Rapport de stage
Master 2 SESI

Systèmes de stockage sur ADN

Étudiant :
Mr. Belaid HAMOUM

Maitre de stage :
Mme. Laura Conde-Canencia

Année : 2018-2019

Table des matières

REMERCIEMENTS

Je tiens tout particulièrement à remercier mon enseignante référente et maitresse de stage, Mme Laura Conde-Canencia, enseignant chercheur à l'Université Bretagne Sud, qui en plus de m'accorder sa confiance pour ce stage, m'a supervisé et accompagné avec le partage de son expertise dans mes différentes missions. Sa disponibilité, son écoute et ses conseils m'ont permis de développer différentes compétences techniques, organisationnelles et relationnelles. Je la remercie également pour sa proposition de continuer en thèse doctorale sur le même sujet et sous sa direction (j'ai évidemment accepté). Cette thèse représente une grande opportunité pour moi et pour mes perspectives d'avenir. Elle a aussi été d'une grande aide dans la rédaction de ce rapport et de différents documents.

Je tiens également à remercier les membres de l'équipe Genscale :

Mr Dominique Lavenier, Directeur de recherche CNRS IRISA/ INRIA Rennes, pour m'avoir expliqué et fait mieux saisir différentes notions de l'ADN et son séquencement. Son expertise et ses suggestions m'ont permis d'avancer sur différents points de ce projet.

Mme Emeline Roux, enseignant-chercheur à l'Université Lorraine, pour son expertise et ses explications qui m'ont permis de mieux appréhender la partie synthèse de l'ADN.

Mme Ariane Badoual, étudiante en bio-informatique à l'Université Rennes 1 et stagiaire, pour ses éclaircissements sur plusieurs notions de la bio-informatique.

Je remercie aussi, les chercheurs et enseignant chercheurs qui ont été favorable à ma candidature en thèse doctorale par leur lettre de motivation ou par leur avis : Mr Dominique Heller, Mr Jean-Philippe Diguet, Mr Cyrille Chavet, Mr Emmanuel Boutillon et Mr Marc Sevaux.

INTRODUCTION

De nos jours, la quantité de données numériques produites dépasse largement les capacités de stockage disponibles. Aussi, malgré la constante amélioration des systèmes de stockage conventionnels, ils nécessitent un espace physique important (surface occupée) et un environnement particulier (basse températures, maintenance, …). Pour sauvegarder toutes ces données nous devons faire des progrès significatifs dans le stockage de données, notamment en termes de densité de stockage. L'utilisation de l'ADN comme support de stockage en est un, car elle est extrêmement dense (1 exabyte par millimètre cubique) et durable (prouvé par les fossiles). Bien qu'elle soit à ses début, cette technologie se développe assez rapidement avec les avancements des biotechnologies. En effet, Microsoft a déjà annoncé qu'ils vont commencer à stocker des données sur ADN en 2020 [1].

Dans le cadre de mon stage de Master 2 Systèmes Embarqués Systèmes Intégrés à l'Université Bretagne Sud, j'ai souhaité réaliser mon stage au Lab-STICC Lorient pour travailler sur le projet « Systèmes de stockage sur ADN ». Ce projet, tente d'apporter ou de se rapprocher d'une solution à la problématique du stockage de données. On a exploré et travaillé sur des techniques de codage (source et canal) qui peuvent transformer les données numériques (binaires) en séquences ADN (formée de bases ou nucléotides 'A', 'C', 'G', 'T') synthétisables (numérique vers ADN réelle). Ceci en améliorant la précision du séquençage ADN (lecture de l'ADN) pour un certain type de séquenceurs. Ce stage a été financé par une action exploratoire du CominLabs et s'est déroulé en collaboration avec l'équipe GenScale de l'INRIA Rennes, composée de bio-informaticiens et d'une biologiste, tous experts en séquencements de génomes.

Pendant ce stage, on a utilisé le séquenceur MinION, qui est un séquenceur de 3ème génération produit par Oxford Nanopore Technologies et qui en plus d'être de petite taille et abordable, offre des lectures longues (donc rapide). Il possède aussi, une connectique USB 3.0. Ce dernier, utilise le séquencement par nanopores pour lire l'ADN d'une manière bien plus rapide (quelques heures) que les autres technologies (n'utilisant pas les nanopores) de séquenceurs (quelques jours). Cependant, son taux d'erreurs lors du séquençage est assez élevé (~15%), d'où la nécessité de chercher des solutions pour améliorer sa précision, ce qu'on a fait pendant ce stage.

Dans ce rapport, je décrirai dans un premier temps, le laboratoire de recherche Lab-STICC où j'ai effectué mon stage, avant de parler des objectifs et de la chronologie définie. Je parlerai ensuite des différentes parties sur lesquelles j'ai travaillé avant de faire un bilan de celui-ci.

I. PRESENTATION LAB-STICC

Le Lab-STICC est le Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance et est présent sur les villes de Brest, Quimper, Lorient et Vannes.

La Bretagne Océane dispose d'importantes forces de recherche académiques dans les le domaine des STIC, mais elles étaient dispersées sur plusieurs laboratoires rattachés aux universités et écoles d'ingénieurs, d'où l'initiative de rassembler ces différentes forces au sein d'une même structure le LAB-STICC. Il fut créé en janvier 2008 avec la volonté de construire autour de son projet fédérateur :

> « Des capteurs à la connaissance : communiquer et décider ». [2]

Le Laboratoire est composé de chercheurs, enseignants-chercheurs et doctorants. J'ai quant à moi travaillé avec l'équipe IAS (Interaction between Algorithms and Silicon) qui fait partie du pôle CACS (Communications, Architectures, Circuits et Systèmes).

II. PROJET « SYSTEME DE STOCKAGE DE DONNEES SUR ADN »

Notre projet a pour objectif de concevoir un système de stockage de données sur ADN qui permettra d'améliorer le séquencement de l'ADN (en rouge) via le séquenceur MinION. Pour se faire on s'est focaliser sur 2 parties en amant de la chaine de stockage ci-dessous, à savoir, la partie codage source (en bleu) et la partie codage de canal (en vert).

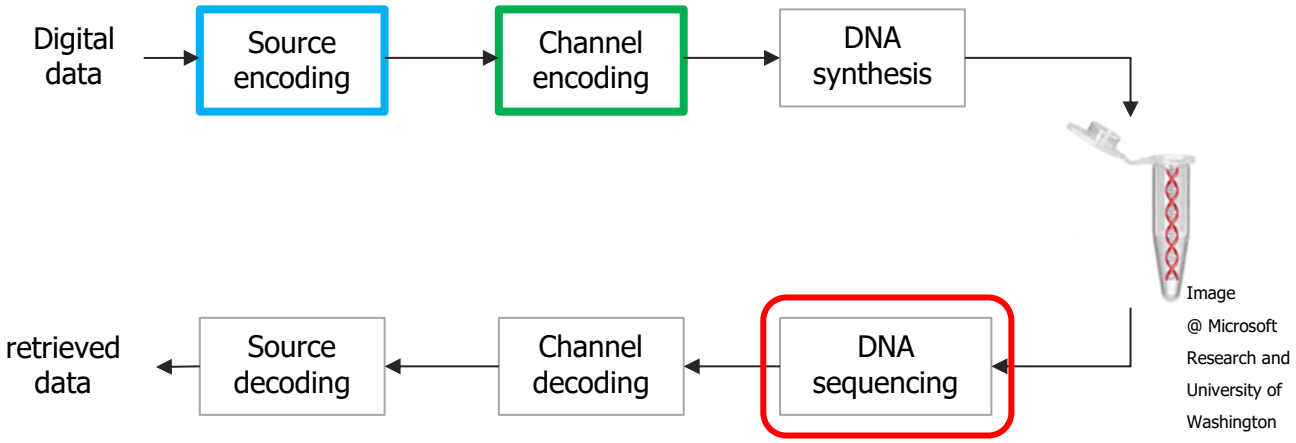


Fig 1 : Chaine de stockage de données sur ADN.

1. Cahier des charges :

Nous avons défini les principales tâches à accomplir pour la concrétisation de notre projet :

- Codage source. (1)
- Simulations. (2)
- Codage de canal (3)

2. Chronologie :

- Tâche (1) : 1,5 mois (temps que devra prendre la tâche). (En vrai 1,5 mois)
- Tâche (2) : 1,5 mois. (En vrai 2 mois)
- Tâche (3) : 2 mois. (En vrai 1,5 mois)

Dans la section qui suit, nous décrivons quelques notions relatives à l'ADN :

3. Notions sur l'ADN et son séquencement :

- ***Nucléotide*** :

Un nucléotide (appelé aussi base) dont il existe 4 types (dans l'ADN) : 'A'(Adenine), 'C'(Cytosine), 'G' (Guanine) et 'T'(Thymine), est l'élément de base de l'ADN.

- ***L'ADN*** :

La molécule d'ADN est une double hélice formée de 2 brins d'ADN enroulés l'un autour de l'autre et dont la succession des nucléotides sur chacun d'entre eux constitue une séquence ADN. Dans l'ADN, une notion de complémentarité entre les nucléotides existe, c.à.d 'A' se rattache à 'T' (et inversement) et 'C' se rattache à 'G' (et inversement). Ceci permet de rattacher les 2 brins d'ADN ensemble. Les 2 brins sont donc complémentaires entre eux.

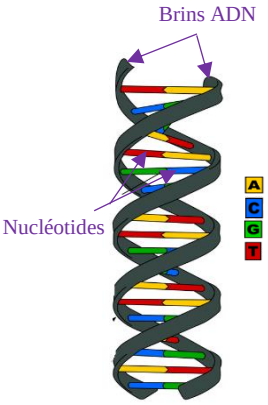


Fig 2 : molécule d'ADN [3].

- ***Séquencement ADN :***

Le séquencement d'ADN permet de lire l'ADN à l'aide d'un séquenceur pour obtenir une chaine de caractères (séquence) représentant les différentes nucléotides (A, C, G et T) de la séquence.

Fig 3 : Séquencement d'ADN [4].

- ***Les k-mers :***

Un k-mer est un ensemble formé de k nucléotides contigus. Dans le cas du dispositif MinION, chaque k-mer est composé de 6 nucléotides (appelé donc 6-mer) et est caractérisé par un niveau d'amplitude du courant électrique propre à lui.

Fig 4 : 6-mers d'une séquence ADN.

- ***Homopolymères (HP) :***

Répétition contigüe (en rouge) d'un même nucléotide dans la séquence.

A A C T A C C C T G G G G

Fig 5 : Séquence ADN contenant des homopolymères.

- ***Séquencement par nanopores (MinION):***

Le séquencement par nanopores utilise des nanopores qui sont des trous nanométriques dans une membrane et par lesquels passent les différents brins d'ADN. L'intensité du courant électrique sur la membrane change en fonction du k-mer (6-mer pour le MinION) se trouvant dans le nanopore. Ce changement d'intensité permet d'identifier les K-mers par la suite (dans la phase de basecalling décrite en bas). Un signal électrique est obtenu à la fin de cette étape.

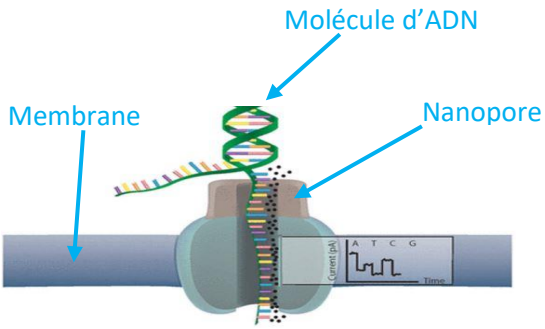


Fig 6 : Séquencement par nanopores [5].

- ***Basecaller (albacore pour le MinION) :***

C'est un outil qui prend en entrée un signal électrique généré lors de la phase du séquencement par nanopore et l'utilise pour identifier les différents k-mers. Il génère en sortie la séquence ADN lue (au format 'A', 'C', 'G', 'T'). La transformation du signal en séquence est appelée phase de « basecalling ».

Fig 7 : Phase de basecalling du MinION [6].

- ***Alignement :***

L'alignement est fait par des outils qui prennent en entrée deux séquences : une appelée référence et l'autre séquence lue. Le principe est d'aligner (positionner) la séquence lue par rapport à la référence (considérée comme juste), de sorte qu'elles aient le plus de similitudes pour pouvoir observer les erreurs.
Exemple :

- Séquence de référence : Bonjour

- Séquence lue : Bxomour

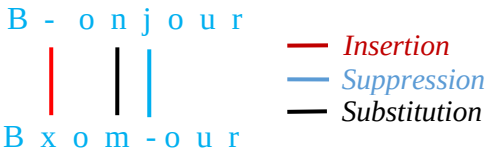


Fig 8 : Alignement de deux séquences.

- ***Synthèse ADN :***

C'est l'inverse du séquencement et permet d'obtenir de l'ADN synthétique à partir d'une séquence de nucléotides.

Fig 9 : Synthèse de l'ADN à partir d'une séquence [4].

III. DEROULEMENT DU STAGE

Pour mettre en œuvre ce système, nous avons exploré et travaillé sur les différents champs suivants :

1. Encodage source 1 (données numériques vers séquence ADN) :

Dans la partie codage source nous avons travaillé sur des algorithmes pour transformer nos données numériques (textes) en séquences ADN (chaine de caractères au format A, C, G et T). On a comme contrainte d'éviter la répétition contiguë d'un même nucléotide et d'obtenir une séquence avec un caractère aléatoire.

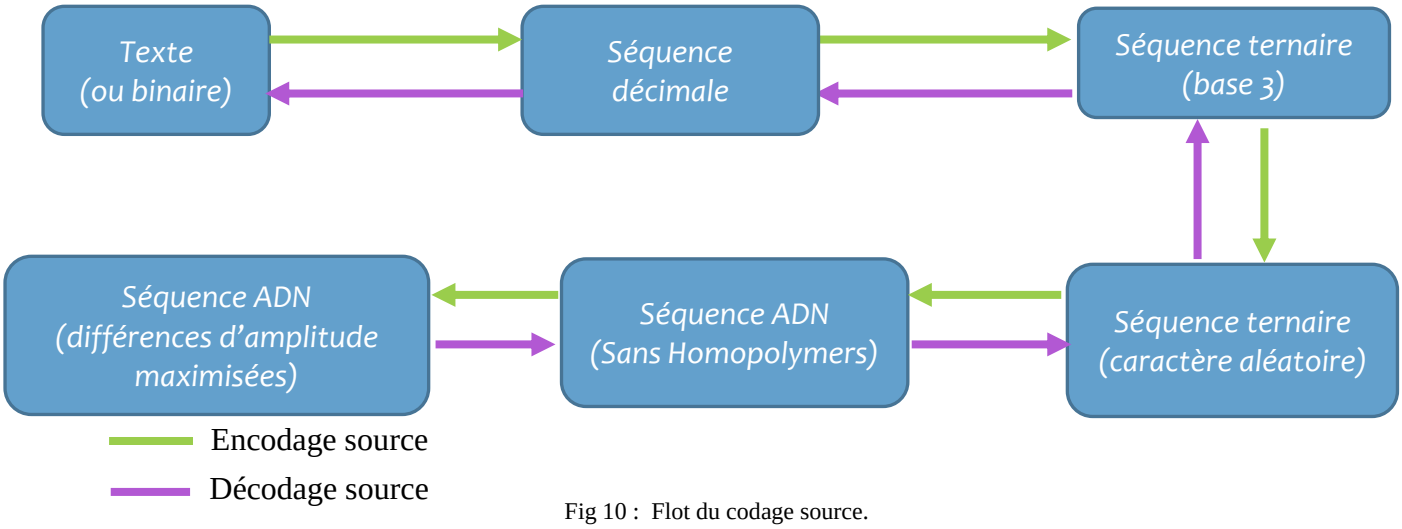


Fig 10 : Flot du codage source.

- ***Étape 1 : Texte vers séquence décimale***

On utilise dans cette étape le code ASCII des différents caractères d'un texte pour obtenir leurs valeurs décimales.

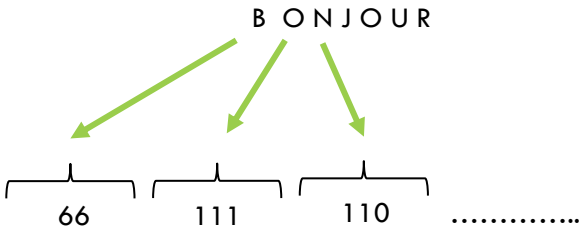


Fig11 : symboles ASCII vers symboles décimaux.

- ***Étape 2 : Séquence décimale vers séquence ternaire (en base 3)***

Dans cette étape on a transformé les caractères décimaux en caractères ternaires. Le passage en alphabet ternaire au lieu du quaternaire est important car, il nous permettra dans la prochaine étape d'éviter les homopolymères (répétition contigüe d'un même nucléotide) dans nos séquences.

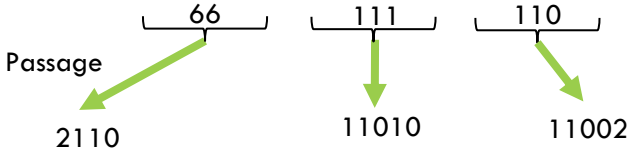


Fig 12 : symboles décimaux vers symboles ternaires.

- ***Étape 3 : Donner un caractère aléatoire à la séquence :***

À cause de la répétition de plusieurs sous séquences à des endroits différents de la séquence ADN (même mot ou phrase qui se répète dans le texte), la synthèse de celle-ci ne peut pas être réalisée. En effet, pour faire la synthèse, la séquence ADN est fractionnée en plusieurs sous séquences (oligos) avant d'être réassemblée. C'est lors de ce réassemblage qu'il est possible de faire des erreurs si une partie importante de l'oligo est semblables à une autre (d'un autre oligo).

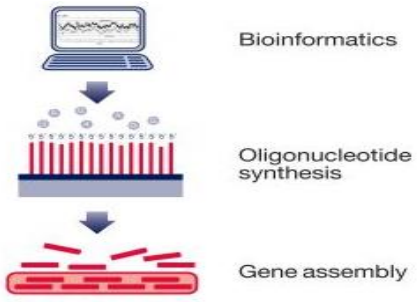


Figure 13 : phases de synthèse d'une séquence ADN [7].

Pour remédier à ce problème, nous avons pensé à générer une chaine aléatoire de trits (1000 valeurs en base 3) comme dans [8]. Cette chaine est ensuite utilisée pour faire la somme en base 3 entre les différents éléments de même position (séquence ADN et chaine aléatoire), avec un résultat modulo 3 (pour rester en base 3 et passer en nucléotides sans répétitions contigües).

Séquence ADN en base 3 : 1 0 0 0 1 2 2 1 0 1 2 2

\+ + + ++ + ++ + + + ...+

Séquence aléatoire en base 3 : 0 2 1 1 2 0 0 1 2 0 2 0

——————————————————————————— (Mod 3)

1 2 1 1 0 2 2 2 2 1 1 2

⇩

Séquence ADN' en base 3 : G T C G C T A G C A C ..... A

Pour revenir à la séquence originale, la chaine aléatoire doit être sauvegardée dans le programme. Ainsi, en soustrayant chaque élément de la chaine ADN' (en base 3) son équivalent (même position) de la séquence aléatoire, on obtient la séquence originale (ADN). Nous avons aussi pris en compte la subtilité du modulo 3 (quand un trit de la séquence ADN est inférieur à son équivalent dans la chaine aléatoire).

Séquence ADN' en base 3 : 1 2 1 1 0 2 2 2 2 1 1 2

≥ ≥ ≥ ≥ ≥ ≥ ≥ ≥ ≥ ≥ ≥ ≥

Séquence aléatoire en base 3 : 0 2 1 1 2 0 0 1 2 0 2 0

⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩

Séquence ADN en base 3 : 1 0 0 0 1 2 2 1 0 1 2 2

$$tritADN[i] = \begin{cases} tritADN'[i] - tritAleatoir[i] \text{ , Si } tritADN'[i] >= tritAleatoir[i] \\ (3+tritADN'[i]) - tritAleatoir[i], \text{ Sinon} \end{cases}$$

- ***Étape 4 : Séquence ternaire vers séquence ADN***

Dans cette étape on a transformé les caractères ternaires en caractères de l'alphabet ADN (A, C, G et T). Nous avons utilisé une table de transition décrite dans [8] dans le but d'éviter l'apparition d'homopolymères.

previous nt written	next trit to encode		
	∅	1	2
A	C	G	T
C	G	T	A
G	T	A	C
T	A	C	G

Fig 14 : table de transition décrite dans [8]

Fig 15 : symboles ternaires vers nucléotides ADN.

En utilisant cet encodage, la séquence ADN obtenue ne présente plus de similitudes entre ses différentes sous-séquences et passe parfaitement le test de différents outils de préparation à la synthèse dont celui de ThermoFisher «GeneArt™ Strings™ Assistant ».

2. Encodage source 2 (Maximisation des distances d'amplitude) :

Après avoir étudié les différentes étapes d'un séquencement par nanopores et pris en compte les observations de différents articles. Il apparait que les homopolymères rendent constante l'amplitude du signal électrique en sortie des nanopores. Ceci rend la phase de basecalling (identification des k-mers sur le signal électrique) compliquée et introduit des erreurs dans celle-ci. Nous avons supposé que l'amélioration de la distance d'amplitude (la rendre plus grande) entre 2 K-mers contigües pourrait permettre d'améliorer la précision du séquencement. Nous avons donc travaillé sur un algorithme qui permet d'améliorer ces distances. Nous nous sommes proposés de vérifier si effectivement le séquencement est meilleur lors de nos simulations et synthèses biologiques.

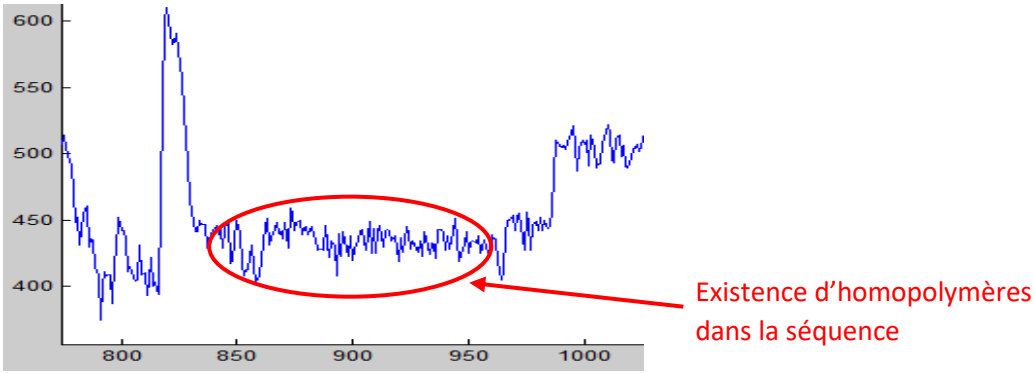


Fig 16 : Signal en sortie du simulateur de séquencement DeepSimulator.

- **Algorithme d'optimisation de distance (entre 2 K-mers consécutifs) :**

Nous avons proposé un algorithme (décrit ci-dessous) pour optimiser la distance entre 2 K-mers consécutifs afin d'observer son effet sur la précision de la lecture. Cet algorithme utilise un seuil défini (fixé à 5 pA et 20 pA pendant nos tests) en dessous duquel la maximisation doit être faite.

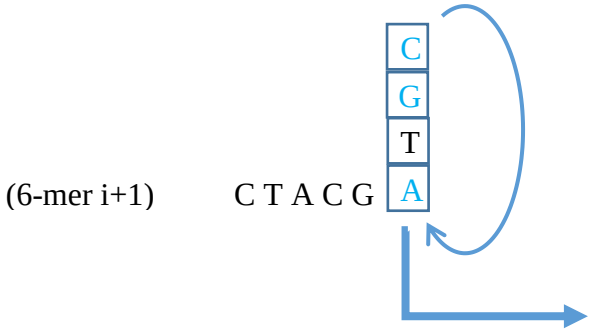
(1) Parcours de la séquence : (pas= 1, en bleu les bases pouvant être substituées)

Si Dist(6-mer i, 6-mer i+1) < seuil

Alors — Sinon

(4)

(2) Substitution de la base (1 ou 2 possibilités de substitution car en trit) :



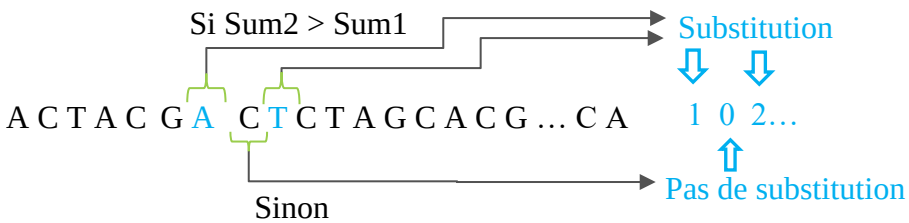
Exemple :
Si base = T elle est substituée soit par A ou C
(Pas par G car ne peut pas être représentée en trit « 0,1,2 » sinon) avec :
0 => pas de substitution
1 => substituée par la 1ère base suivante.
2 => substituée par la 2ème base suivante.
L'ordre de substitution est le suivant :
A → C → G → T

(3) Calcul des sommes avant et après substitution :

Sum1 = 0 (Somme des distances avant substitution)
Sum2 = 0 (Somme des distances après substitution)
Kmer : une base a été substituée dans cette chaine

Pour $0 \leq i \leq$ kmerSize
  Sum1 += | Level(Kmer i) - Level (Kmer i+1) |
  Sum2 += | Level(Kmer i) - Level (Kmer i+1) |
Fin Pour

(4) Concaténation de la séquence :



Pour observer l'effet de ces transformations, nous générons à chaque fois 3 séquences ADN différentes : une contenant des homopolymères, une sans homopolymères et une autres sans homopolymères et dont les distances sont maximisées. Ceci permet de comparer les résultats des lectures lors de la phase de simulation.

Remarque :

La séquence avec homopolymère est obtenue en reprenant l'étape 3 du codage source 1 avec un modulo 4 (au lieu de 3) pour obtenir une chaine contenant des caractères entre 0 et 3 inclus. Ces caractères sont ensuite directement transformés en chaine ADN suivant leur valeur.

'0' => 'A', '1' => 'C', '2' => 'G', '3' => 'T'.

3. Simulations

Pour observer les résultats de la lecture de nos séquences, nous avons dans un premier temps travaillé avec des simulations, car la synthèse biologique est couteuse en termes de temps et d'argent. Nous avons donc utilisé le simulateur « nanoSim » avant de passer à « DeepSimulator ». Il était aussi nécessaire de mettre en œuvre des mécanismes pour réaliser un nombre important (100 dans notre cas) de simulations et des traitements de données automatisées (sur les 3 séquences). Nous avons donc eu recours à des scripts shell et à l'utilisation de Julia et Gawk pour le traitement de nos données.

a. NanoSim

NanoSim est un simulateur open source de lectures par MinION, qui permet de simuler les différentes caractéristiques d'un séquencement par nanopores. Il a été développé par **Chen Yang** au centre canadien « Micheal Smith Genome Science Centre» [9]. Son fonctionnement repose sur 2 étapes : Une étape de caractérisation et une étape de simulation (voir annexe I).

L'étape de caractérisation est une phase d'entrainement qui prend en entrée une lecture réelle (issue du MinION) et un génome de référence (séquence obtenue après traitement de la lecture réelle). Elle permet d'obtenir un profil d'erreur qui sera utilisé lors de la phase de simulation.

L'étape de simulation quant à elle prend en entrée un génome de référence (c.à.d. notre séquence ADN) et un profil d'erreur (obtenu lors de l'étape 1). En plus des lectures simulées (appelés reads ci-dessous), plusieurs fichiers de sorties sont générés lors de cette étape (type d'erreurs, positions des erreurs, …). Nous avons utilisé l'outil Julia pour extraire les données intéressantes et les traiter (calculer nombre d'erreurs). Aussi, on a fait une utilisation un peu détournée de ce simulateur, car la taille de nos séquences ADN ou lectures est connues (contrairement à un génome dans la nature). Nous avons donc dupliqué notre séquence en 3 fois (Ref ' ci-dessous), avant de lancer la simulation, car dans le cas contraire la lecture simulée ne couvre pas tout le segment de la référence mais qu'une partie.

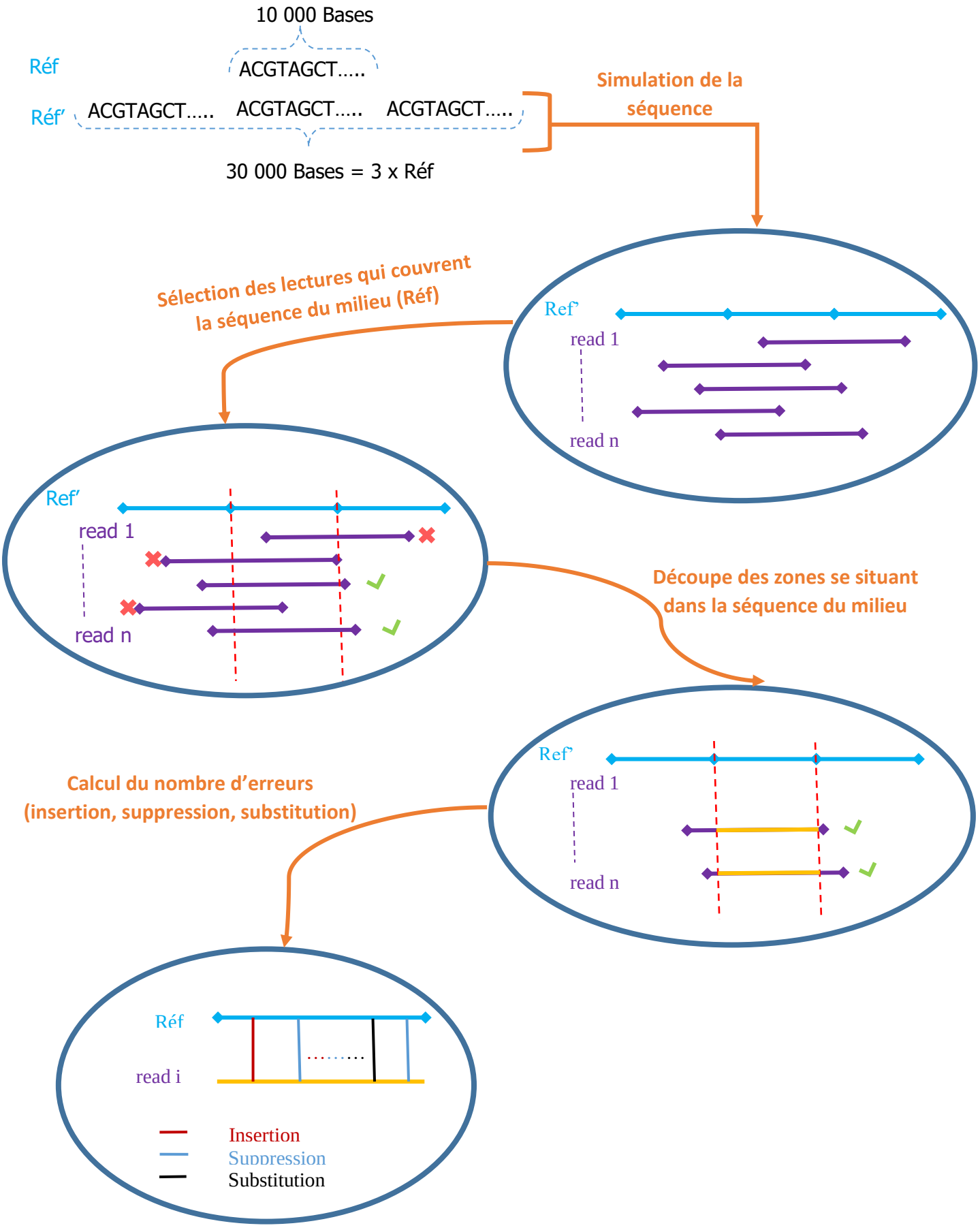


Fig 17 : Flot de simulation et de calcul des erreurs (nanoSim).

- ***Résultats :***

Les résultats obtenus lors de ces simulations se sont révélés incohérents. En effet, il s'avère que nanoSim ne prend pas en compte le contenu de la séquence en entrée mais plutôt sa taille. Pour des références de même taille, le nombre d'erreurs après simulation est plus ou moins pareil, même pour une séquence ne contenant qu'un même nucléotide ('A'). Ces résultats sont erronés, car de nombreuses études ont pointé du doigt l'effet des homopolymères qui introduisent des erreurs dans la phase de basecalling. Nous avons conclu que nanoSim n'est pas adapté pour notre domaine d'application, d'où, le recours à un autre simulateur qui est DeepSimulator.

b. DeepSimulator

DeepSimulator (décrit dans [10]) utilise un réseau de neurones profond (deep learning), pour pouvoir simuler l'ensemble des étapes d'un séquençage par nanopore (voir annexe II). Ce qui le distingue des autres simulateurs, c'est le fait qu'au lieu d'utiliser un modèle (déduit d'un ensemble de données réelles comparées à une référence) pour introduire des erreurs (insertion, suppression et substitution) dans la séquence, il simule l'ensemble des différentes étapes d'un séquençage par nanopores (préparation des brins, capture du signal électrique, basecalling) et laisse l'étape de basecalling introduire elle-même les erreurs (comme dans un séquençage réel). La taille de nos brins d'ADN étant connue, nous avons contraint deepSimulator à générer des lectures de même taille que la référence en entrée (donc pas de phase de préparation des brins). Grâce au basecaller utilisé en fin de la chaine de simulation, DeepSimulator s'intéresse aussi au contenu de la séquence.

- ***Simulation, alignement et calcul des erreurs :***

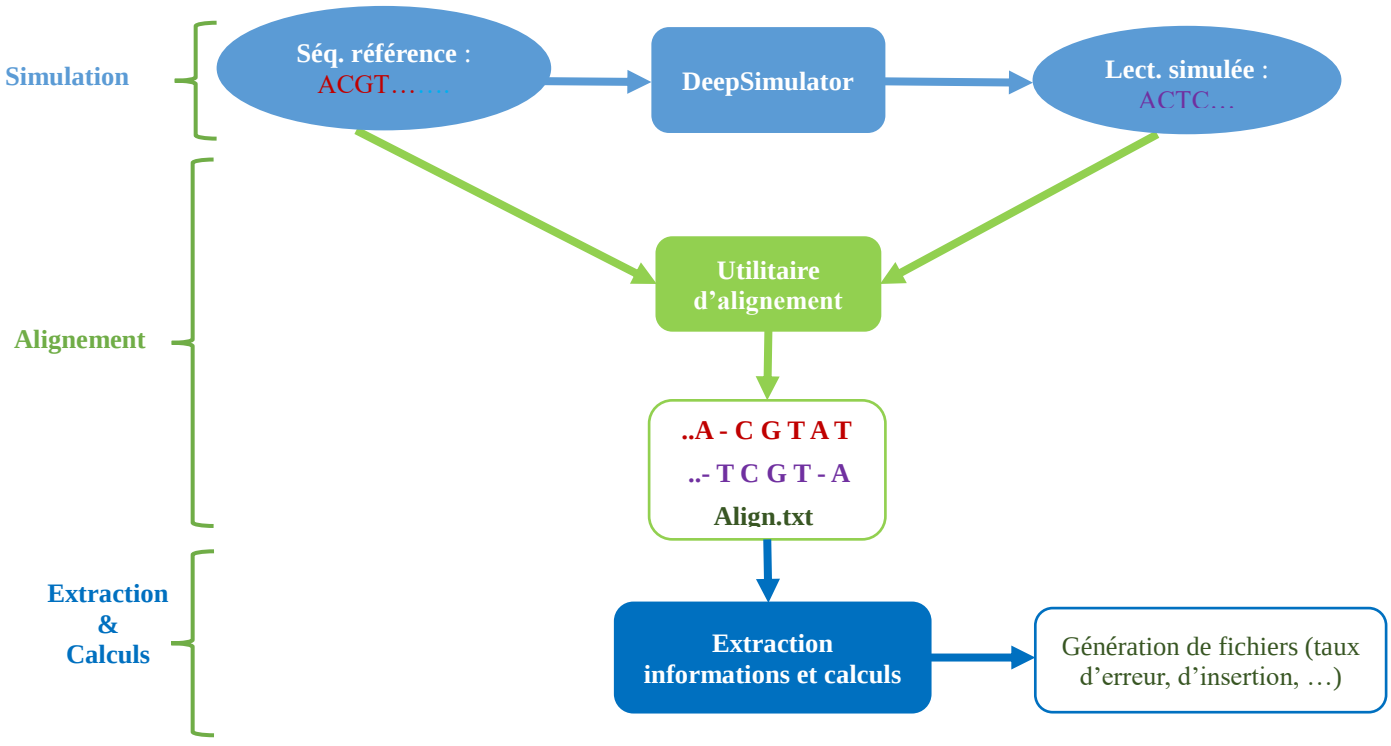


Fig 18 : Flot de simulation, d'alignement et de calcul des erreurs (DeepSimulator).

Nous avons tout d'abord simulé la lecture de chaque séquence en utilisant DeepSimulator. Après cette étape, on obtient trois lectures simulées (Lect1, Lect2, Lect3) de nos séquences de départ.

Pour observer le nombre de correspondances et d'erreurs de chaque lecture par rapport à sa référence (séquence de départ), nous avons utilisé « ggsearch » qui permet de trouver des similarités entre deux différentes séquences en faisant un alignement global de celles-ci. Il prend en entrée 1 séquence de référence dénommée « Subject » et une séquence lue dénommée « Query » et génère en sortie un fichier contenant différentes informations relatives à l'alignement des 2 séquences (nombre de correspondances, d'insertions, de suppressions et de substitutions, taille des séquences, …).

Nous utilisons le fichier obtenu lors de la phase d'alignement pour extraire des informations (voir annexe III) à l'aide de Gawk. Ces informations sont ensuite utilisées pour calculer le taux de correspondance, d'insertion, de suppression et de substitution.

Avant de décrire les taux que nous avons calculé, nous allons expliquer certaines dénominations :

N_m : Nombre de correspondances (match/identities).
N_g : Nombre d'insertions et de suppressions (gaps).
N_{mis} : Nombre de substitutions.
L_1 : Taille de la séquence de référence.
L_2 : Taille de la séquence simulée (lecture).

$$\text{Taux de correspondance} = \frac{N_m}{L_1}$$

$$\text{Taux d'ins/supp} = \frac{N_g}{L_1}$$

$$\text{Taux de substitution} = \frac{L_2 - (N_m + N_g)}{L_1}$$

$$\text{Taux d'erreurs} = \frac{(N_m + N_g)}{L_1}$$

Le travail sur ces différentes étapes étant assez long à faire, nous avons conçu un script « shell » qui permet de faire autant de simulations (avec alignements et calculs) souhaitées sur les 3 différents fichiers. Il génère en sortie des fichiers utilisables avec un script « matlab » pour générer des graphiques.

Avant de décrire les caractéristiques des séquences utilisées, nous rappelons le fait que nous appelons une distance la différence des niveaux de courants entre 2 K-mers consécutifs. La somme des distances représente quant à elle la somme de toutes les distances entre les différents K-mers qui se suivent dans une séquence.

Les séquences simulées sont de taille 1000 et ont les caractéristiques suivantes :

- Séquence avec homopolymères (HP) :
  Somme des distances = **12 417.8** (pA)
  Moyenne de la distance = **12,49** (pA)

- Séquence sans homopolymères (NoHP) :
  Somme des distances = **15 950,9**
  Moyenne de la distance = **16,05**

- Séquence sans homopolymères avec distances maximisées (Max) :
  Somme des distances = **16 883,9**
  Moyenne de la distance = **16.99**

- **Résultats :**
  - **Signal en sortie des nanopores :**

Nous pouvons voir ci-dessous, le signal généré en sortie de DeepSimulator à la fois pour la séquence avec HP et la séquence sans HP avec distances optimisées.

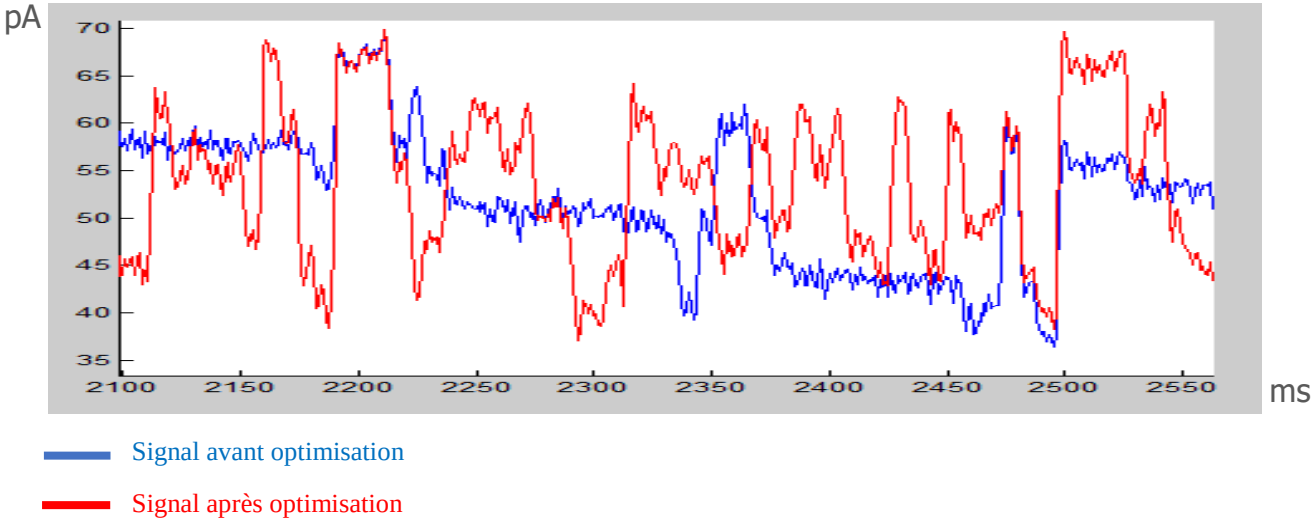


Fig 19 : Signaux en sortie de DeepSimulator

Il apparait clairement que les différences d'amplitude sont plus marquées dans la séquence optimisée que dans l'autre (avec HP), donc l'algorithme arrive bien à augmenter les différences d'amplitude du signal en sortie des nanopores.

- **Taux de correspondances et d'erreurs :**

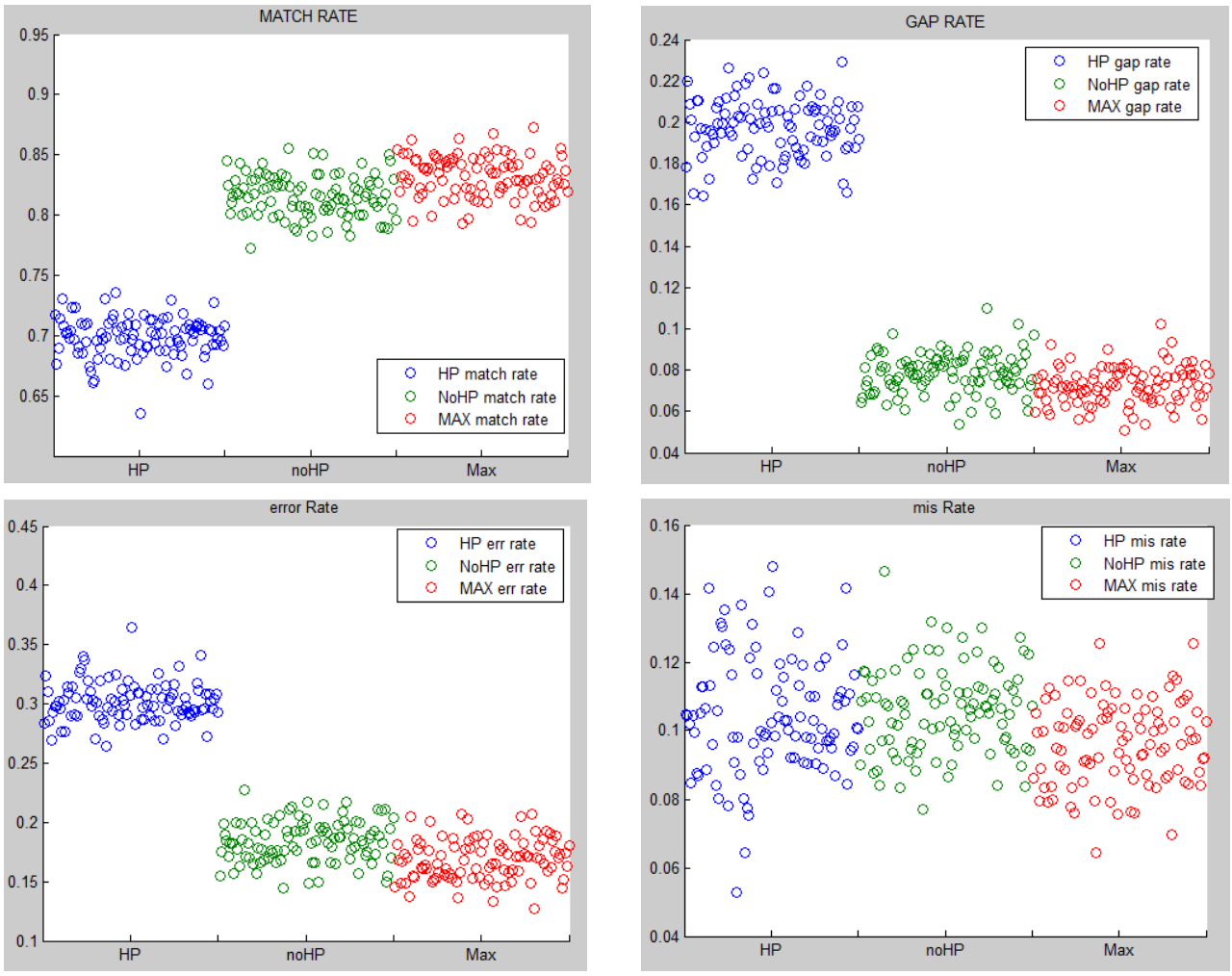


Fig 20 : Taux de correspondances et d'erreurs en sortie de DeppSimulator pour nos différentes séquences.

Nous pouvons remarquer quelques tendances qui se dessinent pour chaque type de séquence. La séquence sans HP (noHP) est bien meilleure que la séquence avec HP (HP), ce qui confirme le fait que les homopolymères rendent le séquencement moins précis. Aussi, la séquence sans HP maximisée (Max) fait mieux que les deux autres séquences, ce qui confirme notre hypothèse de départ c.à.d le fait d'augmenter les différences d'amplitude améliore le résultat du séquencement.

4. Codage de canal :

Dans la partie codage de canal, nous avons travaillé sur des codes correcteurs d'erreurs, en ciblant dans un premier temps les erreurs d'insertion et de suppression. Nous avons donc utilisé les VT codes (Varshamov-Tenengolts codes) pour détecter et corriger une erreur d'insertion ou de suppression dans un mot de codes.

VT codes :

Les VT codes binaires ont été introduits par Varshamov et Tenengolts dans [11] pour les canaux dits asymétriques (suppression ou insertion d'informations) pour la correction d'une erreur de suppression ou d'insertion. Plus tard, Tenengolts introduit dans [12] les VT codes non binaires (alphabet de q symboles).

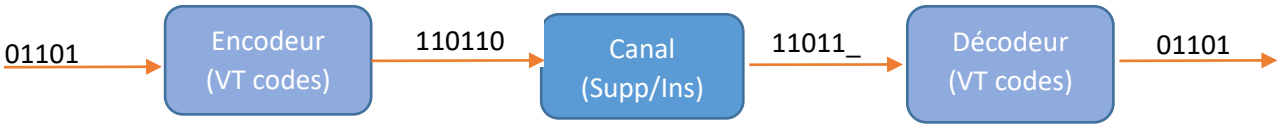


Fig 21 : Flot du codage de canal

- ***Principes VT codes non binaires :***

Dans les VT codes non binaires les mots de code sont formés de q symboles (q = 4 dans le cas de l'ADN). Des positions du mot de code peuvent être utilisées pour coder une partie du message lors de l'encodage. Le décodeur permet de corriger une suppression ou une insertion (une édition) introduite par le canal.

Pour générer les différents mots de codes S, deux paramètres sont fixés : la valeur du syndrome (qu'on appellera « a ») et une somme modulée (qu'on appellera « b »). À partir d'un mot de code, on peut obtenir une séquence auxiliaire A_S (expliquer dans la génération d'un mot de code).

Tout d'abord soit $Z_q = \{0, 1, \ldots, q-1\}$ l'ensemble qui définit l'alphabet utilisé et Z_q^N est un mot de taille N formé des symboles 0 à q-1. $S = s_0 s_2 \ldots s_{N-1}$ est un mot de code de taille N avec $s_i \in Z_q$.

$VT_{a,b}(N)$ est l'ensemble des mots de code de taille N tel que les contraintes a et b fixées sont respectées :

$$VT_{a,b}(N) = \{ S \in Z_q^N : syn(A_S) = a, sum(S) = b \}$$

- ***Génération des mots de code :***

Pour chaque mot S de taille N :

On génère une séquence auxiliaire A_s suivant ses symboles :

$$\boldsymbol{A_s[i]} = \begin{cases} 1 & si\ S[i\text{-}1] \leq S[i] \\ 0 & sinon \end{cases}$$

On calcul ensuite la valeur du syndrome de la séquence A_s et la somme modulée de S tel que :

$$syn\ (A_s) = \sum_{i=0}^{N} \mathrm{i} * \mathrm{s_i}\ mod\ (N+1) = a$$

$$sum(S) = \sum_{i=1}^{N} \mathrm{s_i}\ mod\ (N+1) = a$$

Si la valeur du syndrome ou de la somme modulée est différente des paramètres « a » ou « b » fixés alors ce mot ne fait pas parti de $VT_{a,b}$ (N) (ce n'est pas un mot de code). Le schéma ci-dessous résume ces différentes étapes.

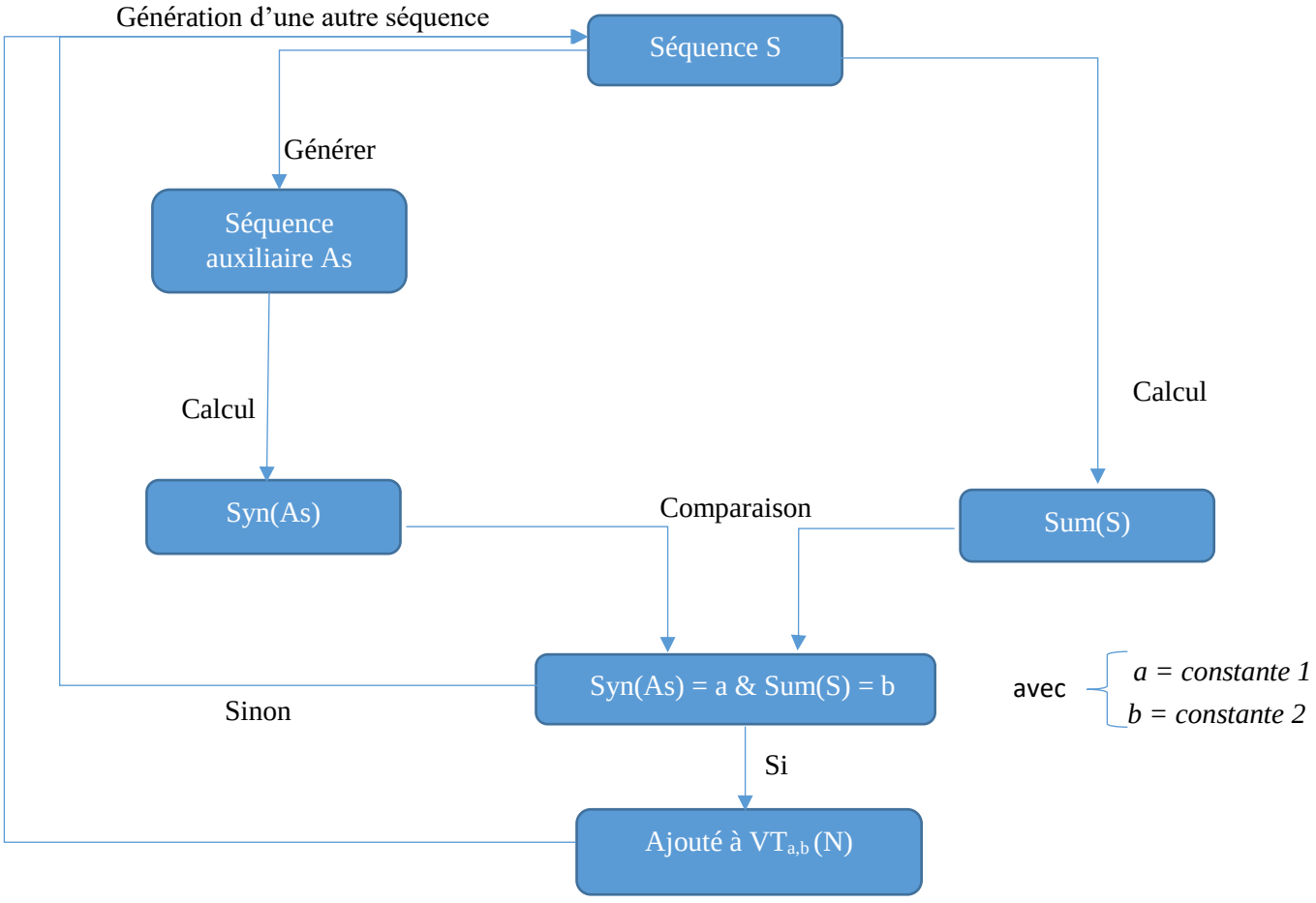


Fig 22 : Génération du dictionnaire de mots de code.

Dans le cas d'un canal qui transmet une séquence contenant des blocs de mots, il est nécessaire de rajouter des préfixes et suffixes à la séquence. Le préfixe est fixé à « 00111 » ou « 11000 » et le suffixe tel que $s_{n-3} = s_{n-2} = s_{n-1} \in Z_q$ comme décrit dans [13].

- ***Décodage des mots de code :***

Dans la partie décodage il est tout d'abord nécessaire de savoir s'il y a eu une suppression ou insertion. Dans le cas d'un canal qui transmet un seul mot de taille n, nous pouvons résoudre ce problème en comptant le nombre de caractères reçus. Cependant, dans le cas d'une séquence contenant plusieurs mots (k mots de taille N), il est plus difficile de savoir s'il y a eu suppression ou insertion ou bien ni l'un ni l'autre. En effet lors du décodage d'une telle séquence, à chaque mot, on hésitera à prendre N+1 symboles (en cas d'insertion), N-1 (en cas de suppression) ou N (si pas d'édition) et la complexité de cette problématique augmentera de plus en plus (au deuxième mot on hésitera à prendre N-2, N-1, N, N+1, N+2 et ainsi de suite). D'où l'utilisation des préfixes et suffixes cités précédemment.

Soit une séquence Y reçue et formée de k mots de taille N chacun. $Y(p_i+1 : p_i+b)$ représente le mot de code allant de la position p_i+1 à p_i+b dans la séquence Y. Le décodage se fait en deux étapes :

- ***État du mot de code :***

La première étape du décodage est de déduire le type de l'édition (s'il y en a eu). Pour déduire le type de l'édition nous avons utilisé l'algorithme décrit dans [13]. Suivant la valeur du syndrome et de la somme modulée lors du décodage, deux possibilités existent :

Si $syn(Aux(Y(p_i+1:p_i+b))) \neq a$ *ou* $sum(Y(p_i+1:p_i+b)) \neq b$ *(il existe une édition dans le mot)*

State of sequence	Type of edit
$y_{p_i+b-1} = y_{p_i+b} = y_{p_i+b+1}$	Insertion
$y_{p_i+b-1} = y_{p_i+b} \neq y_{p_i+b+1}$	Deletion
$y_{p_i+b-1} = y_{p_i+b+1} \neq y_{p_i+b}$ and $\text{syn}(Z) \neq a_0$, where $Z = [Y(p_i+1 : p_i+b-1), y_{p_i+b+1}]$	Deletion
$y_{p_i+b-1} = y_{p_i+b+1} \neq y_{p_i+b}$ and $\text{syn}(Z) = a_0$ and $y_{p_i+b+1} = y_{p_i+b+2} = y_{p_i+b+3}$	Deletion
$y_{p_i+b-1} = y_{p_i+b+1} \neq y_{p_i+b}$ and $\text{syn}(Z) = a_0$ and $(y_{p_i+b+1} \neq y_{p_i+b+2}$ or $y_{p_i+b+1} \neq y_{p_i+b+3})$	Insertion
$y_{p_i+b-1} \neq y_{p_i+b} = y_{p_i+b+1}$ and $y_{p_i+b-2} = y_{p_i+b-1}$	Deletion
$y_{p_i+b-1} \neq y_{p_i+b} = y_{p_i+b+1}$ and $y_{p_i+b-2} \neq y_{p_i+b-1}$	Insertion
$Y_{pi+b-1} \neq Y_{pi+b}$ *and* $Y_{pi+b+1} = Y_{pi+b+2}$	***Deletion***
$Y_{pi+b-1} \neq Y_{pi+b} = Y_{pi+b+2}$	***Deletion***

Si $syn(Aux(Y(p_i+1:p_i+b))) = a$ *et* $sum(Y(p_i+1:p_i+b)) = b$
(mot correcte , on vérifie s'il y a eu insertion à la fin du mot c.à.d à la position p_i+b+1 *)*

$Y(p_i + b + 1 : p_i + b + 5)$	State of y_{p_i+b+1}
$1uvst$	Inserted
$000uv$	Inserted
$011uv$	Not Inserted
01000	Not possible
01001	Inserted
01010	Not possible
01011	Not inserted
00100	Not possible
00101 and $\text{syn}(Z_1)$ matches	Inserted
00101 and $\text{syn}(Z_2)$ matches	Not Inserted
00110	Not Inserted
00111	Not Inserted

Fig 23 : Tables dans [13] représentant l'état du mot de code décodé.

***** *Aux(Y(*p_i*+1:*p_i*+b)) : séquence auxiliaire de Y(*p_i*+1:*p_i*+b)*
***** *Partie du tableau en vert : Ajouté à l'algorithme original pour prendre en compte certains cas non binaires.*

- ***Correction du mot de code (dans le cas d'une édition) :***

Dans l'étape de correction du mot de code, on suppose qu'il y a eu édition et qu'on connait le type de celle-ci (détecter dans l'étape 1). Dans le cas d'une insertion, nous traitons le bloc $Y(p_i+1 : p_i+b+1)$ pour supprimer le symbole inséré et dans le cas d'une suppression, on traite le bloc $Y(p_i+1 : p_i+b-1)$ pour récupérer le symbole perdu.

Dans cette étape les paramètres « a » et « b » sont utilisés pour la correction du mot. Le schéma ci-dessous représente les différentes étapes du décodage tel que décrit dans [14] :

- ***Cas de suppression d'un symbole :***

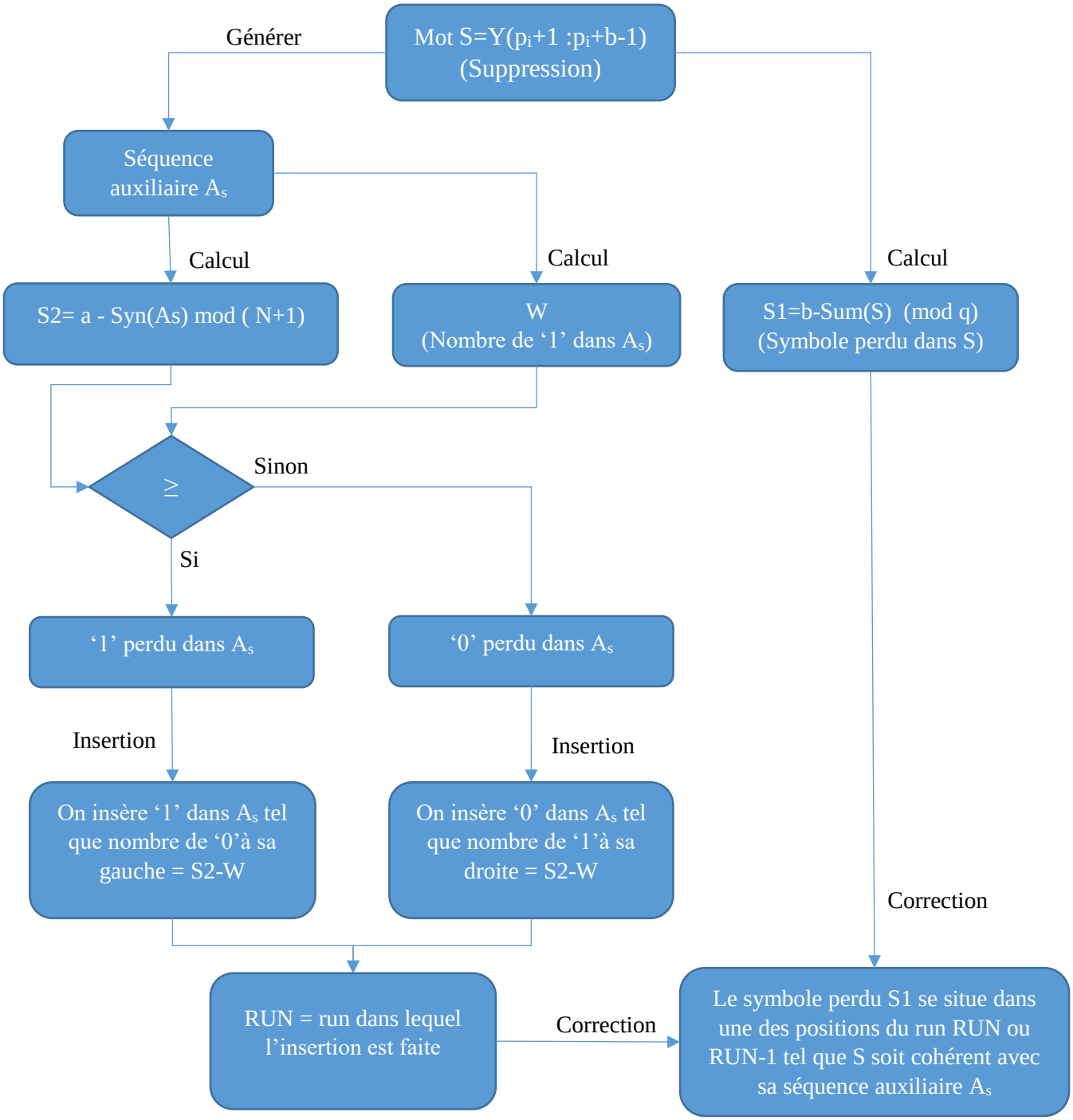


Fig 24 : Algorithme de correction d'une erreur de suppression

***Run : ensemble de symboles d'une même valeur qui se suivent** 00111010111*

- ***Cas d'insertion d'un symbole :***

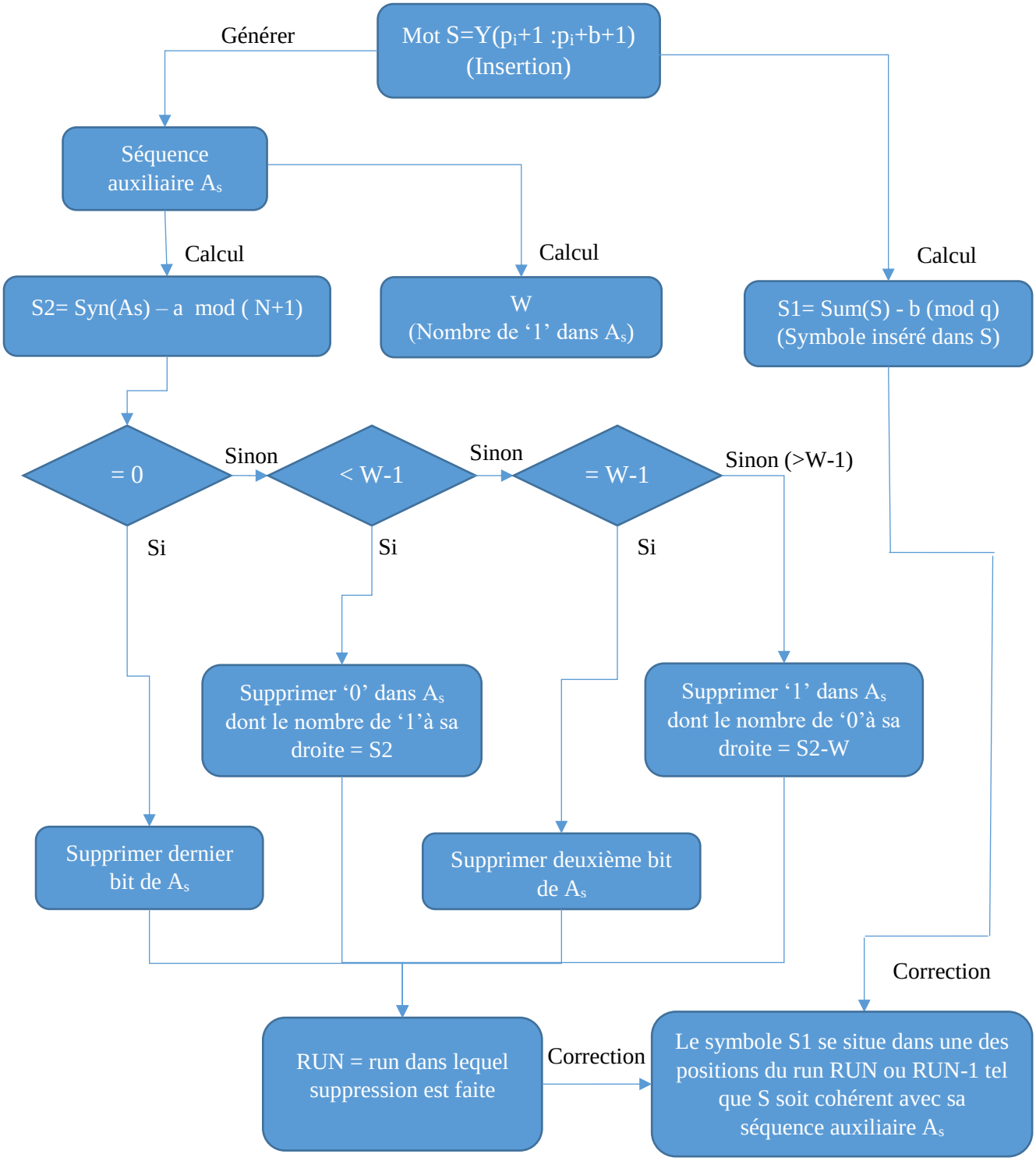


Fig 25 : Algorithme de correction d'une erreur d'insertion

Ce codage permet à nos séquences ADN de pouvoir récupérer d'une éventuelle erreur d'édition tous les 12 nucléotides (mot de code de taille N=12). Cependant, il peut arriver qu'il y ait plus d'une erreur d'édition dans un mot et dans ce cas-là, on ne peut pas récupérer le mot original.

CONCLUSION

Face au défi de proposer des systèmes de stockages denses et capables de stocker la masse de données produite, nous avons à travers ce travail exploré l'alternative que peuvent offrir les systèmes de stockage sur ADN. Nous avons donc travaillé sur différents mécanismes pour mettre en œuvre un système de stockage de données sur ADN, ainsi que les améliorations possibles sur celui-ci, pour rendre le séquencement (lecture des données) par MinION plus précis.

Dans un premier temps, nous avons travaillé sur un codage source qui permet de transformer les données numériques en une séquence de caractères (composée de 'A', 'C', 'G' et 'T'). Nous avons ensuite utilisé et proposé différents algorithmes pour améliorer la précision de la lecture. Ces algorithmes permettent d'éliminer les homopolymères dans nos séquences et de maximiser les distances d'amplitude entre les différents K-mers contigus. Nous avons dans un second temps pu constater l'effet de ces algorithmes avec les simulations. Aussi, nous avons travaillé sur différents outils de simulation parmi lesquels, DeepSimulator qui répond le mieux à nos besoins. Il apparait que les algorithmes que nous avons utilisés permettent d'améliorer le séquencement (donc la précision de la lecture). Dans la dernière partie, nous avons travaillé sur les codes correcteurs d'erreurs (codage de canal). Nous avons implémenté un modèle de VT codes adapté à l'utilisation de séquences formées de plusieurs mots de code. Ce modèle est capable de corriger une erreur d'édition (suppression ou insertion) sur chaque mot de code de la séquence.

Par ailleurs, un vaste travail mériterait d'être mené sur les différentes parties décrites pour améliorer la précision du séquencement par MinION. Il serait intéressant d'approfondir l'étude de la maximisation des distances pour proposer d'autres algorithmes qui réduisent la redondance de données produite par les actuels. Un travail important est aussi nécessaire sur les codes correcteurs d'erreurs. Pour la correction des insertions/suppressions il faudrait prendre en compte les cas où le nombre d'édition est supérieur à 1. Il est aussi nécessaire de travailler sur la correction des erreurs de substitution (non abordée pendant ce stage).

À l'issue de ce stage au LAB-STICC, je peux dire que je suis satisfait de cette expérience où, j'ai eu la possibilité de mettre en pratique les différentes connaissances acquises lors de ma formation. Je pense avoir acquis de solides connaissances sur ce sujet transdisciplinaire qui mêle à la fois électronique, biologie et science des données. Sujet sur lequel j'ai la chance de continuer à travailler en thèse doctorale.

RÉFÉRENCES

1. Microsoft Plans to Have a DNA-Based Computer by 2020, https://bigthink.com/philip-perry/microsoft-plans-to-have-a-dna-based-computer-by-2020 .

2. Lab-STICC, https://www.labsticc.fr/en/francais/ .

3. Image molécule d'ADN, https://iheartmorris.weebly.com/nucleic-acids--nucleotides.html .

4. Molécules d'ADN dans éprouvette, image produite par Microsoft Research and University of Washington.

5. Séquencement par nanopores, image produite par Oxford Nanopore Technologies.

6. Signal utilisé lors de la phase de basecalling, image produite par l'auteur de DeepSimulator.

7. Synthèse d'ADN ThermoFisher, image produite par ThermoFisher.

8. N. Goldman "Method for Encoding and Decoding Arbitrary Computer Files in DNA Fragments", https://www.ebi.ac.uk/sites/ebi.ac.uk/files/groups/goldman/file2features_2.0.pdf .

9. Chen Yang, Justin Chu, René L Warren, Inanç Birol, NanoSim: nanopore sequence read simulator based on statistical characterization, *GigaScience*, Volume 6, Issue 4, April 2017, gix010, https://doi.org/10.1093/gigascience/gix010 .

10. Yu Li, Renmin Han, Chongwei Bi, Mo Li, Sheng Wang, Xin Gao, DeepSimulator: a deep simulator for Nanopore sequencing, *Bioinformatics*, Volume 34, Issue 17, 01 September 2018, Pages 2899–2908, https://doi.org/10.1093/bioinformatics/bty223 .

11. R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," Automatica i Telemekhanica, vol. 26, no. 2, pp. 288–292, 1965.

12. G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion," IEEE Trans. Inf. Theory, vol. 30, no. 5, pp. 766–769, 1984.

13. M. Abroshan, R. Venkataramanan and A. Guillén i Fàbregas, "Coding for Segmented Edit Channels," in *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 3086-3098, April 2018.

14. G. Tenengolts, "Nonbinary codes correcting single deletion or insertion", *IEEE Trans. Inf. Theory*, vol. 30, no. 5, pp. 766-769, Sep. 1984.

Annexe I :

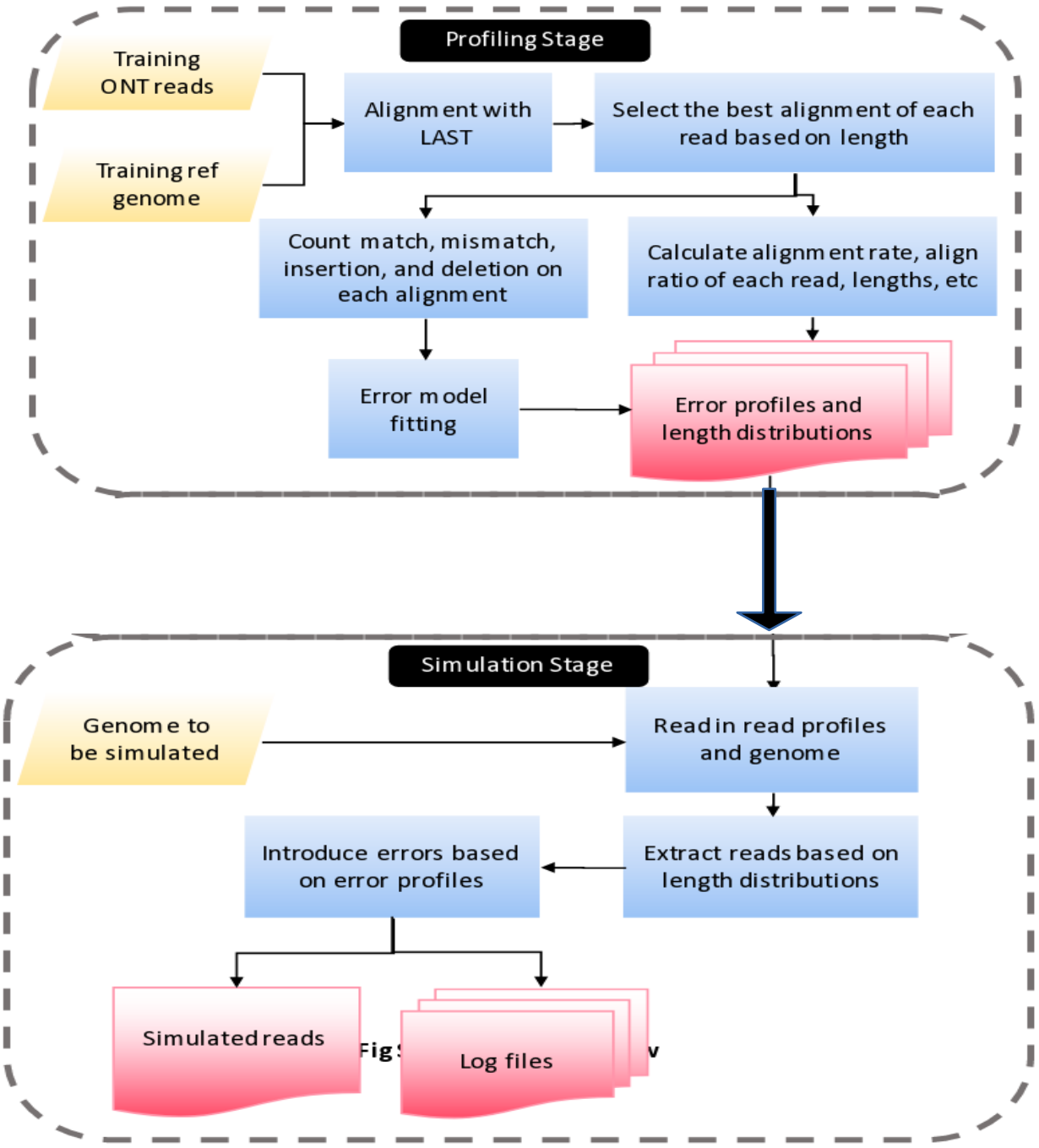


Fig 26 : Phases de caractérisation d'un profil et de simulation.

Annexe II :

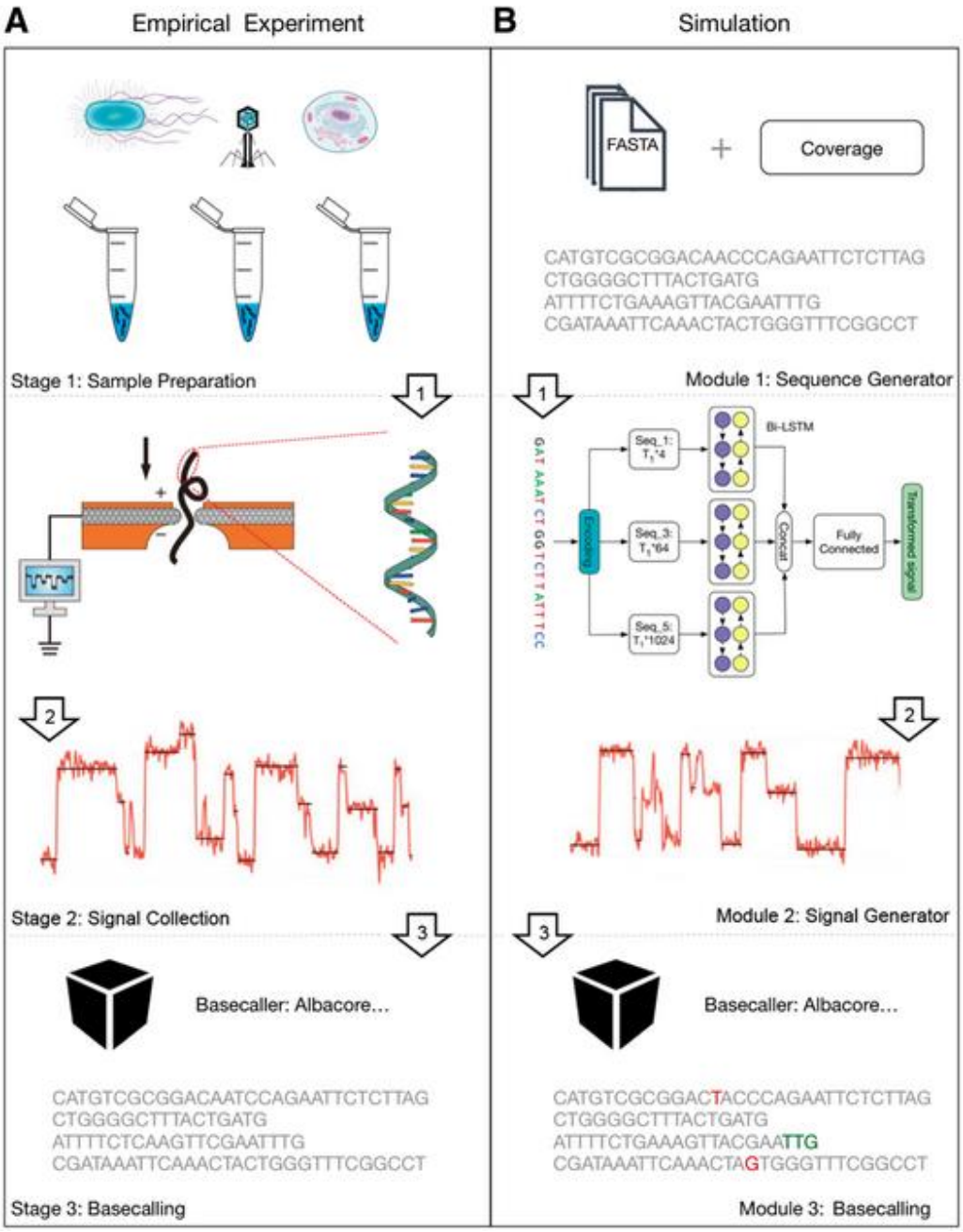


Fig 27 : Phases d'un séquencement réel et simulé par DeepSimulator.

Annexe III :

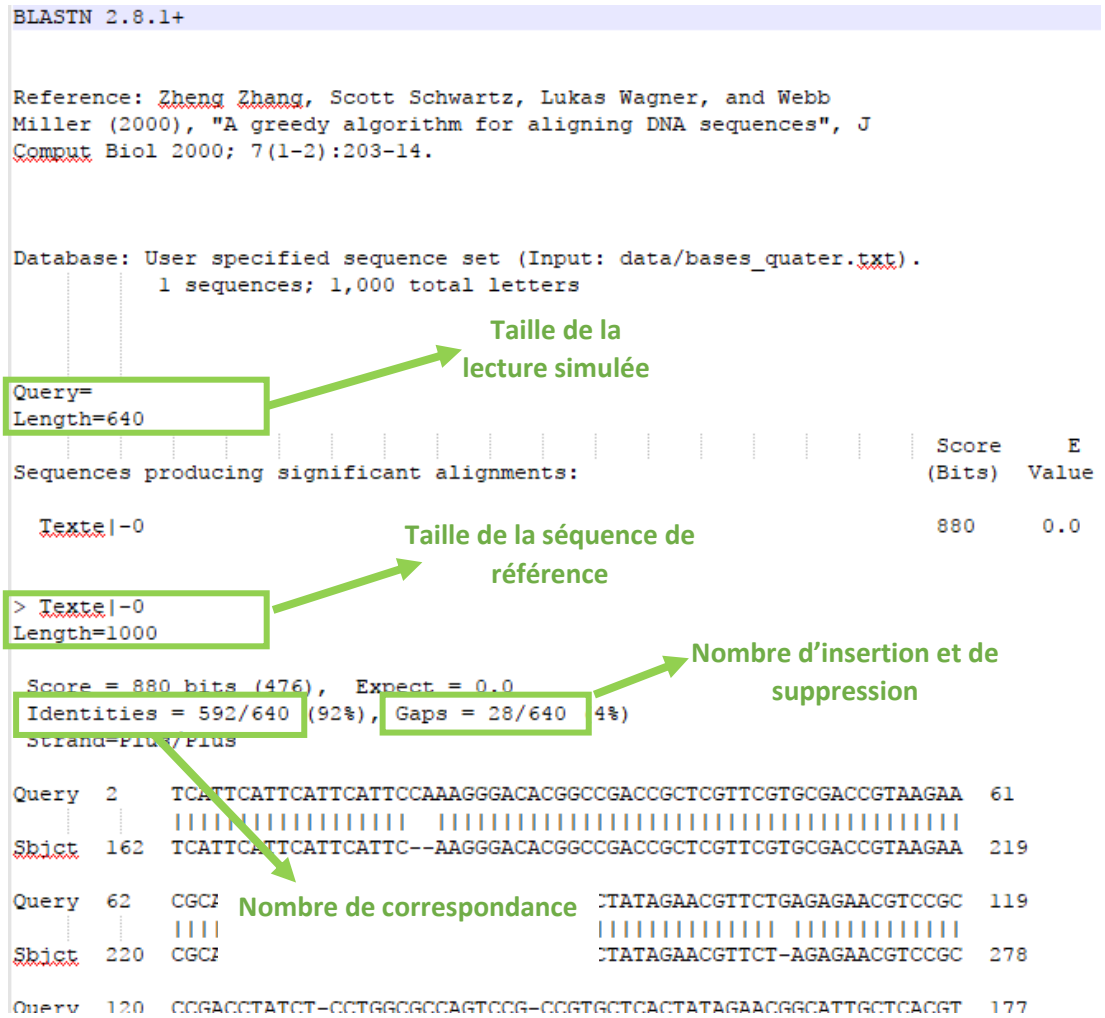


Fig 28 : Fichier en sortie de l'utilitaire d'alignement ggsearch sous le format BLAST.