Rapport de stage M2 : Développements de méthodes d'analyse de l'épissage alternatif à partir de données RNA-seq

Sasha Darmon
Département d'Informatique
ENS Lyon

28 juin 2022

Encadré par :
Vincent Lacroix & Arnaud Mary
LBBE
Université Lyon 1

Table des matières

Introduction **3**

1 Épissage alternatif et graphe de De Bruijn **4**

1.1 Définitions Biologiques 4

1.2 Définitions informatiques 5

1.3 Contexte et enjeux du stage 6

1.4 Stratégies mise en place dans *KisSplice* 7

1.5 Difficultés d'une telle résolution 7

1.6 Note sur l'état de l'art et stratégies préexistantes 8

2 Résolution du problème **9**

2.1 Méthodologie 9

2.2 Hypothèses et Modélisations 9

2.3 Algorithme de simplification du Graphe de De Bruijn 10

2.4 Tests et examen des résultats 11

2.5 Des composantes pour contrecarrer la limitation du branchement 13

3 Application, analyse et améliorations **14**

3.1 Quatre jeux de données 14

3.2 Développement du cas du moustique 16

3.3 Découverte d'éléments transposables 17

3.4 Reproductibilité, Améliorations et Critiques 18

Conclusion **20**

Introduction

D'abord considéré comme exceptionnel lors de sa découverte en 1977 (1), l'**épissage alternatif** est maintenant connu pour être l'un des mécanismes qui participe le plus à la diversité du vivant. Permettant à un même gène d'être exprimé de différentes manières, l'épissage alternatif et sa dérégulation jouent également un rôle dans les cancers ainsi que de nombreuses maladies. En terme d'ordre de grandeur chez l'Homme, ce mécanisme touche environ 95% des gènes et on estime que environ 15% des cancers et maladies héréditaires ont été reportés être en lien avec de l'épissage alternatif (2). Ainsi l'étude et l'analyse de ce mécanisme est capital à la compréhension du vivant, au traitement de maladies variées et dont l'utilité s'étend à un large nombres espèces. Néanmoins, l'existence de répétitions et d'**éléments transposables** (des séquences génétiques capables de se répliquer et se déplacer) contraignent l'efficacité et la précision des méthodes d'analyse de l'épissage alternatif. Il s'agit en particulier du cas pour le programme *KisSplice*(3) développé par des membres du laboratoire *LBBE* de Lyon.

Le but du stage qui m'a été proposé est de développer de nouvelles méthodes d'analyse des données permettant d'éluder ces éléments transposables et autres répétitions, le tout pour être utilisé et appliqué par le logiciel *KisSplice*. Alors pendant cinq mois, j'ai pu m'approprier ce problème biologique afin d'expérimenter différentes approches originales de résolution que j'ai pu, par la suite, les confronter les unes par rapport aux autres dans l'objectif de cibler au mieux le coeur du problème.

Je tiens tout d'abord à remercier Vincent Lacroix et Arnaud Mary de m'avoir proposé ce passionnant sujet. Merci à eux d'avoir su répondre à mes questions et de m'avoir parfaitement encadré dans mon stage. Ensuite, je souhaite remercier l'équipe *Baobab* (également appelée *Érable*), ses permanents, ses doctorants et ses stagiaires pour m'avoir accueilli sur place ainsi que pour la constante bonne humeur qui régnait à cet étage *Mezzanine*. Merci également aux divers chercheurs avec qui j'ai pu échanger pour avoir leur avis et qui ont gentiment partagé leurs jeux de données avec moi. Enfin, j'adresse mes remerciements à toutes les personnes qui m'ont aidées à la rédaction et à la relecture de ce rapport de stage.

J'ai consacré la première partie de mon stage à un travail bibliographique afin d'appréhender au mieux le sujet et le contexte de mon stage. J'ai pu alors situer et mesurer les difficultés de ce problème biologique puis mettre en place et définir les notions et modèles nécessaires à l'élaboration de méthodes pour la résolution du problème.

Par la suite, j'ai pu alors développer une méthode de pré-analyse des données permettant de cibler les éléments transposables et autres répétitions conduisant alors à une amélioration de l'analyse de l'épissage alternatif. Par un raisonnement méthodique, j'ai pu enrichir mes modèles afin de me rapprocher des résultats escomptés. Cela a conduit au développement d'un algorithme que j'ai pu tester dans le but de valider les résultats obtenus.

Finalement, j'ai pu appliquer mes algorithmes sur des jeux de données concrets, et plus particulièrement sur celui du moustique dont les résultats seront présentés dans ce rapport. Par ces méthodes informatiques, j'ai notamment pu découvrir de nouvelles insertions d'éléments transposables, i.e. des régions non annotées du génome où sont exprimés des éléments transposables.

1 Épissage alternatif et graphe de De Bruijn

1.1 Définitions Biologiques

Tout d'abord, le **génome** d'un individu est, formellement, l'ensemble du matériel génétique contenu dans l'**ADN** (ou acide désoxyribonucléique). Biologiquement, l'ADN est composé de deux **brins** constitués de **nucléotides**. De plus, il n'existe que quatre versions de nucléotides qui permettent de coder n'importe quelle information du vivant et que l'on note `A,T,G,C`. D'autre part, les nucléotides d'un brin à l'autre ne sont pas indépendants, chaque séquence d'un brin est toujours le **complémentaire** de l'autre, chaque nucléotide d'un brin s'appariant toujours avec son nucléotide complémentaire selon les règles suivantes : `A` s'apparie toujours avec `T` et `C` s'apparie toujours avec `G` (et réciproquement). De ce fait, une même séquence d'ADN possède deux représentations, l'une dans son sens **forward** (**direct** en français) qui est le sens telle qu'elle est écrite, par exemple `AATGCG`, et l'autre dans le sens **reverse complement** (ou **complémentaire inverse**), qui correspond à la séquence du brin complémentaire, lu alors dans le sens contraire, ce qui donne dans notre exemple `CGCATT`.

Ensuite, chez les eucaryotes [1], on décompose généralement l'ADN en **gènes** : une séquence discrète qui peut s'exprimer chez un individu et/ou dont son expression altère un ou des caractère(s) de l'individu en question. Plus concrètement, le gène se décompose en plusieurs blocs de deux types : des **exons** et des **introns**. La différence entre ces deux types est qu'une fois l'ADN transcrit en **ARN** (un support intermédiaire de l'information génétique nécessaire à la production des protéines constituant le vivant), un processus d'**épissage** va pouvoir être appliqué sur ces molécules et alors, seuls les exons seront conservés et utilisés pour produire des protéines, et les introns quant-à eux seront éliminés de l'ARN. Cependant, lors de cet épissage, il est également possible que des exons soient éliminés permettant alors pour un gène donné d'obtenir de nombreux ARNs différents qui donneront par la suite des protéines distinctes. Ainsi, ce mécanisme est ce qui se nomme l'**épissage alternatif** et en voici un exemple en Figure 1.

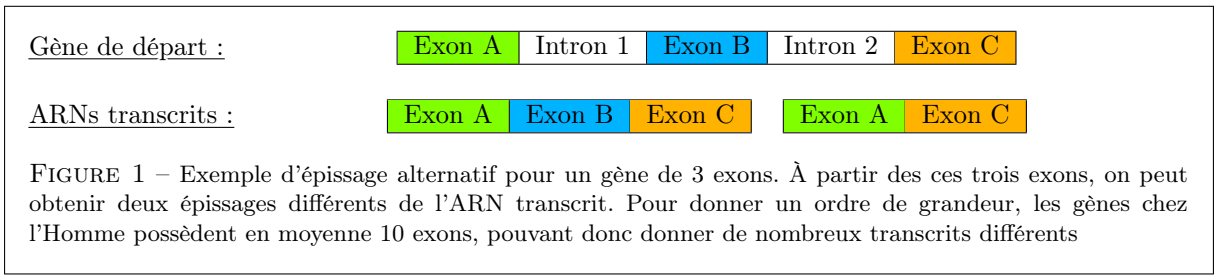

Figure 1 – Exemple d'épissage alternatif pour un gène de 3 exons. À partir des ces trois exons, on peut obtenir deux épissages différents de l'ARN transcrit. Pour donner un ordre de grandeur, les gènes chez l'Homme possèdent en moyenne 10 exons, pouvant donc donner de nombreux transcrits différents

Enfin, aujoud'hui l'épissage est étudié grâce à l'utilisation de **techniques de séquençage à haut débit** appliquées à l'ARN ; on parle alors de protocole **RNA-seq**. Elle consiste à cibler certaines molécules d'ARN d'une cellule, les **ARNs polyadénylés** qui ont la particularité de ne pas être dégradées et de correspondre à l'expression des protéines. Sans prendre cette précaution, on risque de séquencer également de **ARN ribosomique** qui sert uniquement à la formation ribosome [2] mais qui représente 95% de l'ARN présent dans les cellules. Une fois captées, on peut les couper en fragments d'une longueur de quelques centaines de nucléotides puis les marquer, les séparer et les reconvertir en ADN. À l'issue de cette étape, il est désormais possible d'obtenir les séquences via des systèmes de séquençage tels que l'*Illumina*(4). On obtient alors des millions de séquences appelées **lectures** et qui correspondent à des petites portions des ARNs de départ. Il est alors nécessaire de mettre en place diverses méthodes informatiques pour reconstruire les séquences d'origine.

1. Domaine du vivant qui regroupe les animaux, les plantes et les champignons, et qui est caractérisé par le fait de contenir l'ADN dans le noyau de ses cellules

2. Le ribosome est la machinerie cellulaire responsable de la traduction des ARN messagers tels que les ARNs polyadénylés.

1.2 Définitions informatiques

L'une des représentations des lectures les plus utilisées et que j'ai utilisée lors de mon stage est par un **graphe de De Bruijn** (ou **GDB**). Formellement, chaque lecture est un mot sur l'alphabet $\{\texttt{A}, \texttt{T}, \texttt{G}, \texttt{C}\}$ et l'idée est de fixer au préalable un entier k puis de considérer tous les sous-mots de longueurs k de chacune des lectures. On appelle alors ces sous-mots des k-mers et on définit l'ensemble V des sommets du graphe de De Bruijn comme étant tous les k-mers ainsi trouvés. Ensuite, on définit l'ensemble des arêtes orientées E tout simplement en regardant toutes les paires de sommets $(u, v) \in V^2$ tels que les mots u et v se chevauchent, c'est-à-dire tels que les $k-1$ dernières lettres de u soient exactement les $k-1$ premières lettres de v. De cette manière on obtient le graphe orienté $G = (V, E)$, appelé graphe de De Bruijn. Dans le cadre de mon stage, ce paramètre k a été fixé à 41 par défaut, cette valeur étant choisie afin d'éviter que deux k-mers de deux lectures différentes soient égaux par hasard, mais aussi afin que la probabilité qu'il y ait une erreur de séquençage dans le k-mer soit négligeable et/ou corrigible.

En réalité il y a quelques simplifications qui sont réalisées sur ce graphe. Tout d'abord, avec la même idée qu'une séquence d'ADN a deux sens, le sens **forward** et **reverse**, chaque sommet peut également être interprété dans chacun de ces deux sens. Alors, pour distinguer les arêtes partant d'un même sommet mais pas du même sens, on les différencie en 4 types : FF, FR, RF et RR dont la première lettre indique le sens de départ (F pour forward et R pour reverse) et la seconde lettre indique le sens d'arrivée. Ensuite, la seconde simplification est la fusion des sommets adjacents non **branchants** en un nouveau sommet appelé **unitig**. On définit un sommet comme étant branchant si son degré entrant ou sortant est au moins de 2 par le sens forward ou reverse (et inversement, un sommet est dit non branchant s'il a un degré entrant et sortant inférieur à 1). Alors, on peut remarquer que des sommets non branchants adjacents ne peuvent former qu'un chemin qui ne peut alors être interprété que d'une seule manière, uniquement comme la séquence de nucléotides parcourues en suivant le chemin (en prenant en compte le chevauchement des k-mers deux à deux). Cette séquence est traduite en un sommet nommé **unitig** et ainsi, une fois cette étape effectuée sur tous les sommets adjacents, on obtient un graphe dit **graphe de De Bruijn compacté** (ou **GDBC**). Un exemple de ces deux graphes est présenté sur la Figure 2.

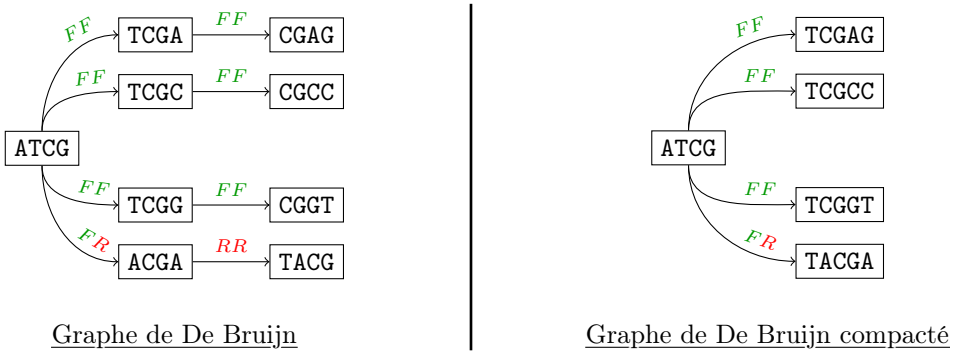

Figure 2 – Exemple de construction d'un graphe de De Bruijn et un graphe de De Bruijn compacté à partir des lectures `ATCGAG`, `ATCGCC`, `ATCGGT`, `ATCGTA` et de $k = 4$. Par un soucis de clarté, seul le sens forward des sommets est dessiné et une seule arête est dessinée entre les paires de sommets, mais formellement, pour toute arête FF d'un sommet u à un sommet v, il existe une arête inverse RR de v à u. D'autre part, le complément inverse de `ACGA` étant `TCGT`, il existe une arête FR de `ATCG` à `ACGA` (et formellement aussi une arête FR de `ACGA` à `ATCG`, non représentée ici).

Ainsi avec cette modélisation, on peut retrouver les informations recherchées. Les molécules d'ARN transcrites correspondent alors aux chemins maximaux de ce graphe, et l'épissage alternatif va correspondre aux **bulles** dans le graphe. Ces dernières sont tout simplement définies par deux chemins qui partagent en commun seulement leur premier et leur dernier sommets. Une représentation de la bulle formée par l'épissage alternatif de la Figure 1 est disponible sur la Figure 3.

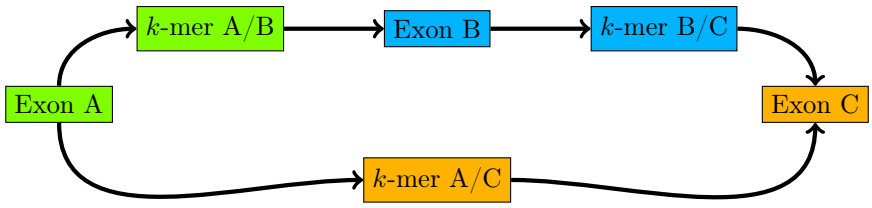

Figure 3 – Bulle représentant l'épissage alternatif de la Figure 1. Les nouveaux sommets "k-mer X/Y" correspondent aux k-mers du GDB à cheval sur l'exon X et l'exon Y et ont été formés car il y a deux façons possibles de continuer la suite de l'exon X. L'ensemble des chemins que l'on peut lire dans cette bulle correspond à l'épissage alternatif de ce gène.

1.3 Contexte et enjeux du stage

De l'importance de ce mécanisme et des maladies avec lesquelles il est en lien, l'épissage alternatif et son étude permettent d'expliquer voire de proposer des traitements ciblés et intéressent tout particulièrement les chercheurs en biologie. Le développement de méthodes propres à l'analyse de l'épissage alternatif est notamment motivé par son application à des **espèces non modèles**[3] sans utiliser de séquence de référence du génome étudié. En effet, la différence entre une espèce modèle et une autre non modèle se joue principalement sur la possession (ou non) d'un génome de référence complet et de bonne qualité de l'espèce, permettant une grande simplification du problème mais qui est malencontreusement bien plus compliquée à obtenir. De ce fait, il n'est pas toujours possible d'y avoir accès, et en terme de chiffres, moins de 0.2% des espèces ont leur génome séquencé (5) et seulement une très faible fraction de ces espèces possèdent un tel assemblage complet et de suffisamment bonne qualité pour pouvoir être efficacement utilisé (6).

Ainsi, le logiciel *KisSplice* (3) a été développé afin d'analyser l'épissage alternatif sans avoir recours à une séquence de référence. Cependant, les répétitions, et en particulier les éléments transposables qui peuvent être dupliqués à des centaines de positions différentes dans le génome. Alors, quand les transcrits vont correspondre à un chemin du graphe, certains de ses chemins ne correspondent à aucun transcrit, à des répétitions, limitant l'efficacité des algorithmes. Et comme les éléments transposables ne sont pas *a priori* des épissages alternatifs, ils peuvent être enlevés ou simplifiés sans risquer de perdre de l'information d'épissage alternatif.

Alors, mon travail lors de ce stage a été de développer de nouvelles méthodes qui pourront permettre d'améliorer l'efficacité du logiciel *KisSplice* en permettant de détecter des épissages alternatifs dans des régions du génome riches en répétions. Ce travail a été de modifier et simplifier le graphe d'assemblage qui est utilisé par *KisSplice*, de manière à prendre en compte ces éléments transposables et les répétitions dans l'objectif de permettre une meilleure énumération des évènements d'épissage alternatif. De plus, il est à noter que le problème de la gestion des régions répétées dans l'**assemblage de novo** de génomes[4] et de **transcriptomes**[5] reste des problèmes ouverts en bioinformatique, dont la résolution n'est possible que sous certaines conditions encore aujourd'hui non suffisantes (7).

Enfin, il y a un intérêt direct à continuer de travailler sur le logiciel *KisSplice* car il est déjà utilisé par de nombreux chercheurs et une telle amélioration dans l'énumération de l'épissage alternatif profiterait donc à une large communauté. Alors, un tel travail de recherche informatique est nécessaire et est motivé pour toutes les raisons mentionnées ci-dessus.

3. Une espèce modèle est une espèce biologique qui est étudiée de manière approfondie afin de comprendre tous les mécanismes biologiques mis en place. En particulier, il s'agit souvent des espèces dont le génome est déjà assemblé (i.e. dont l'ADN des cellules a été fidèlement reconstitué) et très documenté.

4. L'assemblage génétique consiste à reconstituer le génome d'un individu (ou d'un espèce) à partir d'un jeu de données correspondant à un ensemble de séquences ADN tirées des cellules de cet individu. En particulier, l'assemblage **de novo** est un assemblage qui n'utilise aucune séquence de référence pour s'aider à aligner les séquences.

5. Le transcriptome est l'ensemble des molécules d'ARN transcrites à une certaine échelle donnée qui peut être celle de la cellule, d'un organe ou d'un individu par exemple.

1.4 Stratégies mise en place dans *KisSplice*

Le logiciel *KisSplice* fonctionne comme suit : on lui donne en entrée à la fois les lectures de un ou plusieurs individus dont on souhaite analyser l'épissage alternatif ainsi que les lectures d'un ou plusieurs individus de contrôle, permettant *a posteriori* de mettre en évidence les différences dans l'épissage des gènes. Ensuite l'ensemble de ces lectures est transformé en un graphe de De Bruijn compacté via le programme *BCALM* développé par Rayan Chikhi (8). C'est sur ce graphe que le programme *KisSplice* entre en jeu, il va directement énumérer les bulles (les paires de chemins partageant les mêmes premiers et derniers sommets et étant disjoints sur les autres sommets) présentes sur le-dit graphe en y appliquant plusieurs contraintes. Parmi celles-ci, il y a des contraintes biologiques qui imposent aux bulles de vérifier une certaine taille et une certaine structure mais il existe aussi une contrainte algorithmique, nécessaire à l'exécution du programme en un temps imparti, qui est le paramètre que l'on note b : le nombre maximal de sommets branchants (i.e. sommets formant au moins deux chemins possibles à la suite de l'une de ses lectures) dans la bulle.

Ainsi, le nombre de bulles trouvées est en pratique exponentiel en ce paramètre b, principalement à cause des sommets de forte densité (une preuve algorithme est donnée en Annexe *A. Preuve de la croissance exponentielle du nombre de bulles en fonction du paramètre b*). Alors, en pratique on fixe ce paramètre à 5, ce qui signifie que chaque chemin d'une bulle ne passera pas par plus de 5 sommets branchants. Cela permet alors d'éviter de parcourir les zones à forte densité et de devoir étudier un trop grand nombre de chemins pour être calculé empiriquement. Cependant, à cause de ce paramètre les bulles présentes dans ces zones ne sont pas toutes trouvées alors que certaines sont pertinentes biologiquement.

1.5 Difficultés d'une telle résolution

Au cours de l'analyse du sujet et des données, j'ai pu mettre en évidence les différents paramètres faisant de la recherche de l'épissage alternatif un problème algorithmique difficile.

Le premier paramètre est l'explosion combinatoire du nombre de bulles dans le graphe. En effet, lorsque l'on fait l'union de deux sous-graphes, on multiplie leur nombre de bulles ; théoriquement si un sous-graphe possède n bulles et un autre sous-graphe disjoint en possède m, on peut alors montrer que le graphe induit par l'ensemble des sommets considérés peut posséder plus nm bulles (voir exemple en Annexe *B. Exemple de la multiplication du nombre de bulles de deux sous-graphes*). En pratique, un unique chemin peut appartenir à des milliers de bulles différentes, correspondant soit à des répétitions génomiques (par exemple à cause de la présence d'éléments transposables), soit à un ou plusieurs véritables évènements d'épissage alternatif mais qui sont décliné chacun en de multiples versions dans le graphe à cause de la présence de **polymorphisme**[6], le nombre de bulles trouvées au niveau de la variation va jusqu'à être quadruplé (voir preuve en Annexe *C. Preuve de l'impact du polymorphisme sur le nombre de bulles d'un graphe*). Ainsi, d'un point de vue complexité, on souhaite un algorithme de complexité temporelle quasi-linéaire en la taille des données ($O(n \log n)$ et/ou d'une complexité temporelle amortie quasi-linéaire (c'est-à-dire, une complexité telle que l'ensemble du programme tourne est équivalent en moyenne à du quasi-linéaire).

Ensuite, le second paramètre à prendre en compte est lié au protocole expérimental RNA-seq. Pour obtenir de telles données, on place généralement des adaptateurs aux molécules d'ADN que l'on souhaite séquencer ; adaptateurs que l'on peut malencontreusement retrouver dans nos données. De plus, comme j'étudie des données RNA-seq, celles-ci correspondent à de vraies molécules d'ARN qui possèdent bien souvent une **queue poly(A)**, c'est-à-dire une chaîne physique de dizaines de nucléotides A à la fin de la molécule. Alors, ces deux mécanismes créent des structures chimériques dans le graphe. En correspondant à des multitudes de séquences différentes, ces deux structures vont former des zones à forte densité, empêchant toute analyse.

Finalement, le dernier axe de difficulté de ce problème, est la caractérisation des **éléments transposables**, éléments à l'origine de l'explosion combinatoire des bulles. A proprement parler, les éléments transposables sont des séquences d'ADN capables de se déplacer dans le génome. Cependant, au fil du

6. Le polymorphisme est le fait qu'un même gène puisse avoir différents formes, appelées allèles. Des exemples de polymorphismes chez l'Homme sont les groupes sanguins (3 allèles : A, B, O) ou la couleur des yeux.

temps, le génome va accumuler des mutations et en particulier, les séquences issues d'un même élément transposable vont commencer à diverger. On distingue alors les éléments transposables en trois types : les **anciens**, qui ont eu le temps d'accumuler des mutations de voir ses éléments complètement diverger de manière à donner des séquences clairement distinctes, les **récents** qui possèdent des versions exactes ou quasiment exactes, et les **intermédiaires** dont les nombreuses copies ont à peine commencé à diverger.

Les éléments transposables anciens ne posent plus de soucis car leur forte divergence permet de les distinguer facilement entre eux dans le graphe de De Bruijn, et les éléments transposables récents ne posent pas de soucis non plus car correspondent aux mêmes sommets dans le GDB. Le souci se situe alors au niveau des éléments transposables intermédiaires qui possèdent de nombreuses copies et dont les k-mers diffèrent mais sont partagés entre les différents éléments. Cela se traduit dans le graphe de De Bruijn par des centaines voire des milliers de sommets distincts et de surcroît distants.

1.6 Note sur l'état de l'art et stratégies préexistantes

Tout d'abord, un bon point de départ pour faire l'état de l'art des algorithmes pré-existants est la thèse de Camille Sessegolo (9) qui compare les différents méthodes d'analyse de l'épissage génétique à partir de données RNA-Seq. Alors dans sa thèse, Camille Sessegolo soulève l'un des principaux problèmes des méthodes basées sur l'alignement des lectures sur un génome de référence qui est que dès qu'une lecture s'aligne à plusieurs endroits différents, le programme ne peut savoir quelle est l'unique localisation de la lecture et il va la positionner à plusieurs endroits différents, créant de fausses annotations des lectures, suggérant qu'il ne s'agit pas nécessairement la meilleure approche. On peut néanmoins citer le logiciel *SOAP2* (10) qui utilise une méthode se basant sur la recherche des graines dans les lectures puis en les étendant de proche en proche aux autres lectures et qui permet d'obtenir de résultats satisfaisant.
Camille Sessegolo présente également d'autres logiciels qui se basent également sur des graphes de De Bruijn dont les deux principaux sont *Oases* (11) et *Trinity* (12) qui proposent un assemblage globale (reconstruisent les transcrits complets, contrairement à *KisSplice* qui est un assembleur local (à l'échelle de l'exon), même s'ils utilisent la même structure de graphe de De Bruijn.
Un autre logiciel permet également l'analyse de l'épissage alternatif sans références, le logiciel *AbySS* (13) se basant également sur des graphes de De Bruijn avec une approche de parallélisation.
Enfin, d'autres programmes ont été développés pour répondre au problème dans un cadre beaucoup plus restreint. C'est le cas de *GeneChip Human Exon 1.0 ST* (14) et de *ExonPointer* (15) qui fonctionnent dans le contexte de cancers chez l'Homme.
Ainsi, il existe de nombreux programmes ayant tous des approches différentes et des cadres d'utilisation spécifiques. Néanmoins, les méthodes se basant sur les graphes de De Bruijn sont des méthodes qui ont fait leur preuve aussi bien que *KisSplice*, motivant de continuer vers une amélioration de ce programme et non de recommencer de zéro.

D'autre part, la piste explorée lors ce stage n'était pas la seule possible. D'autres méthodes auraient pu être mises en place, telles que utiliser des algorithmes de détection d'éléments transposables sans séquence de référence (tels que *TIF* (16) et *Tedna* (17)) ou bien en ré-implémentant l'énumération des bulles en évitant les zones de forte densité ou en changeant le paradigme du graphe de De Bruijn, stratégies alternatives s'éloignant des outils existants. Alors en pratique, un tel travail à partir de nouvelles implémentations est généralement un travail de thèse, demandant un plus grand approfondissement du problème.

Ainsi, le choix de travailler directement entre deux étapes de la pipeline du programme *KisSplice* et plus précisément sur une simplification du graphe de De Bruijn était le contexte naturel pour développer de nouvelles méthodes pertinentes et pour obtenir des résultats encourageants durant ce stage.

2 Résolution du problème

2.1 Méthodologie

Afin de résoudre le problème que pose la présence d'éléments transposables dans les génomes, je me suis appuyé sur une méthodologie bien précise. Dans un premier temps, il a fallu former des hypothèses sur ce que l'on cherchait, à quelles structures les éléments transposables allaient correspondre dans le graphe de De Bruijn puis trouver des contraintes et des exigences à satisfaire. Cette étape capitale m'a permis par la suite de formuler des modélisations informatiques permettant de répondre à tous ces points prédéfinis. Les modélisations n'étant pas triviales, j'ai aussi dû vérifier que mes modèles étaient cohérents et cela a consisté à la plus grosse partie de mon stage.

Ensuite, une fois une modélisation pertinente du problème obtenue, j'ai pu passer à l'étape de l'implémentation informatique de mon modèle qui consistait à produire une simplification du graphe de De Bruijn utilisable par le logiciel *KisSplice*. La plus grande contrainte de cette étape a été de produire des algorithmes de complexité temporelle quasi-linéaire en la taille des données (un $O(n \log n)$ avec n le taille des données) afin de pouvoir être utilisable sur de vrais jeux de données.

Alors, l'étape suivante était d'appliquer mon algorithme sur des jeux de données afin de regarder les bulles que l'on obtenait. J'ai du alors faire différents programmes afin d'analyser méticuleusement les résultats pour vérifier la cohérence et la pertinence des résultats.

2.2 Hypothèses et Modélisations

Tout d'abord, ma première hypothèse a été de dire que les éléments transposables et les répétitions sont à l'origine de la complexité combinatoire des bulles et forment des régions identifiables dans le graphe de De Bruijn. Concrètement, un élément transposable d'ancienneté intermédiaire va produire une trop forte combinatoire de bulles pour pouvoir être calculée en pratique. Et donc on souhaite pouvoir définir des régions dans le graphe de De Bruijn de forte complexité qui correspondront à cette explosion combinatoire. Alors, ma première étape a été de modéliser ces éléments transposables en définissant un **poids** pour les sommets. J'ai défini formellement le poids d'un sommet u comme étant le cardinal de la boule de centre u et de rayon 10 (la distance entre deux sommets correspond au nombre de nucléotides d'écart entre les deux séquences), noté $|\mathcal{B}(u, 10)|$. Cette métrique est justifiée par le fait que l'on cherche à identifier des séquences de nucléotides peu divergentes mais dont un grand nombre sont distinctes. Alors, le fait de considérer les voisins au plus distants de 10 nucléotides permet d'avoir une métrique représentative du nombre de séquences possibles au niveau d'un sommet donné, tout en restant rapidement calculable. Dans la suite de ce rapport, le **poids** fera donc référence à cette métrique.

Ensuite, j'ai pu tester et vérifier par plusieurs moyens que cette définition de poids proposait bien une modélisation des régions complexes, dont les éléments transposables. J'ai pu dans un premier temps remarquer que la distribution du poids correspondait à une exponentielle inverse (voir un exemple sur la Figure 4) justifiant le fait que les sommets de poids élevés ne correspondent qu'à une fraction des sommets.
Dans un second temps, j'ai pu regarder la distribution des éléments transposables parmi les sommets d'un poids appartenant à un intervalle donné et j'ai pu observer qu'il y a une plus forte proportion d'éléments transposables annotés parmi les sommets de poids élevés. Cependant, on peut quand-même trouver des nombreux éléments transposables parmi les sommets de faible poids. Cela s'explique par plusieurs raisons ; comme dit précédemment, tous les éléments transposables ne se valent pas, les éléments transposables récents ou anciens n'ont aucune raison de provoquer des zones de forte complexité et peuvent donc être facilement annotés. D'un autre côté, cela est appuyé par le fait que les régions complexes des génomes sont généralement peu étudiées en profondeur et manque beaucoup d'annotations. Alors, on sous-estime le nombre d'éléments transposables dans les régions complexes. Ce dernier point a pu *a posteriori* être vérifié, en mettant en évidence des insertions d'éléments transposables non connus dans les régions complexes du génome.

Enfin, c'est l'étude méthodique du problème biologique qui m'a permis de mettre en évidence le soucis

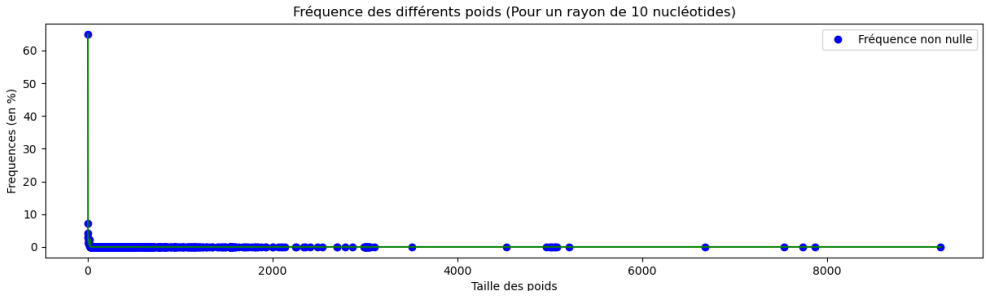

Figure 4 – Distribution des éléments transposables en fonction des poids des sommets. La courbe verte correspond à l'allure générale, tandis que les points bleus correspondent aux fréquences de sommets vraiment trouvées. Cette courbe a été obtenue à partir du jeu de données du moustique. En enlevant les 10% de sommets de poids les plus faibles, on obtient de nouveau une courbe d'allure semblable, mettant en évidence sans caractère exponentielle inverse.

précédemment évoqué des queues poly(A) et des adaptateurs dans le jeu de données. Concrètement, ces deux facteurs correspondaient aux régions les plus denses du graphe de De Bruijn et leur simple élimination a grandement réduit le poids moyen, permettant alors de mieux cibler et cerner le problème.

L'hypothèse suivante a été de formuler qu'il existe un **seuil absolu** tel que si un sommet possède un poids supérieur ou égal à ce seuil, alors il est considéré comme un **sommet de poids élevé** et appartient à une région complexe. Alors, ces sommets forment naturellement des **composantes** définies comme les sous-graphes induits maximaux connexes et dont tous les sommets sont de poids élevé. Puis j'ai pu former la condition suivante sur ces composantes dont la satisfiabilité assure une correcte modélisation : chacune de ces composantes doit capturer entièrement un certain ensemble d'éléments transposables ou de famille de répétitions, ensemble qui se doit d'être unique à la composante en question.

Ainsi, pour obtenir ce résultat recherché, j'ai choisi comme seuil absolu 14 qui répond à cette double contrainte. Plus le seuil absolu est élevé, plus les composantes correspondront à des régions précises du GDB mais au risque de découper un élément transposable dans plusieurs composantes. Inversement, plus le seuil absolu est faible, plus les éléments transposables seront contenus dans des uniques composantes mais au risque de n'avoir plus qu'une seule composante contenant tous les éléments transposables.

Finalement, ma dernière hypothèse a été de dire que si un évènement d'épissage alternatif passe dans une composante alors il existe une bulle dont l'un des chemins passe par la composante et dont l'autre non. Alors, on n'a plus besoin des sommets des composantes, on peut les supprimer pour les remplacer seulement par les arêtes correspondant aux chemins traversant la composante. Cela permet de répondre aux trois contraintes du problème : on ne doit pas perdre les bulles qui ont été déjà trouvées, on ne doit pas complexifier le graphe et on doit casser le nombre exponentiel de bulles au niveau des éléments transposables en proposant un ensemble de bulles exhaustifs des évènements d'épissage alternatif.

Ainsi en donnant comme label aux arêtes rajoutées la séquence du plus court chemin reliant les deux sommets, on obtient une séquence consensus qui permet effectivement de casser la combinatoire des solutions. Néanmoins, cette méthode a pour défaut de quand-même rater les bulles qui commencent (ou terminent) à l'intérieur de cette composante sur des sommets particuliers, ceux qui n'ont pas de chemin sortant ou entrant les reliant à un sommet hors composante. Heureusement, ce problème se règle tout simplement en sortant tous les sommets de degré sortant ou entrant des composantes.

2.3 Algorithme de simplification du Graphe de De Bruijn

Pondération du GDB (Étape 1) Pour obtenir le poids d'un sommet, il suffit de réaliser un **BFS** (un parcours en largeur) du graphe et de compter le nombre de sommets rencontrés, puis dès que l'on descend à plus de 10 nucléotides de profondeur, on arrête le parcours de cette branche. Ainsi, en répétant

cette opération pour chacun des sommets du graphe, on arrive à donner un poids à tous les sommets, poids qui correspond bien à notre définition.

En terme de complexité, la distance maximale de 10 nucléotides à été choisie de façon à ce que le nombre moyen de voisins rencontrés lors du BFS soit négligeable par rapport au nombre de sommets, permettant un calcul global efficace. De plus, par un souci de temps, cette valeur n'a pas pu être retestée et modifiée *a posteriori*, elle a uniquement été choisie de manière à être en accord avec ces contraintes de temps et de façon à apporter du sens aux données.

Création des composantes (Étape 2) Pour générer les différentes composantes j'ai proposé un algorithme quasi-linéaire en le nombre de sommets : on prend tous les sommets de poids élevé (de poids supérieur à 14), on les trie en fonction de leur poids et on les marque comme non visité. Ensuite, il suffit de faire un BFS depuis le sommet de poids le plus élevé, de marquer tous les sommets croisés comme visités et de les enregistrer comme étant une composante. Ensuite, il suffit de recommencer jusqu'à avoir visité tous les sommets et on aura alors bien formé toutes les composantes. Il est à noter que je ne conserve pas les composantes ne contenant qu'un seul sommet car non significatifs.

Niveau complexité, l'étape limitante est le tri des sommets qui se fait en un $O(n \log n)$ avec n le nombre de sommets du graphe. Par la suite, la construction des composantes se fait linéairement en le nombre de sommets, ce qui nous donne une complexité totale quasi-linéaire en la taille des données. Il est à noter que l'algorithme de Tarjan pour la recherche de composantes propose exactement le même résultat mais avec une complexité linéaire en le nombre de sommets et d'arêtes dans le graphe, ce qui permettrait d'obtenir un algorithme linéaire en la taille des données. Cependant dans les faits, cet algorithme est déjà suffisant pour calculer de manière très efficace les composantes et est bien plus facilement implémentable et vérifiable que l'algorithme de Tarjan.

Simplification des composantes (Étape 3) L'algorithme consiste tout simplement, pour chacune des composantes, à rassembler les voisins directs de la composante puis à chercher les chemins reliant deux à deux ces sommets, en ne gardant en mémoire que les plus courts chemins. En terme de complexité, je cherche les plus courts chemins en faisant un BFS à partir de chacun des sommets de la composante, et cela est suffisant pour avoir une complexité quasi-linéaire en la taille des données et par composante. En effet, grâce au critère sur nos composantes la taille n_i des c composantes obtenues ne dépasse jamais la racine carrée du nombre de sommet du graphe noté n et effectuer un BFS de tous les sommets de la composante se fait en visitant n_i^2 sommets. Or, modifier le plus court chemin a une complexité en $O(\log n_i)$, d'où une complexité totale en $O(\sum n_i^2 \log n_i) \leq O(c \times n \log n)$. En pratique, le nombre de composantes n'est pas un problème lors de l'exécution de ce programme car la plupart des composantes ont une taille assez faible (< 1000 sommets) et l'exécution est principalement limitée par les quelques très grosses composantes.

De plus, ajouter les nouvelles arêtes se fait en créant un nouveau sommet correspondant à la séquence du chemin minimal et en l'intercalant entre les deux sommets en utilisant les arêtes appropriées afin de conserver le sens du chemin emprunté. Finalement, on peut ajouter les sommets et les arêtes hors composantes, ce qui se fait linéairement en la taille des données.

Enfin, la Figure 5 récapitule la simplification du graphe de De Bruijn et un exemple illustré et détaillé du fonctionnement de cet algorithme est également disponible en Annexe *D. Exemple d'application de l'algorithme de simplification du GDB sur un mini jeu de donnée.*

2.4 Tests et examen des résultats

Le test et l'examination minutieuse des modèles formés et des résultats trouvés ont été une part importante de mon stage. Pour chacun des aspects à vérifier, j'ai dû produire différents algorithmes pour m'assurer si j'obtenais (ou non) les résultats souhaités.

Le premier point à vérifier, et celui qui m'aura demandé le plus de travail, a été la modélisation des éléments transposables dans le graphe. Pour ce faire, j'ai pu utiliser divers jeux de données de séquençage d'ARN auxquels j'ai couplé les génomes de références des espèces étudiées ainsi qu'une annotation de

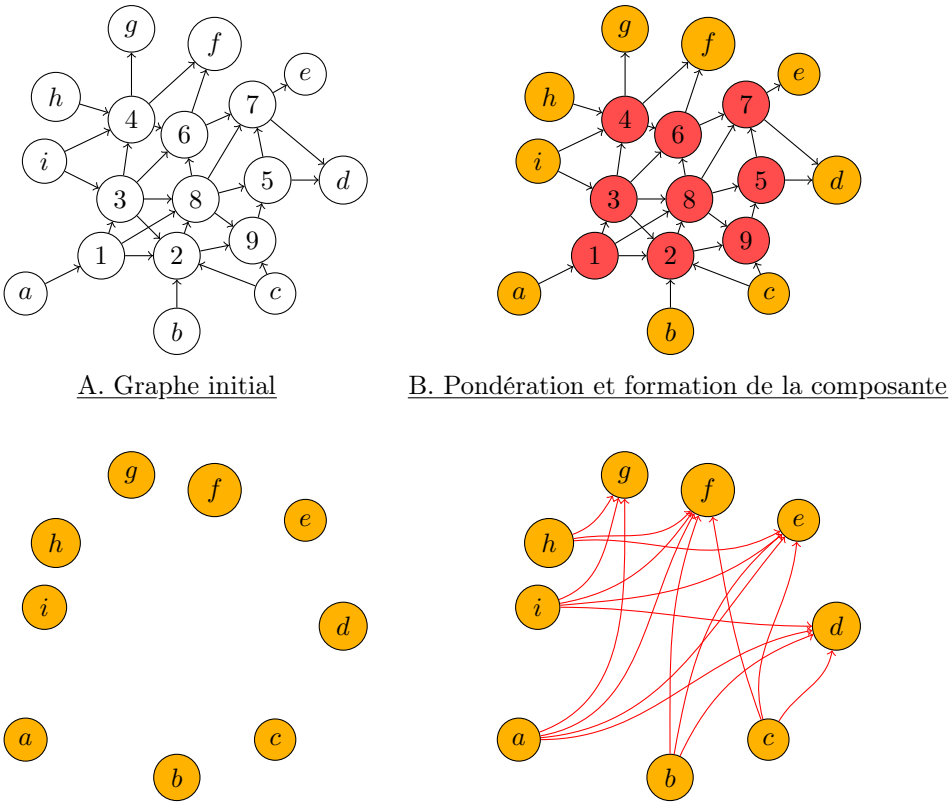

Figure 5 – Schéma de récapitulation de la simplification des composantes du graphe de De Bruijn. Par un soucis de lisibilité, je n'ai dessiné qu'une seule arête par pair de sommets. Les sommets de la composantes sont coloriés en rouge, tandis que les sommets qui sont les voisins directs de la composantes, sont dessinés en orange. Les arêtes consensus sont dessinées en rouge. Dans l'algorithme, ces arêtes labellisées sont construites en rajoutant un sommet qui sera le support de la séquence du chemin choisi et qui fait le lien entre les deux sommets à relier en lui rajoutant deux paires d'arêtes orientées. Enfin sur cet exemple, on n'a considéré qu'une seule composante, au centre.

leurs éléments transposables. En pratique, cela m'a permis, une fois ma pondération obtenue, et mes composantes formées, d'analyser si cela modélise bien les éléments transposables recherchés. Un exemple d'une telle analyse est présentée dans la troisième section. C'est donc par une validation biologique à chaque étape de mes modélisations que j'ai pu continuer à développer les suivantes. De plus, afin d'éviter le biais de produire un algorithme spécifique à un seul jeu de données, j'ai pu utiliser différents jeux de données correspondants à différentes réalités biologiques pour appuyer mes résultats.

Ensuite, j'ai aussi l'occasion de vérifier un point assez important : le fait que mes composantes capturent bien la complexité calculatoire du graphe. Pour ce faire, j'ai tout simplement généré les graphes sans les composantes puis j'ai rentré les graphes dans le logiciel *KisSplice* en essayant d'augmenter le plus possible le paramètre b qui est actuellement le facteur limitant à ce programme. Sans surprise, l'énumération des bulles sur de tels graphes se fait sans problème et même avec des valeurs de b dépassant les 40 ; valeur bien suffisante pour correspondre à la réalité biologique (et bien sûr incalculable sur l'ensemble du graphe). Ainsi, cela m'a permis de mettre en évidence que la complexité du problème se situe effectivement bien dans ces régions complexes, que l'on a identifié en composantes. Alors, le coeur du problème se situe maintenant dans l'identification des bulles des composantes ou à cheval entre les composantes et l'extérieur ; ce qui m'a permis de passer à l'algorithme concret pour construire le graphe recherché, dont les résultats ont aussi du être analysés.

Avant de passer aux tests sur de vrais jeux de données, j'ai d'abord testé mon algorithme sur un exemple minimal que j'ai pu enrichir au fur et à mesure des essais concluants. De cette manière, il a été aisé de vérifier que cet algorithme fonctionnait ce qui m'a permis de le lancer sur plusieurs jeux de données, pour

ensuite récupérer une énumération des bulles présentes via *KisSplice*, bulles qu'il a fallu analyser pour vérifier plusieurs points : quelles sont les nouvelles bulles par rapport à l'utilisation de *KisSplice* sur le graphe d'origine ? Est-ce que ces bulles correspondent à de nouveaux évènements d'épissage alternatif ? Est-ce que les éléments transposables sont bien capturés par des composantes ?

Alors, pour répondre à ces interrogations, j'ai du définir un critère simple pour caractériser un unique évènement d'épissage alternatif. Il s'agit de considérer le **chemin du bas** des bulles, chemin défini comme étant le plus court des deux chemins de la bulle (et inversement, on définit le **chemin du haut** par le plus long chemin des deux) et de l'associer à l'évènement d'épissage alternatif. De cette manière, ne considérer que les chemins du bas permet de ne considérer que les séquences qui excluent l'exon et on évite ainsi tous les séquences possédant du polymorphisme ou des variations dans l'exon en question. Cela permet alors d'avoir une idée de quels évènements sont ratés mais aussi, lesquels sont trouvés en plus.

Ensuite, j'ai pu aussi réaliser un programme qui analyse les bulles en fonction des composantes qu'elles traversent. En effet, comme chaque composante arrive à capturer un certain type de séquence qui crée de la complexité dans le graphe, cela permet de facilement identifier les types de bulles. On peut les décomposer en quatre catégories : les bulles d'épissage alternatif, de répétitions, d'éléments transposables et d'insertion/délétion. Et ce que j'ai pu alors vérifier est que toutes les bulles traversent une composante donnée avaient le même type. Ainsi, ce programme d'analyse est aussi utile lors de mes phases de recherche et d'expérimentation, que lorsque je souhaite effectivement analyser les bulles en tant qu'épissage alternatif. Un exemple sera détaillé dans la troisième section.

Finalement, j'ai également pu utiliser deux autres programmes, l'un de l'équipe nommé *KisSplice2RefGenome* permettant d'obtenir les évènements d'épissage alternatif et un autre, *STAR*(18) permettant d'aligner les bulles sur un génome de référence. Le premier programme m'a permis de voir quelles bulles étaient directement reconnues et lesquelles demandaient une analyse plus poussée, que j'ai réalisée efficacement à l'aide de *STAR* qui associe un certain code particulier à chaque type de bulles. Cependant, il est à noter que comme ces deux algorithmes se basent sur des génomes de référence, génomes qui dans mon cas ne sont pas complets, et alors, toutes les bulles ne sont pas retrouvées via ces programmes.

2.5 Des composantes pour contrecarrer la limitation du branchement

Pour résumer ce que l'on a fait, on a simplifié les régions complexes en transformant tous les chemins passant par de nombreux sommets branchants en simples arêtes annotées dans le but que le programme *KisSplice* puisse trouver des évènements d'épissage alternatif dans ces régions sans être limité par le paramètre b ; le nombre maximal de sommets branchants. Et en effet, cela est un point important à relever car l'idée intuitive aurait été de faire des améliorations dans le but de pouvoir augmenter, voire se débarrasser du paramètre b puisqu'il s'agit du paramètre qui affecte le plus le temps de calcul du logiciel *KisSplice*. Cependant, il y a plusieurs raisons pour lesquelles se débarrasser du paramètre b n'est pas souhaitable.

Un premier argument est qu'à partir d'une certaine valeur, augmenter le paramètre b ne permet plus d'obtenir de nouveaux évènements d'épissage alternatif. Ce point là a pu être vérifié expérimentalement en essayant de rechercher les bulles dans les régions hors des composantes, régions qui correspondent à un sous-graphe où il est très aisé de chercher des bulles grâce à notre modélisation des régions complexes. De la sorte, rechercher des bulles dans ce sous-graphe permet d'obtenir un ensemble exhaustif et suffisant de bulles à partir de $b = 15$, alors qu'en pratique, on peut lancer une recherche de bulles avec $b = 40$, ce qui donne des millions de bulles supplémentaires mais qui correspondent presque toutes à des évènements d'épissage alternatif déjà trouvés. Une telle bulle est illustrée en Figure 9.

De plus, comme un seul évènement d'épissage alternatif peut correspondre à des centaines voire des milliers de bulles différentes dans le graphe (à cause de mutations, répétitions ou du polymorphisme), il n'est pas nécessaire et pas souhaitable de conserver toutes les bulles, seulement un ensemble exhaustif de l'épissage alternatif des gènes. Et même au contraire, cela signifie aussi qu'un évènement d'épissage alternatif est représenté par une dizaine de bulles en moyenne pour $b = 15$ mais par un millier de bulles en moyenne pour $b = 40$, valeur qui n'est plus du tout envisageable pour que de tels résultats puissent être exploités par d'autres chercheurs. Ainsi, le fait de vouloir un ensemble de bulles exhaustif est le second argument pour garder une valeur fixée de b.

Enfin, mon approche de simplification de graphe est complètement compatible avec le fait de vouloir garder un paramètre b fixé puisque que ma méthode permet la suppression des sommets branchants afin de remplacer les multitudes de chemins les parcourant par des uniques arêtes consensus. Cela a bien agi sur les deux points précédents : une même bulle qui avait trop de sommets branchants pour être trouvée, l'est désormais puisque ses nombreux sommets branchants ont été simplifiés en une arête et pour un évènement d'épissage, et l'ensemble des chemins auparavant possibles et qui faisaient exploser le nombre de bulles correspondantes, a été réduit à un seul chemin consensus.

3 Application, analyse et améliorations

3.1 Quatre jeux de données

Afin d'éviter de faire de la sur-interprétation d'un jeu de données, nous avons choisi quatre jeux différents :

- Le moustique, et plus précisément l'espèce *Aedes aegypti*, a été choisi car ses éléments transposables sont également étudiés par une autre équipe du laboratoire. J'ai ainsi pu récupérer une bonne annotation de ses éléments transposables, ce qui m'a permis de tester mes modélisations de ces éléments sur une espèce non modèle. De plus, ce jeu de données comporte les séquences d' individus résistant à certains insecticides et celles d'individu de contrôle. S'agissant du cas que j'ai le plus étudié, une analyse plus profonde sera développée dans la prochaine partie.
- Le chien, de nom scientifique *Canis lupus familiaris* est également une espèce non modèle et a été choisi car est le sujet d'étude d'une équipe partenaire mais aussi car ces derniers disposent aussi de données de séquençages d'une autre génération, dont l'étude croisée aurait pu mener à une autre orientation de mon stage. Le jeu de données est composé des séquences ARN d'un chien atteint de cancer ainsi que d'un individu de contrôle. Ce jeu de données m'aura été très utile car le chien possède énormément d'éléments transposables, et j'ai pu avoir un cas limite en terme de quantité d'éléments transposables. En effet, certains éléments transposables ont été dupliqués des milliers voire des centaines de milliers de fois. Néanmoins, la majorité de ces éléments transposables sont anciens et donc cela m'a permis un exemple bien différent des autres cas, et m'a permis de me rendre compte de différents biais comme la présence de queue poly(A) et d'adaptateurs qui polluaient les données ainsi du fait que les éléments transposables peuvent être très anciens (et très étendus).
- Ensuite, j'ai également eu à disposition de séquences du *rhodnius prolixus*, un insecte responsable de la maladie de Chagas. Ces éléments transposables étant en cours d'étude, il s'agit du jeu de données initial au début de mon stage, mais dont l'étude n'a pas dépassé le stade des éléments transposables car semblable au jeu de données du moustique.
- Enfin, le dernier jeu de données à disposition était celui de l'Homme (*Homo sapiens*) qui est un bon jeu de données de contrôle au niveau des éléments transposables, mais qui reste une espèce modèle, permettant au passage d'avoir des éléments transposables bien annotés. Cependant, l'étude du chien et moustique étant plus pertinente car étant des espèces non modèles, j'ai préféré avoir des analyses plus poussées pour ces deux espèces que pour l'Homme, dont l'analyse complète a été principalement contrainte par la durée limitée de mon stage.

Ainsi, j'ai principalement analysé et comparé les jeux de données du chien et du moustique et j'ai pu en tirer plusieurs caractéristiques :

- Poids moyen : Le poids moyen des sommets du graphe de De Bruijn est 2 fois plus élevé chez le chien que chez le moustique. Cela s'explique pour deux raisons : le chien ayant beaucoup plus d'éléments transposables qui s'interfèrent les uns les autres, cela crée des régions plus complexes que chez le moustique. D'autre part, on étudie des séquences issues de cellules cancéreuses chez le chien qui induisent alors de nombreuses mutations, complexifiant d'avantage le graphe.

- Nombre de composantes : Étonnamment, la distribution et la valeur de poids moyen n'influent pas sur le nombre de composantes dans les graphes respectifs. Le nombre de composantes est principalement lié au seuil absolu choisi. Une différence à noter est que chez le chien un élément transposable, le *SINEC* crée une très grosse composante et il a été intéressant de la redécouper en plusieurs composantes afin d'obtenir une plus profonde analyse en relançant tout simplement les algorithmes.
- Éléments transposables : Tous deux possèdent beaucoup d'éléments transposables (comparé à l'Homme) mais ils ne sont pas de la même nature. Le chien possède en majorité des éléments transposables anciens et très nombreux tandis que le moustique possède plutôt des éléments transposables récents. Cette différence se manifeste en deux points : pour le chien, certains éléments transposables sont présents à des centaines de milliers de positions différentes dans le génome alors que l'on peut trouver de nombreuses nouvelles insertions d'élément transposable, non annotées chez le moustique. Une représentation du nombres d'éléments transposables en fonction des composantes où on peut les trouver est illustrée en Figure 6 dont nous allons prendre le temps d'analyser.

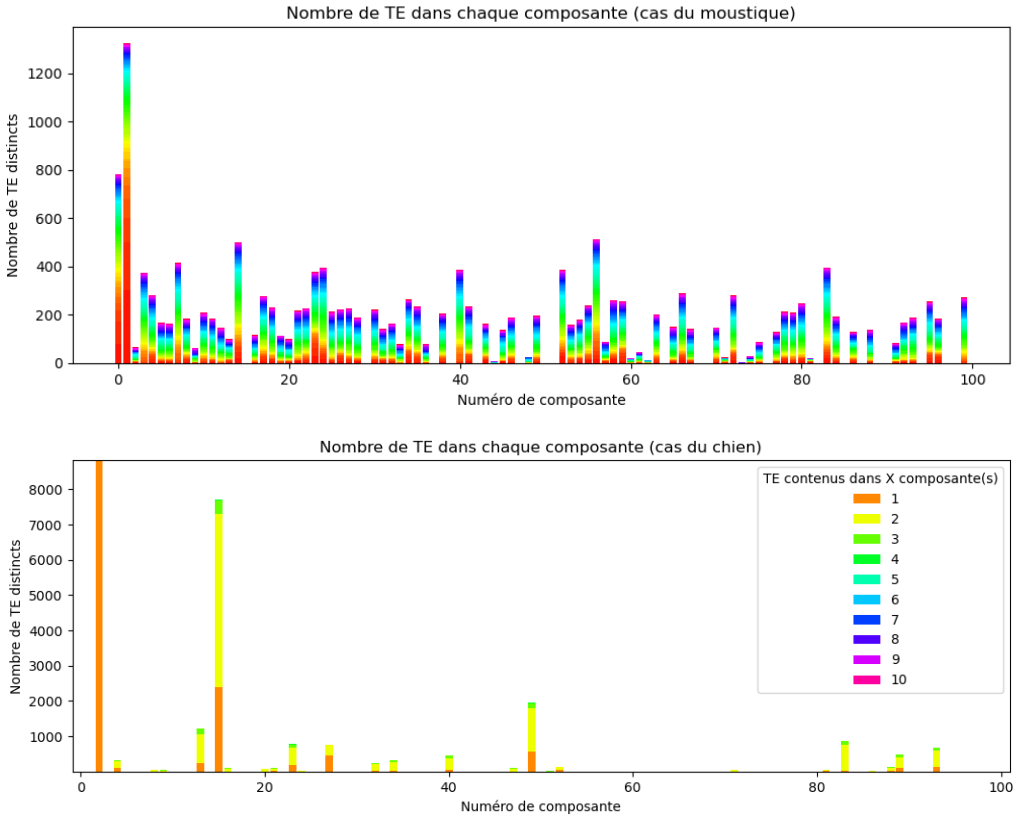

Figure 6 – Nombres d'éléments transposables trouvés dans chacune des 100 plus grandes composantes. En haut, le cas du moustique et en bas celui du chien. De plus, chaque élément transposable se voit attribuée une couleur correspondant au nombre de composantes différentes auxquelles il appartient. Pour le moustique, les couleurs vont du rouge (appartient à une unique composante, ce que l'on souhaite modéliser) au violet (appartient à 72 exemplaires) et pour le chien, cela va du orange (une unique composante) au violet (appartient à 10 composantes).

Tout d'abord, on peut observer sur la courbe du moustique qu'une forte proportion des composantes correspondent à des éléments transposables totalement capturés. Cependant, elles contiennent quand-même des éléments transposables qui sont communs à plusieurs composantes différents, chose que l'on ne souhaitait pas. Cela s'explique en regardant plus en détail les composantes et les éléments transposables. Ces derniers peuvent être partagés en deux types : les **SINE**[7] et les **LINE**[8] qui se différencient principalement par leur mode de fonctionnement. De manière simplifié, un LINE va être capable de déplacer et se dupliquer de lui-même alors que le SINE lui va avoir besoin d'utiliser un LINE pour faire tout cela

7. SINE signifie en anglais short interspersed repetitive elements
8. Inversement, LINE signifie en anglais long interspersed repetitive elements

(et dans le génome, on retrouve une proximité physique entre ses deux types d'éléments). Alors, on peut avoir un unique LINE qui se retrouve dans les composantes de SINE bien différents, et inversement. Ainsi, même s'il n'est possible de capturer chaque élément transposable dans une composante distincte, il est en réalité possible que chaque composante contienne des éléments transposables propres à elles-même, les caractérisant. Il s'agit là bien de la propriété que l'on a utilisé dans la modélisation des composantes.

Enfin, l'exemple du chien est assez différent de celui du moustique. On peut remarquer que beaucoup composantes ne correspondent à aucun élément transposable car elles correspondent également à des répétitions, à des éléments transposables non annotés ou non retrouvés, ou à des régions riches en polymorphisme. De plus, les éléments transposables sont bien plus nombreux que chez le moustique, et sont bien bien capturé par les composantes. Cela induit par le fait que les éléments transposables du chien sont bien plus anciens et mieux annotés. D'autre part, il existe en réalité des centaines de milliers d'éléments transposables ayant divergés dont la quasi-totalité est contenu dans la composante numéro 2. En effet, comme le seuil absolu de 14 permet de capturer l'ensemble d'un élément transposable, il va également englober les centaines de milliers de variations en une composante. Alors, la solution a été d'utiliser un autre seuil absolu pour cette composante qui est de 22 et qui permet de bien séparer les éléments transposables anciens.

3.2 Développement du cas du moustique

Le jeu de données sur le moustique étant mon jeu de référence, j'ai pu analyser mes résultats obtenus et les comparer avec l'utilisation classique du logiciel *KisSplice*. J'ai préféré présenter ce jeu de données plutôt que celui du chien car il m'a permis de mettre en évidence de nouveaux évènements d'épissage alternatif, non trouvés par *KisSplice* auparavant ainsi que de nombreuses nouvelles insertions d'éléments transposables dans le génome du moustique.

Concrètement, j'ai pu trouvé des bulles dont seulement l'un des chemins passe par une composante et qui ont pu être trouvé uniquement en nombre raisonnable, uniquement grâce à ma simplification du graphe de De Bruijn. Une illustration d'une telle bulle est sur la Figure 7.

De manière plus synthétique, j'ai ainsi obtenu les résultats suivants :

- 42527 composantes formées.
- 5647 nouvelles bulles trouvées sur un total de 24402 bulles trouvées. On considère qu'une nouvelle bulle est trouvée lorsque cette dernière a son chemin du haut qui passe par une composante et tandis que son chemin de bas n'y passe pas et n'avait jamais été trouvé auparavant par *KisSplice*.
- 181 composantes donnent ces nouvelles bulles. Parmi ces composantes, on a :
  - 53 composantes qui sont directement associées à des évènements d'épissage alternatif par le logiciel *KisSplice2RefGenome*. Un exemple est présenté en Figure 8
  - 26 composantes sont associées avec des insertions ou des délétions, qui se comportent en terme de bulles exactement comme des évènements d'épissage alternatif pour une taille bien précise de l'insertion/délétion.
  - 24 composantes sont associées à des zones de répétitions inexactes qui donnent des chemins alternatifs formant alors des bulles.
  - 16 composantes sont associées à des insertions d'éléments transposables, point sur lequel nous reviendrons dans la partie suivante.
  - 62 composantes qui ne sont pas automatiquement associées par les logiciels pré-existants à l'une des quatre précédentes catégories. Il faut alors les analyser à la main afin de les réattribuer à leur catégorie correspondante.

D'autre part, en poussant l'analyse des bulles très branchantes situées en-dehors des composantes, j'ai pu mettre en évidence une dizaine d'évènements d'épissage alternatif dans des régions à très fort polymorphisme, sur un total de plus de 12000 évènements d'épissage alternatif hors des composantes.

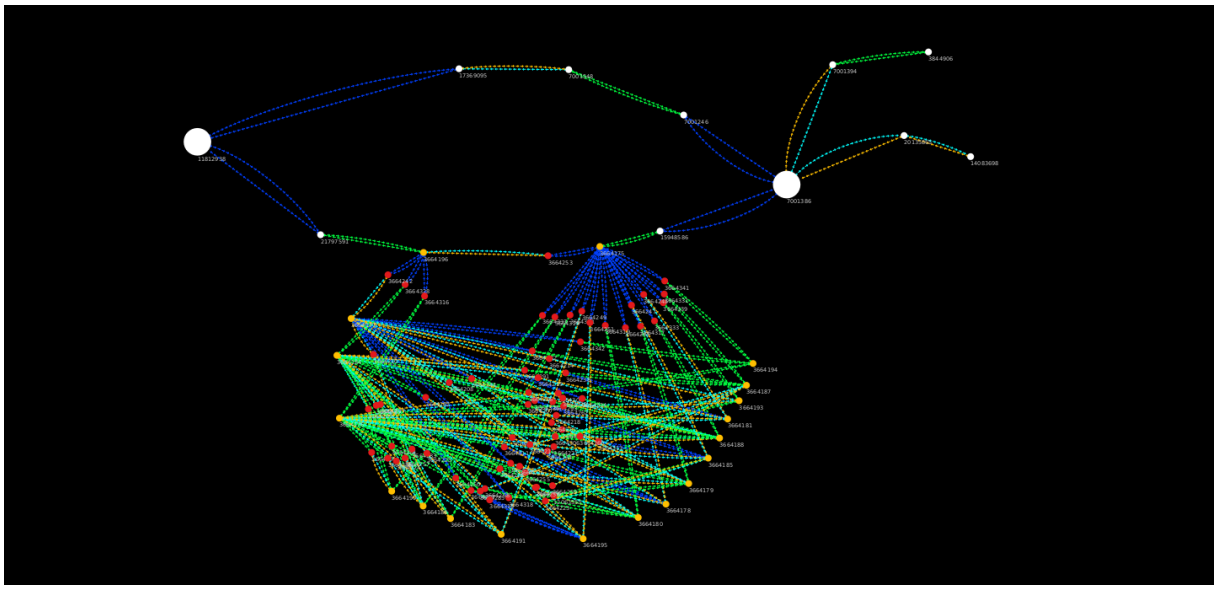

Figure 7 – Exemple d'une bulle après simplification des composantes correspondant à de l'épissage alternatif et traversant un élément transposable. Les couleurs des arêtes sur cette figure n'ont pas vraiment d'importance, il faut les lire allant de gauche à droite. Les deux gros sommets blancs correspondant au début et à la fin de la bulle qui est ainsi bien formée par deux chemins disjoints (hormis au départ et à la fin de la bulle). Sont affichés uniquement les sommets à au plus cinq arêtes de distance de ces gros sommets blancs. Les sommets rouges sont les labels des arêtes que l'on a rajouté pour remplacer une composante, et en orange sont les voisins directs de l'ancienne composante. Ensuite, on peut remarquer les sommets du chemin du bas sont branchants et donnent sur la composante. Après la composante, les arêtes continuent de descendre jusqu'à finalement se rejoindre en bas. De plus, on retrouve bien la composante sous forme d'arêtes labellisées, qui sont modélisées en reliant deux sommets en y intercalant un sommet consensus et deux paires d'arêtes entre. Ces nouvelles arêtes labellisées correspondaient avant à des multitudes de chemins différents, la composante contenant avant plusieurs centaines de sommets. Ainsi, on a pu capturer cette bulle en traversant moins de 5 sommets branchants grâce à ces arrêtes labellisées qui contraignent la combinatoire des chemins au sein de la composante. Cet exemple illustre également la raison pour laquelle on ne souhaite par considérer des bulles dont les deux chemins passent par la même composante ; on ne ferait que toujours regarder le même élément mais sous des répétitions légèrement différentes.

L'évènement en question, illustré en Figure 9, est très certainement une rétention d'intron, et est associée en sortie du logiciel *KisSplice* à plus de 2 millions de bulles différentes !

Cela est dû au très fort polymorphisme de cette région ; on ne dénombre pas moins de 200 différences avec le génome de référence, et quasiment autant au sein des lectures du jeu de données. Or pour arriver à 2 millions de chemins différents, il ne suffit que d'une vingtaine de branchements différents. Alors avoir une vingtaine de polymorphisme espacés d'au moins un k-mer est suffisant pour obtenir une telle combinatoire. De plus, une question intéressante à relever est pourquoi cette région n'est pas capturée par l'une de nos composantes ? La raison est toute simple : pour former une composante, il faut avoir un poids élevé, qui nécessite d'avoir au moins 14 voisins différents à moins de 10 nucléotides de distance donc autant de branchements différents en moins de 10 nucléotides, mais dans cet exemple, on est plutôt à une différence tous les 100 nucléotides.

3.3 Découverte d'éléments transposables

L'une des surprises lors de ce stage est qu'en modélisant les éléments transposables par une unique séquence consensus, il a été possible de mettre en évidence l'insertion de nouveaux éléments transposables dans le génome du moustique et qui n'avaient jamais été annotés jusqu'à présent. En effet, l'une des particularités du moustique est que les éléments transposables sont toujours très actifs dans les populations étudiées, ce qui fait que certains individus ont plus de copies que d'autres. Alors, comme on étudie un individu résistant à certaines molécules en même temps qu'un individu de contrôle, l'insertion d'un nouvel élément transposable dans le génome de l'individu résistant va former une bulle avec la séquence

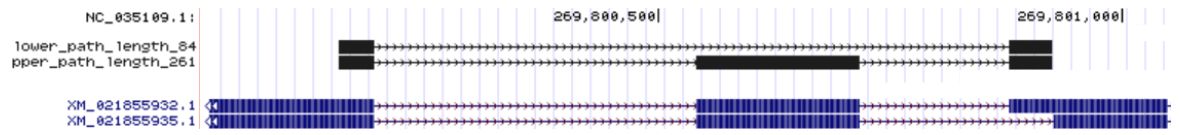

Figure 8 – Exemple d'une bulle correspondant à un évènement d'épissage alternatif extraite à l'aide de *Blat* (19). Les blocs noirs correspondent à l'alignement des chemins par rapport à un génome de référence, les flèches noires correspondent à un passage sauté de la référence tandis que les traits rouges correspondent à une différence. Les lignes bleues `XM_021855932.1` et `XM_021855935.1` correspondent à des gènes annotés de la référence et dont on connait déjà les exons et introns.
Cet évènement se situe sur le chromosome `NC_035109.1`. On peut remarquer que le premier chemin, `lower_path_length_84` est composé de deux blocs en extrémité correspondant à la fin d'un exon à gauche et au début d'un nouvel exon à droite. Le second chemin, `upper_path_length_261` s'aligne en trois blocs distincts, avec celui du milieu qui correspond à un exon intermédiaire.

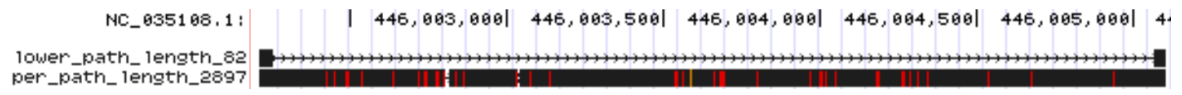

Figure 9 – Exemple d'une bulle hors composante, très branchante, extraite à l'aide de *Blat* (19). Il s'agit d'une rétention d'intron au niveau du chromosome `NC_035108.1`. On peut remarquer que le premier chemin, `lower_path_length_82` est composé de deux 41-mers en extrémité tandis que le second chemin, `upper_path_length_2897` mesure 2897 nucléotides et s'aligne à 99.0% avec l'intron au milieu. Non annoté dans le génome de référence, il fait partie des rares bulles que l'on peut trouver avec un branchement élevé.

originelle de l'individu de contrôle. Alors, on observe des motifs bien particuliers pour les chemins de la bulle, schématisé sur la Figure 10.

e tels motifs et en alignant les bulles sur un génome de référence à l'aide du logiciel *STARlong*, il est possible de filtrer les bulles en fonction de si elles suivent ce motif ou non. De cette manière, j'ai pu trouver de nombreuses nouvelles insertions d'éléments transposables. Un exemple concret est illustré sur la Figure 11.

D'autre part, il a également été possible de mettre en évidence un autre type d'insertion d'élément transposable : ceux qui s'insèrent dans les introns des gènes mais qui vont ensuite se comporter comme un exon lors de l'épissage de l'ARN. On dit alors que l'élément transposable a été **exonisé**. Un tel exemple est illustré sur la figure 12.

Ainsi, ces exemples ont pu montrer que les éléments transposables ne sont pas encore bien connus malgré le fait qu'ils soient effectivement présents dans les génome. Cela suggère aussi que le nombre d'éléments transposables est sous-estimé, notamment dans les régions complexes, faisant écho aux points précédemment mentionnés.

Finalement, les modélisations étant mises en place afin de simplifier le plus possible la diversité des éléments transposables sous une seule arête consensus, il serait imaginable de modifier et orienter les stratégies élaborées afin de trouver tous ces nouveaux éléments, et non seulement un élément consensus choisi arbitrairement dans le cadre la résolution d'un autre problème.

3.4 Reproductibilité, Améliorations et Critiques

De manière générale en recherche, et plus particulièrement en bioinformatique, il est capital d'être en mesure de reproduire les résultats obtenus. Pour ce faire, tous mes programmes réalisés sont disponibles sur un Git : `https://github.com/sdarmon/stage-M2/` et tous les autres logiciels utilisés sont en libre accès. De plus, un mini jeu de donnée est disponible pour tester les algorithmes et un fichier `README.md` est disponible afin de décrire comment utiliser les programmes. En effet, le but de ce stage étant de produire

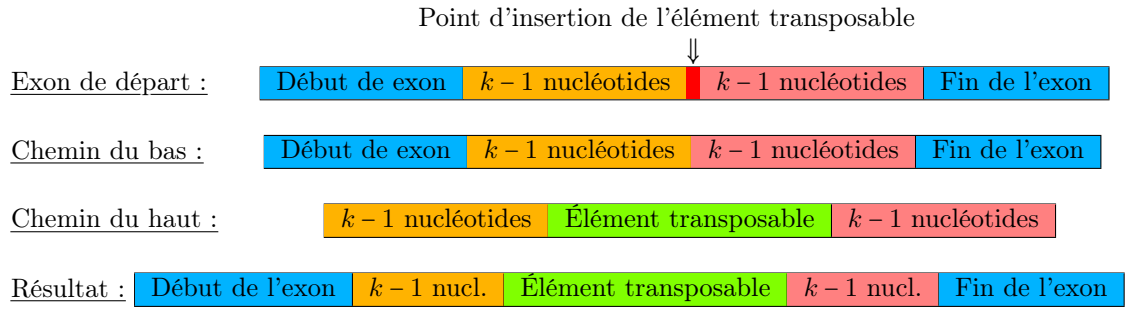

Figure 10 – Exemple de motifs attendus dans une bulle correspondant à l'insertion d'un élément transposable. Quand le chemin du bas s'aligne complètement sur un unique exon mais que le chemin du haut correspond à la séquence d'un élément transposable, encadrée par deux séquences d'au plus $k-1$ nucléotides, cela signifie que dans l'une des lectures, il y a cet élément transposable qui s'est inséré au milieu de l'exon en question. En effet, comme le chemin du bas et le chemin du haut forment une bulle, les séquences de $k-1$ nucléotides sont deux à deux identiques sur les deux chemins et donc une telle structure d'alignement ne peut correspondre qu'à un tel phénomène.

un travail réutilisable et implémentable dans le logiciel *KisSplice*, cette étape est une étape indispensable pour permettre un développement futur de cet algorithme.

En outre afin de faciliter la reproductibilité de mes résultats, j'ai utilisé des scripts *Nextflow* qui une fois correctement initialisés permettent de reproduire tous les résultats en une ligne de commande. Cela sera également très utile aux prochains membres du laboratoire qui aimeraient retester ou améliorer mes algorithmes.

En effet, il reste plusieurs points à encore améliorer. Les différents paramètres mis en place dans cette algorithme (la distance maximale pour la pondération, le seuil absolu pour former les composantes, la valeur du paramètre b) mériteraient de plus profondes analyses et ajustements. De plus, actuellement l'étape 3 (la simplification du graphe) est l'étape limitante de mon algorithme et il serait judicieux de trouver les arêtes à rajouter via un algorithme linéaire en la taille des données, quitte à abandonner l'idée de l'associer à la séquence du plus court chemin (puisque ce qui nous importe c'est seulement d'avoir une séquence consensus). De plus, certaines composantes correspondent à des répétitions dans le génome, alors comme ce ne sont pas de l'épissage alternatif, toutes les bulles que l'on obtient à partir de ces composantes ne nous intéressent pas. Alors, une amélioration toute simple, qui sort du cadre des éléments transposables mais qui permettrait de gagner en efficacité par la suite, serait de vérifier si les composantes correspondent ou non à des répétitions, et si c'est le cas, tout simplement les supprimer. De cette manière, on aura beaucoup moins de répétitions à gérer, et comme détecter des répétitions dans un sous-graphe réduit est déjà quelque chose que l'on sait faire aisément, il s'agit d'une amélioration facilement implémentable.

Une autre piste d'amélioration possible est directement dans le logiciel *KisSplice*. En effet, on ne souhaite par énumérer les bulles dont les deux chemins passent par la même composante. Actuellement, seul un filtrage *a posteriori* est effectué mais cela améliorerait grandement le temps d'exécution de l'énumération des bulles si les composantes (et cette interdiction) étaient prises en compte.

Finalement, malgré ces améliorations à encore développer, cette méthode garde plusieurs points positifs. N'ayant pas besoin de génome de référence pour fonctionner, la simplification des régions complexes a permis d'éviter les biais dû à l'utilisation de références. Ainsi, mon algorithme permet de trouver de nouveaux évènements d'épissage alternatif mais il a aussi mis en évidence la présence de nouvelles insertions d'éléments transposables dans le génome.

Ainsi, en faisant les quelques correctifs suggérés, il deviendrait viable d'implémenter cet algorithme directement dans le logiciel *KisSplice* et ce qui améliorerait de surcroît les connaissances de l'épissage alternatif dans les régions complexes du génome.

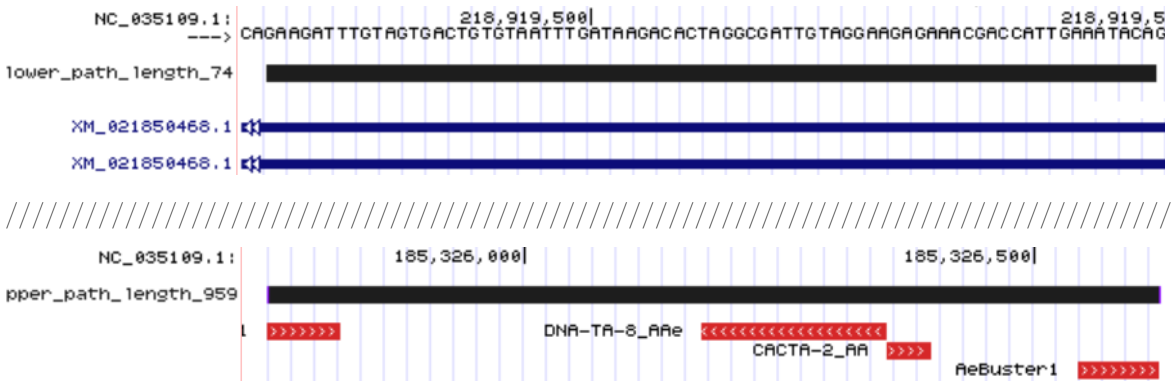

Figure 11 – Exemple d'une bulle correspondant à un élément transposable, extraite à l'aide de *Blat* (19). Cet évènement se situe sur le chromosome `NC_035109.1`. On peut remarquer que le premier chemin, `lower_path_length_74` correspond à un seul exon (s'aligne avec les gènes `XM_021851011.1` et `XM_0218510115.1`). D'autre part, le second chemin, `upper_path_length_959`, s'aligne à pleins d'endroits différents du génome dont en particulier sur le chromosome `NC_035109.1` mais à une position complètement différente du chemin du bas. De plus, on remarque en rouge la présence d'éléments transposables (et répétitions) et en regardant en détail la séquence, on peut remarquer qu'il lui manque $k-1$ nucléotides au début et à la fin de la séquence ce qui correspondent bien au chevauchement des nucléotides situées autour du point d'insertion.

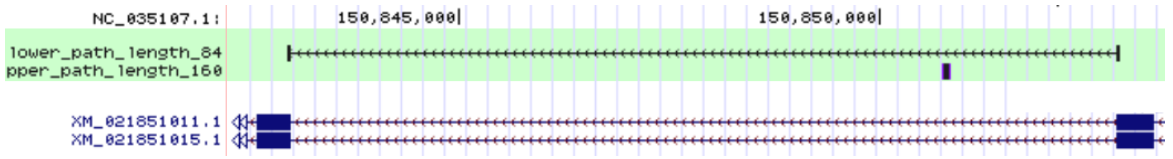

Figure 12 – Exemple d'une bulle correspondant à un élément transposable exonisé. Il s'agit également d'exactement la même bulle que celle de la Figure 7. Cet évènement se situe sur le chromosome `NC_035108.1`. On peut remarquer que le premier chemin, `lower_path_length_84` est composé de deux blocs en extrémité correspondant à la fin d'un exon à gauche et au début d'un nouvel exon à droite. Le second chemin, `upper_path_length_160` s'aligne sur un seul bloc au milieu de l'intron, et en y regardant de plus près, il lui manque $k-1$ nucléotides au début et à la fin de la séquence qui correspondent bien à l'extrémité droite de l'exon à gauche de l'intron et à l'extrémité gauche de l'exon à droite de l'intron. De plus, on peut remarquer la présente de deux gènes annotés `XM_021851011.1` et `XM_021851015.1` qui ne présentent pas cet élément transposable exonisé.

Conclusion

Pendant ces cinq mois de stage j'aurais pu me plonger pleinement dans un sujet de recherche passionnant et je suis heureux que l'on m'ait laissé l'opportunité de me confronter à un problème biologique et de proposer une solution algorithmique concrète. J'espère que cette dernière pourra être utilisée afin d'améliorer le logiciel *KisSplice* et de ce fait, aider les chercheurs biologistes à analyser l'épissage alternatif.

Par ailleurs, mon immersion dans la recherche en bioinformatique, et plus particulièrement dans le développement de méthodes de résolution de problèmes biologiques, m'a permis de confirmer mon attrait pour ces domaines ainsi que de développer de nouvelles compétences qui, je l'espère, me permettront de mener à bien mes prochains projets de recherche en bioinformatique.

Encore perfectible, et possédant plusieurs développements réalisables, cet algorithme aura également permis d'identifier de nouvelles insertions d'éléments transposables inconnues jusqu'à présent, et suggère possiblement une meilleure compréhension des éléments transposables chez les espèces non modèles, voire de nouvelles stratégies pour la recherche d'éléments transposables.

Références

(1) Walter Gilbert. "Why genes in pieces?" In : Nature 271.5645 (1978), p. 501-501.

(2) Wei Jiang et Liang Chen. "Alternative splicing: Human disease and quantitative analysis from high-throughput sequencing". In : Computational and Structural Biotechnology Journal 19 (2021), p. 183-195.

(3) Gustavo AT Sacomoto et al. "KIS SPLICE: de-novo calling alternative splicing events from RNA-seq data". In : BMC bioinformatics. T. 13. 6. Springer. 2012, p. 1-12.

(4) Michael A Quail et al. "A large genome center's improvements to the Illumina sequencing system". In : Nature methods 5.12 (2008), p. 1005-1010.

(5) Scott Hotaling, Joanna L. Kelley et Paul B. Frandsen. "Toward a genome sequence for every animal: Where are we now?" In : Proceedings of the National Academy of Sciences 118.52 (2021), e2109019118. doi : 10.1073/pnas.2109019118. url : https://www.pnas.org/doi/abs/10.1073/pnas.2109019118.

(6) Arang Rhie et al. "Towards complete and error-free genome assemblies of all vertebrate species". In : Nature 592.7856 (2021), p. 737-746.

(7) Mihai Pop. "Genome assembly reborn: recent computational challenges". In : Briefings in bioinformatics 10.4 (2009), p. 354-366.

(8) Rayan Chikhi, Antoine Limasset et Paul Medvedev. "Compacting de Bruijn graphs from sequencing data quickly and in low memory". In : Bioinformatics 32.12 (2016), p. i201-i208.

(9) Camille Sessegolo. "Développement de méthodes bio-informatiques pour l'étude de l'épissage chez les espèces non modèles : épissage complexe et apport des technologies de séquençage de 3eme génération". Theses. Université de Lyon, oct. 2021. url : https://tel.archives-ouvertes.fr/tel-03662745.

(10) Ruiqiang Li et al. "SOAP2: an improved ultrafast tool for short read alignment". In : Bioinformatics 25.15 (2009), p. 1966-1967.

(11) Marcel H Schulz et al. "Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels". In : Bioinformatics 28.8 (2012), p. 1086-1092.

(12) Manfred G Grabherr et al. "Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data". In : Nature biotechnology 29.7 (2011), p. 644.

(13) Jared T Simpson et al. "ABySS: a parallel assembler for short read sequence data". In : Genome research 19.6 (2009), p. 1117-1123.

(14) Paul J Gardina et al. "Alternative splicing and differential gene expression in colon cancer detected by a whole genome exon array". In : BMC genomics 7.1 (2006), p. 1-18.

(15) Fernando J de Miguel et al. "Identification of alternative splicing events regulated by the oncogenic factor SRSF1 in lung cancer". In : Cancer research 74.4 (2014), p. 1105-1115.

(16) Mariko Nakagome et al. "Transposon Insertion Finder (TIF): a novel program for detection of de novo transpositions of transposable elements". In : BMC bioinformatics 15.1 (2014), p. 1-9.

(17) Matthias Zytnicki, Eduard Akhunov et Hadi Quesneville. "Tedna: a transposable element de novo assembler". In : Bioinformatics 30.18 (2014), p. 2656-2658.

(18) Alexander Dobin et al. "STAR: ultrafast universal RNA-seq aligner". In : Bioinformatics 29.1 (2013), p. 15-21.

(19) W James Kent. "BLAT—the BLAST-like alignment tool". In : Genome research 12.4 (2002), p. 656-664.

Annexes

A. Preuve de la croissance exponentielle du nombre de bulles en fonction du paramètre b

Fixons un sommet de départ u et un sommet d'arrivée v ainsi que le paramètre b d'une bulle. Le paramètre b, le branchement, est défini comme étant le nombre maximal de sommets branchants présents dans les chemins de la bulle. De cette manière, l'un des chemins possèdent b sommets branchants et on peut alors considérer toutes les bulles de branchement b allant de u à v.

Or, un sommet en branchant s'il peut créer un branchement entre deux chemins lors que l'on parcourt.

Ainsi, si tous les sommets branchants donnent sur deux autres sommets possibles, le pire des cas donne alors 2^b chemins alternatifs possibles pour l'un des deux chemins formant la bulle, donnant alors au moins 2^b bulles différentes.

Et donc, on peut alors obtenir un nombre exponentielle de bulles en fonction de b et pour des sommets de départ et d'arrivée fixé. En pratique, ce phénomène est également amplifié à cause du fait qu'un sommet branchant peut donner sur bien plus de sommets différents.

B. Exemple de la multiplication du nombre de bulles de deux sous-graphes

Considérons un sous-graphe H_1 contenant n bulles allant du sommet $u_1 \in V(H_1)$ au sommet $v_1 \in V(H_1)$, puis considérons un second sous-graphe H_2 disjoint de H_1 contenant m bulles allant du sommet $u_2 \in V(H_2)$ au sommet $v_2 \in V(H_2)$.

Alors, en définissant le graphe G comme étant l'union des graphes H_1 et H_2 et auquel on a rajouté les sommets u et v ainsi que les arêtes $(u, u_1), (u, u_2), (v_1, v), (v_2, v)$. De cette manière, il existe au moins $n+1$ chemins distincts allant de u à v et en ne passant que par des sommets de $V(H_1)$ (n bulles de u_1 à v_1) et il existe également au moins $m+1$ chemins distincts allant de u à v (m bulles de u_2 à v_2) et en ne passant que par des sommets de $V(H_2)$.

Or, H_1 et H_2 sont disjoints.

D'où le fait que le graphe G possède $(n+1)(m+1)$ bulles du sommet u au sommet v.

C. Preuve de l'impact du polymorphisme sur le nombre de bulles d'un graphe

Considérons un évènement d'épissage commençant au sommet u et finissant au sommet v. Supposant qu'il y a n bulles partant du sommet u et finissant au sommet v avec n un carré parfait, on note $m = \sqrt{n}$.

Le polymorphisme peut faire doubler le nombre de bulles lorsque :

- Les bulles sont formées par les pairs de chemins (C_i, D_j) pour $i \in \{1, ..., m\}$ et $j \in \{1, ..., m\}$ et tels que C_i et D_j soient à support disjoint (i.e. n'ont en commun que leur premier et dernier sommet) pour pour $i \in \{1, ..., m\}$ et $j \in \{1, ..., m\}$.
- Le polymorphisme affecte un somment présent chez tous les C_i pour $i \in \{1, ..., m\}$. On note alors C'_i pour $i \in \{1, ..., m\}$ les chemins possédant la variation du polymorphisme.

Ainsi, on a également que D_j et C'_i sont à support disjoint pour tout pour $i \in \{1, ..., m\}$ et $j \in \{1, ..., m\}$ car on a rajouté un polymorphisme à des chemins qui étaient déjà à support disjoint avec les D_j.

Donc, après ce polymorphisme, il y a $2m \times 2m$ bulles de u à v soit un total de $4n$ bulles. En pratique, c'est ce cas là qui se passe avec certains exons qui vont avoir beaucoup de polymorphisme. On a bien multiplié par quatre le nombre de bulles !

D. Exemple d'application de l'algorithme de simplification du GDB sur un mini jeu de donnée

Afin d'illustrer mon algorithme, je me suis servi d'un jeu de données factice. Il est composé de six lectures :

- `ACTTGCATTACTTTCTAAATGG` : une première séquence.
- `ACTTGCATTAGTTTCTAAATGG` : une séquence possédant une différence d'un nucléotide par rapport la première lecture.
- `ACCTGCTACTTTAGG` : une séquence qui partage les mêmes dix derniers nucléotides que la première lecture.
- `CACGGTACATAGTTTAGG` : une séquence qui partage les mêmes dix derniers nucléotides que la seconde lecture.
- `ACTTGCAAATGG` : un épissage alternatif de la première lecture, où l'exon `TTACTTTCTAA` est épissé.
- `ACTTGCATTACATGCATTACTTTAGGAT` : une séquence possédant la répétition `CATTAC`.

Ainsi, ces quelques lectures me permettent de couvrir plusieurs situations différentes : des répétitions, exactes ou inexactes, du polymorphisme et de l'épissage alternatif. De plus, pour étudier cet exemple avec mes algorithme, j'ai fixé comme paramètre $k = 5$ pour pouvoir générer facilement beaucoup de sommets, et dans toute cette partie, les types d'arêtes seront associés à un code couleur :

- Les arêtes **FF** seront en bleu clair.
- Les arêtes **RR** seront en orange.
- Les arêtes **RF** seront en vert clair.
- Les arêtes **FR** seront en bleu foncé.

On peut alors former le graphe de De Bruijn compacté ce qui nous donne alors le résultat suivant sur la Figure 13 :

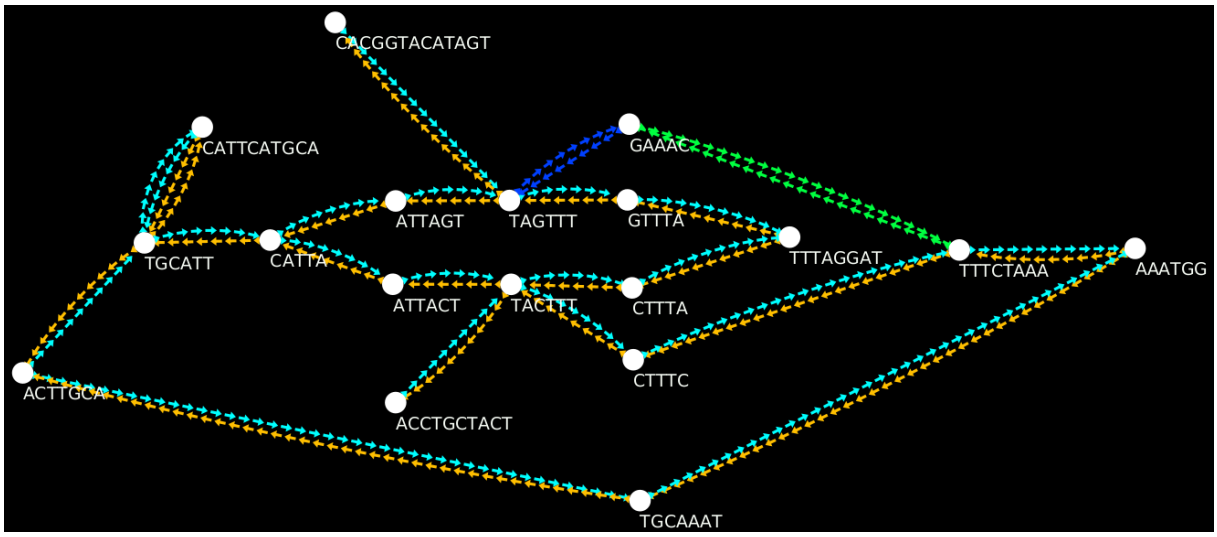

Figure 13 – Exemple de graphe de De Bruijn sur les lectures pré-définies. Seul le sens forward est indiqué sur le graphe. Les lectures se lisent de gauche à droite dans le sens direct (en forward), sauf pour le 5-mer `GTTTA` qui est écrit en reverse complement.

Ensuite, j'ai pu y appliquer mon algorithme de pondération avec là encore un rayon de 3, adapté à ce petit jeu de donnée, et pour former mes composantes j'ai choisi comme seuil absolu 4. Cela permet d'obtenir une composante dont les sommets sont coloriés en rouge sur la Figure 14 :

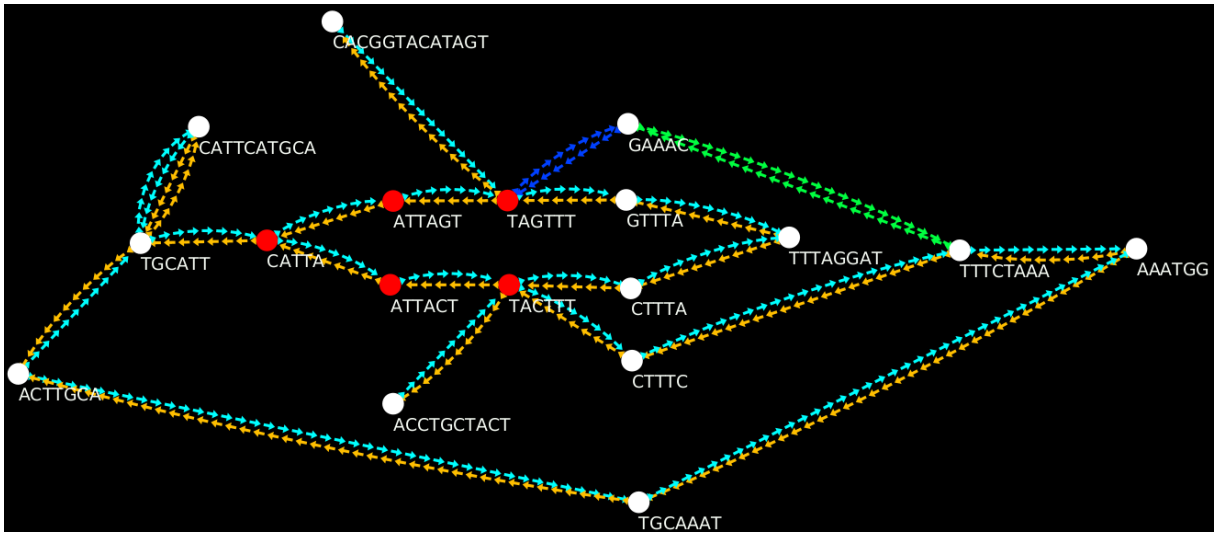

Figure 14 – Exemple de la formation d'une composante. Tous les sommets rouges ont exactement un poids de 4 et tous les autres ont un poids strictement inférieur à 3. De plus, le sommet `TGCATT` a un poids de 3 et non de 4, car on calcule les boules $|\mathcal{B}(u,3)|$ à partir de chaque k-mer et non à partir de chaque unitig. Alors pour le sommet `TGCATT`, en partant du k-mer `TGCAT`, on est bien à moins de 3 nucléotides de tous les 5-mers de `ACTTGCA` et de `CATTA`, pas pour ceux de `CATTCATGCA`.

Alors, après l'unique composante formée, j'ai pu passer à l'étape de la simplification du graphe, ce qui donne sur la Figure15 :

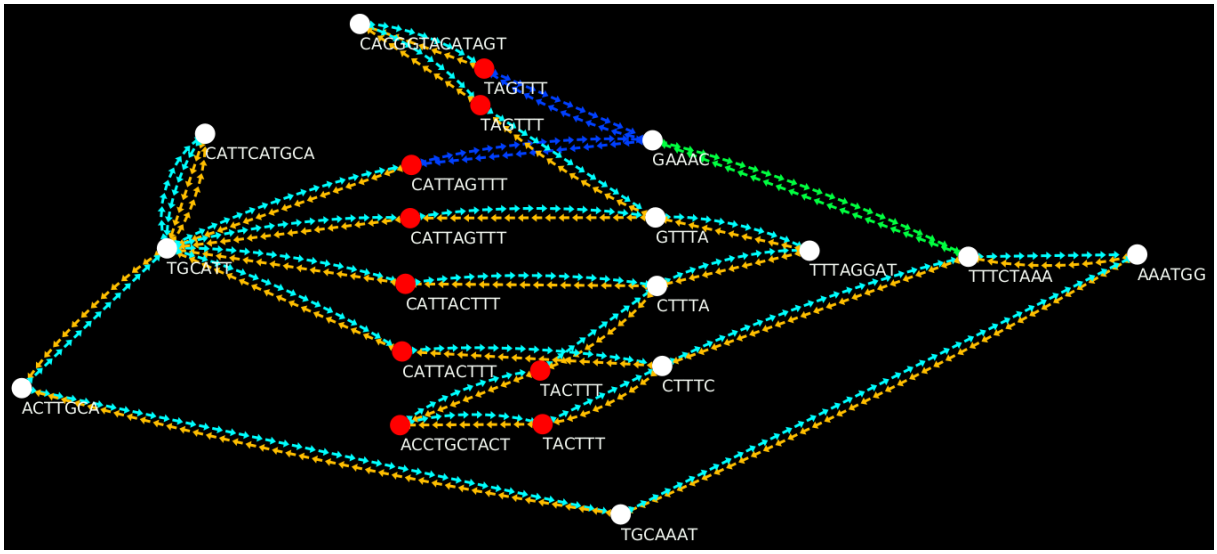

Figure 15 – Exemple de la simplification de la composante. Cette dernière est remplacée par des sommets jouant le rôle d'arêtes labellisées permettant de n'avoir plus que des sommets non branchants.

Ainsi, sur ce graphe on peut y observer quatre bulles différentes :

- Deux bulles dues au polymorphisme (voir Figure 16

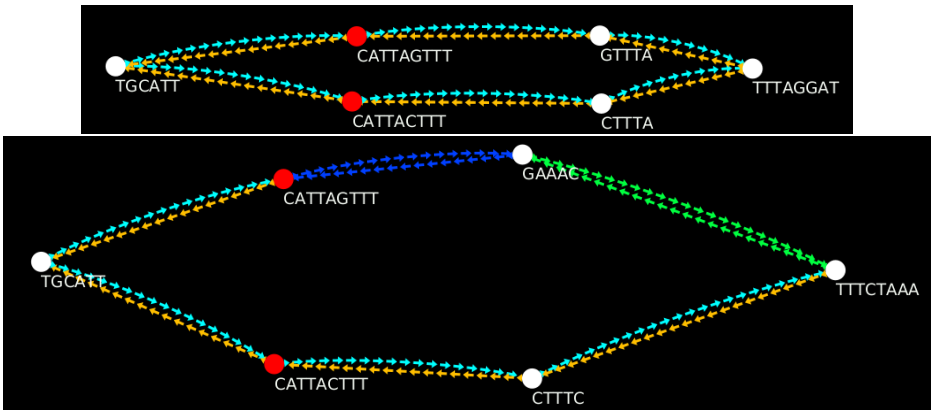

Figure 16 – Les deux bulles correspondant au polymorphisme. On peut remarque que les deux chemins passent par la composante ; on a donc bien modélisé une répétition inexacte et le fait de prendre en compte cette composante en filtrant ces types de bulles, permet un préfiltrage efficace.

- Deux bulles attestant de l'épissage alternatif (voir Figure 17)

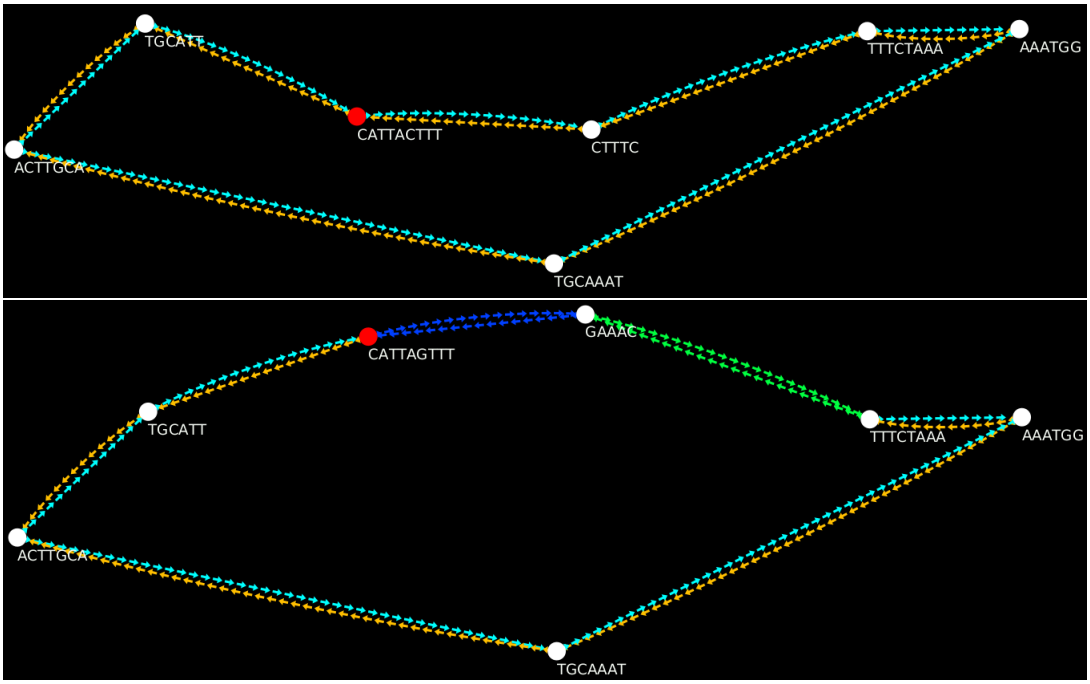

Figure 17 – Les deux bulles correspondent à un même et seul évènement d'épissage alternatif, issu de l'épissage de l'exon `TTACTTTCTAA`. Cependant, on peut remarquer qu'à première vue, il est impossible de distinguer s'il s'agit de l'exon `TTACTTTCTAA` ou `TTAGTTTCTAA` qui a été épissé et le seul moyen de faire la différence est de trouver une lecture dans jeu de données supportant un tel épissage.

Pour finir, on peut également remarquer que la répétition exacte forme un cycle (`CATTCATGCA` peut être répété une infinité de fois) mais elle ne crée pas de bulle. C'est donc pour cela que l'on s'intéresse particulièrement à des répétitions inexactes.