultat déjà filtré, ce qui est un réel problème pour l'équipe du CHU UCL Namur.

Des algorithmes de criblage de complémentarité (rechercher facilement des bases complémentaires entre deux amorces) entre des ADN courts ont été décrits. Aujourd'hui, la science compte une multitude de progiciels gratuits pour la sélection d'amorces PCR qui emploient des variantes de ces algorithmes de base pour déterminer la complémentarité [82].

L'objectif principal de ce point est de cribler des ensembles de paires d'amorces PCR pré-sélectionnées, pour identifier les amorces qui ont plus de bases complémentaires avec d'autres. Le programme effectuera des inter-comparaisons amorce-amorce tout en évaluant les interactions selon les règles traditionnelles d'appariement des bases Watson-Crick. Les résultats pourront être inspectés visuellement ou enregistrés dans un fichier texte et excel afin de donner la main au biologiste, de pouvoir interagir avec les résultats obtenus. La sortie d'information consiste en une composante visuelle ainsi qu'un score qui représente le degré d'interaction. Pour une température de fusion de transition (Tm), l'énergie libre de fusion ($\Delta$G) est calculée pour chacune des interactions potentielles amorce-amorce.

#### 3.2.5.1 Méthodes de détection des dimers et hairpins

L'algorithme de détermination de la complémentarité inter-brins est similaire à celui décrit dans l'article "*A computer program for choosing optimal oligonucleotides for filter hybridization, sequencing and in vitro amplification of DNA* " [69] : deux primers se chevauchent progressivement, la présence ou l'absence d'appariement de bases est évaluée et tabulée pour chaque chevauchement. La figure 3.2.17 illustre le glissement de deux amorces de 20 nucléotides dans l'algorithme pour l'inter-comparaison de celles-ci [82].

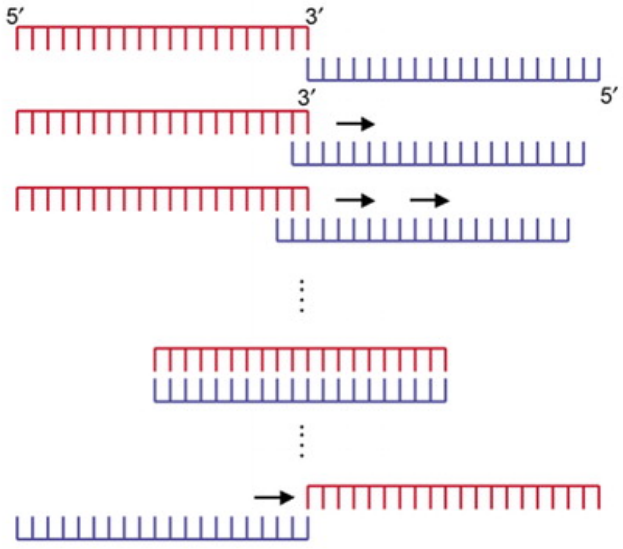



FIGURE 3.2.17 – Glissement de deux amorces pour l'inter-comparaison de celles-ci [82] .

Deux séquences sont comparées à chaque état d'appariement de bases de Walson-Crick possible (à l'exclusion des boucles et des lacunes). Dans l'exemple de la figure 3.2.17, un duplex avec deux parties de longueur X et Y où X = 20 et Y = 20, donnent 400 comparaisons de bases au total. Pour un ensemble d'amorces PCR, le nombre de comparaison duplex-duplex effectuées est égal à $2 * n^2 + n$, où n = nombre de paire d'amorces possibles. Pour 5-plex (10 amorces), 55 comparaisons amorce-amorce sont effectuées [82].

Une variante de cet algorithme a été employée pour cribler de courtes structures en épingle à cheveux intramoléculaires. Une épingle à cheveux intramoléculaire peut se former lorsqu'un seul brin d'ADN contenant des régions de séquences complémentaires se lie à lui-même. Le même brin unique a également le potentiel de se dimériser (de manière intermoléculaire), dans lequel un duplex avec un renflement incompatible sera présent au centre du duplex. L'homodimère intermoléculaire ne sera criblé que dans l'algorithme de criblage amorce-dimère. Les épingles à cheveux d'ADN ont tendance à se former à de faibles concentrations de sel et de brins par rapport à l'homodimère correspondant. Traditionnellement, une épingle à cheveux se compose d'une tige (région duplex) et d'une boucle simple brin. Les désignations de tige et de boucle sont illustrées à la figure 3.2.18. La formation d'une boucle simple brin est défavorable et est typiquement associée à une énergie libre de formation positive. La stabilité de la région de la tige duplex doit être suffisamment élevée pour surmonter l'énergie libre de la boucle déstabilisatrice. Des études sur les effets de la variation de la taille et de la séquence des boucles ont été rapportées. Les boucles de 4 et 5 nucléotides se sont avérées les moins déstabilisantes. Sur la base de ces informations, l'algorithme de criblage en épingle à cheveux permet des boucles de 4 et 5 bases avec un minimum de 2 paires de bases dans la tige [82, 7]. Un exemple de l' algorithme de dépistage en épingle à cheveux est illustré à la figure 3.2.18.

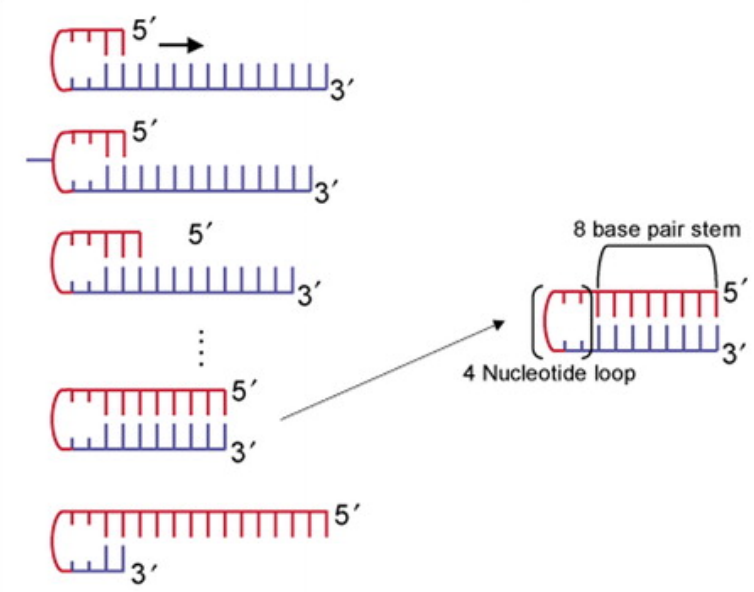



FIGURE 3.2.18 – Algorithme glissant pour le criblage de structures secondaires intramoléculaires (épingle à cheveux) [82]

Dans l'exemple décrit, un primer d'ADN simple brin est replié sur lui-même et criblé pour l'appariement de bases Watson-Crick [82]. Le criblage de l'amorce de 20 nucléotides a donné lieu à des comparaisons de 116 paires de bases. Les régions de la tige duplex et de la boucle simple brin de l'épingle à cheveux sont indiquées à droite de la figure 3.2.18 [7].

L'outil de vérification des dimers d'amorces prend une série de fichiers TSV en entrée et génère en sortie un fichier de type TSV afin de permettre aux utilisateurs d'avoir main mise sur le choix des primers. La figure 3.2.19 illustre le fonctionnement global de cette partie de vérification des primers.

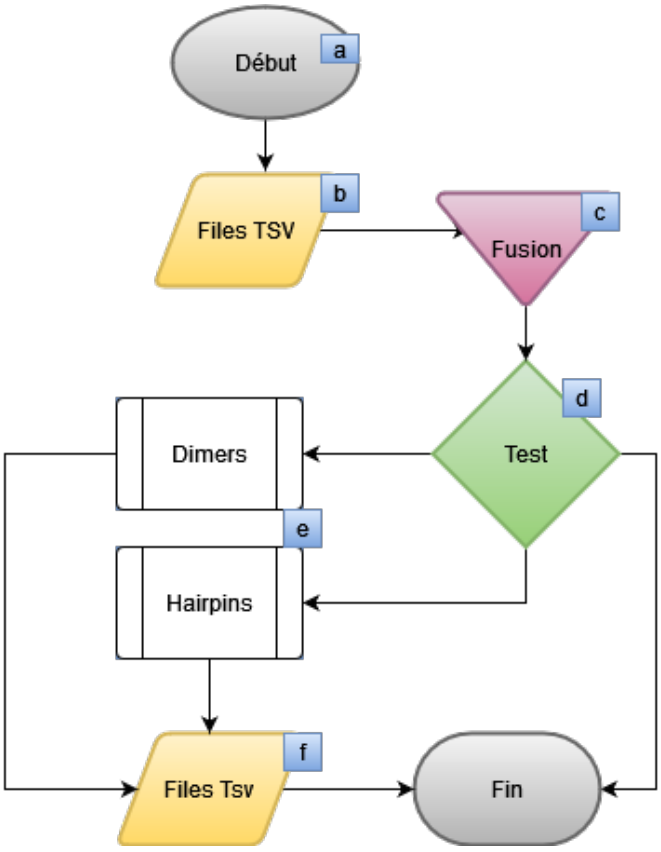



FIGURE 3.2.19 – Diagramme de vérification des dimers et hairpins lors de la conception d'amorces : b) l'outil permet de charger plusieurs fichiers, c) les fichiers chargés en entrée sont fusionnés en un seul fichier, d) une vérification des dimers ou hairpins est effectué sur chaque paire d'amorces, f) les résultats sont stockés dans un fichier .tsv et .txt pour permettre leur consultation.

Une valeur score est déterminée en combinant le nombre de paires de bases Watson-Crick. Cette valeur score est représentée sous la forme :

MaxScore(score, direction, index, position, deltaG)

— score : est le nombre d'appariements de bases consécutives entre deux amorces pour un dimer et une amorce pour le casa de hairpin.

— direction : est le criblage en amont (forward) et en aval (backward)

— index : est la position où débute l'appariement des bases pour un dimer ou un hairpin

— position : est une liste qui reprend la position de chaque appariement de base entre les amorces

— deltaG : est l'énergie libre de l'amorce calculée à l'aide de la méthode du plus proche voisin de "Breslauer, KJ et al" qui dépend d'une température de fusion ce qui permet de vérifier l'attraction des bases de nucléotides [78].

Un exemple de séquence auto-complémentaire est représenté sur la figure 3.2.20 ci-dessous.

```
1  ----------------------------------------------------------------------
2  5'> TCCTGGACAGAAACATCTCTGGA >3'   scheme_2_LEFT
3           | |||
4     3'< CACGGTCACACGGTTTTAGCTT <5'   scheme_2_RIGHT
5  ----------------------------------------------------------------------
6  Max scored dimer is  (3, 'forward', 3, [2, 4, 5, 6], '-1.060 Kcal/mol')
7  ----------------------------------------------------------------------
8  5'> TCCTGGACAGAAACATCTCTGGA >3'   scheme_2_LEFT
9              |  |     ||||
10          3'< TGATGTGCAAGACTACCGACAC <5'   scheme_4_LEFT
11 ----------------------------------------------------------------------
12 Max scored dimer is  (4, 'forward', 8, [0, 3, 9, 10, 11, 12], '-1.710 Kcal/mol')
13 ----------------------------------------------------------------------
```

FIGURE 3.2.20 – Résultat du criblage de séquence auto-complémentaire stocké dans un fichier .txt.

### A. Format de séquence d'entrée

Les fichiers d'entrée de l'outil sont des simples fichiers .tsv qui représentent les primers qui ont été produits. Le programme reconnaît la concaténation des lettres A,C,T,G majuscules et minuscules sans espace des colonnes "seq" du fichier. Des exemples de formats d'entrée valides acceptables sont illustrés sur la figure 3.2.21 ci-dessous :

| | name | pool | seq | size | %gc | tm (use 65) | 20-mers_css | 19-mers_css | 18-mers_css |
|---|---|---|---|---|---|---|---|---|---|
| 1 | test_1_LEFT | 1 | ACATGCCCCCTAAATACAGCTG | 22 | 50.00 | 60.61 | 0 | 0 | 0 |
| 2 | test_1_RIGHT | 1 | TGAAGAATAGCCTTCTGGTTTATTCTAAAC | 30 | 33.33 | 60.78 | 0 | 0 | 0 |
| 3 | test_2_LEFT | 2 | GCCTATGCATCACTTTGGGTCA | 22 | 50.00 | 61.13 | 0 | 0 | 0 |
| 4 | test_2_RIGHT | 2 | AGTGATTGAAGAATAGCCTTCTGGTT | 26 | 38.46 | 60.97 | 0 | 0 | 0 |
| 5 | test_3_LEFT | 1 | GGCTCTCCGTTCAGAGTTGTTT | 22 | 50.00 | 60.99 | 0 | 0 | 0 |
| 6 | test_3_RIGHT | 1 | AGCCCTGGTTGTTTGCTTGT | 20 | 50.00 | 60.70 | 0 | 0 | 0 |

FIGURE 3.2.21 – Fichier d'entrée acceptable pour les dimers d'amorces

Une fois les fichiers .tsv entrés dans le programme, l'utilisateur peut définir le nom du dossier de sortie (dont les fichiers de sorties porteront le nom), le seuil de score pour les dimers et les hairpins, la température à laquelle $\Delta G$ est calculé. Le seuil de score permet de trier quelles interactions de séquence seront stockées dans le fichier de sortie.

Une interaction de paire de séquences dont le score est inférieur au score seuil n'est ni enregistrée ni affichée dans le programme. La ré-exécution du programme (avec le même score de seuil ou un score de seuil différent) entraîne une ré-exécution complète de l'algorithme (c'est-à-dire que les résultats précédents ne sont pas conservés à moins d'être enregistrés manuellement).

#### 3.2.5.2 Résultats

Après avoir chargé les séquences dans le programme et fait varier les paramètres souhaités, le programme se charge de faire le traitement des inputs d'entrée. Il génère un fichier .txt et un fichier .tsv qui affiche les séquences (une à une) dont les interactions répondent aux critères de seuil. La figure 3.2.22 illustre l'interface paramétrable de l'outil, la figure 3.2.23 représente un exemple de fichier .tsv de sortie et la figure 3.2.20 (présentée précédemment) représente un exemple de fichier .txt reprenant les résultats de criblage de séquence auto-complémentaire.

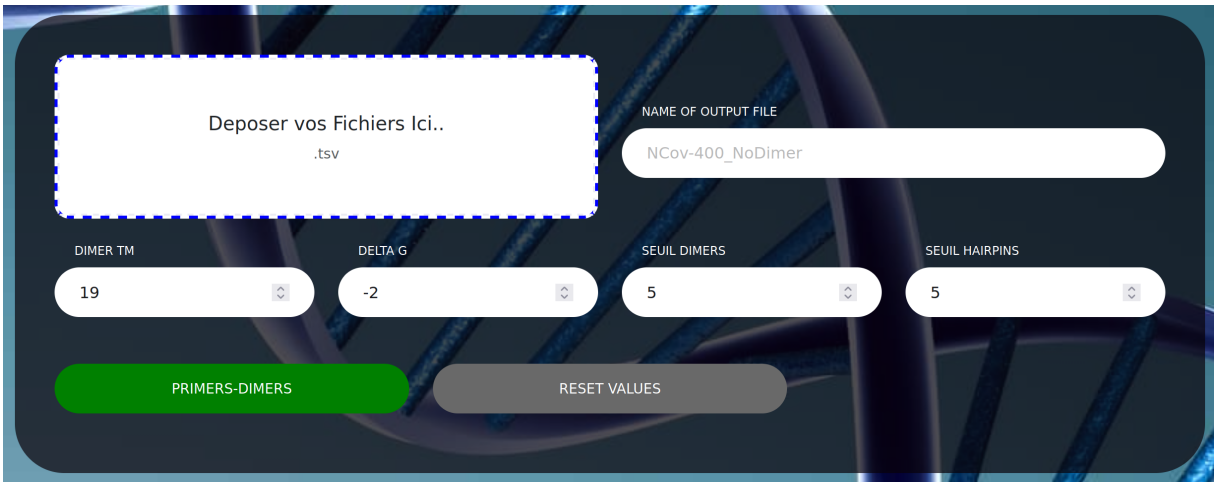



FIGURE 3.2.22 – Interface paramétrable de l'outil

| | C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|---|
| 1 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | TCGGCCTGAACCTTCTGAATTG | scheme_4_RIGHT | 0.750 Kcal/mol | 57 |
| 2 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | AACCTAGTTACCAGGCAAGACC | scheme_8_LEFT | -3.850 Kcal/mol | 57 |
| 3 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | AATGGCCGGAGTCCTTTAAGAA | scheme_14_LEFT | -2.540 Kcal/mol | 57 |
| 4 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | CGCAGTCTATGGAGATGTGCTC | scheme_14_RIGHT | -3.050 Kcal/mol | 57 |
| 5 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | GTGTTTACCGGCAGCACTGA | scheme_18_RIGHT | -2.370 Kcal/mol | 57 |
| 6 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | ATATCCGGCGCCTGTGTACT | scheme_20_LEFT | -2.380 Kcal/mol | 57 |
| 7 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | GCAATGTGTTGACGAACAGAGT | scheme_22_RIGHT | -2.150 Kcal/mol | 57 |
| 8 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | CAAAAGAAAAAGAAGCCGGGCC | scheme_26_LEFT | -2.370 Kcal/mol | 57 |
| 9 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | CATTGTTCCAGTAATCGTGCACG | scheme_28_RIGHT | -2.110 Kcal/mol | 57 |
| 10 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | GACACCCCAGATCGCACATTAA | scheme_30_LEFT | -2.730 Kcal/mol | 57 |
| 11 | TCCTGGACAGAAACATCTCTGGA | scheme_2_LEFT | GTTTGGTTCTTCTCCCATGTTCC | scheme_30_RIGHT | 0.140 Kcal/mol | 57 |

FIGURE 3.2.23 – Fichier de sortie .tsv de l'outil

Une interface simple permet à l'utilisateur d'ouvrir un fichier de séquence. Le programme fournit des informations sur le nombre d'échantillons lus à partir du fichier

d'entrée. L'utilisateur peut faire varier le seuil de score. Le score attribué à chaque interaction est un critère très basique utilisé pour juger le degré de complémentarité de deux séquences. Le score reflète la stabilité générale ou la tendance de l'interaction potentielle qui pourrait exister entre celles-ci.

### 3.2.6 Améliorations futures

Concernant le testing, quelques approches ont été abordées : pour les nouveaux modules ajoutés, à savoir la calcul de k-mers et l'évaluation des dimers d'amorces, des *tests unitaires* ont été effectuées afin de tester le bon fonctionnement de ceux-ci et surtout vérifier la génération de résultats cohérents. Ensuite ils ont été intégrés au reste du programme primalscheme afin de subir des *tests d'intégration*. Pour ceux-ci il s'agissait d'une série de 5 tests prenant respectivement en compte les gènes FUT1, RHD, GYPA, GYPB. Ces séries ont parfois été exécutées sur le génome humaine, parfois sur le bactérien. De plus une série de tests jouant avec les paramètres de l'interface a permis de démontrer leur impact sur les amorces conçues. Ayant eu une contrainte de temps, certains niveaux de tests n'ont pas pu être abordés mais mériteraient néanmoins qu'on s'y intéresse tels que les tests de couverture.

Pour revenir sur le point de complexité abordé dans la section 3.2.4, il pourrait être intéressant de faire une étude de complexité en prenant compte des paramètres autres que les fichiers d'entrée. De plus, des versions compilées de certaines parties du programme pourraient également être introduites afin d'améliorer les performances de temps et de calculs.

De plus, par manque de temps, seules les exigences de bases ont été traitées. Des fonctionnalités plus poussées n'ont pas été développées. Il pourrait par exemple être envisagé d'implémenter, sur base de machine learning, des paramètres optimaux sur base de la taille du fichier .fasta d'entrée du programme de conception d'amorces ou encore une IA qui détecteraient, par essai-erreur, le meilleur chevauchement d'amorce sur base d'images.

### 3.2.7 Conclusion

Dans ce chapitre, il était question de développer un outil permettant la construction des amorces qui doivent délimiter la portion d'ADN à amplifier afin d'être traitée, comme point d'entrée pour la prochaine étape (séquençage). Il est important d'avoir des amorces de bonne qualité afin de favoriser un assemblage correct par rapport aux différentes références qui lui seront attribuées. Ceci passe donc par un contrôle de la présence des dimers ou hairpins dans les amorces construites. La partie suivante sera consacrée à la mise sur point d'un aligneur permettant de générer une séquence consensus d'ADN au travers d'un pipeline.

# Chapitre 4

# Automatisation et amélioration d'un pipeline

## 4.1 Mise en place d'un pipeline automatisé pour le séquençage d'ADN

Dans le chapitre précédent, le séquençage était amélioré selon une approche prétraitement. L'idée consistait à concevoir de meilleures amorces afin de voir la zone ciblée amplifiée lors du séquençage et ainsi augmenter la qualité des données qui transitent dans le pipeline.

Ce chapitre-ci ciblera essentiellement la mise au point d'un pipeline automatique pour gérer *l'alignement*, *la détection des variants* [1] (qui sera également améliorée à travers le fichier contenant les variants) et *la génération d'une séquence consensus* par rapport à une référence. Une analyse des performances du pipeline sera également décrite à la fin de cette section.

### 4.1.1 Introduction

#### 4.1.1.1 Alignement

Il s'agit de l'étape d'alignement des séquences d'ADN (appelées read) prélevées sur un humain sur une séquence de référence connue au préalable. Cet alignement est réalisé grâce à des algorithmes d'alignement qui positionnent chaque «read» sur une position génomique selon une analyse probabiliste. Aujourd'hui, il existe de nombreux aligneurs. Il est important de noter que certains sont adaptés selon un traitement ou une utilisation bien précise, adaptés au SNP (single nucléotide polymorphisme), SNV (single nucléotide variation) ou Indels (Insertion Deletion).

Selon les normes bio-informatiques, un aligneur prend en entrée un fichier FASTQ et génère en sortie un fichier BAM (Binary Alignment Map) ou SAM (Simple Align-

1. aussi appelé "variant calling"

ment Map) associant à chaque read ses coordonnées génomiques [40]. Ce qui permet l'existence d'une panoplie d'outils d'aligneurs en fonction des besoins spécifiques.

#### 4.1.1.2 Détection des variants

Il s'agit d'un ensemble des méthodes visant à identifier des SNVs ou des Indels ou des CNV (Copy Number Variation) à partir des résultats de l'alignement. Dans la plupart des cas, cette étape est différenciée de l'alignement mais ses résultats sont très souvent dépendants de ceux de l'aligneur : effectuer son appel en tenant compte de l'aligneur choisi est un plus pour l'obtention de bons résultats [42]. Le variant est toute différence observée entre la séquence d'un individu et celle de référence utilisée. Les résultats sont stockés dans un fichier standard de type VCF (Variables Call Files) ou BCF (Binary Call Files). Il existe plusieurs logiciels de détection ou d'appel de variants.

#### 4.1.1.3 Génération d'une séquence consensus

Une fois l'alignement fait, il est intéressant de générer une séquence consensus afin de visualiser les zones couvertes et non couvertes par les alignements de lectures ainsi que de représenter pour chaque position, la base majoritaire présente dans les différentes lectures qui la chevauchent (sauf si c'est une base non couverte). Plus de détails se trouvent dans la section 4.3.

### 4.1.2 Panorama des logiciels existants

L'étude des logiciels évoqués dans cette partie n'est pas une étude exhaustive de tous les outils d'aligneurs, mais juste de ceux utilisés par le laboratoire du CHU UCL Namur et de quelques logiciels supplémentaires qui viennent les compléter en fonction des mises à jour et des performances. De même, cette étude ne vise pas à effectuer une comparaison entre outils, mais vise à déterminer les caractéristiques et les fonctionnalités offertes par ces logiciels par rapport aux besoins et aux objectifs du CHU.

Le tableau 4.1 récapitule les logiciels les plus importants et les plus cités dans les articles pour l'alignement de séquences.

Le tableau 4.2 récapitule les logiciels les plus importants pour la détection des variants en fonction des résultats attendus en sortie.

Le tableau 4.3 récapitule les logiciels les plus importants pour la manipulation et la visualisation des fichiers.

| Logiciel | Fonctionnalités général | Input | Output | Système | Licence |
|---|---|---|---|---|---|
| BWA | Logiciel d'alignements de <<Short-Read>>.<br>Réaliser des alignements sur des<br>génomes de référence.<br>Réaliser trois types d'alignements :<br>1. BWA-backtrack.<br>2. BWA-SW.<br>3. BWA-MEM. | Fastq | Sam | Linux,<br>MacOS,<br>Windows | libre droit (GPLv3,MIT License) |
| BOWTIE | Logiciel d'alignements de « short reads ».<br><br>Le programme utilise<br>une doubleindexation<br>du génome de référence. | Fastq | Sam | Linux,<br>MacOS,<br>Windows | libre droit |
| BOWTIE2 | Logiciel d'alignements de reads courts.<br><br>Capable de réaliser deux types dalignement :<br>1. -End-to-end alignment<br>2. -Local alignment | Fastq | Sam | Linux,<br>MacOS,<br>Windows | libre droit (GPLv3) |
| NOVAALIGN | Logiciel d'alignements de reads courts.<br>réaliser des alignements sur des génomes ambigus. | Fastq | / | Linux,<br>MacOS, | libre droit |
| MINIMAP2 | Logiciel d'alignements de séquences polyvalent.<br>Réaliser des alignements sur des<br>grandes base de données de référence.<br>Réaliser trois types d'alignements :<br>1. BWA-backtrack.<br>2. BWA-SW.<br>3. BWA-MEM.<br><br>aligne les séquences D'ADN et D'ARN | Fastq | Sam | Linux,<br>MacOS,<br>Windows | libre droit (MIT) |

TABLE 4.1 – Alignement de séquence [14, 12, 62, 53]

| Logiciel | Fonctionnalités général | Input | Output | Système | Licence |
|---|---|---|---|---|---|
| SAMTOOLS / MPILEUP | L'outil samtools mpilepup permet de convertir les reads alignés (fichiers BAM) en comptages par position génomique. | | VCF, BCF | Linux, MacOS | libre droit (MIT) |
| BCFTOOLS/ MPILEUP/CALL | L'outil BCFTools call met ensuite en œuvre une méthode statistique basée sur un modèle bayésien, afin d'identifier des sites variants par rapport à la référence (SNP et indels). | BAM | VCF, BCF | Linux, MacOS | libre droit (MIT) |
| GATK | outil dont l'objectif premier est la détection de variants Haplotype Caller et le génotype. | SAM , BAM, VCF | SAM , BAM, VCF | Linux, MacOS | libre droit (Apache) |
| VCFTOOLS | Logiciel d'alignements de reads courts.<br>réaliser des alignements sur des génomes ambigus. | Fastq | / | Linux, MacOS, | libre droit (LGPLv3) |
| FreeBayes | FreeBayes est un détecteur de variants génétiques conçu pour trouver de petits polymorphismes (SNP, indels, MNP et événements complexes). | BAM , VCF | BAM , VCF | Linux | libre droit (MIT) |

TABLE 4.2 – Détection de variants [71, 26, 85]

| Logiciel | Fonctionnalités général | Input | Système | Licence |
|---|---|---|---|---|
| SAMTOOLS | permettant de manipuler des fichiers au format sam (conversion au format bam (binaire correspondant), tri, création d'index, statistiques sur l'alignement, nettoyage de potentiels biais de PCR, ...). | SAM | Linux, MacOS | libre droit (MIT) |
| BCFTOOLS | Bcftools est un ensemble de programmes pour manipuler des fichiers de variants au format vcf ou bcf. | BAM | Linux, MacOS | libre droit (MIT) |
| IGV | Integrative Genomics Viewer (IGV) est un outil de visualisation pour l'exploration interactive de grands jeux de données génomiques. permet de visualiser un grand nombre de formats de fichiers : fichiers bam (triés par position et indexés), bed, gff, vcf, P | SAM , BAM, VCF | Linux, MacOS, Windows | libre droit (MIT) |
| PICARD TOOLS | fournit un grand nombre de programmes (Java) pour manipuler des fichiers aux formats sam/bam/cram ou vcf permettent d'obtenir des statistiques sur les alignements | sam/bam/cram ou vcf | Linux, MacOS, | libre droit (MIT) |
| PYSAM | module python de lecture et de manipulation de fichiers | SAM, BAM | Linux, MacOS, | libre droit (MIT) |

TABLE 4.3 – Manipulation et visualisation des fichiers [37, 36, 71]

### 4.1.3 Automatisation du pipeline d'alignement avec Nextflow

#### 4.1.3.1 Récapitulatif des apports d'un WorkFlow automatisé

Aujourd'hui, le laboratoire du CHU UCL Namur utilise un pipeline non automatisé. Il est principalement basé sur l'exécution des processus d'analyse bio-informatique en ligne de commande. Les scripts sont construits sous la forme de fichiers textes, et le tout est lancé en effectuant un copier-coller bout par bout dans un terminal dédié.

En résumé, les principaux problèmes des pipelines d'analyse de données du projet érythrocytaire actuels sont :

1. des pipelines inaccessibles ou difficiles à utiliser pour les utilisateurs non techniques car il y a actuellement beaucoup de bugs dans l'exécution du pipeline
2. la réécriture d'un nouveau script pour un nouveau set de test : ce qui rend lourde la maintenance en cas d'erreur ou de réutilisation du script, qui à la longue peut créer une perte de temps dans la recherche d'un script pour une amélioration, une modification ou une réutilisation de celle-ci.
3. la non-structuration du pipeline : il est facile de se perdre rapidement dans un script.

L'annexe F, dont voici un extrait, présente un exemple de pipeline actuellement utilisé au laboratoire du CHU. L'objectif principal de cette partie est de mettre à disposition de ce laboratoire un workflow universel pour le projet érythrocytaire qui conviendrait à tous les tests sans ne rien changer dans le code mais en jouant juste avec les données d'entrées.

```
minimap2 -ax map-ont /home/labo-user/Typage_erythro/
    sequences_references/Duffy_seq.fasta donnees_nettoyees/
    fastq_merged_cleaned.fq > merged.sam
samtools view -bS merged.sam > merged.bam
samtools sort merged.bam -o merged_sorted.bam
samtools index -b merged_sorted.bam
#6 polissage par RACON (4 iterations)
mkdir racon/
bwa index Duffy_seq.fasta
bwa mem -t 14 -x ont2d Duffy_seq.fasta donnees_nettoyees/
    fastq_merged_cleaned.fq > racon/mapping1.sam
racon -m 8 -x -6 -g -8 -w 500 -t 14 donnees_nettoyees/
    fastq_merged_cleaned.fq racon/mapping1.sam Duffy_seq.fasta > racon
    /racon1.fasta
```

#### 4.1.3.2 Matériel utilisé

**A. Données biologiques**

Les données biologiques utilisées sont découpées en deux sets ;

— Les données de référence pour l'alignement : ce set de données biologiques utilisé ici est celui provenant d'un ensemble de données du génome humain généré par le CHU. Comme les données issues du séquençage d'ADN sont volumineuses, les bases du génome humain sont divisées en sous-ensembles à des positions bien particulières pour l'obtention de résultats concrets.

| Nom | Bases | Tailles |
|---|---|---|
| GRCH38 (genome humain) | 3.099.706.404 | > 3 GB |
| GYPA_seq | 32520 | 33,6 kB |
| GYPB_seq | 32520 | 33,6 kB |
| KELL | 21300 | 22 kB |
| Duffy | 4254 | 4,4 kB |

TABLE 4.4 – Données utilisées comme références et visualisation pour l'étape de l'alignement

— Les données pour les reads : elles sont également générées par le CHU sur base de la technique Nanopore pour le traitement et le séquençage, il s'agit de fichiers FASTQ.

**B. Outils informatiques**

1. **Système de gestion de flux de travail bio-informatique**

   Les systèmes de gestion des flux de travail représentent, gèrent et exécutent des analyses informatiques en plusieurs étapes. De manière plus précise, un système de gestion de flux de travail est un outil qui peut être utilisé par un bioinformaticien pour intégrer tous ses scripts bash/python/perl/autres dans un seul pipeline cohérent qui est portable, reproductible, évolutif et contrôlé [73]. De nombreux systèmes de gestion de flux de travail prennent en charge l'utilisation des conteneurs, de systèmes de calcul haute performance (HPC) et de clouds.

   Ces systèmes peuvent être utilisés pour l'analyse de données bio-informatiques qui comportent de nombreuses étapes. La partie cruciale consiste à exécuter les bonnes étapes de traitement dans le bon ordre, de manière fiable [75].

   Pour sélectionner un système de flux de travail, une enquête rapide a été menée sur deux outils : Snakemake [1] [43] et Nextflow [2] [19]. Celle-ci a été menée selon plusieurs critères que sont :

   — la popularité (le nombre de fois que l'outil apparaît dans les recherches et le nombre de personnes qui ont contribué au développement de l'outil)

   — le type de licence open-source

   — la facilité de prise en main (mesuré à travers les recommandations par un tiers)

   Le tableau 4.5 résume les résultats obtenus pour chaque critère d'évaluation afin de pouvoir faire le choix entre les deux outils.

1. https ://github.com/snakemake/snakemake
2. https ://github.com/nextflow-io/nextflow

| Workflow | Licence | Contributions | Résultats de recherches | Recommandations |
|---|---|---|---|---|
| Snakemake | MIT | 122 | 23000 | 0 |
| Nextflow | Apache 2.0 | 230 | 23800 | 3 |

TABLE 4.5 – Statistique sur les systèmes de gestion de workflow sélectionnés [38]. Comparé à Nextflow, Snakmake à une licence MIT, est cité 2300 fois, avec 122 contributions qui ont été apportés et 0 recommandation par une tiers personne pour ce travail.

Le choix définitif a été porté sur Nextflow suite aux résultats obtenus précédemment et aussi, car, en bio-informatique, l'analyse de données implique de nombreuses étapes qui nécessitent que les données soient regroupées, traitées et analysées à l'aide d'une succession de progiciels indépendants. Nextflow est un excellent gestionnaire de flux (workflow manager) qui utilise la technologie des conteneurs pour assurer un déploiement et une reproductibilité efficace des pipelines d'analyse computationnelle. Les pipelines tiers peuvent être portés dans Nextflow avec un recodage minimum [79]. Nextflow permet des workflows scientifiques évolutifs et reproductifs à l'aide de conteneurs logiciels (Docker, Singularity) et permet également l'adaptation de pipelines écrits dans les langages de script les plus courants tels que [2] :

— Python,

— Perl,

— Bash,

— etc.

Plusieurs caractéristiques de Nextflow valent son choix pour le déploiement du nouveau pipeline au sein du laboratoire d'analyses du CHU telles que :

- *Prototypage rapide* : Nextflow permet d'écrire un pipeline de calcul en simplifiant l'assemblage de nombreuses tâches différentes. Les scripts peuvent être réutilisés, donc pas besoin d'apprendre de nouveaux langages ou de nouvelles API [2].
- *Reproductibilité* : Nextflow prend en charge les technologies de conteneurs Docker et Singularity. Grâce au partage de code sur GitHub, des pipelines autonomes peuvent être écrits, la gestion des versions peut être prise en compte ainsi que la reproduction rapide de toute configuration antérieure [2].
- *Parallélisme unifié* : Nextflow est basé sur le modèle de programmation par flux de données qui simplifie grandement l'écriture de pipelines distribués complexes [2].

  La parallélisation est implicitement définie par des déclarations d'entrée et de sortie des processus. Les applications qui en résultent sont intrinsèquement parallèles et peuvent être mises à l'échelle de manière transparente, sans avoir à s'adapter à une architecture de plate-forme spécifique [2].
- *Portabilité* : Nextflow fournit une couche d'abstraction entre la logique du pipeline et la couche d'exécution, afin qu'il puisse être exécuté sur plusieurs plates-formes sans qu'il ne change [2].

- *Point de contrôle continu* : tous les résultats intermédiaires produits lors de l'exécution du pipeline sont automatiquement suivis [2].

2. **Système de conteneurisation pour la mise en production**

   La conteneurisation consiste à rassembler le code du logiciel et tous ses composants (bibliothèques, frameworks et autres dépendances) de manière à les isoler dans leur propre conteneur [47]. Celui-ci permet au logiciel, ou à une application, d'être déplacé et exécuté de façon cohérente dans tous les environnements et sur toutes les infrastructures, indépendamment de leur système d'exploitation.

   Docker et Singularity sont deux systèmes permettant de gérer des conteneurs afin de faciliter la mise en production d'un outil/programme sur plusieurs machines. Ceux-ci permettent de créer des environnements (appelés containers) de manière à isoler des applications. Beaucoup ont tendance à comparer Docker ou Singularity à une machine virtuelle (VM), or ce sont deux choses différentes.

   Un conteneur dans Docker est une instance d'une image qui peut ensuite être modifiée, reconstruite et exécutée. Les conteneurs sont accessibles de plusieurs manières, en fonction de leur configuration et de leurs objectifs [81]. Docker met également à la disposition de ses utilisateurs un référentiel appelé DockerHub pour permettre aux utilisateurs d'héberger leurs images en ligne afin que d'autres puissent s'en servir.

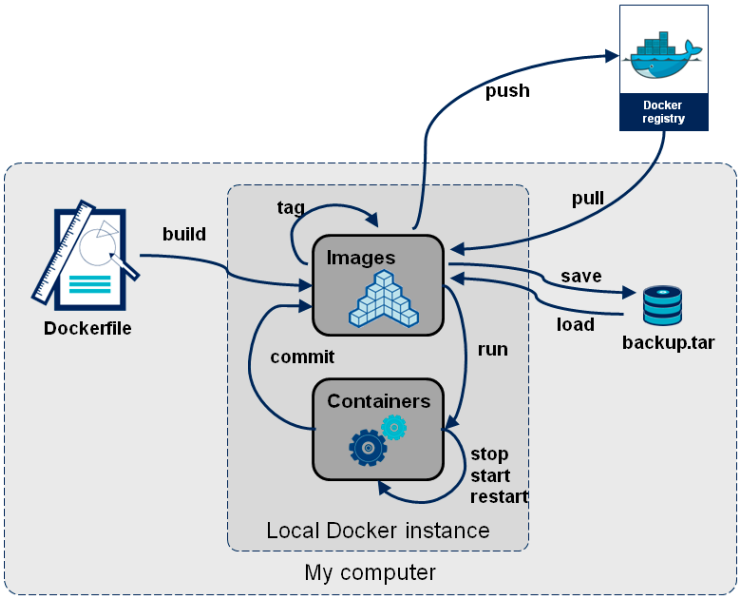



FIGURE 4.1.1 – Flux de travail d'image docker. Cette figure illustre le fonctionnement global de Docker. Tout débute par la mise en place d'un ficher *Dockerfile* nécessaire à la création (build) d'une image docker qui peut être renommée (tag), de même l'image peut être mise (push) ou récupérée (pull) depuis un environnement externe appelé *DockerHub*. Cette image est donc nécessaire pour la création d'un conteneur (run) sur n'importe quel système d'exploitation.

La figure 4.1.1 illustre le fonctionnement d'une image Docker (qui est similaire à une image Singularity). Les images sont stockées puis récupérées à partir d'un ré-

pertoire DockerHub. Plusieurs conteneurs peuvent être créés à partir d'une image docker puis être exécutés sur la machine hôte. L'image est créée sur base d'un fichier appelé *Dockerfile*.
Docker et Singularity sont deux outils de conteneurisation largement utilisés. L'un comme l'autre possèdent leurs défauts et qualités pour ce travail. L'outil Docker a été choisi suite à sa flexibilité, sa prise en main simple, son système de copie automatique de tous les fichiers du projet (là où dans Singularity, la copie de fichiers se fait manuellement) [80]. Il facilite également la construction du fichier de définition pour la mise en place d'une image Docker. Par conséquent, il pose deux problèmes : l'isolation du système de fichiers dans son propre environnement et son ID d'utilisateur (différent de celui du processus) qui doit être utilisé pour lancer l'exécution du container. L'outil Singularity est choisi car il solutionne ces deux problèmes : il favorise l'intégration plutôt que l'isolement tout en préservant les restrictions de sécurité sur le container et en fournissant des images reproductibles.

Le *tableau 4.6* présente les caractéristiques des différents outils informatiques utilisés pour la conception du pipeline du projet *Erythrocytaire*.

| **Outils / Bibliothèques** | **Version** | **Utilité** |
|---|---|---|
| Minimap2 | 2.17-r941 | Étape d'assamblage |
| Samtools | samtools 1.10 | Indexation |
| Racon | v1.4.3 | Polissage |
| Vcftools | vcftools (0.1.16) | Variant Calling |
| BcfTools | bcftools (0.10.2) | Variant Calling |
| Mummer/nucmer | Nucmer version 3.1 | alignement de génomes |
| vcf-stats | / | Get Stat Variant |
| pysam | / | Modificartton du Vcf |
| fpdf | / | Rapport de séquençage |

TABLE 4.6 – Outils utilisés pour la mise en place du pipeline

#### 4.1.3.3 Méthodes de développement

**A. Aperçu global du pipeline développé**

Le traitement de données se fait en une succession d'étapes. Le pipeline est repris en deux phases (la phase d'alignement et de génération de la séquence consensus et la phase de détection de variants). Ces deux phases sont directement liées : la sortie de la première est l'entrée de la suivante. Il est important de noter que certaines étapes des processus sont réalisées directement par l'automate de séquençage, tandis que d'autres sont réalisées de manière manuelle ou semi-automatisée par des scripts (python, bash) informatiques. La *figure 4.1.2* illustre les principales étapes de l'analyse du pipeline avec l'alignement et la génération de la séquence consensus. La *figure 4.1.3* est la suite du pipeline avec l'étape de la détection de variants. Il est également important de faire remarquer que la succession des processus du pipeline, illustrée à la figure 4.1.2 et 4.1.3, n'est pas la même que lors de l'exécution réelle du code.

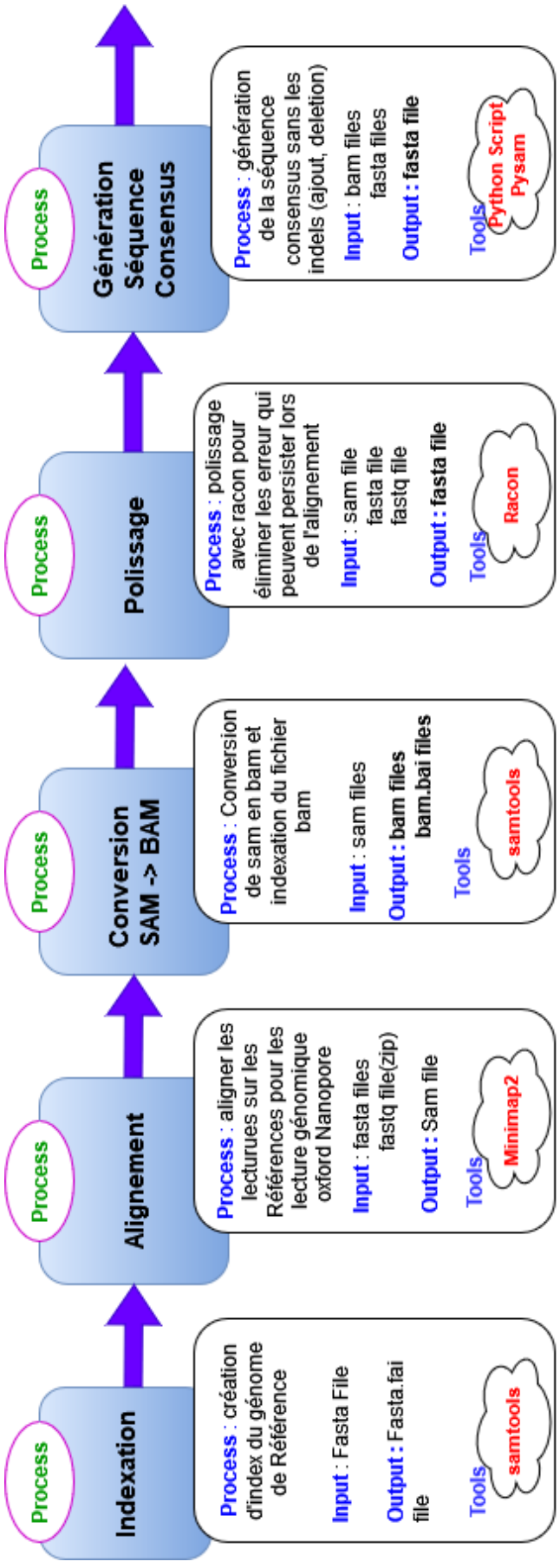



FIGURE 4.1.2 – Pipeline bio-informatique d'analyse de données partie alignement et séquence consensus du CHU

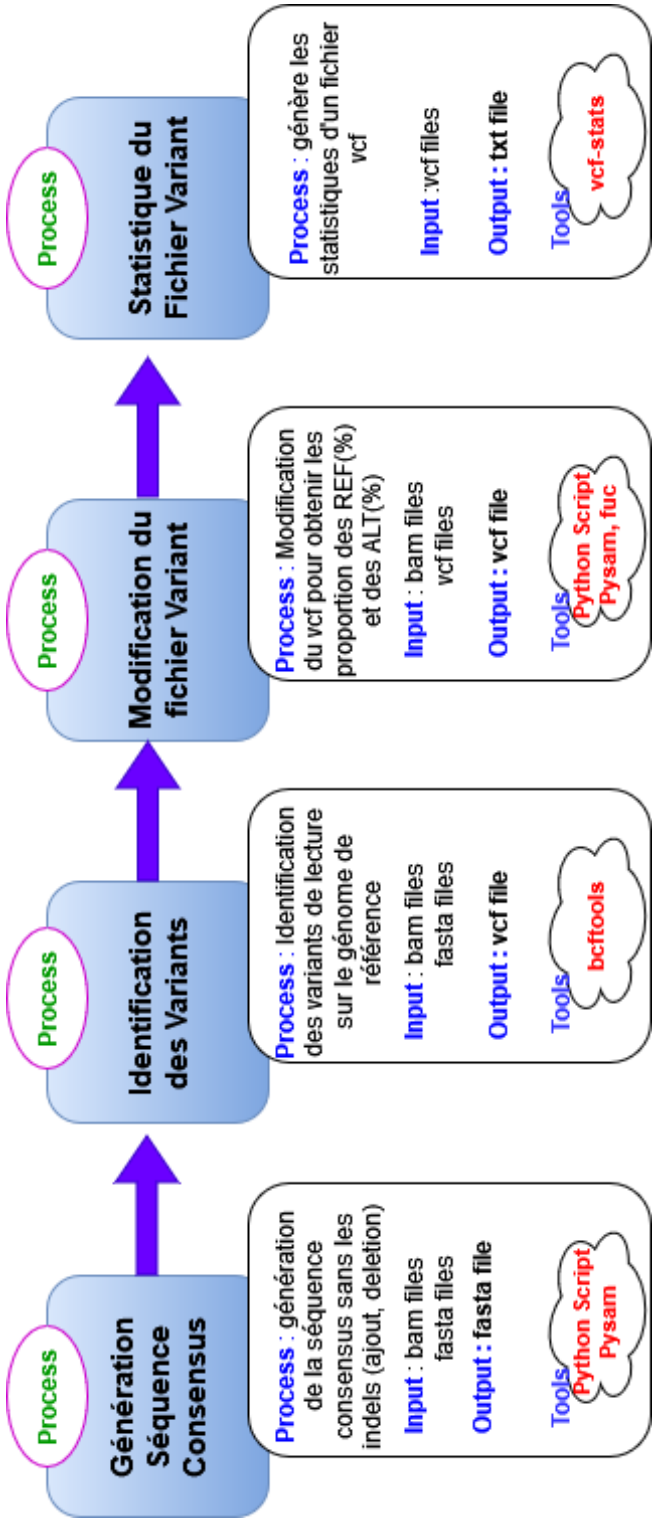



FIGURE 4.1.3 – Pipeline bio-informatique d'analyse de données partie détection de variants du CHU

**B. Description détaillée du pipeline d'analyse développé**

Le pipeline présenté la figure 4.1.2 et 4.1.3 est un pipeline qui comporte onze processus où chacun joue un rôle bien particulier. Ils sont regroupés en quatre grandes étapes :

1. **L'alignement** (4 processus) :

— **Processus d'indexation**

L'indexation du fichier de référence (FASTA) permet un accès aléatoire indexé à celui-ci. Ceci est utile pour récupérer rapidement des régions (par exemple chr1 :123-10004) ou des séquences uniques (par exemple chr1) à partir de fichiers FASTA [35]. Le fichier décrit les décalages d'octets dans le fichier FASTA pour chaque contig, permettant de calculer exactement où trouver une base de référence particulière à des coordonnées génomiques spécifiques dans le fichier FASTA [23].
Cela produit un fichier texte nommé .fai avec un enregistrement par ligne pour chacun des contigs FASTA. Chaque enregistrement reprend le contig, la taille, l'emplacement, des basesPerLine et des bytesPerLine. Le fichier d'index produit ressemble à ceci :

```
20 63025520 4 60 61
```

FIGURE 4.1.4 – Exemple de fichier indexé (.fai)

- Input : une collection de fichiers .fasta (références)
- Output : une collection de fichiers .fai

Ce processus utilise *samTools*. Ce choix suit celui de l'équipe du CHU car cet outil est rapide.

— **Processus d'alignement**

L'alignement avec le génome de référence : cette partie consiste à mapper les lectures (Reads) sur la référence (Ref) avec une efficacité et une précision élevées. Celle-ci dépend des outils utilisés et aussi du type de lecture (lecture longue ou lecture courte). Pour ce processus, l'outil minimap2 a été utilisé pour trouver une correspondance d'alignement optimale dans un temps de calcul réduit. Il est 3x plus rapide et aussi précis sur les données simulées que les outils BWA-MEN et Bowtie2 et précise les commandes en fonction du type de technologie utilisée lors de l'étape de séquençage [52].

- précondition : l'existence de fichiers .fai non-vide
- postcondition : les fichiers .fai inchangés
- Input : une collection de fichier .fasta et un fichier .fastq ou .fastq.gz (reads)
- Output : une collection de fichiers .sam

— **Processus de conversion Sam en Bam**

En bio-informatique, les données d'alignement pour un grand nombre de lectures alignées sont produites sous forme de fichier SAM (sequence alignment map). Afin que le fichier sam puisse être traité par d'autres outils bio-informatiques, il est nécessaire de le convertir en fichier binaire BAM (binary alignment map).

- Input : une collection de fichiers .sam
- Output : une collection de fichiers .bam et .bam.bai (qui représentent respectivement un fichier .bam indexé)

Ce processus utilise samTools. Ce choix suit celui de l'équipe du CHU car il est nécessaire de garder une certaine cohérence entre les outils utilisés. Il existe rapidement une incompatibilité liée à plusieurs facteurs (version, longueurs des reads, système de la machine, etc. ) entre les outils, d'où la cohérence favorise les résultats de meilleurs qualités.

— **Processus de polissage**

Le but ici est de corriger les contigs bruts générés par des méthodes d'alignements rapides qui n'incluent pas d'étape de consensus. L'objectif est de générer un consensus génomique de qualité similaire ou supérieure à la sortie générée par les méthodes d'assemblage qui utilisent à la fois des étapes de correction d'erreurs et de consensus, tout en offrant une accélération de la vitesse d'exécution multipliée par rapport à celles de ces méthodes. Il prend en charge les données produites par Pacific Biosciences et Oxford Nanopore Technologies [76].

- précondition : existence de fichiers .fai et .sam non-vides
- postcondition : les fichiers .fai et .sam inchangés
- Input : une collection de fichier .fasta et un fichier .fastq ou .fastq.gz (reads)
- Output : une collection de ficher .fasta

Le processus de polissage, qui fonctionne par itération, utilise une succession d'outils afin de corriger les contigs bruts générés : sammtools pour indexer le fichier fasta, minimap2 pour l'alignement, et *Racon* pour le polissage. Le choix de ce dernier outil suit celui du laboratoire du CHU, présent dans le pipeline actuel, car c'est l'un des outils les plus utilisés lors d'une analyse de séquençage à lectures longues et courtes et le plus cité dans les articles scientifiques.

2. **La détection de variants** : le processus de détection de variants est la génération d'un fichier vcf (Variant call file). L'objectif de cette partie est de modifier le fichier vcf fourni afin d'avoir les proportions de toutes les mutations rencontrées lors du séquençage. Cette étape, détaillée dans la section 4.2, est regroupée en trois processus :

— **Impression du fichier Vcf**. Ce processus consiste à identifier tous les variants présents dans de l'alignement.

- précondition : existence de fichiers .bam.bai non-vides
- postcondition : les fichiers .bam.bai inchangés
- Input : une collection de fichiers .fasta et .bam
- Output : une collection de fichiers .vcf

Ce processus utilise bcftools. qui est une suite de la librairie Tools qui représentent une collection unique d'outils pour la conversion et la manipulation de formats de fichiers, le tri, l'interrogation, les statistiques, l'appel de variantes et l'analyse des effets, entre autres méthodes au même titre que vcftools.

— **Impression des statistiques du fichier vcf**. Ce processus indique à l'utilisateur quelques informations sur le fichier vcf produit par le processus d'impression du fichier vcf telle que : les indels, SNP, les mutations hétérozygotes et homozygotes.

- précondition : existence de fichiers .bam.bai non-vides
- postcondition : les fichiers .bam.bai inchangés
- Input : collection de fichiers .vcf
- Output : collection de fichiers .txt

— **Amélioration du fichier vcf**.

- précondition : existence de fichiers .bam.bai et non-vides
- postcondition : les fichiers .bam.bai inchangés
- Input : collection de fichiers .fasta et .bam
- Output : collection de fichiers .vcf

Ce processus est détaillé dans la section 4.2.

3. **La génération de la séquence consensus :**
   - précondition : existence de fichiers .bam.ba non-vides
   - postcondition : les fichiers .bam.bai inchangés
   - Input : collection de fichiers .fasta et .bam
   - Output : collection de fichiers .fasta

   Ce processus est détaillé dans la section 4.3
4. **La génération d'un rapport d'analyse :** ce processus se charge de générer un rapport d'analyse sur le typage érythrocytaire des données introduites. C'est le résultat final de tout le processus de séquençage effectué. Cette analyse est donc générée au format PDF.
   - Input : collection de fichiers .vcf et fichier .xls non-vides
   - Output : collection de fichiers .pdf

   Ce processus est détaillé dans la section 4.4.

Les figures 4.1.2 et 4.1.3 présentaient le pipeline général qui a été développé pour le projet erythrocytaire. La figure 4.1.5 présente le pipeline, le chemin d'exécution, les fichiers d'entrées et de sorties de chaque processus du pipeline. L'exécution d'un pipeline avec Nextflow est parallèle sauf qu'un processus qui dépend d'un autre doit attendre la fin de l'exécution de cet autre avant de démarrer. Donc un processus situé en dixième appel peut démarrer avant un processus en deuxième ou troisième appel.

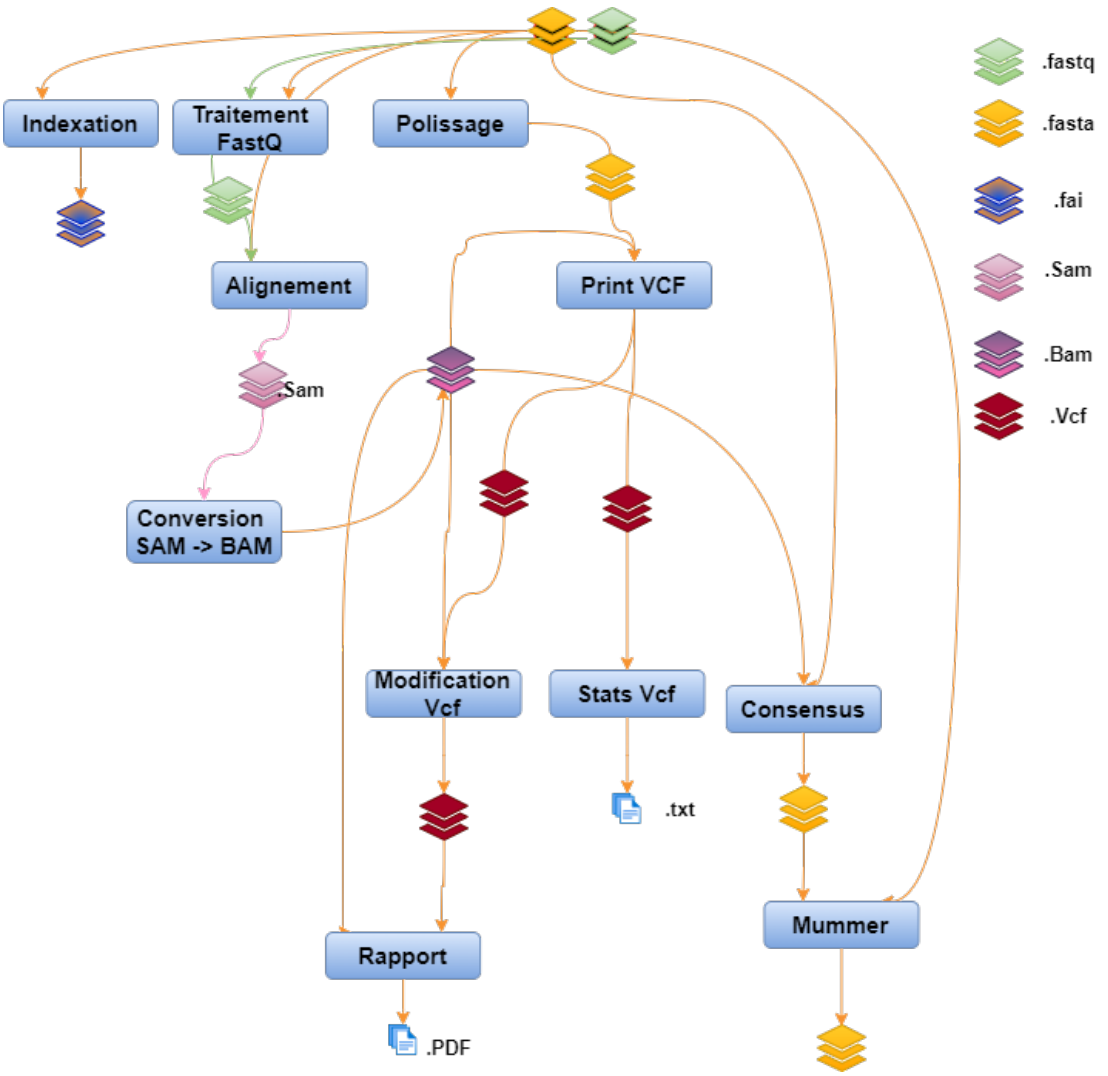



FIGURE 4.1.5 – Chemin d'exécution du pipeline conçu pour l'analyse érythrocytaire : ou chaque couleur fait référence à un type de fichier particulier fonction du processus en cours.

Le pipeline mis en place se lance grâce à la commande Nextflow :

*"nextflow run Nom_Fichier_NextFlow.nf"*

Néanmoins, Nextflow offre d'autres d'options qui sont très intéressantes pour la compréhension ou encore pour une analyse profonde du pipeline développé.

- *-with-report* : génère un fichier .html qui donne un aperçu de la distribution de l'utilisation des ressources (CPU, RAM, Durée du Travail, Run time, etc) pour

chaque processus tout en mentionnant le nombre de processus qui ont réussi et échoué.

- *-with-trace* : génère un fichier .txt reprenant les caractéristiques de chaque processus (date d'exécution, ROM, etc.). Cette commande est similaire à celle *-with-report*.
- *-with-timeline* : génère un fichier .html qui donne un aperçu sur la chronologie d'exécution des processus.
- *-with-dag* : produit une image .png qui illustre l'exécution des processus pour la totalité du pipeline.

#### 4.1.3.4 Performance du Pipeline

Les performances d'un pipeline se mesurent sur base des outils utilisés pour l'exécution de chaque processus que comporte le pipeline. Il n'existe aucun benchmark de comparaison étant donné que chaque pipeline est conçu pour un cas d'utilisation bien précis et est donc unique. Cependant, Nextflow offre quelques lignes de commande afin de donner la main à l'utilisateur et de pouvoir avoir une idée sur la performance globale en termes de ressources utilisées du pipeline. Il est à noter que celui-ci dépend en partie des outils bio-informatiques qui sont utilisés.

Ces facilités sont :

1. *Run time* : qui est la durée d'exécution du pipeline
2. *Ressources utilisées (CPU)* : donne un aperçu de la distribution de l'utilisation des ressources pour chaque processus.
3. *Mémoire utilisée (RAM)* : donne un aperçu de la mémoire physique (RAM) et de la mémoire virtuelle (RAM + échange de disque).
4. *Durée du travail* : donne un aperçu de l'utilisation brute pour chaque processus.
5. *Nombre de bytes des entrées / sorties en lecture / écriture*

En bio-informatique, étant donné la manipulation des fichiers volumineux à l'exemple du GRCh38/hg38 qui est l'assemblage du génome humain publié en décembre 2013 [33], parfois la performance du pipeline ne primera pas sur la qualité des résultats qui doivent être attendus. Un exemple typique de ce cas est illustré à la figure 4.1.6. La détection de variant est effectuée par rapport à deux outils d'alignement avec les mêmes données d'entrée :

— Référence : GYPB (Chromosome 4)

— Reads : Barcode94 (ensemble de fichier fastq)

— outils d'alignement : BWA/minimap2

La figure 4.1.6 compare le temps d'exécution de deux outils différents utilisant les mêmes entrées-sorties pour un seul processus (PrintVCF), mais ici la qualité du résultat primera sur la performance des processus. Il est parfois important de perdre en performances et de gagner en qualité du résultat attendu.

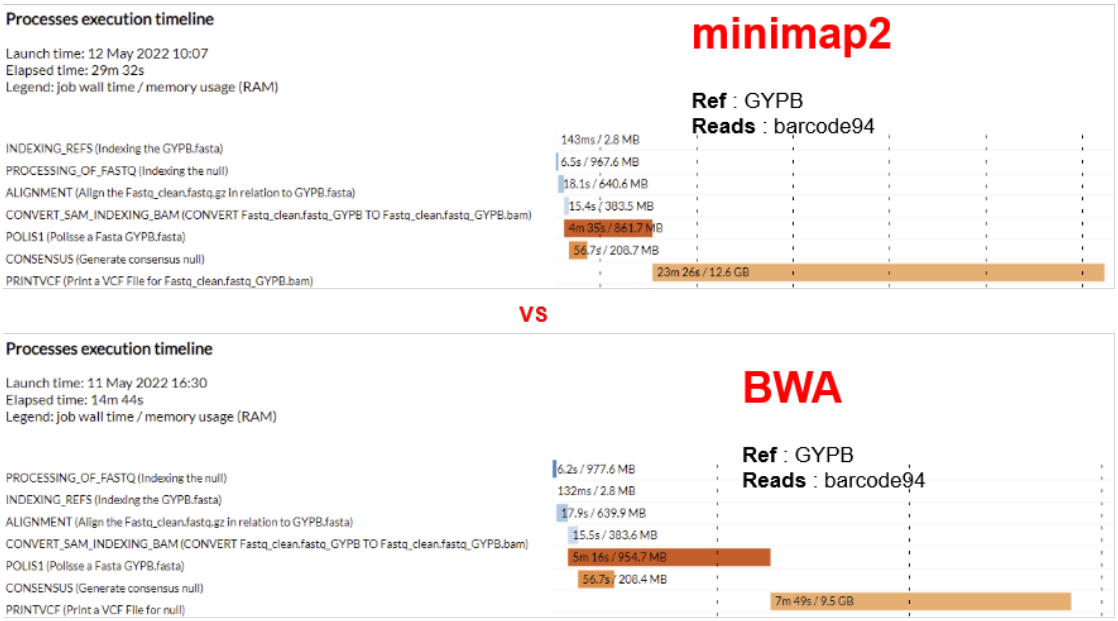



FIGURE 4.1.6 – Comparaison du temps d'exécution d'un même processus pour deux outils de détection de variant : pour l'outil minimap2, la détection de variant prend environ 23 minutes, mais la qualité du résultat est bien meilleure que l'outil BWA qui ne prend que 7 minutes pour s'exécuter.

### 4.1.4 Conclusion

L'alignement de séquences est la partie la plus importante du séquençage d'ADN ou ARN car elle utilise des algorithmes qui doivent être rapides et précis. Cette partie, lors du séquençage, suit une succession d'étapes bien précises afin d'avoir un résultat de bonne qualité. Celui-ci s'effectue par le biais d'un pipeline qui lui-même doit montrer sa robustesse pour gérer les outils nécessaires à cet effet. Aucun pipeline unique partant de l'alignement à la génération des variants n'a démontré sa supériorité dans la détection de tous les variants. L'application de plusieurs outils sans aucune analyse faite peut entraîner des résultats trompeurs. Il a également été rapporté que les aligneurs de lecture influencent la précision de la détection de variants avec la combinaison optimale d'aligneurs et d'appelants de variants qui peuvent produire des appels de variants précis, y compris des variants de nucléotide unique (SNV) et de petites insertions et suppressions (IndDels) [45].

## 4.2 Amélioration du fichier .vcf

Pour avoir une idée d'où la partie qui suit se trouve dans les étapes d'un pipeline, celui de l'annexe B peut être repris. Il s'agit alors de la phase d'alignement.

Lorsqu'un alignement est effectué, il aboutit à un fichier .SAM (fichier texte) qui sera ensuite converti en version binaire, le BAM. Pour visualiser cet alignement, l'outil

IGV [1] est utilisé. La figure 4.2.1 représente un aperçu de IGV pour un fichier .BAM généré à partir de données du CHU en prenant comme génome "Duffy" :

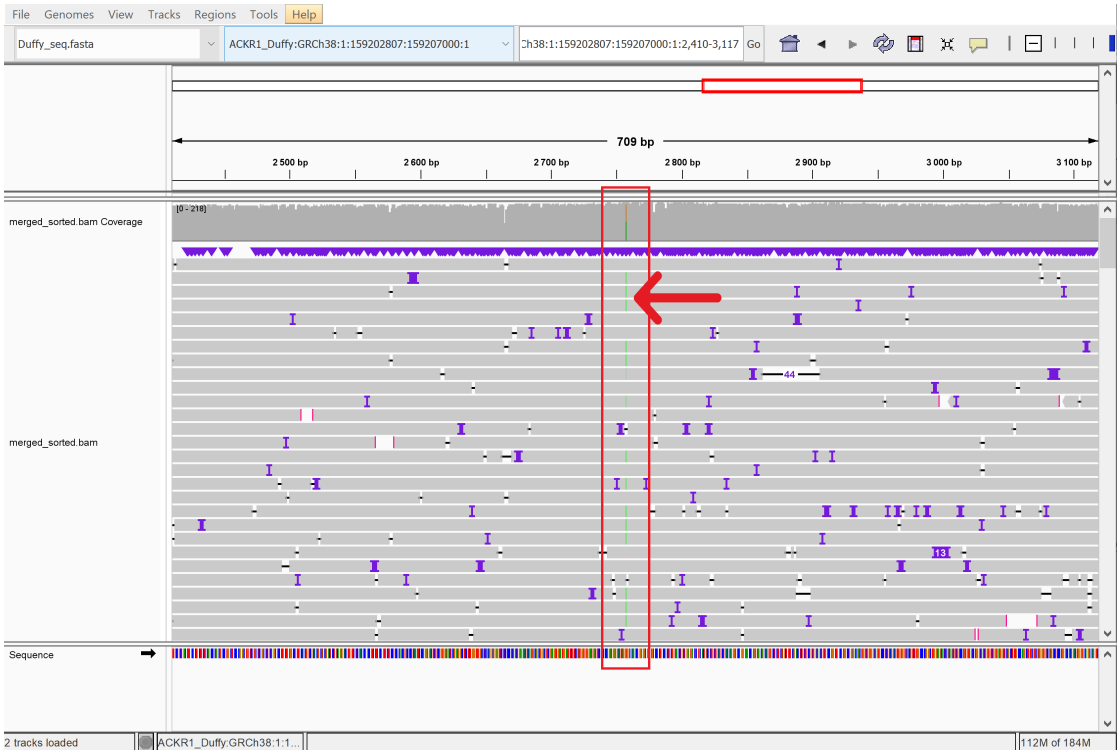



FIGURE 4.2.1 – Visualisation des reads alignés sur le génome de référence Duffy avec le logiciel IGV. La flèche montre une mutation à la position 2 758 visible sur l'ensemble des reads

Cet alignement visuel pourrait être parcouru manuellement afin de chercher toutes les mutations pour les différentes positions. Cependant, il est préférable de procéder de façon automatique grâce à un variant caller (.vcf)[72].

Comme expliqué précédemment, un des outils utilisé dans l'alignement aboutit à la génération d'un fichier .vcf [2] Ce fichier sert d'outil d'analyse pour les médecins/laborantins.

### 4.2.1 Autopsie d'un fichier VCF

L'annexe G représente le contenu d'un fichier VCF. Par simplicité, l'exemple ci-dessous, plus petit, est considéré [9] :

1. Integrative Genomics Viewer : https ://software.broadinstitute.org/software/igv/
2. Le Variant Call Format ( VCF ) spécifie le format d'un fichier texte utilisé en bio-informatique pour stocker les variations de séquences génétiques[83].

```
##fileformat=VCFv4.0
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS ID  REF ALT  QUAL FILTER INFO                      FORMAT      Sample1        Sample2
2      10  id1 G   A    29   .      NS=2;DP=13;AF=0.5         GT:GQ:DP:HQ 0|0:48:1:52,51 1|0:48:8:51,51
2      20   .  T   A    3    q10    NS=2;DP=12;AF=0.017       GT:GQ:DP:HQ 0|0:46:3:58,50 0|1:3:5:65,3
2      30  id3 A   G,T  67   PASS   NS=2;DP=10;AF=0.333,0.667 GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2
```

Ce fichier donne des informations sur les variants trouvés dans un génome. Les lignes d'en-tête sont précédées de ##, notamment [9, 84, 22] :

- ##INFO : il décrit les informations que l'on collecte pour chaque variant. L'ensemble des informations possible varie avec l'outil utilisé, et ne sera pas décrit ici. Le format est le suivant.
  - ID (NS) : l'identifiant que l'on retrouvera dans le fichier.
  - Number (1) : le nombre de fois que l'on verra l'identifiant pour chaque variant (le point signifie "un nombre arbitraire").
  - Type (Integer) : le type d'information, comme un entier, un mot, etc.
  - Description (Number of Samples with Data) : une description de l'information.
- ##FILTER : l'ensemble des étapes de filtrage utilisées pour générer ce fichier.
  - ID (q10) : identifiant du filtrage.
  - Description (Quality below 10) : description du filtrage.
- ##FORMAT : données du génotypage trouvées pour chaque variant.
  - ID (GQ) : identifiant du génotypage.
  - Number (1) : le nombre de fois que l'on verra cette donnée pour chaque variant.
  - Type (Integer) : le type d'information, comme un entier, un mot, etc.
  - Description (Genotype Quality) : une description du génotypage.

Chaque ligne contient au moins 10 champs. Les champs 10 et plus décrivent les génotypages dans les différents échantillons [9, 84, 22].

1. (2) le chromosome
2. (10) la position sur le chromosome
3. (id1) l'identifiant du variant
4. (G) variant sur la référence
5. (A) variant sur les lectures
6. (29) score de qualité sur le variant (score Phred)

7. (.) filtres passés : PASS si tous les filtres sont passés ; un point si aucun n'est passé ; sinon on énumère la liste des filtres qui sont passés (tels que définis par les en-tête ##FILTER), séparés par un point-virgule.
8. (NS=2 ;DP=13 ;AF=0.5) Informations sur le variant, qui utilise la nomenclature définie dans les en-têtes INFO. Ici, il s'agit d'un variant que l'on observe dans les 2 échantillons, avec une profondeur de 13 lectures, et une fréquence de 50%.
9. (GT :GQ :DP :HQ) Format des données de génotypages pour chaque échantillon, séparés par des "deux-points".
10. (0|0 :48 :1 :52,51) Données de génotypage pour l'échantillon 1. Ici, nous avons le génotypage (GG), qualité du génotypage (score Phred), profondeur de séquençage (une lecture), qualité des haplotypes (un score Phred par haplotype).
11. et suivant Données de génotypage pour l'échantillon n.

En particulier, les champs *variant sur la référence* et *variant sur les lectures* intéressent les médecins. Cependant le variant sur les lectures n'est qu'une base. Or, dans la pratique, il peut y avoir plusieurs bases différentes représentées pour une même position à travers les lectures. Avoir le pourcentage de représentation de chacune de ces bases permettrait de faciliter l'interprétation du résultat. En voyant les pourcentages, un médecin pourra plus facilement trancher et dire qu'il a à faire à un patient qui a vraiment les deux formes du gène ou il pourra déterminer par exemple si un faible pourcentage rend la présence d'un des deux allèles négligeables. Ce serait une information pertinente pour aider à interpréter les résultats.

### 4.2.2 Autopsie d'un fichier SAM

Afin d'ajouter cette information dans le fichier .vcf, l'idée est d'utiliser le fichier BAM comme le fait IGV. Ce type de fichier est simplement la version binaire et compressée d'un format SAM.
Un fichier SAM (Sequence Alignment/Map) prend la forme suivante :

```
@SQ SN:chr1 LN:45
r001  99   chr1  7  30  8M2I4M1D3M =  37 39  TTAGATAAAGGATACTG *
r002  0    chr1  9  30  3S6M1P1I4M *  0  0   AAAAGATAAGGATA    *
r003  0    chr1  9  30  5S6M       *  0  0   GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004  0    chr1  16 30  6M14N5M    *  0  0   ATAGCTTCAGC       *
r003  2064 chr1  29 17  6H5M       *  0  0   TAGGC            * SA:Z:ref,9,+,5S6M,30,1;
r001  147  chr1  37 30  9M         =  7  -39 CAGCGGCAT         * NM:i:1
```

Son contenu se décompose en deux parties : l'en-tête et le corps[9]. L'en-tête donne des informations sur le génome ou le sur le mapping. Les lignes d'en-têtes commencent toutes par un @, suivi de deux lettres. La ligne @SQ SN :chr1 LN :45 se lit :

— @ : nous sommes dans un en-tête
— SQ : qui a trait à une séquence de référence (les chromomes)
— SN :chr1 : le nom d'une séquence est chr1
— LN :45 : sa taille est de 45 pb

Il existe beaucoup de type d'en-têtes différents. Le corps est un format tabulé :

1. (r001) le nom d'une lecture
2. (99) informations binaires sur la lecture. Un unique chiffre enregistre les informations suivantes : fragment mappé ou non, fragment d'un paired-end (ou d'un single-end), premier de la paire, etc. Pour savoir ce que signifie un nombre, il suffit d'aller sur le site http ://broadinstitute.github.io/picard/explain-flags.html.
3. (chr1) la séquence sur lequel est mappée la lecture ; on utilise * si la lecture n'est pas mappée
4. (7) position la plus 5' de la lecture
5. (30) qualité du mapping, soit 10 log10(p), où p est la probabilité, estimée par l'outil de mapping, que la lecture soit mappée à la mauvaise place
6. (8M2I4M1D3M) format CIGAR de la lecture (cf infra)
7. (=) séquence sur laquelle est mappé l'autre fragment, en cas de paired-end ; on utilise = si le chromosome est le même, et * si la lecture est single-end
8. (37) en cas de paired-end, osition la plus 5' de l'autre fragment ; on utilise 0 si l'on est en single-end, si l'autre fragment ne mappe pas, etc.
9. (39) en cas de paired-end, taille de la lecture, soit la différence entre la position la plus 5' du fragment 5' et la position la plus 3' du fragment 3'
10. (TTAGATAAAGGATACTG) séquence du fragment
11. (*) qualité du fragment (similaire au FASTQ) ; on utilise * si on ne souhaite pas renseigner ce champ
12. (SA :Z :ref,29,-,6H5M,17,0 ;) d'autres informations (cf infra)

### 4.2.3 Pseudo algorithme et implémentation

La librairie *pysam* permet de manipuler les données stockées dans un fichier BAM. Une façon simple d'accéder à celles-ci consiste à itérer sur chaque base d'une région spécifiée à l'aide de la méthode pileup(). Chaque itération renvoie une PileupColumn qui représente toutes les lectures dans le fichier SAM qui chevauchent une position particulière dans la séquence de référence. La liste des lectures est représentée sous forme d'objets PileupRead dans l'attribut PileupColumn.pileups [31] :

```
import pysam
samfile = pysam.AlignmentFile("ex1.bam", "rb" )
for pileupcolumn in samfile.pileup("chr1", 100, 120):
    print("\ncoverage at base %s = %s" % (pileupcolumn.pos, pileupcolumn.n))
    for pileupread in pileupcolumn.pileups:
        if not pileupread.is_del and not pileupread.is_refskip:
            # query position is None if is_del or is_refskip is set.
            print('\tbase in read %s = %s' %
                  (pileupread.alignment.query_name,
                   pileupread.alignment.query_sequence[pileupread.query_position]))

samfile.close()
```

Une implémentation consisterait à avoir :

— une méthode *processBamFile* qui traite le fichier BAM afin de compter les bases par position

— une méthode *isHomoHetero* qui détecte la présence de mutation pour une position

— une méthode intermédiaire *countpourcentbases* qui traduit le comptage de bases en % de bases pour une même position

— une méthode *modifyVCF* qui réécrit le fichier vcf sur base des mutations détectées et les bases comptabilisées

Dès lors un pseudo-algorithme pour *processBamFile* serait :

```
avg_base = {position: {%A,%G..}}

pour chaque pileupcolumn dans le bam:
    count_base = {A:0, G:0..}

    pour chaque lecture dans la colonne :
        comptabiliser la base dans count_base

    appeler isHomoHetero pour count_base

    si mutation, ajouter count_base dans avg_base a la position de
   pileupcolumn et en convertissant count_base en %

return avg_base
```

L'idée est de parcourir le fichier bam (qui reprend le résultat de l'alignement), et de regarder pour chaque position couverte, l'ensemble des lectures (la colonne) qui la chevauchent. Pour ces lectures chevauchant cette position, il faut compter le nombre de chaque type de bases afin d'avoir les proportions. Sur base de cette proportion appeler la fonction isHomoHetero qui détectera une mutation. S'il y a une mutation, alors on enregistre la position ainsi que les proportions, converties en %, des différentes bases. Le pseudo-algorithme de *isHomoHetero(count_base)* est :

```
calculer la somme du nombre de bases presentes dans count_base

si le nombre de base differentes est >= 2 :
    calculer le % de la base la plus representative par rapport au
   nombre total de base pour cette position

    si ce % est compris dans un intervalle, c'est une mutation
```

L'idée est d'additionner le nombre de bases différentes afin d'avoir le total de bases et ainsi la proportion en % de chacune. S'il y a plus de 1 base, il faut regarder si % de la base la plus représentative se situe dans une certaine intervalle. Si tel est le cas, alors il s'agit d'une mutation. Cette intervalle sera compris entre 40 et 70%.

Celui de *modifyVCF(avg_base)* est :

```
pour chaque position dans avg_base :
    chercher si elle est dans le fichier vcf, si oui :
        changer l'attribut ALT correspondant dans le fichier vcf par
   les bases et leur %
```

L'idée est ici de parcourir chaque mutation pour laquelle on a la position et les % de représentations des bases. Si la position est reprise dans le fichier .vcf, alors son attribut ALT de sa ligne correspondante est changé pour indiquer le % de chacune des bases.

L'implémentation de ces pseudo-codes est disponible à l'annexe H.

### 4.2.4 Résultats

Une fois l'implémentation faite, le corps du fichier .vcf généré par le pipeline, disponible à l'annexe I, devient (l'en-tête reste le même) :

| #CHROM | POS | ID | REF | ALT | QUAL | ... |
|---|---|---|---|---|---|---|
| ACKR1_Duffy :GRCh38 :... | 258 | . | G | A47% ;G52% ; | 141 | . |
| ACKR1_Duffy :GRCh38 :... | 548 | . | C | C59% ;G1% ;T38% ; | 134 | . |
| ACKR1_Duffy :GRCh38 :... | 943 | . | C | C43% ;T56% ; | 162 | . |
| ACKR1_Duffy :GRCh38 :... | 1527 | . | C | C45% ;T54% ; | 222 | . |
| ACKR1_Duffy :GRCh38 :... | 2324 | . | C | C56% ;T43% ; | 211 | . |
| ACKR1_Duffy :GRCh38 :... | 2758 | . | G | A53% ;G46% ; | 190 | . |

TABLE 4.7 – Corps d'un fichier .vcf amélioré pour indiquer le pourcentage de chaque base lors d'une mutation à une position précise

La colonne ALT (variant sur les lectures) est bien modifiée en indiquant la proportion de chaque base représentée au travers de l'ensemble des lectures couvrant la même position (colonne POS) sur le génome de référence.

Après la comparaison de plusieurs positions dans IGV avec les résultats obtenus dans le fichier .vcf modifié, certaines incohérences apparaissent. La figure 4.2.2 montre le nombre de chaque base pour la position 548 ouverte dans IGV.

Dans le fichier .vcf modifié, pour cette même position, la ligne correspondante est :

| #CHROM | POS | ID | REF | ALT | QUAL | ... |
|---|---|---|---|---|---|---|
| ACKR1_Duffy :GRCh38 :... | 548 | . | C | C59% ;G1% ;T38% ; | 134 | . |

TABLE 4.8 – Exemple d'une ligne d'un fichier .vcf après modification

La différence est donc de 2% pour les C et 1% pour les T. Celle-ci n'a pas encore trouvé d'explication valable, mais serait peut-être liée au fait que *pileupcolumn* ne considère que des reads selon un certain seuil de qualité ou bien du fait que les attributs *pileupread.is_del* and not *pileupread.is_refskip* doivent être faux. Dans tous les cas, ce pourcentage adapté semble toujours respecter les mêmes proportions. Elle est donc négligeable mais justifierait un point d'attention par la suite.

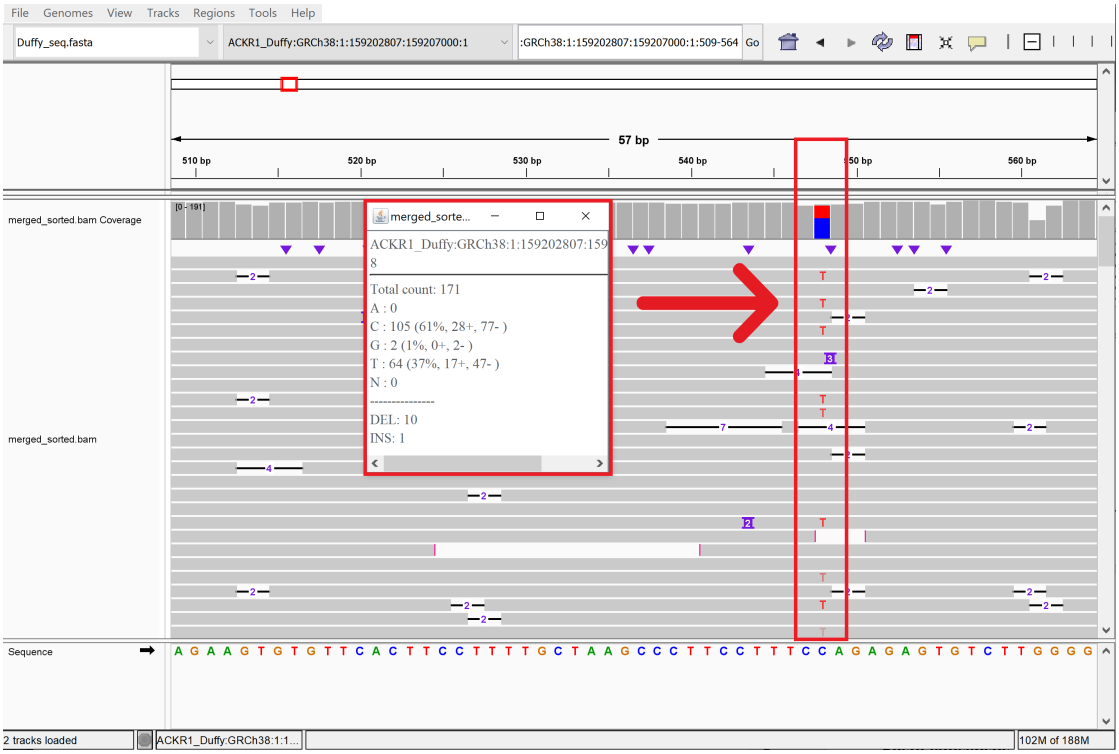



FIGURE 4.2.2 – Pourcentage de représentation des bases présente dans les alignements pour la position 548 : C : 105(61%), G : 2(1%), T : 64(37%)

### 4.2.5 Conclusion

Dans cette section c'est le fichier .vcf qui s'est vu être modifié afin de retenir la proportion de chaque base pour une position dans un gène d'un patient lorsque celle-ci comporte une mutation. Cette nouvelle information facilitera l'interprétation des médecins qui pourront plus facilement trancher et dire si un patient a vraiment les deux formes d'un gène ou il pourra déterminer par exemple si un faible pourcentage rend la présence d'un des deux allèles négligeables. Cette nouvelle information sera utilisée dans le point 4.4 pour identifier le typage érythrocytaire d'un individu.

## 4.3 Fasta consensus

Pour avoir une idée d'où la partie qui suit se trouve dans les étapes d'un pipeline, celui de l'annexe B peut être repris. Il s'agit alors de la phase d'analyse d'alignement. Une fois cet alignement effectué et les différentes lectures qui se chevauchent stockées dans un fichier SAM/BAM, une séquence consensus peut en être produite. Il s'agit d'une séquence idéalisée d'une région donnée d'un acide nucléique ou d'une protéine dans laquelle chaque position représente la base ou l'acide aminé rencontré le plus fréquemment[18]. Dans le cas où aucune lecture ne chevauche une position précise, dans la plupart des cas, la base de cette position "manquante" devient celle correspondante à

la même position du génome de référence. C'est le cas des séquences consensus générées par IGV. L'image 4.3.1, qui représente un alignement sur Duffy, illustre une zone non-couverte par des lectures (le cadre en rouge) et une zone couverte (le cadre en bleu)

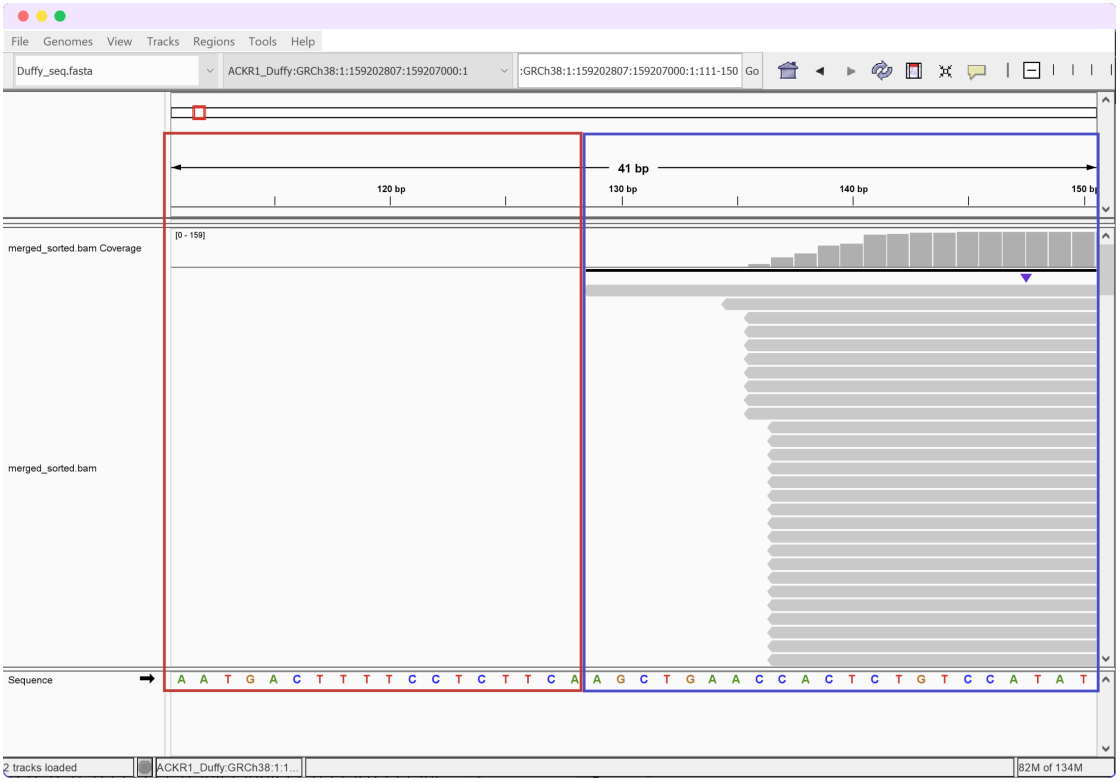



FIGURE 4.3.1 – Représentation IGV pour Duffy illustrant dans le cadre rouge une zone non-couverte par des alignements et en bleu une zone couverte par des alignements

L'exigence de la part de l'équipe du CHU est de garder ces "trous" dans le consensus généré. Un pseudo-code serait :

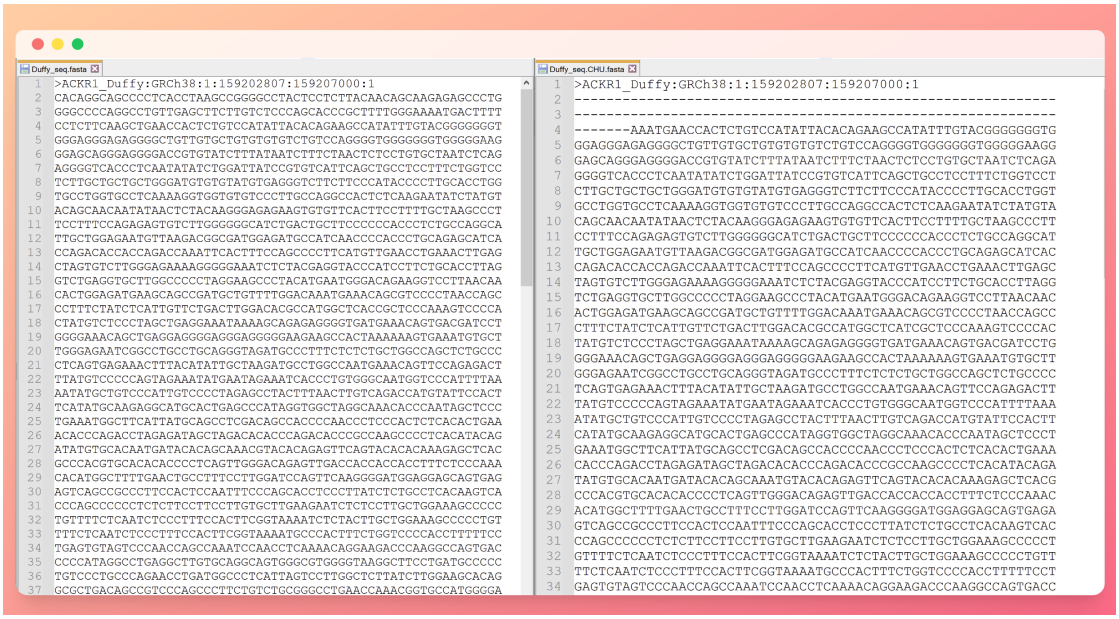
```
pour chaque position dans la reference :
    si il n'y a pas de pileupcolumn dans le bam de la même position:
        ajouter "-" dans le génome consensus

    sinon :
        count_base = {A:0, G:0..}
        pour chaque lecture de la colonne correspondante:
            comptabiliser la base dans "count_base"
        ajouter la base la plus representée de count_base dans le gé
   nome consensus

    renvoyer pileupcolumn
```

L'idée derrière cet algorithme est de parcourir chaque position de la référence et d'écrire le fasta consensus en même temps. Pour chacune des positions, il suffit de regarder si des lectures la chevauchent dans le fichier .bam. Si tel est le cas, la base majoritaire est ajoutée dans la nouvelle référence. Dans le cas contraire, un "-" est ajoutée afin de savoir qu'il s'agit d'une position non-couverte. L'implémentation de ce pseudo-algorithme se trouve à l'annexe J.

Une fois la solution implémentée et lancée sur le *fasta de Duffy* (fichier ouvert *à gauche* ci-dessous), le *nouveau consensus* (fichier ouvert *à droite*) devient :



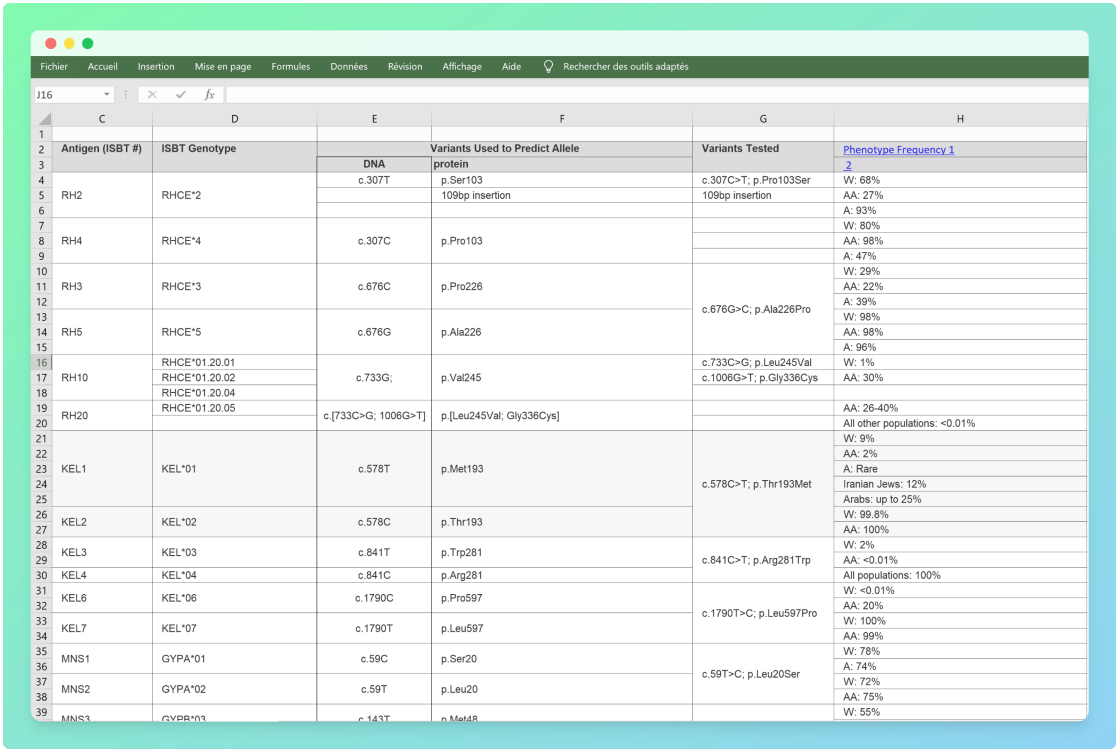

Dans le fichier de droite, les premières lignes qui contiennent des "-" correspondent au cadre rouge dans la figure 4.3.1, tandis que le reste au cadre bleu de cette même figure.

## 4.4 Identification du type érythrocytaire d'un patient

Pour avoir une idée d'où la partie qui suit se trouve dans les étapes d'un pipeline, celui de l'annexe B peut être repris. Il s'agit alors de la phase d'analyse en aval. Cette dernière partie consiste à identifier le type érythrocytaire d'un patient, c'est-à-dire la carte d'identité de ses systèmes d'antigènes.

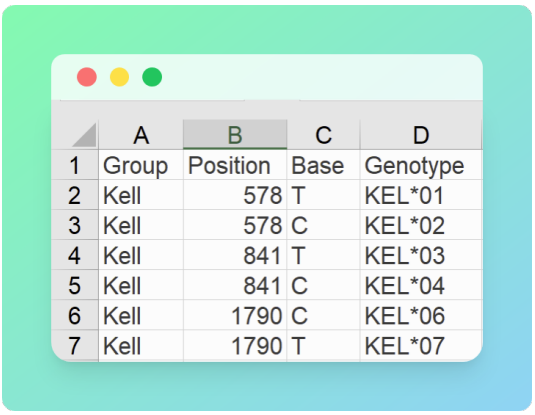
Habituellement, cette recherche se fait de manière manuelle. Les laborantins ont à disposition le fichier excel illustré à la figure 4.4.1. Ce fichier peut-être lu de la manière suivante : à la ligne 7,8,9, il s'agit de l'antigène RH4 (*colonne Antigen (ISBT )*) qui correspond au génotype RHCE*4 (*ISBT Genotype*) qui apparaît lorsqu'on observe une base C à la position 307 (*DNA*).

Le reste des colonnes est ignorée lors de cette analyse, elles constituent donc du bruit. Ce fichier pouvant être sujet à modification et afin d'éviter le bruit inutile de certaines colonnes, un nouveau fichier .xlsx en a été dérivé. Il est illustré à la figure 4.4.2.



| Antigen (ISBT #) | ISBT Genotype | Variants Used to Predict Allele | | Variants Tested | Phenotype Frequency 1 |
|---|---|---|---|---|---|
| | | DNA | protein | | 2 |
| RH2 | RHCE*2 | c.307T | p.Ser103 | c.307C>T; p.Pro103Ser | W: 68% |
| | | | 109bp insertion | 109bp insertion | AA: 27% |
| | | | | | A: 93% |
| RH4 | RHCE*4 | c.307C | p.Pro103 | | W: 80% |
| | | | | | AA: 98% |
| | | | | | A: 47% |
| RH3 | RHCE*3 | c.676C | p.Pro226 | | W: 29% |
| | | | | | AA: 22% |
| | | | | c.676G>C; p.Ala226Pro | A: 39% |
| RH5 | RHCE*5 | c.676G | p.Ala226 | | W: 98% |
| | | | | | AA: 98% |
| | | | | | A: 96% |
| RH10 | RHCE*01.20.01 | c.733G; | p.Val245 | c.733C>G; p.Leu245Val | W: 1% |
| | RHCE*01.20.02 | | | c.1006G>T; p.Gly336Cys | AA: 30% |
| | RHCE*01.20.04 | | | | |
| RH20 | RHCE*01.20.05 | c.[733C>G; 1006G>T] | p.[Leu245Val; Gly336Cys] | | AA: 26-40% |
| | | | | | All other populations: <0.01% |
| KEL1 | KEL*01 | c.578T | p.Met193 | | W: 9% |
| | | | | | AA: 2% |
| | | | | | A: Rare |
| | | | | c.578C>T; p.Thr193Met | Iranian Jews: 12% |
| | | | | | Arabs: up to 25% |
| KEL2 | KEL*02 | c.578C | p.Thr193 | | W: 99.8% |
| | | | | | AA: 100% |
| KEL3 | KEL*03 | c.841T | p.Trp281 | | W: 2% |
| | | | | c.841C>T; p.Arg281Trp | AA: <0.01% |
| KEL4 | KEL*04 | c.841C | p.Arg281 | | All populations: 100% |
| KEL6 | KEL*06 | c.1790C | p.Pro597 | | W: <0.01% |
| | | | | c.1790T>C; p.Leu597Pro | AA: 20% |
| KEL7 | KEL*07 | c.1790T | p.Leu597 | | W: 100% |
| | | | | | AA: 99% |
| MNS1 | GYPA*01 | c.59C | p.Ser20 | | W: 78% |
| | | | | c.59T>C; p.Leu20Ser | A: 74% |
| MNS2 | GYPA*02 | c.59T | p.Leu20 | | W: 72% |
| | | | | | AA: 75% |
| MNS3 | GYPB*03 | c.143T | p.Met48 | | W: 55% |

FIGURE 4.4.1 – Contenu d'un fichier excel utilisé pour identifier le type érythrocytaire des patients



| | A | B | C | D |
|---|---|---|---|---|
| 1 | Group | Position | Base | Genotype |
| 2 | Kell | 578 | T | KEL*01 |
| 3 | Kell | 578 | C | KEL*02 |
| 4 | Kell | 841 | T | KEL*03 |
| 5 | Kell | 841 | C | KEL*04 |
| 6 | Kell | 1790 | C | KEL*06 |
| 7 | Kell | 1790 | T | KEL*07 |

FIGURE 4.4.2 – Contenu d'un fichier excel amélioré de la version utilisée pour identifier le type érythrocytaire des patients

Sur base de ce fichier, et du .vcf amélioré dans la partie précédente (voir le tableau 4.7 et 4.8) un algorithme naïf afin de détecter le génotype d'un patient serait :

```
pour chaque position ligne du fichier excel:
    regarder si la base associée apparaît dans le fichier .vcf a la position correspondante pour un pourcentage supérieur à un certain seuil:
        si tel est le cas, le patient possède ce génotype
```

L'idée est de vérifier pour chaque position du fichier excel si la base associée apparaît dans le fichier .vcf à la position correspondante. Si c'est le cas et que la base est représentée au dessus d'un certain seuil, alors on enregistre le génotype correspondant. Bien évidemment, l'ajout de fonctions intermédiaires pour traiter le fichier xls et vcf serait pertinent. L'implémentation de ce pseudo-code ainsi que des fonctions intermédiaires se trouvent à l'annexe K.

Une fois la liste des génotypes d'un patient acquises, il suffit de générer un rapport au format .pdf afin qu'il puisse être envoyé aux médecins. La figure ci-dessous illustre un bout d'un rapport PDF au format A4 :

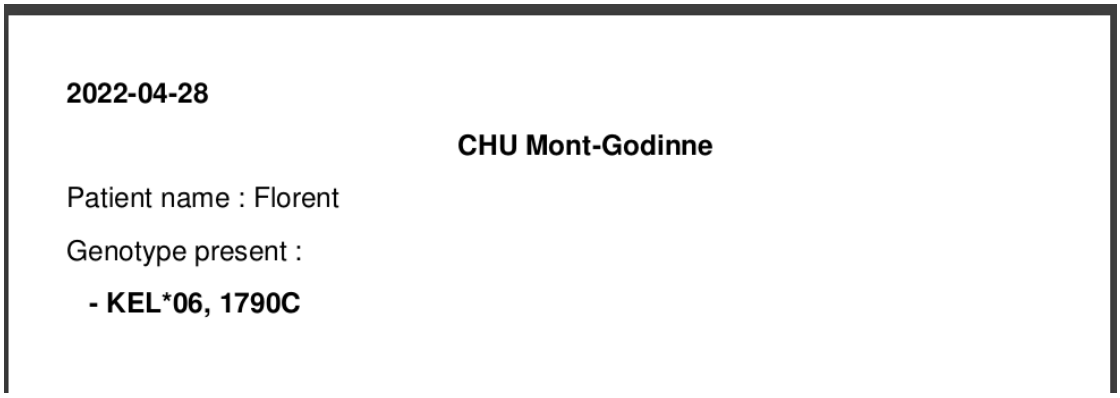



FIGURE 4.4.3 – Contenu d'une rapport au format .pdf reprenant le type érythrocytaire d'un patient

Ce rapport reprend la date à laquelle il a été fait, le nom du patient (ou tout autre moyen d'identification de celui-ci), la liste des génotypes présents dans ses globules rouges. Une amélioration consisterait à personnaliser ce rapport sur base du laboratoire qui effectue les tests, du médecin à qui ce document est destiné. Également créer une base de données SQL, SQL Lite ou autre type pour stocker les différentes mutations afin de faciliter la mise à jour de ses informations.

# Conclusion

Ce travail s'inscrivait dans le cadre du projet *"ROBIN : Hight level reconstruction of biological pipelines"*, une collaboration entre le CHU UCL Namur et l'université de Namur. Suite aux enjeux de l'identification du typage érythrocytaire d'individus, notamment pour la transfusion sanguine, la question était *"Comment adapter un pipeline de séquençage pour identifier le profil érythrocytaire d'individus ?"*. Notre contribution a été dans l'amélioration de l'automatisation d'un pipeline d'analyse de données de séquençage entreprit par le CHU mais aussi dans le développement d'outils pour venir enrichir ce dernier.

Dans un premier temps, c'est une approche pré-séquençage qui a été abordée. Le but était d'améliorer la qualité des amorces, nécessaires à délimiter les portions d'ADN à amplifier lors du séquençage, en vue d'un meilleur traitement et surtout favoriser un alignement/assemblage correct. L'outil développé à cette fin est une amélioration de l'outil primalscheme sur base des exigences de l'équipe du CHU. Il a nécessité une rétro-ingénierie de ce dernier afin de mieux intégrer les nouvelles fonctionnalités qui sont : la paramétrisation dynamique à travers une interface utilisateur, une amélioration d'un fichier tsv de sortie en indiquant les k-mers, et la détection des dimers d'amorces pour une vérification des amorces construites. Ainsi, l'outil développé a bien épousé les attentes de l'équipe du laboratoire du CHU. Celui-ci est aujourd'hui utilisé dans leur laboratoire pour la construction des amorces nécessaires au séquençage. Il pourrait s'adapter à n'importe quel génome de n'importe quelle espèce.

Il convenait ensuite de s'intéresser au pipeline qu'utilisait l'équipe du CHU pour l'analyse des données de séquençage. Les premières exigences étaient la mise au point d'un aligneur, permettant de générer une séquence d'ADN *consensus* à partir des résultats obtenus lors de l'étape du séquençage, et la modification du fichier vcf pour rapporter les proportions à chaque base donnant lieu à une mutation. Un parti fort a été adopté dans cette étape : celui d'améliorer, de mieux automatiser le pipeline existant sur base d'outils informatiques pouvant être utilisés pour la mise en place d'un pipeline de haute performance tel qu'un outil de gestion de flux de travail et de conteneurisation. Celui-ci a également été enrichi par une sélection des outils nécessaires à l'analyse de données de séquençage (alignement, polissage, détection de variants, etc.) pour obtenir des résultats de qualités.

L'amélioration de l'automatisation du pipeline existant est faite sur base du gestionnaire de flux de travail Nextflow qui intègre directement un système de conteneurisation Docker et Singularity nécessaire à la mise en production de l'outil. Conçu comme un outil de gestion de flux de travail, Nextflow offre une modélisation qui se compose d'un

ensemble de processus qui définissent les entrées, sorties et les commandes permettant de créer les sorties à partir des entrées. Le nouveau pipeline créé compte à ce jour onze processus nécessaires à l'analyse des données du typage érythrocytaire. Ces processus s'organisent en 5 étapes :

— L'étape d'alignement (positionnement des séquences générées par rapport à la séquence de référence stocké dans un fichier ".sam") composée de quatre processus : un processus pour le nettoyage des données de lectures nécessaires à l'alignement, un processus pour indexer le fichier fasta, un autre pour l'alignement proprement dit et un quatrième pour la conversion du fichier ".sam" en ".bam".

— L'étape du polissage composée d'un seul processus pour corriger les contigs bruts générés lors de l'alignement dans un fichier ".fasta".

— L'étape d'identification de variants de séquence (comparaison des séquences générées avec la séquence de référence, stockée dans un fichier ".vcf"). qui comporte trois processus : un premier processus pour imprimer les variants dans un fichier ".vcf" , un second processus pour imprimer les statiques de ce fichier (ceux-ci reprennent toutes les caractéristiques telles que Indels, SNP, hétérozygotes, homozygotes présentes dans un fichier vcf) et un troisième processus pour la modification du fichier ".vcf". Cette modification est faite sur base du fichier ".bam", de son fichier indexé ".bam.bai" et du fichier ".fasta" obtenu lors du processus de polissage.

— L'étape d'analyse de variants composée de deux processus : l'un pour la génération de la séquence consensus sur base de l'alignement effectué, et l'autre pour la comparaison entre le fichier de référence et la séquence consensus.

— L'étape de reporting. Sur base de l'analyse des données effectuées, il convenait de s'intéresser à la question d'identification du type érythrocytaire (la carte d'identité du patient) à partir d'une base de donnée reprenant les différentes mutations et le type érythrocytaire associé. Pour ce faire, un rapport d'analyse a été généré au format ".pdf" qui reprend dans un premier temps l'identification du patient et l'identification des génotypes auxquels il appartient. Cette étape ne comporte que le dernier processus du nouveau pipeline amélioré du CHU.

Afin de faciliter la mise en production et le déploiement, un système de conteneurisation (Docker et Singularity) est directement intégré dans le pipeline amélioré.

Tous les tests nécessaires à la validation du pipeline n'ont pas encore été clôturés par l'équipe du CHU. Celui-ci reste tout de même disponible et opérationnel pour l'analyse des données sorties d'un séquenceur pour le typage érythrocytaire.

Ce travail a permis de nous rendre compte de la richesse des pipelines, de l'unicité de chacun. Malgré un besoin précis, nous avons pu développer l'outil de conception d'amorces qui peut s'adapter à n'importe quel génome de n'importe quelle espèce. Concernant l'outil de conception d'amorces, ce que nous avons réalisé pourrait être complété par une analyse plus approfondie de la complexité en fonction des paramètres autres que les fichiers d'entrée. En ce qui concerne le pipeline développé, nous nous sommes concentrés sur deux étapes particulières : l'alignement et les analyses en aval telles que l'identification des variants d'intérêts. Le travail mériterait de pousser la réflexion en s'intéressant de près aux autres étapes, en particulier en faisant une analyse

approfondie des différents outils. La complexité des pipelines, dues à leurs grandes diversités et leurs nombreux outils utilisés, représente un enjeux majeur pour le futur de la bio-informatique. Il faudra beaucoup de temps avant que tous les domaines de la médecine de précision soient couverts.

# Bibliographie


[1] 16.5 : Primer-BLAST. Biology LibreTexts. 30 juill. 2019. URL : https://bio.libretexts.org/Bookshelves/Biotechnology/Bio-OER_(CUNY)/16%3A_Bioinformatics/16.05%3A_Primer-BLAST (visité le 27/03/2022).

[2] A DSL for parallel and scalable computational pipelines | Nextflow. URL : https://www.nextflow.io/ (visité le 25/04/2022).

[3] Acide ribonucléique. In : Wikipédia. Page Version ID : 189891726. 14 jan. 2022. URL : https://fr.wikipedia.org/w/index.php?title=Acide_ribonucl%C3%A9ique&oldid=189891726 (visité le 09/02/2022).

[4] ADN : définition, test, structure, rôle, maladies. URL : https://sante.journaldesfemmes.fr/fiches-anatomie-et-examens/2726491-adn-definition-test-structure-role-maladies/ (visité le 12/05/2022).

[5] Janine ALTMÜLLER, Birgit S. BUDDE et Peter NÜRNBERG. "Enrichment of target sequences for next-generation sequencing applications in research and diagnostics". In : Biological Chemistry 395.2 (1er fév. 2014). Publisher : De Gruyter, p. 231-237. ISSN : 1437-4315. DOI : 10.1515/hsz-2013-0199. URL : https://www.degruyter.com/document/doi/10.1515/hsz-2013-0199/html (visité le 18/05/2022).

[6] Shanika L. AMARASINGHE et al. "Opportunities and challenges in long-read sequencing data analysis". In : Genome Biology 21.1 (fév. 2020), p. 30. ISSN : 1474-760X. DOI : 10.1186/s13059-020-1935-5. URL : https://doi.org/10.1186/s13059-020-1935-5.

[7] V P ANTAO et I TINOCO. "Thermodynamic parameters for loop formation in RNA and DNA hairpin tetraloops." In : Nucleic Acids Research 20.4 (25 fév. 1992), p. 819-824. ISSN : 0305-1048. URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC312023/ (visité le 20/03/2022).

[8] ARN ribosomique : définition et explications. AquaPortail. URL : https://www.aquaportail.com/definition-14593-arn-ribosomique.html (visité le 24/05/2022).

[9] Ateliers « 2 en 1 » : Production et Analyses de données NGS | IFB. URL : https://ressources.france-bioinformatique.fr/fr/elearning/Atelier_NGS (visité le 21/04/2022).

[10] BAnQ numérique. URL : http://numerique.banq.qc.ca/ (visité le 18/03/2022).

[11] Bio-informatique, une révolution pour la cancérologie ? URL : https://www.fondation-arc.org/actualites/2015/bio-informatique-une-revolution-pour-la-cancerologie (visité le 06/02/2022).

[12] Bowtie : An ultrafast, memory-efficient short read aligner. URL : http://bowtie-bio.sourceforge.net/index.shtml (visité le 24/04/2022).

[13] Caroline BUOTE. "Application clinique du séquençage de l'exome pour le diagnostic moléculaire des syndromes polymalformatifs". In : (2015). Accepted : 2015-06-30T18 :34 :35Z pages7 Publisher : Université de Sherbrooke. URL : https://savoirs.usherbrooke.ca/handle/11143/6941 (visité le 18/03/2022).

[14] Burrows-Wheeler Aligner. URL : http://bio-bwa.sourceforge.net/ (visité le 24/04/2022).

[15] Sophie CHARGÉ et Kendra HODGKINSON. Le sang : concepts de base. fr. URL : https://professionaleducation.blood.ca/fr/transfusion/publications/le-sang-concepts-de-base (visité le 12/05/2022).

[16] Jian-Qun CHEN et al. "Variation in the Ratio of Nucleotide Substitution and Indel Rates across Genomes in Mammals and Bacteria". In : Molecular Biology and Evolution 26.7 (juill. 2009), p. 1523-1531. ISSN : 0737-4038. DOI : 10.1093/molbev/msp063. URL : https://doi.org/10.1093/molbev/msp063 (visité le 07/02/2022).

[17] Comment faire des amorces pour PCR / Science. La différence entre des objets et des termes similaires. URL : https://fr.sawakinome.com/articles/science/how-to-make-primers-for-pcr.html (visité le 27/03/2022).

[18] DALLAIRE L. Glossaire de Génétique Médicale et Moléculaire. URL : https://atlasgeneticsoncology.org/teaching/30143/glossaire-de-g-eacute;n-eacute;tique-m-eacute;dicale-et-mol-eacute;culaire (visité le 25/04/2022).

[19] Paolo DI TOMMASO et al. "Nextflow enables reproducible computational workflows". In : Nature biotechnology 35.4 (2017), p. 316-319.

[20] Différence entre l'acide désoxyribonucléique et l'acide ribonucléique. La différence entre des objets et des termes similaires. URL : https://fr.sawakinome.com/articles/molecular-biology/difference-between-deoxyribonucleic-acid-and-ribonucleic-acid.html (visité le 09/02/2022).

[21] Différence entre la structure de l'ADN et de l'ARN / Biologie moléculaire. La différence entre des objets et des termes similaires. URL : https://fr.sawakinome.com/articles/molecular-biology/difference-between-dna-and-rna-structure.html (visité le 13/05/2022).

[22] Susan FAIRLEY et al. “The International Genome Sample Resource (IGSR) collection of open human genomic variation resources”. In : Nucleic Acids Research 48.D1 (jan. 2020), p. D941-D947. ISSN : 0305-1048. DOI : 10.1093/nar/gkz836. URL : https://doi.org/10.1093/nar/gkz836 (visité le 22/04/2022).

[23] FASTA - Reference genome format. GATK. URL : https://gatk.broadinstitute.org/hc/en-us/articles/360035531652-FASTA-Reference-genome-format (visité le 09/05/2022).

[24] FASTQ. In : Wikipédia. Page Version ID : 171863457. 10 juin 2020. URL : https://fr.wikipedia.org/w/index.php?title=FASTQ&oldid=171863457 (visité le 10/02/2022).

[25] Formats de fichiers utilisés dans le NGS - PDF Free Download. URL : https://docplayer.fr/31993574-Formats-de-fichiers-utilises-dans-le-ngs.html (visité le 13/05/2022).

[26] GATK. URL : https://gatk.broadinstitute.org/hc/en-us (visité le 24/04/2022).

[27] Gènes et chromosomes - Fondamentaux. Manuels MSD pour le grand public. URL : https://www.msdmanuals.com/fr/accueil/fondamentaux/g%C3%A9n%C3%A9tique/g%C3%A8nes-et-chromosomes (visité le 17/03/2022).

[28] GENET - Les outils de génétique moléculaire. URL : http://genet.univ-tours.fr/gen001300_fichiers/CHAP5D/GEN05D1EC26.HTM (visité le 17/03/2022).

[29] Genomic DNA Extraction and Genomic DNA Isolation - BE. URL : https://www.thermofisher.com/uk/en/home/life-science/dna-rna-purification-analysis/genomic-dna-extraction.html (visité le 25/05/2022).

[30] Anne-Laure GEORGE-MOLLAND. “Innovation technique dans les studios d’animation et d’effets visuels : la Recherche et Développement au service du pipeline”. In : La Création Collective au Cinéma 2 (2019).

[31] Andreas HEGER et KEVIN JACOBS. Introduction — pysam 0.19.0 documentation. URL : https://pysam.readthedocs.io/en/latest/api.html#sam-bam-cram-files (visité le 22/04/2022).

[32] P. HOGEWEG. “The Roots of Bioinformatics in Theoretical Biology”. In : PLOS Computational Biology 7.3 (31 mars 2011). Publisher : Public Library of Science, e1002021. ISSN : 1553-7358. DOI : 10.1371/journal.pcbi.1002021. URL : https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002021 (visité le 19/05/2022).

[33] Human genome reference builds - GRCh38 or hg38 - b37 - hg19. GATK. URL : https://gatk.broadinstitute.org/hc/en-us/articles/360035890951-Human-genome-reference-builds-GRCh38-or-hg38-b37-hg19 (visité le 12/05/2022).

[34] Idée reçue : La bioinformatique, c'est l'analyse du génome humain. URL : https://interstices.info/idee-recue-la-bioinformatique-cest-lanalyse-du-genome-humain/ (visité le 23/05/2022).

[35] Indexed FASTA I/O — SeqAn master documentation. URL : https://seqan.readthedocs.io/en/master/Tutorial/InputOutput/IndexedFastaIO.html (visité le 09/05/2022).

[36] Integrative Genomics Viewer. URL : https://www.illumina.com/products/by-type/informatics-products/basespace-sequence-hub/apps/integrative-genomics-viewer.html (visité le 24/04/2022).

[37] Introduction — pysam 0.19.0 documentation. URL : https://pysam.readthedocs.io/en/latest/api.html (visité le 24/04/2022).

[38] Michael JACKSON, Kostas KAVOUSSANAKIS et Edward W. J. WALLACE. "Using prototyping to choose a bioinformatics workflow management system". In : PLOS Computational Biology 17.2 (25 fév. 2021). Publisher : Public Library of Science, e1008622. ISSN : 1553-7358. DOI : 10.1371/journal.pcbi.1008622. URL : https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008622 (visité le 26/05/2022).

[39] A KAHN. "Le code génétique et la spécificité des ARN de transfert". In : (1989).

[40] Thomas KARAOUZENE. "Bioinformatique et infertilité : analyse des données de séquençage haut-débit et caractérisation moléculaire du gène DPY19L2". Issue : 2017GREAS041. Theses. Université Grenoble Alpes, nov. 2017. URL : https://tel.archives-ouvertes.fr/tel-01738127 (visité le 22/04/2022).

[41] Daniel C. KOBOLDT et al. "The Next-Generation Sequencing Revolution and Its Impact on Genomics". In : Cell 155.1 (26 sept. 2013), p. 27-38. ISSN : 0092-8674. DOI : 10.1016/j.cell.2013.09.006. URL : https://www.sciencedirect.com/science/article/pii/S0092867413011410 (visité le 18/05/2022).

[42] Daniel C. KOBOLDT et al. "VarScan : variant detection in massively parallel sequencing of individual and pooled samples". In : Bioinformatics 25.17 (1er sept. 2009), p. 2283-2285. ISSN : 1367-4803. DOI : 10.1093/bioinformatics/btp373. URL : https://doi.org/10.1093/bioinformatics/btp373 (visité le 17/05/2022).

[43] Johannes KÖSTER et Sven RAHMANN. "Snakemake—a scalable bioinformatics workflow engine". In : Bioinformatics 28.19 (2012), p. 2520-2522.

[44] Martin KRAHN, Nicolas LÉVY et Marc BARTOLI. "Le séquençage de nouvelle génération (Next-Generation Sequencing, ou NGS) appliqué au diagnostic de maladies monogéniques hétérogènes - Notions essentielles pour le dialogue entre cliniciens et généticiens". In : Les Cahiers de Myologie 13 (1er juin 2016). Number : 13 Publisher : EDP Sciences, p. 31-33. ISSN : 2108-2219, 2496-1558. DOI : 10.1051/myolog/201613008. URL : https://www.cahiers-myologie.org/articles/myolog/abs/2016/01/myolog201613p31/myolog201613p31.html (visité le 13/05/2022).

[45] Manojkumar KUMARAN, Umadevi SUBRAMANIAN et Bharanidharan DEVARAJAN. "Performance assessment of variant calling pipelines using human whole exome sequencing and simulated data". In : BMC Bioinformatics 20.1 (17 juin 2019), p. 342. ISSN : 1471-2105. DOI : 10.1186/s12859-019-2928-9. URL : https://doi.org/10.1186/s12859-019-2928-9 (visité le 25/04/2022).

[46] L'alignement des séquences. URL : https://www.classicistranieri.com/wikipediaforschoolsfr/wp/s/Sequence_alignment.htm (visité le 08/02/2022).

[47] La conteneurisation, qu'est-ce que c'est ? URL : https://www.redhat.com/fr/topics/cloud-native-apps/what-is-containerization (visité le 26/05/2022).

[48] J. LAMORIL et al. "Les techniques de séquençage de l'ADN : une révolution en marche. Première partie". In : Immuno-analyse & Biologie Spécialisée 23.5 (oct. 2008), p. 260-279. ISSN : 0923-2532. DOI : 10.1016/j.immbio.2008.07.016. URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7147846/ (visité le 13/05/2022).

[49] Ben LANGMEAD et Jacob PRITT. Algorithms for DNA Sequencing. fr. URL : https://www.coursera.org/learn/dna-sequencing/home/info (visité le 25/03/2022).

[50] Le génotype : quelles sont les caractéristiques d'un gène | RN' Bio. URL : https://rnbio.upmc.fr/genetique_genotype1 (visité le 17/03/2022).

[51] Le séquençage d'ADN à haut débit en pratique clinique. DOI : 10.1016/j.arcped.2017.01.008. URL : https://reader.elsevier.com/reader/sd/pii/S0929693X17300337?token=3E4602EEE274F72B90141320B9826F3B5801C12741D5F200703D3%20E2B1DCB91451ECD0CD4B607B43164920ABB093FBBC9&originRegion=eu-west-1&originCreation=20220518095705 (visité le 18/05/2022).

[52] Heng LI. lh3/minimap2. original-date : 2017-07-18T15 :04 :53Z. 9 mai 2022. URL : https://github.com/lh3/minimap2 (visité le 09/05/2022).

[53] Heng LI. "Minimap2 : pairwise alignment for nucleotide sequences". In : Bioinformatics (Oxford, England) 34.18 (15 sept. 2018), p. 3094-3100. ISSN : 1367-4811. DOI : 10.1093/bioinformatics/bty191.

[54] Stacie LOFTUS. Primer. en. URL : https://www.genome.gov/genetics-glossary/Primer (visité le 17/03/2022).

[55] Steven M. CARR. Transitions vs transversions. URL : https://www.mun.ca/biology/scarr/Transitions_vs_Transversions.html (visité le 07/02/2022).

[56] Elaine R. MARDIS. "The impact of next-generation sequencing technology on genetics". In : Trends in Genetics 24.3 (1er mars 2008), p. 133-141. ISSN : 0168-9525. DOI : 10.1016/j.tig.2007.12.007. URL : https://www.sciencedirect.com/science/article/pii/S0168952508000231 (visité le 18/05/2022).

[57] Aude MASSÉ et Luc BUHANNIC. "Vers un traitement personnalisé de la dégénérescence maculaire liée à l'âge". In : Actualités Pharmaceutiques 56.565 (1er avr. 2017), p. 26-29. ISSN : 0515-3700. DOI : 10.1016/j.actpha.2017.02.007. URL : https://www.sciencedirect.com/science/article/pii/S0515370017300691 (visité le 12/05/2022).

[58] Francesca MERLIN. "Le « hasard évolutionnaire » de toute mutation génétique, ou la vision consensuelle de la Synthèse Moderne". In : Bulletin dhistoire et depistemologie des sciences de la vie 18.1 (2011). Bibliographie_available : 0 Cairndomain : www.cairn.info Cite Par_available : 0 Publisher : Éditions Kimé, p. 79-108. ISSN : 1279-7243. URL : https://www.cairn.info/revue-bulletin-d-histoire-et-d-epistemologie-des-sciences-de-la-vie-2011-1-page-79.htm (visité le 24/05/2022).

[59] Etienne MULLER. "Les défis du séquençage à haut débit dans l'exploration génétique des cancers du sein et de l'ovaire." In : (), p. 259.

[60] Multiplex PCR method for MinION and Illumina sequencing of Zika. URL : https://www.nature.com/articles/nprot.2017.066 (visité le 30/05/2022).

[61] NAHOY. Le séquençage, une histoire de générations. blog bioinformatique communautaire scientifique. 5 juin 2012. URL : https://bioinfo-fr.net/le-sequencage (visité le 17/03/2022).

[62] NovoAlign | Novocraft. URL : http://www.novocraft.com/products/novoalign/ (visité le 24/04/2022).

[63] PrimalScheme : panels d'amorces pour la PCR multiplex. URL : https://primalscheme.com/ (visité le 27/03/2022).

[64] Primer Designer Tool for PCR & Sequencing - BE. URL : //www.thermofisher.com/uk/en/home/life-science/sequencing/sanger-sequencing/pre-designed-primers-pcr-sanger-sequencing.html (visité le 27/03/2022).

[65] Principe de la PCR. URL : http://www.ens-lyon.fr/RELIE/PCR/principe/principe.htm (visité le 18/05/2022).

[66] Isabelle QUINKAL. "Quelques termes-clef de biologie moléculaire et leur définition". In : INRIA Rhône-Alpes (2003).

[67] Frédérique ROUSTANT. Définition ARN - ARM. Futura. Section : médecine. URL : https : / / www . futura - sciences . com / sante / definitions/medecine-arn-messager-98/ (visité le 24/05/2022).

[68] Steve ROZEN et Helen SKALETSKY. "Primer3 on the WWW for general users and for biologist programmers". In : Bioinformatics methods and protocols. Springer, 2000, p. 365-386.

[69] W RYCHLIK et R E RHOADS. "A computer program for choosing optimal oligonucleotides for filter hybridization, sequencing and in vitro amplification of DNA". In : Nucleic acids research 17.21 (1er nov. 1989), p. 8543-8551. ISSN : 1362-4962. DOI : 10 . 1093 / nar / 17 . 21 . 8543. URL : https : / / europepmc.org/articles/PMC335026 (visité le 20/03/2022).

[70] Chadi SAAD. "Caractérisation des erreurs de séquençage non aléatoires, application aux mosaïques et tumeurs hétérogènes". In : (), p. 193.

[71] Samtools. URL : http://www.htslib.org/ (visité le 24/04/2022).

[72] Sacha SCHUTZ. Analyse de génomes de SARS-CoV-2. URL : https : / / dridk.me/covid_ngs.html (visité le 22/04/2022).

[73] Andrew SEVERIN. Introduction to NextFlow. en. URL : https : / / bioinformaticsworkbook . org / dataAnalysis / nextflow / 01_ introductionToNextFlow.html (visité le 02/06/2022).

[74] Osman Ugur SEZERMAN et al. Bioinformatics Workflows for Genomic Variant Discovery. Publication Title : Bioinformatics Tools for Detection and Clinical Interpretation of Genomic Variations. IntechOpen, 14 juin 2019. ISBN : 978-1-78923-800-6. DOI : 10 . 5772 / intechopen . 85524. URL : https : //www.intechopen.com/chapters/67673 (visité le 25/04/2022).

[75] Ashley SHADE et Tracy K TEAL. "Computing workflows for biologists : a roadmap". In : PLoS biology 13.11 (2015), e1002303.

[76] Ivan SOVIC. Racon. original-date : 2016-02-23T15 :41 :42Z. 5 mai 2022. URL : https://github.com/isovic/racon (visité le 10/05/2022).

[77] Jamie K. TEER et al. "Systematic comparison of three genomic enrichment methods for massively parallel DNA sequencing". In : Genome Research 20.10 (10 jan. 2010). Company : Cold Spring Harbor Laboratory Press Distributor : Cold Spring Harbor Laboratory Press Institution : Cold Spring Harbor Laboratory Press Label : Cold Spring Harbor Laboratory Press Publisher : Cold Spring Harbor Lab, p. 1420-1431. ISSN : 1088-9051, 1549-5469. DOI : 10.1101/gr. 106716.110. URL : https://genome.cshlp.org/content/20/ 10/1420 (visité le 18/05/2022).

[78] Theories and Formulas. URL : http : / / www . premierbiosoft . com / netprimer/netprlaunch/Help/Theories_and_Formulas.htm (visité le 20/03/2022).

[79] Paolo Di TOMMASO et al. “Nextflow : un outil efficace pour l’amélioration de la stabilité numérique des calculs en analyse génomique”. In : Biologie Aujourd’hui 211.3 (2017). Number : 3 Publisher : EDP Sciences, p. 233-237. ISSN : 2105-0678, 2105-0686. DOI : 10.1051/jbio/2017029. URL : https://www.biologie-journal.org/articles/jbio/abs/2017/03/jbio170029/jbio170029.html (visité le 25/04/2022).

[80] Itamar TURNER-TRAURING. Docker vs. Singularity for data processing. PythonSpeed. 11 mars 2020. URL : https://pythonspeed.com/articles/containers-filesystem-data-processing/ (visité le 09/05/2022).

[81] Tuto Docker - Démarrer Docker (Partie 2) - Wanadev. WanadevDigital - Agence de développement d’applications au service de la digitalisation des entreprises. URL : https://www.wanadev.fr/24-tuto-docker-demarrer-docker-partie-2/ (visité le 09/05/2022).

[82] Peter M. VALLONE et John M. BUTLER. “AutoDimer : a screening tool for primer-dimer and hairpin structures”. In : BioTechniques 37.2 (août 2004). Publisher : Future Science, p. 226-231. ISSN : 0736-6205. DOI : 10.2144/04372ST03. URL : https://www.future-science.com/doi/full/10.2144/04372ST03 (visité le 20/03/2022).

[83] Variant Call Format. fr. Page Version ID : 192171212. Mars 2022. URL : https://fr.wikipedia.org/w/index.php?title=Variant_Call_Format&oldid=192171212 (visité le 21/04/2022).

[84] VCF (Variant Call Format) version 4.0 | 1000 Genomes. URL : https://www.internationalgenome.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-40/ (visité le 22/04/2022).

[85] VCFtools. URL : http://vcftools.sourceforge.net/ (visité le 24/04/2022).

[86] Vég-Di@g - PCR. URL : http://ephytia.inra.fr/fr/C/23575/Veg-Di-g-PCR (visité le 27/03/2022).

[87] Génome - Définition et Explications. Techno-Science.net. URL : https://www.techno-science.net/glossaire-definition/Genome.html (visité le 07/02/2022).

[88] Séquençage - Définition et Explications. Techno-Science.net. URL : https://www.techno-science.net/glossaire-definition/Sequencage.html (visité le 06/02/2022).

# Annexes

# Annexe A

# Opérations informatiques basiques sur l'ADN

Supposons deux mots de taille 4, s = 'ATCG' et t = 'GGTT' définis sur cet alphabet. Voici quelques opérations élémentaires qui pourraient être effectuées dessus dans un langage de programmation tel que Python :

| | |
|---|---|
| $len(s) \rightarrow 4$ | retourne la taille du mot s |
| $len('') \rightarrow 0$ | retourne la taille du mot vide |
| $s[0] \rightarrow A$ | retourne le caractère situé à la position 0 dans le mot s |
| $s[2] \rightarrow T$ | retourne le caractère situé à la position 2 dans le mot s |
| $s + t \rightarrow ATCBGGTT$ | la concaténation de s et t |
| $s[1, 2] \rightarrow T$ | renvoie les caractères se situant de la position 1 jusqu'à la 2 |
| $s[0 : 2]$ ou $s[: 2] \rightarrow AT$ | renvoie le préfixe jusqu'à la position 2 (non-comprise) |
| $s[-2 :] \rightarrow B$ | renvoie les deux derniers caractères |
| $s[2 :] \rightarrow CB$ | renvoie le suffix composé de 2 caractères |
| $s[2 : 4] \rightarrow CB$ | renvoie le caractère de la position 2 à la position 4 |

Dans ces opérations élémentaires, la concaténation prend en compte deux strings. Il pourrait dès lors être intéressant de définir des fonctions utiles prenant en compte 2 strings.

Par exemple un code qui retourne le plus long préfixe commun à deux strings.

```
"""
Retourne le plus long préfixe commun chaînes de caractères s1 & s2
Param:
- s1 et s2 deux chaînes de caractères
Return:
- le plus long préfixe commun chaînes de caractères s1 et s2
"""
def longuestCommonPrefix(s1, s2):
i = o
while i < len(s1) and i < len(s2) and s1[i] == s2[i]:
  i += 1
return s1[:i]
```

Dans le cas de l'ADN, une fonction qui retourne le complément d'une séquence sur base des compléments de bases est également pertinente :

```
"""
Renvoie le complément d'ADN de s
Param:
- s un string défini sur un alphabet {A,T,G,C}
Return:
- le complément de s
"""
def reverseComplement(s):
  complement = {'A': 'T', 'T': 'A', 'G': 'C', 'C': 'G'}
    t = ''
    for base in s :
      t = complement[base] + t
    return t
```

Il est important de noter l'utilisation d'une structure de données appelée **dictionnaire** qui associe à chaque clé une valeur.

# Annexe B

# Pipeline d'analyse à lectures longues

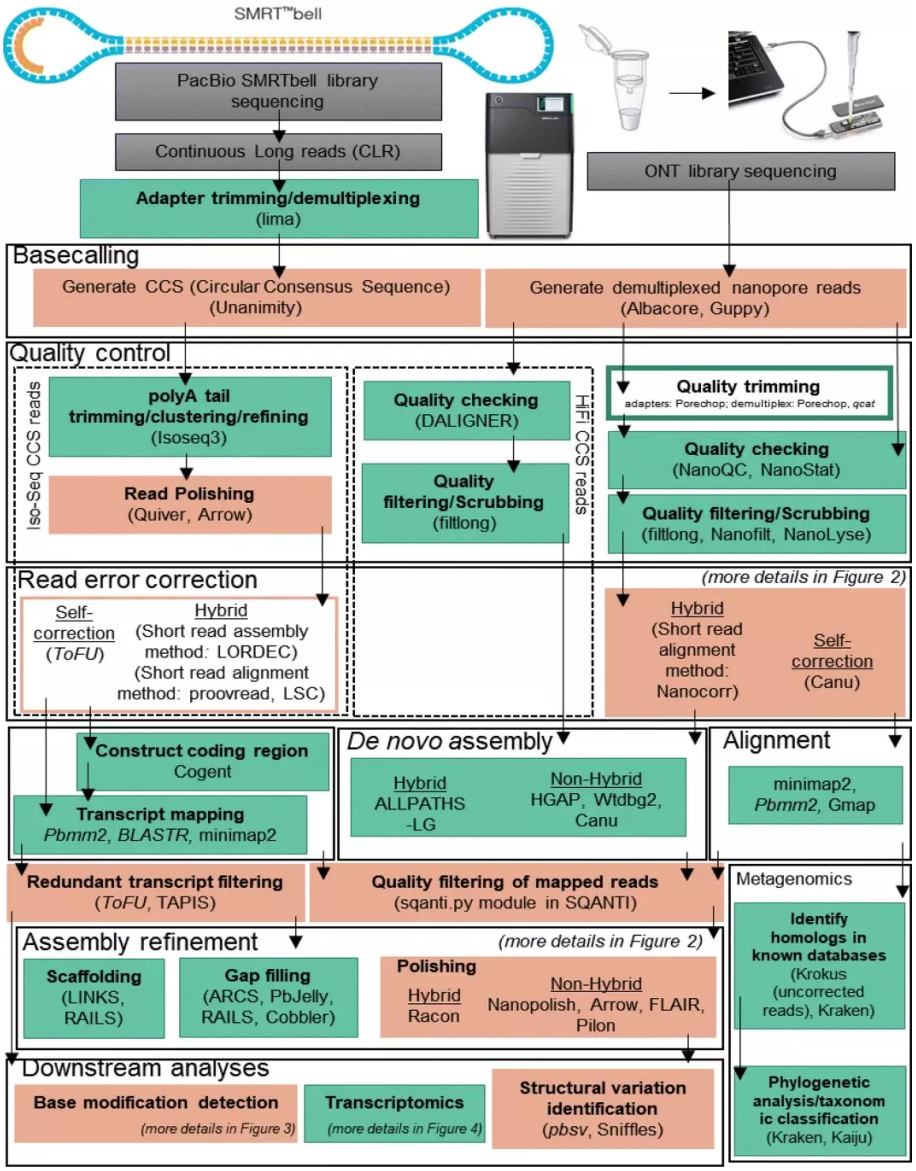
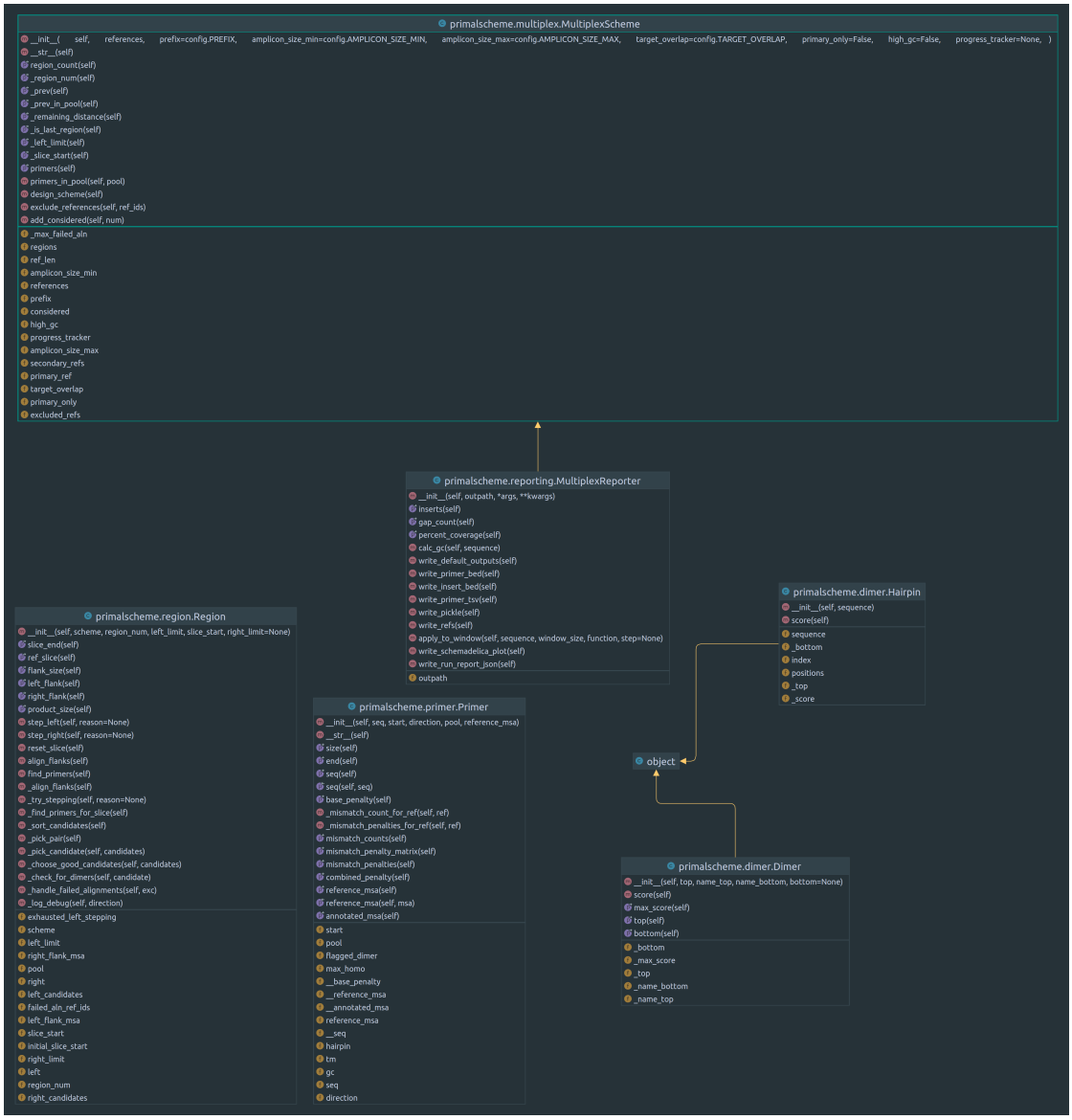



FIGURE B.1 – Vue d'ensemble des outils d'analyse utilisés dans un pipeline d'analyse à lecture longue pour des données de troisième génération

Pipelines d'analyse à lecture longue typiques pour les données SMRT et nanopores. Six étapes principales sont identifiées à travers la présentation flux de travail (appel de base, contrôle qualité, correction des erreurs de lecture, assemblage/alignement, raffinement de l'assemblage et analyses en aval). Les cases de couleur verte représentent les processus communs aux analyses à lecture courte et à lecture longue. Les cases de couleur orange représentent les processus propres aux analyses à lecture longue. Les cases vides représentent des étapes facultatives. Les outils couramment utilisés pour chaque étape de l'analyse à lecture longue sont entre parenthèses. Les italiques signifient des outils développés par les sociétés PacBio ou ONT, et les non-italiques signifient des outils développés par des parties externes. Les flèches représentent la direction du flux de travail [6].

# Annexe C

# Diagramme de classe de primalscheme



# Annexe D

# Temps d'exécution en fonction de la taille des gènes

Cet annexe reprend les données utilisés pour réaliser le graphique D.1 repris dans le point *A. Croissance de la taille des gènes* de la section 3.2.4. Primalscheme a été exécuté sur le génome bactérien, avec les mêmes paramètres mais sur des gènes existants de tailles différentes. Pour chaque exécution, le temps (en secondes) a été enregistré dans le but de voir l'évolution du temps d'exécution de primalscheme en fonction de la taille du gène.

Les paramètres sont :

| | | |
|---|---|---|
| *Primer Size Min* | *Primer Size Opt* | *Primer Size Max* |
| 19 | 22 | 34 |
| *Primer Tm Min* | *Primer Tm Opt* | *Primer Tm Max* |
| 59,5 | 61,0 | 62,5 |
| *Product Size Min* | *Product Size Max* | *Target Overlap* |
| 380 | 420 | 42 |

Le tableau D.1 reprend le nom de différents gènes avec pour chacun la taille du fichier .fasta ainsi que le temps qu'a duré l'exécution de primalscheme.

Le graphique D.1 représente l'évolution de ce temps d'exécution en fonction de la taille du gène. La relation est linéaire.

| Nom gène | Taille (Kib) | Temps (en secondes) |
|---|---|---|
| VEL | 5,9 | 24 |
| Duffy | 5,9 | 29 |
| FUT1 | 11,6 | 53 |
| Gata1 | 12,1 | 58 |
| FUT2 | 15,2 | 80 |
| Lutheran | 18,5 | 82 |
| KEL | 30,8 | 174 |
| GloB | 31,2 | 214 |
| GypB | 33,6 | 247 |
| P1PK | 41,7 | 268 |
| GypA | 44,9 | 350 |
| Gype | 49,4 | 360 |
| ABO | 60,5 | 432 |
| Kx | 65,4 | 561 |
| RHD | 83,0 | 592 |
| Kidd | 91,9 | 683 |
| RCHE | 95,3 | 704 |

TABLE D.1 – Listes des gènes avec leur nom, taille et nombre de bases

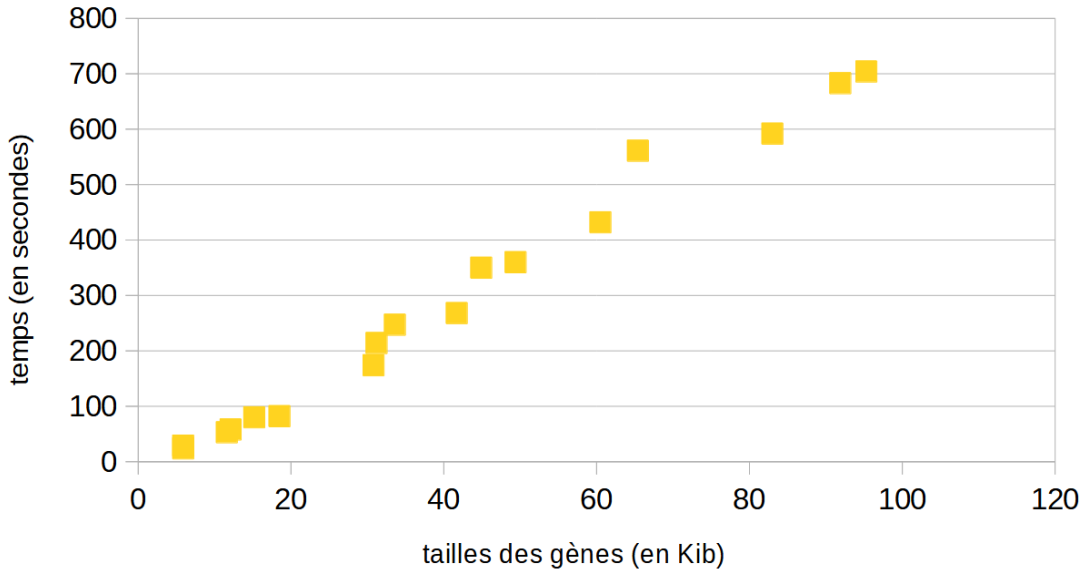



FIGURE D.1 – Évolution du temps de calcul de primalscheme en fonction de la taille des gènes

# Annexe E

# Temps d'exécution en fonction de la taille des génomes

Cet annexe reprend les données utilisés pour réaliser le graphique E.1 repris dans le point *B. Croissance de la taille des génomes de référence* de la section 3.2.4. Primalscheme a été exécuté sur le gène RHD avec les mêmes paramètres mais avec des génomes existants de tailles différentes. Pour chaque exécution, le temps (en minutes) a été enregistré dans le but de voir l'évolution du temps d'exécution de primalscheme en fonction de la taille du génome de référence.

Les paramètres sont :

| *Primer Size Min* | *Primer Size Opt* | *Primer Size Max* |
|---|---|---|
| 20 | 25 | 30 |
| *Primer Tm Min* | *Primer Tm Opt* | *Primer Tm Max* |
| 59,5 | 61,0 | 62,5 |
| *Product Size Min* | *Product Size Max* | *Target Overlap* |
| 8000 | 8400 | 840 |

Le tableau E.1 reprend les différents génomes de référence avec pour chacun son nom, sa taille ainsi que le temps qu'a duré l'exécution.

Le graphique E.1 représente l'évolution de ce temps d'exécution en fonction de la taille du génome consensus. La relation est linéaire.

| Nom génome | Taille (millions de bases) | Temps (min) |
|---|---|---|
| Bactérie | 5,59628 | 0,31 |
| Arabidopsis thaliana | 119,763 | 5,01 |
| Drosera capensis | 263,788 | 11,1 |
| Populus trichocarpa | 434,29 | 17,18 |
| Catharanthus roseus | 522,654 | 20,31 |
| Datisca glomerata | 688,404 | 27,56 |
| Pueraria montana | 1386,38 | 54,13 |
| Sporobolus alterniflorus | 1525,94 | 61,13 |
| Cocos nucifera | 2102,42 | 75,31 |
| Mus musculus | 2588,62 | 103,37 |
| Human | 2864,72 | 123,73 |

TABLE E.1 – Liste des génomes consensus avec pour chacun son nom, sa taille, et le nombre de minutes que l'exécution de primalscheme prend pour ce génome, le gène RHD et des paramètres identiques

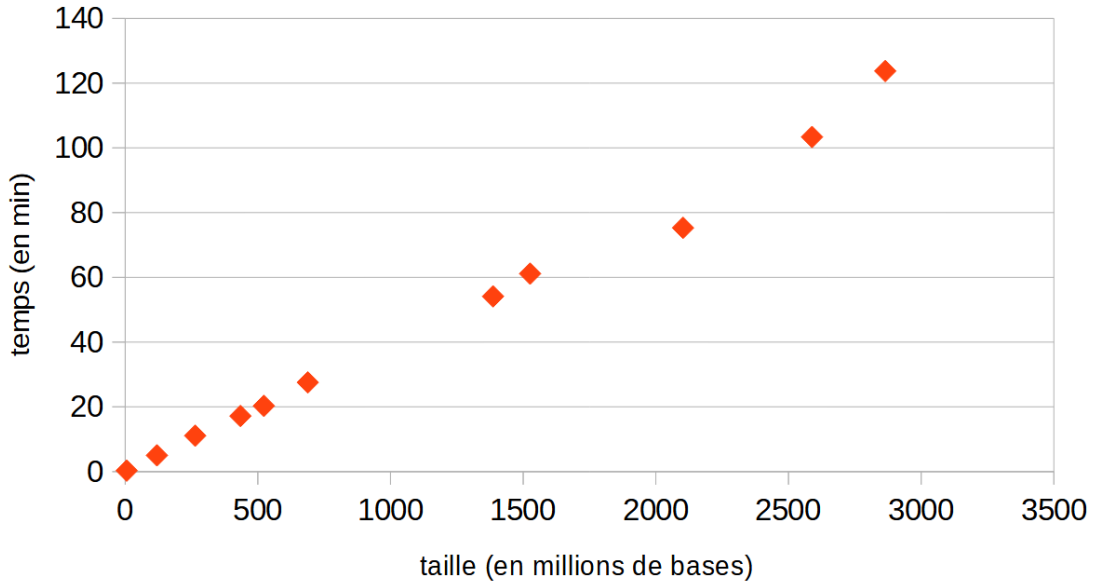



FIGURE E.1 – Évolution du temps de calcul de primalscheme en fonction de la taille du génome consensus

# Annexe F

# Pipeline commencé par le CHU

```
#NOTES:
# Le script nécessite l'installation des programmes suivants:
# - BBTools (https://jgi.doe.gov/data-and-tools/bbtools/)
# - FALCO (https://github.com/smithlabcode/falco)
# - trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic)
# - quast (https://github.com/ablab/quast)
# - MLST (https://github.com/tseemann/mlst)
# - Staramr (https://github.com/phac-nml/staramr)
# - Numactl (https://github.com/numactl/numactl)
#       - Unicycler (https://github.com/rrwick/Unicycler)
#       - Mummer3 (https://github.com/marbl/MUMmer3)
# - Masurca (https://github.com/alekseyzimin/masurca)
# - FASTQC (https://github.com/s-andrews/FastQC) Attention, en cas de "permission non accordée", l'exécutable FASTQC doit d'abord être "chmodé" (chmod +x fastqc)
#
# - les données brutes FASTQ du nanopore et de l'Illumina (forward et reverse) doivent se trouver dans des dossiers différents
# - les données Illumina dans un dossier "donnees_illumina" (attention aux majuscules)
# - les données Nanopore dans un dossier "donnees_nanopore" (idem, attention aux majuscules)
#
# L'analyse QC par QUAST (étape #9) nécessite de renseigner un génome de référence pour faire l'analyse. L'espèce de l'organisme ciblé est donné par SOURMASH. Cette référence doit être fournie par QUAST.

#0 indications des chemins des différents bin
export PATH=/home/labo-user/blast/ncbi-blast-2.12.0+/bin:$PATH
export PATH=/home/labo-user/Spades/SPAdes-3.15.2-Linux/bin:$PATH
export PATH=/usr/bin:$PATH
export PATH=/home/labo-user/BBTools/bbmap:$PATH
export PATH=/home/labo-user/Masurca/MaSuRCA-4.0.5/bin:$PATH
export PATH=/home/labo-user/fastqc/FastQC:$PATH

#1 decompresser et merger les fichiers FASTQ Nanopore en un seul
mkdir donnees_brutes
mkdir donnees_nettoyees
```

```
gzip -d *.gz
cat *.fastq > fastq_merged.fq

#2 nettoyage des reads Nanopore avec un mauvais score QC avec BBDUK (de la suite BBTools)
#NOTE: vérifier quel est le kit de barcoding utilisé, changer l'adapteur à enlever si nécessaire
trimmomatic SE -phred33 fastq_merged.fq fastq_merged_minlength.fq MINLEN:500
export PATH=/home/labo-user/BBTools/bbmap:$PATH
bbduk.sh qin=33 in=fastq_merged_minlength.fq out=fastq_merged_adaptor_removed.fq ref=/home/labo-user/Sequence_adaptor_nanopore/SQK_RBK110.fasta ktrim=l k=39 mink=31 hdist=0 tpe tbo
bbduk.sh qin=33 in=fastq_merged_adaptor_removed.fq out=fastq_merged_left_trimmed.fq ftl=10
bbduk.sh qin=33 in=fastq_merged_left_trimmed.fq out=fastq_merged_cleaned.fq maq=13
rm fastq_merged.fq
rm fastq_merged_left_trimmed.fq
rm fastq_merged_left_trimmed_clean.fq
rm fastq_merged_minlength.fq
mv fastq_merged_cleaned.fq donnees_nettoyees/
mv *.fastq donnees_brutes/

#4 création d'un fichier .bam et index pour lecture dans "IGV"
# NOTE 2: fichiers nécessaires: séquence de référence en fasta et fichier fastq mergé
# NOTE 3: nécessite l'installation de minimap2 et de samtools
export PATH=/home/labo-user/minimap2:$PATH
minimap2 -ax map-ont /home/labo-user/Typage_erythro/sequences_references/Duffy_seq.fasta donnees_nettoyees/fastq_merged_cleaned.fq > merged.sam
samtools view -bS merged.sam > merged.bam

#stop-pose problème


samtools sort merged.bam -o merged_sorted.bam
samtools index -b merged_sorted.bam
cp /home/labo-user/Typage_erythro/sequences_references/Duffy_seq.fasta Duffy_seq.fasta
#le fichier merged_sort.bam peut être ouvert dans IGV.
#Attention, la séquence exemple.fasta utilisée pour l'aligement doit être téléchargée dans IGV avec la fonction "genome uploaded from file"

#6 polissage par RACON (4 iterations)
mkdir racon/
bwa index Duffy_seq.fasta
bwa mem -t 14 -x ont2d Duffy_seq.fasta donnees_nettoyees/fastq_merged_cleaned.fq > racon/mapping1.sam
racon -m 8 -x -6 -g -8 -w 500 -t 14 donnees_nettoyees/fastq_merged_cleaned.fq racon/mapping1.sam Duffy_seq.fasta > racon/racon1.fasta

```

```
71 bwa index racon/racon1.fasta
72 bwa mem -t 14 -x ont2d racon/racon1.fasta donnees_nettoyees/
       fastq_merged_cleaned.fq > racon/mapping2.sam
73 racon -m 8 -x -6 -g -8 -w 500 -t 14 donnees_nettoyees/
       fastq_merged_cleaned.fq racon/mapping2.sam racon/racon1.fasta >
       racon/racon2.fasta
74
75 bwa index racon/racon2.fasta
76 bwa mem -t 14 -x ont2d racon/racon2.fasta donnees_nettoyees/
       fastq_merged_cleaned.fq > racon/mapping3.sam
77 racon -m 8 -x -6 -g -8 -w 500 -t 14 donnees_nettoyees/
       fastq_merged_cleaned.fq racon/mapping3.sam racon/racon2.fasta >
       racon/racon3.fasta
78
79 bwa index racon/racon3.fasta
80 bwa mem -t 14 -x ont2d racon/racon3.fasta donnees_nettoyees/
       fastq_merged_cleaned.fq > racon/mapping4.sam
81 racon -m 8 -x -6 -g -8 -w 500 -t 14 donnees_nettoyees/
       fastq_merged_cleaned.fq racon/mapping4.sam racon/racon3.fasta >
       racon/racon4.fasta
82
83 # conda activate medaka
84 # medaka_consensus -i donnees_nettoyees/fastq_merged_cleaned.fq -d
       racon/racon4.fasta -o medaka -t 14
85 # conda deactivate
86
87 #génération du fichier .vcf à partir du sorted bam
88
89 bwa index racon/racon4.fasta
90 samtools view -bS donnees_nettoyees/fastq_merged_cleaned.fq racon/
       mapping3.sam > mapping3.bam
91 samtools sort mapping3.bam -o mapping3_sorted.bam
92 samtools index -b mapping3_sorted.bam
93
94 bcftools mpileup -Ov -f racon/racon3.fasta mapping3_sorted.bam |
       bcftools call -mv -o sample3.vcf
95 # bcftools mpileup -Ov -f /home/labo-user/Typage_erythro/
       sequences_references/Duffy_seq.fasta merged_sorted.bam | bcftools
       call -mv -o merged_sorted.vcf
96
97 # export PATH=/home/labo-user/Mummer3/MUMmer3.23:$PATH
98 # export PATH=/home/labo-user/Mummer3/MUMmer3.23/aux_bin:$PATH
99 # nucmer --prefix=SNPsChr /home/labo-user/Typage_erythro/
       sequences_references/Duffy_seq.fasta racon/racon4.fasta
100 # show-snps -Clr SNPsChr.delta > SNPsChr.snps
101
102 #fin du programme
103
104 #7 generation la sequence consensus FASTA avec MEDAKA en utilisant la
        séquence polie avec RACON comme template
105
106 # mv racon4.fasta.fai medaka/
107 # mv racon4.fasta.mmi medaka/
108 # cd medaka/
109 # mv consensus.fasta resultats_medaka_consensus.fasta
110
```

```
111 # fin du programme
```

# Annexe G

# Contenu d'un long fichier .vcf

```
##fileformat=VCFv4.2
##FILTER=<ID=PASS,Description="All filters passed">
##bcftoolsVersion=1.7+htslib-1.7-2
##bcftoolsCommand=mpileup -Ov -f FUT1_seq.fasta merged_sorted.bam
##reference=file://FUT1_seq.fasta
##contig=<ID=FUT1:GRCh38:19:48747511:48755890:1,length=8380>
##ALT=<ID=*,Description="Represents allele(s) other than observed.">
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">
##INFO=<ID=IDV,Number=1,Type=Integer,Description="Maximum number of reads supporting an indel">
##INFO=<ID=IMF,Number=1,Type=Float,Description="Maximum fraction of reads supporting an indel">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Raw read depth">
##INFO=<ID=VDB,Number=1,Type=Float,Description="Variant Distance Bias for filtering splice-site artefacts in RNA
    -seq data (bigger is better)",Version="3">
##INFO=<ID=RPB,Number=1,Type=Float,Description="Mann-Whitney U test of Read Position Bias (bigger is better)">
##INFO=<ID=MQB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality Bias (bigger is better)">
##INFO=<ID=BQB,Number=1,Type=Float,Description="Mann-Whitney U test of Base Quality Bias (bigger is better)">
##INFO=<ID=MQSB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality vs Strand Bias (bigger
    is better)">
##INFO=<ID=SGB,Number=1,Type=Float,Description="Segregation based metric.">
##INFO=<ID=MQ0F,Number=1,Type=Float,Description="Fraction of MQ0 reads (smaller is better)">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="List of Phred-scaled genotype likelihoods">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##INFO=<ID=ICB,Number=1,Type=Float,Description="Inbreeding Coefficient Binomial test (bigger is better)">
##INFO=<ID=HOB,Number=1,Type=Float,Description="Bias in the number of HOMs number (smaller is better)">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes for each ALT allele, in the same
    order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##INFO=<ID=DP4,Number=4,Type=Integer,Description="Number of high-quality ref-forward , ref-reverse, alt-forward
    and alt-reverse bases">
##INFO=<ID=MQ,Number=1,Type=Integer,Description="Average mapping quality">
##bcftools_callVersion=1.7+htslib-1.7-2
##bcftools_callCommand=call -mv -o merged_sorted.vcf; Date=Tue Apr 19 11:32:24 2022
#CHROM  POS ID  REF ALT QUAL  FILTER  INFO  FORMAT  merged_sorted.bam
FUT1:GRCh38:19:48747511:48755890:1  75  .  C G,A 41.9001 .  DP=255;VDB=8.19059e-42;SGB=-0.693147;RPB=1;MQB=1;MQSB
```

```
  =0.944757;BQB=1;MQ0F=0.00392157;AC=1,1;AN=2;DP4=0,1,33,20;MQ=12  GT:PL 1/2:100,105,38,122,0,53
31 FUT1:GRCh38:19:48747511:48755890:1  90  . C T 45.2825  . DP=272;VDB=4.47437e-17;SGB=-0.693136;RPB=0.913962;MQB
    =0.998205;MQSB=0.632085;BQB=0.919253;MQ0F=0.00367647;ICB=1;HOB=0.5;AC=1;AN=2;DP4=33,15,25,10;MQ=13 GT:PL
    0/1:78,0,95
32 FUT1:GRCh38:19:48747511:48755890:1  93  . T C 160  . DP=266;VDB=3.92915e-38;SGB=-0.693147;MQSB=0.828488;MQ0F
    =0.0037594;AC=2;AN=2;DP4=0,0,56,25;MQ=12 GT:PL 1/1:190,241,0
33 FUT1:GRCh38:19:48747511:48755890:1  100 . T C 129  . DP=278;VDB=1.25549e-36;SGB=-0.693147;RPB=0.683979;MQB
    =0.999968;MQSB=0.942814;BQB=0.115195;MQ0F=0.00359712;AC=2;AN=2;DP4=4,2,60,25;MQ=13 GT:PL 1/1:156,178,0
34 FUT1:GRCh38:19:48747511:48755890:1  106 . G A 146  . DP=284;VDB=2.70976e-35;SGB=-0.693147;RPB=0.87037;MQB
    =0.401235;MQSB=0.651053;BQB=0.975309;MQ0F=0.00352113;AC=2;AN=2;DP4=2,0,52,29;MQ=12  GT:PL 1/1:173,197,0
35 FUT1:GRCh38:19:48747511:48755890:1  128 . A G 67  . DP=305;VDB=7.5519e-19;SGB=-0.693147;RPB=0.108071;MQB
    =0.735966;MQSB=0.493382;BQB=0.161871;MQ0F=0.00327869;ICB=1;HOB=0.5;AC=1;AN=2;DP4=32,6,32,17;MQ=13 GT:PL
    0/1:100,0,46
36 FUT1:GRCh38:19:48747511:48755890:1  132 . T C 145  . DP=238;VDB=3.45764e-27;SGB=-0.693147;RPB=0.313152;MQB
    =0.981184;MQSB=0.0932737;BQB=0.568275;MQ0F=0.00420168;AC=2;AN=2;DP4=5,2,52,18;MQ=12  GT:PL 1/1:172,117,0
37 FUT1:GRCh38:19:48747511:48755890:1  133 . G A 128  . DP=237;VDB=1.18827e-28;SGB=-0.693147;RPB=0.156945;MQB
    =0.19842;MQSB=0.16108;BQB=0.0163042;MQ0F=0.00421941;AC=2;AN=2;DP4=11,2,44,18;MQ=13 GT:PL 1/1:155,62,0
38 FUT1:GRCh38:19:48747511:48755890:1  134 . T C 123  . DP=310;VDB=1.23537e-25;SGB=-0.693147;RPB=0.454416;MQB
    =0.178555;MQSB=0.0748816;BQB=0.0270213;MQ0F=0.00322581;AC=2;AN=2;DP4=4,3,51,16;MQ=13 GT:PL 1/1:150,100,0
39 FUT1:GRCh38:19:48747511:48755890:1  154 . G A 181  . DP=327;VDB=2.81953e-37;SGB=-0.693147;RPB=0.971604;MQB
    =0.449755;MQSB=0.673395;BQB=0.250373;MQ0F=0.0030581;AC=2;AN=2;DP4=3,0,92,31;MQ=12  GT:PL 1/1:208,255,0
40 FUT1:GRCh38:19:48747511:48755890:1  165 . T C 81  . DP=322;VDB=3.15689e-34;SGB=-0.693147;RPB=0.916824;MQB
    =0.0256369;MQSB=0.437749;BQB=0.315476;MQ0F=0.00310559;AC=2;AN=2;DP4=13,3,83,20;MQ=12 GT:PL 1/1:108,121,0
41 FUT1:GRCh38:19:48747511:48755890:1  209 . T C 7.7815  . DP=368;VDB=3.1207e-13;SGB=-0.693147;RPB=0.907743;MQB
    =0.701319;MQSB=0.659155;BQB=0.118695;MQ0F=0.00271739;ICB=1;HOB=0.5;AC=1;AN=2;DP4=80,15,45,13;MQ=12  GT:PL
    0/1:40,0,94
42 FUT1:GRCh38:19:48747511:48755890:1  214 . A G 52  . DP=366;VDB=2.02694e-11;SGB=-0.693147;RPB=0.966601;MQB
    =0.651748;MQSB=0.714094;BQB=0.935588;MQ0F=0.00273224;ICB=1;HOB=0.5;AC=1;AN=2;DP4=64,14,60,13;MQ=12 GT:PL
    0/1:85,0,94
43 FUT1:GRCh38:19:48747511:48755890:1  223 . A C 59  . DP=297;VDB=6.4577e-11;SGB=-0.693147;RPB=0.679172;MQB
    =0.298531;MQSB=0.448627;BQB=0.286309;MQ0F=0.003367;AC=2;AN=2;DP4=4,0,76,19;MQ=11  GT:PL 1/1:92,114,6
44 FUT1:GRCh38:19:48747511:48755890:1  224 . A G 58  . DP=340;VDB=2.77369e-11;SGB=-0.693147;RPB=0.0589008;MQB
    =0.905565;MQSB=0.802165;BQB=0.92195;MQ0F=0.00294118;ICB=1;HOB=0.5;AC=1;AN=2;DP4=41,12,42,9;MQ=10  GT:PL
    0/1:91,0,55
```

```
45 FUT1:GRCh38:19:48747511:48755890:1  227 . T C 88  . DP=321;VDB=1.62252e-10;SGB=-0.693147;RPB=0.692198;MQB
     =0.3829;MQSB=0.677914;BQB=0.0195;MQ0F=0.00311526;AC=2;AN=2;DP4=16,5,69,18;MQ=12  GT:PL 1/1:115,76,0
46 FUT1:GRCh38:19:48747511:48755890:1  233 . G A 65  . DP=376;VDB=3.16728e-15;SGB=-0.693147;RPB=0.522176;MQB
     =0.998824;MQSB=0.996884;BQB=0.182249;MQ0F=0.00265957;ICB=1;HOB=0.5;AC=1;AN=2;DP4=57,13,66,15;MQ=11  GT:PL
     0/1:98,0,74
47 FUT1:GRCh38:19:48747511:48755890:1  260 . C G 97  . DP=337;VDB=2.16528e-25;SGB=-0.693147;RPB=0.449059;MQB
     =0.425726;MQSB=0.510427;BQB=0.0442033;MQ0F=0.00296736;AC=2;AN=2;DP4=10,3,91,18;MQ=11  GT:PL 1/1:124,172,0
48 FUT1:GRCh38:19:48747511:48755890:1  261 . C G 131 . DP=339;VDB=5.32897e-26;SGB=-0.693147;RPB=0.350266;MQB
     =0.989316;MQSB=0.576335;BQB=0.232892;MQ0F=0.00294985;AC=2;AN=2;DP4=2,1,95,19;MQ=12  GT:PL 1/1:158,255,0
49 FUT1:GRCh38:19:48747511:48755890:1  281 . T G 77  . DP=308;VDB=1.84405e-18;SGB=-0.693147;RPB=1;MQB=1;MQSB
     =0.474769;BQB=1;MQ0F=0.00324675;AC=2;AN=2;DP4=1,0,70,18;MQ=10  GT:PL 1/1:104,213,0
50 FUT1:GRCh38:19:48747511:48755890:1  290 . G A 132 . DP=314;VDB=2.56566e-34;SGB=-0.693147;MQSB=0.964136;MQ0F
     =0.00318471;AC=2;AN=2;DP4=0,0,89,24;MQ=10  GT:PL 1/1:162,255,0
51 FUT1:GRCh38:19:48747511:48755890:1  300 . A G 42.1461 . DP=313;VDB=8.76153e-21;SGB=-0.693147;RPB=0.940053;MQB
     =0.623163;MQSB=0.707024;BQB=0.250215;MQ0F=0.00319489;AC=2;AN=2;DP4=23,9,53,14;MQ=10  GT:PL 1/1:70,20,0
52 FUT1:GRCh38:19:48747511:48755890:1  302 . A G 13.473  . DP=315;VDB=1.93446e-17;SGB=-0.693147;RPB=0.937299;MQB
     =0.268642;MQSB=0.953727;BQB=0.760999;MQ0F=0.0031746;ICB=1;HOB=0.5;AC=1;AN=2;DP4=27,10,43,14;MQ=10  GT:PL
     0/1:46,0,19
53 FUT1:GRCh38:19:48747511:48755890:1  304 . T C 96  . DP=311;VDB=3.87646e-17;SGB=-0.693147;RPB=0.807927;MQB
     =0.826017;MQSB=0.959508;BQB=0.086617;MQ0F=0.00321543;AC=2;AN=2;DP4=4,1,67,19;MQ=10  GT:PL 1/1:123,195,0
54 FUT1:GRCh38:19:48747511:48755890:1  324 . C A 29.5125 . DP=291;VDB=5.63489e-16;SGB=-0.693147;RPB=0.736011;MQB
     =0.937416;MQSB=0.243048;BQB=0.945258;MQ0F=0;AC=2;AN=2;DP4=5,0,71,23;MQ=9  GT:PL 1/1:83,95,28
55 FUT1:GRCh38:19:48747511:48755890:1  332 . A G 21.4134 . DP=281;VDB=2.43706e-25;SGB=-0.693147;RPB=0.543635;MQB
     =0.984843;MQSB=0.149949;BQB=0.744996;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=29,6,38,15;MQ=9  GT:PL 0/1:54,0,19
56 FUT1:GRCh38:19:48747511:48755890:1  339 . G C 44.9633 . DP=190;VDB=1.50577e-22;SGB=-0.693147;RPB=0.354559;MQB
     =0.0588999;MQSB=0.472367;BQB=0.533052;MQ0F=0;AC=2;AN=2;DP4=3,0,47,17;MQ=8  GT:PL 1/1:72,109,0
57 FUT1:GRCh38:19:48747511:48755890:1  445 . A C 26.4287 . DP=209;VDB=2.18849e-15;SGB=-0.692352;MQSB=0.542747;MQ0F
     =0;AC=2;AN=2;DP4=0,0,3,18;MQ=7  GT:PL 1/1:56,60,0
58 FUT1:GRCh38:19:48747511:48755890:1  448 . C A 31.4175 . DP=218;VDB=2.52245e-15;SGB=-0.692831;MQSB=0.58632;MQ0F
     =0;AC=2;AN=2;DP4=0,0,6,18;MQ=7  GT:PL 1/1:61,72,0
59 FUT1:GRCh38:19:48747511:48755890:1  1364  . C A 122 . DP=122;VDB=1.0211e-40;SGB=-0.693147;RPB=0.820606;MQB
     =0.640918;MQSB=0.698343;BQB=0.970917;MQ0F=0.155738;AC=2;AN=2;DP4=3,0,42,26;MQ=20  GT:PL 1/1:149,110,0
60 FUT1:GRCh38:19:48747511:48755890:1  1371  . C G 163 . DP=188;VDB=0;SGB=-0.693147;RPB=0.898037;MQB=0.139647;MQSB
     =0.999428;BQB=0.547397;MQ0F=0.111702;AC=2;AN=2;DP4=5,1,71,46;MQ=20  GT:PL 1/1:190,230,0
```

```
61 FUT1:GRCh38:19:48747511:48755890:1  1379  . C T 31.6305  . DP=254;VDB=1.28038e-29;SGB=-0.693147;RPB=0.0112297;MQB
     =0.00207239;MQSB=0.862305;BQB=0.829629;MQ0F=0.0866142;ICB=1;HOB=0.5;AC=1;AN=2;DP4=79,33,58,14;MQ=21 GT:PL
     0/1:65,0,189
62 FUT1:GRCh38:19:48747511:48755890:1  1380  . A G 94  . DP=254;VDB=0;SGB=-0.693147;RPB=3.41805e-05;MQB=0.00141267;
     MQSB=0.613617;BQB=0.0864835;MQ0F=0.0866142;ICB=1;HOB=0.5;AC=1;AN=2;DP4=49,19,81,30;MQ=21  GT:PL 0/1:127,0,192
63 FUT1:GRCh38:19:48747511:48755890:1  1381  . T C 228  . DP=256;VDB=0;SGB=-0.693147;RPB=0.159848;MQB=0.595141;MQSB
     =0.826709;BQB=0.00688173;MQ0F=0.0898438;AC=2;AN=2;DP4=7,4,126,47;MQ=21 GT:PL 1/1:255,255,0
64 FUT1:GRCh38:19:48747511:48755890:1  1384  . A G 164  . DP=261;VDB=0;SGB=-0.693147;RPB=0.813943;MQB=0.0711222;MQSB
     =0.640299;BQB=0.0547105;MQ0F=0.0842912;AC=2;AN=2;DP4=13,8,113,46;MQ=22  GT:PL 1/1:191,174,0
65 FUT1:GRCh38:19:48747511:48755890:1  1388  . A T 228  . DP=265;VDB=0;SGB=-0.693147;RPB=0.380952;MQB=0.280423;MQSB
     =0.777863;BQB=0.0608466;MQ0F=0.0867925;AC=2;AN=2;DP4=2,0,140,49;MQ=22  GT:PL 1/1:255,255,0
66 FUT1:GRCh38:19:48747511:48755890:1  1396  . T C 176  . DP=271;VDB=0;SGB=-0.693147;RPB=0.0312898;MQB=0.400402;MQSB
     =0.512696;BQB=0.0294817;MQ0F=0.0664207;AC=2;AN=2;DP4=20,6,111,42;MQ=21  GT:PL 1/1:203,156,0
67 FUT1:GRCh38:19:48747511:48755890:1  1398  . C T 228  . DP=262;VDB=0;SGB=-0.693147;RPB=0.828089;MQB=0.995807;MQSB
     =0.543361;BQB=0.00418484;MQ0F=0.0877863;AC=2;AN=2;DP4=7,2,138,46;MQ=21 GT:PL 1/1:255,255,0
68 FUT1:GRCh38:19:48747511:48755890:1  1407  . C T 39.0061  . DP=276;VDB=6.7467e-37;SGB=-0.693147;RPB=0.13097;MQB
     =0.180311;MQSB=0.618599;BQB=0.473227;MQ0F=0.076087;ICB=1;HOB=0.5;AC=1;AN=2;DP4=95,32,42,17;MQ=21 GT:PL
     0/1:72,0,103
69 FUT1:GRCh38:19:48747511:48755890:1  1408  . A G 153  . DP=278;VDB=4.6439e-42;SGB=-0.693147;RPB=0.330736;MQB
     =0.794894;MQSB=0.548583;BQB=0.918903;MQ0F=0.0683453;ICB=1;HOB=0.5;AC=1;AN=2;DP4=57,24,73,30;MQ=21 GT:PL
     0/1:186,0,127
70 FUT1:GRCh38:19:48747511:48755890:1  1412  . T C 122  . DP=277;VDB=8.39956e-33;SGB=-0.693147;RPB=0.208349;MQB
     =0.34345;MQSB=0.773783;BQB=0.256567;MQ0F=0.0722022;ICB=1;HOB=0.5;AC=1;AN=2;DP4=71,27,67,18;MQ=23 GT:PL
     0/1:155,0,148
71 FUT1:GRCh38:19:48747511:48755890:1  1416  . A T 3.10166  . DP=280;VDB=5.99748e-13;SGB=-0.693147;RPB=0.000125614;
     MQB=0.16877;MQSB=0.94443;BQB=0.471555;MQ0F=0.075;ICB=1;HOB=0.5;AC=1;AN=2;DP4=71,26,54,26;MQ=22 GT:PL
     0/1:34,0,176
72 FUT1:GRCh38:19:48747511:48755890:1  1420  . C T 158  . DP=265;VDB=0;SGB=-0.693147;RPB=5.47397e-08;MQB=0.00111759;
     MQSB=0.996102;BQB=0.00143203;MQ0F=0.0716981;AC=2;AN=2;DP4=28,5,96,39;MQ=21  GT:PL 1/1:185,111,0
73 FUT1:GRCh38:19:48747511:48755890:1  1421  . T C 165  . DP=278;VDB=3.7658e-30;SGB=-0.693147;RPB=0.0177648;MQB
     =0.330407;MQSB=0.999151;BQB=0.00113304;MQ0F=0.0719424;ICB=1;HOB=0.5;AC=1;AN=2;DP4=62,19,65,24;MQ=21  GT:PL
     0/1:198,0,143
74 FUT1:GRCh38:19:48747511:48755890:1  1422  . G A 127  . DP=258;VDB=1.98244e-28;SGB=-0.693147;RPB=0.11098;MQB
     =0.150876;MQSB=0.954131;BQB=0.00207754;MQ0F=0.0852713;ICB=1;HOB=0.5;AC=1;AN=2;DP4=63,24,61,18;MQ=21 GT:PL
```

```
   0/1:160,0,140
FUT1:GRCh38:19:48747511:48755890:1  1429  . C G 130 . DP=282;VDB=2.19713e-31;SGB=-0.693147;RPB=0.0310305;MQB
   =0.0443378;MQSB=0.778573;BQB=0.955339;MQ0F=0.0815603;ICB=1;HOB=0.5;AC=1;AN=2;DP4=70,23,61,27;MQ=20  GT:PL
    0/1:163,0,137
FUT1:GRCh38:19:48747511:48755890:1  1430  . C T 64  . DP=288;VDB=1.41552e-32;SGB=-0.693147;RPB=0.78251;MQB
   =0.36887;MQSB=0.692442;BQB=0.254457;MQ0F=0.0798611;ICB=1;HOB=0.5;AC=1;AN=2;DP4=82,31,50,19;MQ=21  GT:PL
    0/1:97,0,140
FUT1:GRCh38:19:48747511:48755890:1  1439  . G C,T 194  . DP=292;VDB=5.71616e-18;SGB=-0.693147;RPB=0.12299;MQB=0.
```

# Annexe H

# Implémentation de la modification du fichier VCF

```
import pysam
# import matplotlib.pyplot as plt
from fuc import pyvcf
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from vcf_parser import VCFParser


"""
Converti base_count_dict en %

Param:
- base_count_dict (dict) : {'A': 25, 'G': 12, 'C': 13, 'T': 0}

Return:
- pourcentBases (dict) : {'A': '50%', 'G': '25%', 'C': '25%'}
"""
def countpourcentbases(base_count_dict):
    bases = []
    pourcentBases = {}
    sumBasesOccurences = 0
    # Vérifie si chaque base est dans le dictionnaire et additionne
    le nombre total
    for base in ['A', 'C', 'G', 'T']:
        assert base in base_count_dict, "Wrong dictionnary in
            argument"
        if base_count_dict[base] > 0:
            sumBasesOccurences += base_count_dict[base]
            bases.append(base)

    # Calcule le % pour chaque base
    for base in bases:
        pourcentBases[base] = str(int(((base_count_dict[base] /
            sumBasesOccurences) * 100) // 1))+"%"

    return pourcentBases
```

```
37
38
39 """
40 Retourne vrai si le % de la base la plus représentée est compris
41 entre 40 et 70%
42 Param:
43 - base_count_dict: {'A': x, 'G': y, etc} où les clés sont les
44 bases et les valeurs le nombre d'occurence de la base
45 Return:
46 - true : si c'est une mutation (entre 40 et 70%)
47 - false : sinon
48 """
49 def isHomoHetero(base_count_dict):
50     bases = ['A', 'C', 'G', 'T']
51     # Nombre de bases avec une valeur > 0
52     nbBases = 0
53     # Somme des occurences des bases
54     sumBasesOccurences = 0
55
56     for base in bases:
57         assert base in base_count_dict, "Wrong dictionnary in
    argument"
58         # Vérifie que la valeur de la base est > 0
59         if base_count_dict[base] > 0:
60             nbBases += 1
61             sumBasesOccurences += base_count_dict[base]
62
63     if nbBases >= 2:
64         # Prend la base qui a la plus grande valeur
65         find_max = max(base_count_dict, key=base_count_dict.get)
66         #  Calcule son % en fonction de sumBasesOccurences
67         pourcentage_max = base_count_dict[find_max] /
    sumBasesOccurences
68         if 0.4 < pourcentage_max < 0.7:
69             return True
70
71     return False
72
73
74 """
75 Gère le fichier bam, regarde toutes les positions et vérifie si il y
    a une mutation, retiens la position et le nombre d'occurences des
    bases si c'est le cas.
76
77 Param:
78 - path (str) : le chemin du fichier bam
79
80 Return:
81 - vcf_id (dict) : un dictionnaire où chaque clé est la position de la
     séquence où il y a une mutation homozygote ou hetero. La valeur
    est un dictionnaire où chaque bases qui apparaît dans la pile de
    lecture a pour valeur sa couverture en %
82 in %
83 """
84 def processBamFile(path):
85     # Lis le fichier
```

```
86      samfile = pysam.AlignmentFile(path, "rb")
87      total_size = 0
88      # Dictionnaire de sortie
89      vcf_id = {}
90
91      # Itère sur les pileupcolumns du fichier bam (une colonne d'
        alignement par position)
92      for pileupcolumn in samfile.pileup():
93          total_size += 1
94
95          # Compte les bases pour cette colonne
96          base_count = {'A': 0, 'G': 0, 'C': 0, 'T': 0}
97          # Itère sur chaque read aligné sur cette colonne
98          for pileupread in pileupcolumn.pileups:
99              if not pileupread.is_del and not pileupread.is_refskip:
100                 # Incrémente le compte de la base de 1
101                 base_count[pileupread.alignment.query_sequence
102                     [pileupread.query_position]] += 1
103
104         # Vérifie la répartition des bases pour cette colonne
105         if isHomoHetero(base_count):
106             vcf_id[pileupcolumn.reference_pos + 1] =
107                 countpourcentbases(base_count)
108
109     print(f"Total pileupcolumns is {total_size}")
110     samfile.close()
111     return vcf_id
112
113
114 """
115 Converti dic en string
116
117 Param:
118 - dic (dict) : {'A': '50%', 'G': '25%', 'C': '25%'}
119
120 Return:
121 - s (str) : le résultat de la concaténation des éléments de dic
122     ex : A50%;G25%;C25%
123 """
124 def vcf_id_convertor(dic):
125     s = ''
126     for key in dic:
127         s += f'{key}{dic[key]};'
128     return s
129
130
131 """
132 Modifie un fichier vcf pour indiquer le % de représentation des bases
      pour les positions pour lesquelles il y a une mutation.
133
134 Param:
135 - path (str) : le chemin du fichier vcf
136 - vcf_id (dict) : un dictionnaire reprenant des positions avec pour
    valeur un dictionnaire indiquant les bases de ces positions avec
    leur taux de représentation en %
137 """
```

```
138 def modifyVCF(path, vcf_id):
139     print("Writing VCF file..")
140     vf = pyvcf.VcfFrame.from_file(path)
141     data = vf.df.values
142 
143     # Modifie le champ ALT du fichier vcf si une position dans vcf_id
        matche
144     for i in range(len(data)):
145         position = data[i][1]
146         if position in vcf_id:
147             vf.df.at[i, 'ALT'] = vcf_id_convertor(vcf_id[position])
148 
149     # Garde le même nom de fichier mais avec une extension .CHU.vcf
150     if path[-4:] == '.vcf':
151         path = f'{path[:-4]}.CHU.vcf'
152     else:
153         path = f'{path}.CHU.vcf'
154     vf.to_file(path)
155 
156 
157 # Bam and vcf file
158 def mainBam(path):
159     vcf_id = processBamFile(f"{path}/merged_sorted.bam")
160     modifyVCF(f'{path}/merged_sorted.vcf', vcf_id)
```

# Annexe I

# Contenu d'un fichier vcf non amélioré généré par le pipeline

```
##fileformat=VCFv4.2
##FILTER=<ID=PASS,Description="All filters passed">
##bcftoolsVersion=1.7+htslib-1.7-2
##bcftoolsCommand=mpileup -Ov -f Duffy_seq.fasta merged_sorted.bam
##reference=file://Duffy_seq.fasta
##contig=<ID=ACKR1_Duffy:GRCh38:1:159202807:159207000:1,length=4194>
##ALT=<ID=*,Description="Represents allele(s) other than observed.">
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">
##INFO=<ID=IDV,Number=1,Type=Integer,Description="Maximum number of reads supporting an indel">
##INFO=<ID=IMF,Number=1,Type=Float,Description="Maximum fraction of reads supporting an indel">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Raw read depth">
##INFO=<ID=VDB,Number=1,Type=Float,Description="Variant Distance Bias for filtering splice-site artefacts in RNA
    -seq data (bigger is better)",Version="3">
##INFO=<ID=RPB,Number=1,Type=Float,Description="Mann-Whitney U test of Read Position Bias (bigger is better)">
##INFO=<ID=MQB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality Bias (bigger is better)">
##INFO=<ID=BQB,Number=1,Type=Float,Description="Mann-Whitney U test of Base Quality Bias (bigger is better)">
##INFO=<ID=MQSB,Number=1,Type=Float,Description="Mann-Whitney U test of Mapping Quality vs Strand Bias (bigger
    is better)">
##INFO=<ID=SGB,Number=1,Type=Float,Description="Segregation based metric.">
##INFO=<ID=MQ0F,Number=1,Type=Float,Description="Fraction of MQ0 reads (smaller is better)">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="List of Phred-scaled genotype likelihoods">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##INFO=<ID=ICB,Number=1,Type=Float,Description="Inbreeding Coefficient Binomial test (bigger is better)">
##INFO=<ID=HOB,Number=1,Type=Float,Description="Bias in the number of HOMs number (smaller is better)">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes for each ALT allele, in the same
    order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##INFO=<ID=DP4,Number=4,Type=Integer,Description="Number of high-quality ref-forward , ref-reverse, alt-forward
    and alt-reverse bases">
##INFO=<ID=MQ,Number=1,Type=Integer,Description="Average mapping quality">
##bcftools_callVersion=1.7+htslib-1.7-2
##bcftools_callCommand=call -mv -o merged_sorted.vcf; Date=Tue Apr 19 11:24:59 2022
#CHROM  POS ID  REF ALT QUAL  FILTER  INFO  FORMAT  merged_sorted.bam
ACKR1_Duffy:GRCh38:1:159202807:159207000:1  258 . G A 141 . DP=164;VDB=4.60576e-15;SGB=-0.693147;RPB=0.942039;
```

```
   MQB=0.953571;MQSB=0.952643;BQB=0.729162;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=8,53,10,48;MQ=59 GT:PL
     0/1:174,0,246
31 ACKR1_Duffy:GRCh38:1:159202807:159207000:1  548 . C T 134 . DP=171;VDB=0.00230182;SGB=-0.693147;RPB=0.762335;MQB
     =1;MQSB=1;BQB=0.372201;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=12,57,14,33;MQ=60 GT:PL 0/1:167,0,216
32 ACKR1_Duffy:GRCh38:1:159202807:159207000:1  943 . C T 162 . DP=172;VDB=0.00395344;SGB=-0.693147;RPB=0.820277;MQB
     =1;MQSB=1;BQB=0.703139;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=22,20,24,29;MQ=60 GT:PL 0/1:195,0,159
33 ACKR1_Duffy:GRCh38:1:159202807:159207000:1  1527  . C T 222 . DP=184;VDB=6.2661e-06;SGB=-0.693147;RPB=0.973049;
     MQB=1;MQSB=1;BQB=9.50862e-07;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=44,19,36,36;MQ=60  GT:PL 0/1:255,0,174
34 ACKR1_Duffy:GRCh38:1:159202807:159207000:1  2324  . C T 211 . DP=193;VDB=0.0337609;SGB=-0.693147;RPB=0.848011;
     MQB=1;MQSB=1;BQB=0.307632;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=65,26,43,21;MQ=60  GT:PL 0/1:244,0,255
35 ACKR1_Duffy:GRCh38:1:159202807:159207000:1  2758  . G A 190 . DP=214;VDB=0.00318366;SGB=-0.693147;RPB=0.896639;
     MQB=0.989618;MQSB=0.982917;BQB=3.55054e-05;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=55,22,55,35;MQ=59  GT:PL
     0/1:223,0,169
```

# Annexe J

# Implémentation de la génération du fasta consensus

```
import pysam
from fuc import pyvcf
from Bio import SeqIO
from Bio.Seq import Seq

"""
Crée un nouveau fichier fasta qui sera la nouvelle séquence consensus

Param:
- bampath (str): le chemin du fichier .bam qui reprend les
    alignements faits sur le fichier fasta
- fastapath (str) : le chemin du fichier .fasta sur lequel les
    alignements ont été faits
"""
def assemblyReadsv2(bampath, fastapath):
    file_in = fastapath
    # Le nouveau fichier fasta porte le même nom mais avec l'
   extension .CHU.fasta
    file_out = f'{fastapath[:-6]}.CHU.fasta'

    with open(file_out, 'w') as f_out:
        for seq_record in SeqIO.parse(open(file_in, mode='r'),
                'fasta'):
            # retire .id de .description record (retire tout
            # avant le premier espace)
            seq_record.description = ' '.join(seq_record.description.
                split()[1:])

            # Instancie la nouvelle séquence consensus
            new_seq = ""
            # Lis le fichier .bam
            samfile = pysam.AlignmentFile(bampath, "rb")
            # Récupère l'ancienne séquence
            old_seq = str(seq_record.seq)
            old_seq = Seq(old_seq)

            start = 0
```

```
            weAt = 0

            # Itère sur chaque colonne de l'alignement
            for pileupcolumn in samfile.pileup():
                # Itère, à partir de start,
                # sur chaque base de l'ancienne séquence
                for position in range(start, len(old_seq)):
                    weAt += 1

                    # Écrit - si les positions sont différentes,
                    # signifie que position n'est pas couverte
                    if pileupcolumn.reference_pos != position:
                        new_seq += "-"

                    # Si la position est couverte, écrire la base
                    # la plus représentée dans cette colonne
                    # d'alignement, incrémenter start de 1
                    else:
                        start = position+1
                        base_count = {'A': 0, 'G': 0, 'C': 0, 'T': 0}
                        for pileupread in pileupcolumn.pileups:
                            if not pileupread.is_del and not
                                pileupread.is_refskip:

                                base_count[pileupread.
                                alignment.query_sequence
                                [pileupread.query_position]] += 1
                        new_seq += max(base_count,key=base_count.get)
                        break

            # Si les alignments n'ont pas couvert toutes les
            # positions, ajouter - pour chaque position non couverte
            if weAt < len(old_seq):
                for i in range(weAt, len(old_seq)):
                    new_seq += "-"

            # Écris la nouvelle séquence dans le nouveau fasta
            seq_record.seq = Seq(new_seq)
            # write new fasta file
            r = SeqIO.write(seq_record, f_out, 'fasta')
            if r != 1:
                print('Error while writing sequence:  '
                    + seq_record.id)
```

# Annexe K

# Implémentation de l'identification du type érythrocytaire

```
import pandas as pd
from fuc import pyvcf
import pdfGenerator
import os


"""
Convertis le fichier .xls en dictionnaire

Param:
- xls (str) : le chemin d'accès au fichier .xls

Return:
- genotype (dict) : un dictionnaire construit sur base du fichier xls
    ex : {'KEL': {578: {'T': 'KEL*01', 'C': 'KEL*02'}, 841: ... }}
    KEL est un un gène, si à la position 578 de ce gène, un T est
    observé en majorité, alors il s'agit du génotype KEL*01.

"""
def xlsToDic(xls):
    # Lis le fichier xls, instancie le dictionnaire
    df = pd.read_excel(xls, sheet_name=0)
    df = df.reset_index()
    genotype = {}

    # Itère sur chaque ligne du fichier xls
    for index, row in df.iterrows():
        # Récupère la base et le génotype de cette ligne
        tmp = {row['Base']: row['Genotype']}
        # Si le nom du gène est déjà dans le dict,
        # ajoute au gène déjà existant
        if row['Group'] in genotype:
            # si la position existe déjà, ajoute la
            # nouvelle basenew base et son genotype
            if row['Position'] in genotype[row['Group']]:
                if row['Base'] not in genotype[row['Group']]
                                                [row['Position']]:
```

```
                    genotype[row['Group']][row['Position']]
                                [row['Base']] = row['Genotype']
                else:
                    print("Multiple genotypes for the same base")

            # Sinon on ajoute la position, la base et le genotype
            else:
                genotype[row['Group']][row['Position']] = tmp

        # Sinon on l'ajoute
        else:
            genotype[row['Group']] = {row['Position']: tmp}

    return genotype


"""
Converti element issu d'un fichier .CHU.vcf

Param:
- element (str): par exemple A44%;C1%;G54%;T0%; le contenu d'un champ
    d'un fichier .CHU.vcf

Return:
- dic (dic) : elem converti en dict
    ex : {'A': '44', 'C': '1', 'G': '54', 'T': '0'}
"""
def vcfElemToDict(element):
    splipp = element.split('%;')
    while '' in splipp:
        splipp.remove('')

    dic = {}
    for element in splipp:
        dic[element[0]] = element[1:]
    return dic


"""
Converti les données du fichier vcf en dictionnaire

Param:
- path (str) : le chemin vers le fichier .CHU.vcf

Return:
- dictionary_from_vcf (dict) : exemple :
    {59: {'C': '50'}, 1416: ..}
    le fichier vcf indique à la position 59, un pourcentage pour la
    base C de 50%
"""
def vcfToDict(path):
    # Lis le fichier vcf
    vf = pyvcf.VcfFrame.from_file(path)
    data = vf.df.values

    dictionary_from_vcf = {}
```

```
92
93      # Itère sur les lignes
94      for i in range(len(data)):
95          # Si le 5 champs a une taille > 1, alors les pourcentages
96          # sont indiqués
97          if len(data[i][4]) > 1:
98              # converti ce champ en dict et le stocke dans le
99              # dictionnaire de sortie
100             pourcentage_dic = vcfElemToDict(data[i][4])
101             dictionary_from_vcf[data[i][1]] = pourcentage_dic
102
103     return dictionary_from_vcf
104
105
106 """
107 Détecte les génotypes et les stocke dans le context du pdf qui sera
108 généré
109 """
110 def getGenotype(genotypedic, dictionary_from_vcf, family='Kell'):
111     # Vérifie si la famille de gènes existe dans genotypedic
112     try:
113         my_family = genotypedic[family]
114     except Exception as e:
115         print(f'Family {family} doesn\'t exists in xlsx file')
116         return
117
118     # Regarde pour chaque position si elle est reprise dans le
    fichier vcf,
119     # si c'est le cas, vérifie si le % de la base à cette position
120     # est supérieur à 35%, si une génotype correspond à cette base
121     # pour cette position, on le stocke dans le contexte du pdf
122     for position in my_family:
123         if position in dictionary_from_vcf:
124             # Check the base and %
125             for base in dictionary_from_vcf[position]:
126                 if int(dictionary_from_vcf[position][base]) > 35 and
    base in my_family[position]:
127                     print(f" Genotype : {my_family[position][base]}")
128                     positionandbase = f"{str(int(position))}{base}"
129                     pdfGenerator.pdfcontext["Genotype"].append(f"{
    my_family[position][base]}, {positionandbase}")
130
131
132 def main(genotypeList, vcf):
133     genotypedict = xlsToDic(genotypeList)
134     vcfDict = vcfToDict(vcf)
135     filename, _ = os.path.splitext(vcf)
136
137     for family in genotypedict:
138         if family in filename or family == "MNS":
139             if family == "MNS" and not('GYPA' in filename or 'GYPB'
    in filename):
140                 continue
141             getGenotype(genotypedict, vcfDict, family)
142
143     pdfGenerator.writePDF()
```

```
144
145
146 main("genotype.xlsx", "KELL_seq.vcf")
```

Avec le code de la génération de pdf :

```
1  from fpdf import FPDF
2  from datetime import date
3
4  today = date.today()
5  print("Today's date:", today)
6
7  pdf = FPDF()
8
9  # Style for pdf
10 pdf_size = 15
11 pdf_font = "Arial"
12
13
14 pdfcontext = {
15     "Title": 'CHU Mont-Godinne',
16     "Date": date.today(),
17     "Id": "Florent",
18     "Genotype": []
19 }
20
21
22 def setBold():
23     pdf.set_font(pdf_font, 'B', size=pdf_size)
24
25
26 def removeBold():
27     pdf.set_font('')
28     pdf.set_font(pdf_font, size=pdf_size)
29
30
31
32 def writePDF():
33     """
34     Write the pdf for genotype report
35     """
36     # Add a page
37     pdf.add_page()
38
39     # Title in bold
40     setBold()
41     pdf.cell(200, 10, txt=f"{pdfcontext['Date']}",
42              ln=1, align='L')
43     pdf.cell(200, 10, txt=f"{pdfcontext['Title']}",
44              ln=1, align='C')
45     removeBold()
46
47     # Patient name
48     pdf.cell(200, 10, txt=f"Patient name : {pdfcontext['Id']}", ln=1,
       align='L')
49     # Genotype list
```

```
50      pdf.cell(200, 10, txt="Genotype present : ", ln=2, align='L')
51
52      # List of genotype
53      setBold()
54      for genotype in pdfcontext['Genotype']:
55          pdf.cell(200, 10, txt=f"    - {genotype}", ln=2, align='L')
56      # Removes bold
57      removeBold()
58
59      # save the pdf with name .pdf
60      pdf.output(f"{pdfcontext['Id']}-genotype.pdf")
```