Comparaison et visualisation de graphes de pangénomes

Siegfried Dubois, Claire Lemaitre, Thomas Faraut, Matthias Zytnicki

Master 2 Bioinformatique

Université de Rennes

Année 2022-2023

Siegfried DUBOIS

Comparaison et visualisation de graphes de pangénomes

Sous la supervision de

Claire Lemaitre [1], **Thomas Faraut** [2], **Matthias Zytnicki** [3]

INRIA Centre Rennes-Bretagne
Campus de Beaulieu
263 Av. Général Leclerc
35042 Rennes

1. CR INRIA - Symbiose, équipe Genscale
2. CR INRAe - GenPhySE, équipe CYTOGENE
3. CR INRAe - MIAT, équipes SaAB/GenToul-Bioinfo

Master bioinformatique

UNIVERSITÉ DE RENNES 1

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e) ……Siegfried…DUBOIS……
étudiant(e) en……Master…2…Bioinformatique……
déclare être pleinement informé que le plagiat de documents ou d'une partie de document publiés sur toute forme de support, y comprit l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour la rédaction de ce document.

Date : 5 juin 2023

Signature :

Document à compléter de manière manuscrite et à insérer obligatoirement en première page du rapport de stage.

Table des matières

1 Introduction

1.1 D'un génome de référence à un graphe

L'émergence des technologies de séquençage longues lectures a considérablement modifié et simplifié le processus d'assemblage de génomes, permettant de réaliser des assemblages de qualité pour un nombre important d'individus à un coût abordable. Chez l'Homme par exemple, la mise à disposition de 350 génomes assemblés est annoncée pour l'été 2024 [a], principe qui sera étendu à d'autres espèces. Plutôt que de considérer ces génomes de manière individuelle comme un complément à l'information fournie par l'assemblage de référence, l'enjeu est de réussir à exploiter simultanément toute l'information disponible, de manière efficace et cohérente.

Une *séquence de référence* est un assemblage d'un génome d'un individu, que l'on estime étant représentatif de l'espèce ou souche observée, qui sert de système de coordonnées unifié et de séquence grâce à laquelle on va pouvoir identifier les variations. Historiquement, cette solution s'est imposée pour des raisons techniques et budgétaires : un assemblage de haute qualité représentait un travail considérable et coûtait très cher. Dans l'analyse de variations, un protocole classique consiste à aligner des lectures issues de différents individus sur cette référence, puis à détecter les variations entre les lectures et cette séquence de référence, et donc les variations au sein de la population. Cela pose des questions de *design expérimental* : comment choisir cette référence pour introduire le minimum de biais dans le protocole ? Plus la lecture est différente de la référence, plus l'alignement est difficile : le variant qu'elle portera pourra alors être perdu[2]. Le choix d'une référence, indépendamment de sa qualité, ne peut pas représenter toute la diversité génétique d'une population : il est important de garder à l'esprit qu'une séquence de référence n'est qu'un génome parmi d'autres.

Ce biais de référence est d'autant plus important en présence de grandes variations, ce que l'on appelle les variations de structure, de longues ($\geq$ 50bp) insertions, délétions, ainsi que d'autres plus complexes. En effet, une grande délétion chez l'individu utilisé pour la référence aura pour conséquence que la séquence correspondante ne sera pas prise en compte dans l'analyse. On sait par ailleurs que ces grandes variations contribuent de manière substantielle à la variabilité génétique[3], elles sont même plus importantes en nombre de nucléotides que les variations ponctuelles (SNPs/INDELS), elles ne sont donc pas à négliger.

L'alternative est de considérer simultanément de multiples génomes, et c'est l'objet et l'ambition de la pangénomique. La pangénomique computationnelle est un champ de recherche qui tend à changer les méthodes d'analyse des séquences biologiques. Le terme de *pangénome* désigne l'union des génomes d'un clade. Celui-ci peut être intra-spécifique (souches bactériennes, plusieurs individus d'une espèce), inter-spécifique (un bovin domestique et yak), ou même les deux à la fois (plusieurs individus bovins

a. Le Human Pangenome Reference Consortium (HPRC) prévoit de fournir ces assemblages pour l'été 2024, dans le prolongement de leur effort pour proposer un pangénome de référence humain[1], voir `https://humanpangenome.org`.

domestiques et un yak). Celui-ci peut être divisé en trois catégories : le *core genome* [b], le *shell genome* [c] et le *cloud genome* [d].

Parmi les approches proposées par la pangénomique, les *graphes de variations*, aussi parfois appelés *graphes de pangénome*, sont des structures de données récentes proposant une représentation de l'information génomique sous forme d'un graphe dirigé[4], exprimant l'information des séquences non plus de manière linéaire mais sous la forme d'une collection de sous-séquences reliées. Un graphe de variations permet de représenter naturellement les variations au sein d'une population : dans un tel graphe, les chemins partagés par plusieurs génomes correspondent à des régions conservées et les chemins alternatifs aux variations ; on comprend que ces graphes sont particulièrement adaptés à la représentation des grandes variations qui ne peuvent se résumer à l'énumération d'une ou plusieurs positions variables au sein d'un unique génome linéaire.

Le passage d'une séquence de référence linéaire unique à un graphe de référence comprenant l'information de multiple génomes pourrait permettre d'obtenir plus d'informations depuis les données pour un faible surcoût calculatoire, simplement en tirant parti d'informations telles que la topologie de portions du graphe, la répartition des segments... Dans ce contexte, trouver des métriques pertinentes et des méthodes d'analyse de ces structures est fondamental pour tirer parti au maximum de la structure, et se ré-approprier le graphe pour qu'il témoigne d'une réalité biologique plutôt que d'une forme de compression de la donnée génomique.

1.2 Bénéfices de la représentation en graphe

Les graphes de variations offrent, par leur structure, des caractéristiques avantageuses par rapport aux séquences linéaires. D'une part, il est aisé de retrouver les portions de séquence partagées entre génomes : comme les nœuds sont porteurs de l'information génomique, si tous les chemins décrivant les génomes passent par un nœud donné, alors la séquence du nœud fait partie du *core genome*. De manière analogue, le *shell genome* se trouve en se demandant quels nœuds sont partagés par au moins deux génomes, et le *cloud genome* quels sont les nœuds spécifiques à certains génomes. D'autre part, comme on ne représente qu'une seule fois une séquence partagée entre au moins deux génomes, on réduit l'espace de stockage nécessaire pour conserver le pangénome ; de plus, sous condition que les génomes fournis soient alignables entre eux, plus le nombre de génomes dans le graphe est grand, plus le taux de compression sera élevé.

Ces graphes sont utilisés dans des travaux récents de génotypage et de découverte de variants[5-7], se montrant particulièrement adaptés pour du *mapping* de lectures, réduisant effectivement le biais d'alignement ; permettant d'augmenter le nombre de variants détectés en améliorant la précision de l'alignement par rapport à ce qu'on obtiendrait avec une référence linéaire[8] : des variants structuraux sur des lectures courtes comme des allèles rares dans la population ont plus de probabilité de réussir à être alignés sur un graphe que sur une séquence linéaire, où ils pourraient être rejetés

b. Part commune à tous les génomes
c. Part partagée par au moins deux génomes
d. Part unique à chaque génome

car disposant d'un score d'alignement trop faible[4]. Le choix d'une séquence linéaire de référence n'est pas neutre, et pose la question de « qu'est-ce qu'un génome de référence optimal ? » que l'on ne cherche plus à répondre dans le cadre d'un graphe de variation, car c'est la réunion des génomes, ce qu'ils partagent et ce qui les différencie, qui devient la référence de l'espèce ou même de la population.

1.3 Construction de graphes de variations

Dans cette section, je vais d'abord présenter les différents outils et méthodes pour créer des graphes de variations. Ensuite, je vais expliquer comment stocker et représenter l'information d'un graphe de variations. Enfin, je vais présenter succinctement les outils permettant d'explorer et de représenter ces graphes.

1.3.1 Depuis un génome et un ensemble de variants

Si l'on dispose d'un fichier de variants au format VCF [e] (Variant Call Format) et d'une séquence de référence, il est possible de construire un graphe de variations à partir de ces informations. Avec les positions des variants le long de la référence et la connaissance des séquences alternatives, on peut enrichir le graphe itérativement en ajoutant chaque variant à la structure, en fractionnant la séquence de référence et en ajoutant chaque segment correspondant à une variation : c'est le principe de *Variation Graph toolkit* (*VG*)[8]. Pour de tels graphes, on a deux biais : celui de la méthode de détection des variants, et celui dû à l'usage d'une référence.

De plus, si les VCF ne sont pas phasés [f], de tels graphes peuvent écraser de l'information par rapport aux séquences d'origine, en plaçant tous les chemins comme équiprobables : une variation qui ne serait observée que chez un individu de la population aura le même poids qu'une variation touchant la quasi-totalité de celle-ci.

Dans cette situation, on perd le chaînage des variations : si on ne possède pas les chemins, on ne peut pas supposer à qui appartient quel ensemble de variations : le nombre de chemins que l'on peut tracer dans le graphe peut être bien supérieur au nombre d'haplotypes dans la population [g].

Dans le cadre de ce stage, *VG* n'a pas été retenu, car cela implique de disposer d'un ensemble de variants phasés et que nous souhaitions nous positionner d'emblée dans la trajectoire actuellement prise par la pangénomique : celle de l'usage d'un groupe d'assemblages et non d'une référence enrichie de variations[1] grâce à la quantité d'assemblages de qualité disponibles.

e. Format de fichier permettant de représenter des variations dans un système de coordonnées fourni par une assemblage de référence, associant à une position la séquence de référence et sa séquence alternative.

f. Un VCF phasé est un VCF qui comporte une information haplotypique, c'est-à-dire l'origine parentale de chaque variant (donc de chaque allèle). En particulier, il permet de distinguer les variations portées par un même chromosome parental : ce chromosome est un chemin dans le graphe de variations, chemin qui est donc observé dans la population.

g. Le nombre d'haplotypes d'une population augmente de manière linéaire avec le nombre de variants, car chaque nouveau variant apparaît sur un unique haplotype, et se traduit par l'apparition d'un nouvel haplotype, tandis que le nombre de chemins possibles dans le graphe augmente de manière exponentielle avec le nombre de variants.

1.3.2 Depuis un jeu de génomes

Pour construire un graphe de variations depuis un jeu de génomes, il faut résoudre la question de la détection de différences (variations) et de zones d'identité (*core genome*) entre les séquences génomiques : c'est un problème d'alignement. Cet alignement sera toujours la première étape du processus : sans lui, il n'est pas possible de savoir où sont les variations (que ce soit par rapport à une référence, ou à toutes les autres séquences) ni même de savoir si les séquences devront faire partie de la même composante connexe du graphe de variations (ce qui sous-entend qu'elles appartiennent au même chromosome).

Si la pangénomique computationnelle est un champ récent, l'algorithmique des séquences travaille sur la question de l'alignement de génomes depuis longtemps. Grâce à une fonction de score que l'on cherche à maximiser, favorisant les similarités entre les génomes, on va chercher l'appariement optimal entre les génomes. Un graphe de variations est une généralisation de l'alignement multiple, qui peut contenir des inversions et des duplications. Comme pour l'alignement multiple de génomes, même avec un système de score simple, il n'est pas possible d'utiliser un algorithme exact et on a recours à des heuristiques, en particulier des méthodes d'alignement progressives.

A notre connaissance, trois outils existent à ce jour pour construire un graphe de variations sur la base d'un jeu de séquences[9] : *minigraph*[10], historiquement la première approche proposée, qui renvoie des graphes comportant relativement peu de nœuds comparativement aux autres méthodes, *PanGenome Graph Builder* (PGGB)[11] et *minigraph-cactus*[12], qui en dépit d'une différence d'approche dans l'alignement renvoient des graphes au nombre de nœuds comparables. Dans le cadre de ce stage, tous trois ont été utilisés.

Approches guidées par référence. Cette méthode consiste en une introduction progressive d'assemblages sur une référence choisie par l'utilisateur. On va chercher à aligner chaque assemblage sur la référence, et, itérativement, ajouter des segments au graphe. Cette approche est celle utilisée par *minigraph* : il va s'agir de construire de manière itérative le graphe, en partant d'un génome de référence et en augmentant le graphe par des chemins alternatifs lorsque des variations de plus de 50 bases de long [h] sont détectées. L'identification de ces grandes variations repose sur un algorithme d'alignement d'un génome sur un graphe qui est assuré par des mots choisis judicieusement, chaînés, qui vont servir d'ancres pour le génome que l'on cherche à ajouter. Les petites variations ne sont pas considérées, et c'est le génome choisi comme référence qui sert à représenter la séquence commune : en conséquence, le graphe ne contient pas l'information de tous les génomes utilisés lors de sa construction.

Un autre outil élaborant sur cet usage d'une référence est le pipeline *minigraph-cactus*. Après la construction d'un graphe *minigraph* selon la méthodologie décrite plus haut, un ré-alignement avec *cactus*[12] assemblages d'origine est effectué sur le graphe, afin de retrouver les SNPs, INDELs et petites variations perdus par *minigraph*. Après avoir réaligné, il va « pincer » l'assemblage réaligné contre le graphe là où les bases sont

h. Ce seuil arbitraire de 50 bases de long a été historiquement sélectionné pour distinguer les SNPs et INDELS des variants structuraux.

identiques, et conserver en tant que nouveaux nœuds les petites variations perdues par *minigraph*.

Le défaut majeur de ces approches réside dans la nécessité de choisir une référence pour la construction du graphe. S'il ne s'agit pas d'une référence au sens aussi strict qu'une référence linéaire sert de modèle pour tous les individus d'une population, c'est en fonction de celle-ci que l'on va qualifier si une séquence est une insertion, une délétion... Le choix de la référence reste arbitraire et cela est susceptible d'introduire un biais.

Approches sans biais de référence. Une approche pour obtenir un graphe identique quel que soit l'ordre d'introduction des séquences dans le graphe est d'utiliser un alignement *tous-contre-tous*. Si cela permet théoriquement de construire sans biais de référence, l'aspect calculatoire n'est pas à négliger. On prend ici toutes les paires de séquences, et calculer tous les alignements entre ces paires pour construire le graphe. C'est le principe de *PGGB*, avec son algorithme *wfmash*[13], qui assure de tels alignements. Ensuite, un second algorithme, *seqwish*,[11] construit le graphe à partir de ces alignements.

La question restant en suspens est la capacité à étendre ces graphes après construction : si on ne reconstruit pas tout le graphe (et donc que l'on ne ré-effectue pas toute l'étape d'alignement) on se retrouve dans une situation similaire aux approches guidées par référence pour l'extension. Les premières versions de *PGGB* nécessitaient une reconstruction complète du graphe, mais désormais alignent la séquence à ajouter sur le graphe avant d'éditer celui-ci pour lui ajouter nœuds et arêtes.

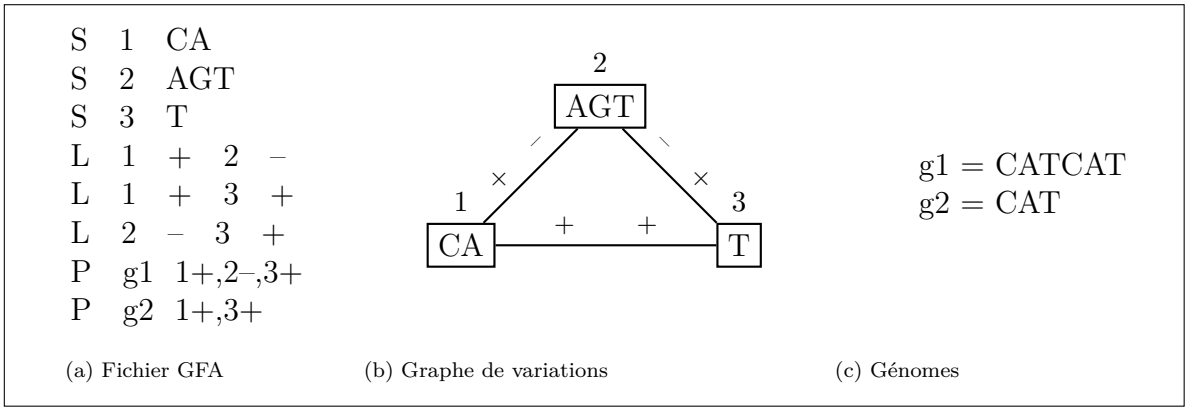
1.3.3 Représenter un graphe de variations : le format GFA

Vouloir utiliser des graphes de variations nécessite de se poser la question de leur stockage. On est dans le cas d'un graphe dirigé particulier, qui représente une collection de séquences liées par des arêtes, dont les chemins décrivent les séquences d'origine. Dans ce formalisme, les séquences sont inscrites dans les nœuds du graphe, et les arêtes donnent le sens de lecture des séquences associées aux nœuds : le format de stockage doit avoir la flexibilité pour contenir cette diversité d'information.

La spécification du *Graphical Fragment Assembly* (GFA) est un standard collaboratif, où les différents acteurs de la recherche du domaine contribuent à faire évoluer la syntaxe[14]. Ce format a été théorisé en 2014 par Heng Li[15] pour tenter d'apporter un standard générique pour représenter un assemblage de fragments génomiques ; ce format a ensuite été utilisé pour les graphes de variations, recevant des mises à jour dans sa spécification pour accommoder ce nouvel usage.

Si tous les outils présentés utilisent ce format en retour pour leur graphe, il existe de nombreuses sous-spécifications, qui ne contiennent pas nécessairement les mêmes informations, ou qui ne sont pas formatés exactement de la même manière. Chaque sous-spécification accepte un sous-ensemble des types existant dans le GFA, mais au demeurant, la structure des fichiers reste identique : chaque ligne correspond à un élément (nœud, arête, chemin), les informations sur la ligne sont séparées par des tabulations, les champs obligatoires sont identifiés par leur position au sein de la

ligne, et les champs facultatifs sont identifiés par un code à deux lettres suivi d'un code à une lettre donnant le type de la donnée (Figure 1).



(a) Fichier GFA (b) Graphe de variations (c) Génomes

Figure 1 – Pour passer du format GFA à la représentation sous forme de graphe, on doit interpréter chacune des lignes. Les lignes commençant par S (*segment*) représentent les nœuds du graphe auxquels sont associés les séquences, celles commençant par L (*link*) sont les arêtes reliant les nœuds entre eux, et celles commençant par P (*path*) sont les chemins, qui permettent de retrouver les séquences d'origine en stockant l'ordre des nœuds traversés et leur sens de lecture : un signe moins indique que l'on va lire le *reverse-complément* de la séquence, et un signe plus signifie qu'on va la lire telle quelle.

L'existence de plusieurs sous-standards implique que deux graphes contenant la même information peuvent ne pas donner lieu au même fichier s'ils sont enregistrés en des sous-formats différents, ce qui signifie que l'on ne peut pas vérifier que deux graphes sont identiques par une simple comparaison des fichiers.

1.3.4 Explorer un graphe de variations

Afin d'explorer des graphes de variations, plusieurs outils ont été développés. *VG toolkit*[8] regroupe un ensemble d'outils en ligne de commande pour manipuler des graphes au format GFA, permettant notamment d'aligner des lectures courtes sur les graphes, d'augmenter ces graphes, ou encore de faire de la recherche de variants. *ODGI*[16] propose une suite d'outils en ligne de commande permettant (entre autres) de calculer des métriques sur un graphe, convertir en divers formats et de nettoyer les régions complexes. *BubbleGun*[17] permet d'explorer les *bubbles*[i] au sein d'un graphe de variations. *GFApy*[18] propose une librairie Python pour parcourir des graphes de variations.

Côté visualisation, on peut citer *Bandage*[19], qui intègre une fonctionnalité pour afficher des graphes dans le format GFA. *ODGI* et *VG* disposent également chacun de commandes pouvant renvoyer une image statique des chemins du graphe, représentés sous forme de rubans.

i. Structures acycliques ramifiées au sein d'un graphe, toutes reliées à un nœud d'entrée et un nœud de sortie. Elles peuvent correspondre à de simples SNPs/INDELs (*bubbles* simples) tout comme à des variations complexes (*bubbles* complexes.

1.4 Challenges à relever

Le passage d'un assemblage de référence — d'une séquence linéaire — à un graphe de variations pose plusieurs problèmes. (1) Tout d'abord, la question de la sélection des génomes et la construction de leurs assemblages de qualité. (2) Ensuite, la construction de graphes de variations à partir de ces génomes. (3) Le graphe construit, l'analyse des graphes de variations obtenus pour caractériser la variabilité génétique de la population. (4) Enfin, la transposition des méthodes d'analyse génomique de l'usage d'une séquence linéaire de référence à un pangénome de référence sous la forme d'un graphe de variations.

Ces *graphes de variations* manquent d'outils pour en analyser les structures : dans de récents travaux[1,20], les graphes de variations produits sont seulement analysés au regard du nombre de segments, de la taille de ces derniers... mais ces métriques sont plus une conséquence des paramètres du programme utilisé pour répondre à la question (2) que d'une réalité biologique, variant significativement selon les outils utilisés.

1.5 Objectifs du stage

L'objectif de ce stage est d'aborder la question (3). Avec la multiplicité des méthodes de création de graphe et les évolutions perpétuelles de ces outils, et en tant que structures récentes, les graphes de variation manquent d'outils étant en capacité de comparer les approches, et de s'assurer que la méthode sélectionnée pour son analyse est pertinente ; ainsi, pouvoir séparer artefacts de construction du graphe liés aux implémentations et véritables variations de séquence est un des maillons nécessaires à cette analyse.

Suivant l'usage que l'on souhaite faire du graphe, les conditions qui devront être remplies par le graphe seront différentes : pour une annotation de gènes basée sur les positions, le respect des séquences d'origine est nécessaire ; en revanche, pour du *variant calling* ou du *mapping* de lectures, un graphe incomplet peut suffire.

Dans ce contexte, l'enjeu du stage est d'être en mesure de comprendre l'objet qu'est le graphe, d'en explorer la donnée, de pouvoir en offrir une représentation simple pour en analyser la topologie, d'être en capacité de le manipuler et d'être en capacité de pointer les différences entre deux de ces objets.

Dans la suite de ce rapport, je vais présenter plus en détail l'objet qu'est le graphe de variations, puis détailler comment j'ai exploré sa donnée. J'expliquerai ensuite comment j'ai proposé un algorithme de comparaison de graphes de variations et en expliciterai le fonctionnement.

2 Matériel et méthodes

2.1 Définition formelle d'un graphe de variations

L'objectif de cette partie est de poser une série de définitions et concepts concernant les éléments qui composent le graphe de variations que j'exploite pour proposer des algorithmes qui seront appliqués sur ces graphes.

Définition 1 (Génome) *Un génome g est représenté par un texte sur un alphabet de quatre lettres $\Sigma = \{A, T, G, C\}$, désigné par un identifiant id, tel que $g = s_1 \cdots s_n$, où n est la taille du génome.*

Définition 2 (Segment) *Un intervalle génomique, segment génomique ou segment σ de g désigne un intervalle $g[i, j] = s_i \cdots s_j$ respectant $1 \leq i \leq j \leq n$, et la chaîne de caractères (le label) associée à cet intervalle sera notée seq(σ). Deux segments de g, $g[i, j]$ et $g[k, l]$ seront dits contigus si $k = j + 1$ ou $i = l + 1$.*

Le segment correspond à l'information portée par un nœud du graphe : un nœud peut être le support de plusieurs segments aux séquences (aux labels) identiques mais aux positions et génomes d'appartenance différents. Un segment peut être lu dans deux sens : un intervalle génomique comme son reverse complément sont représentés dans le graphe par un unique segment. L'orientation, *forward* ou *reverse*, est donnée par les chemins (s'ils exitent) ou à défaut par les arêtes. Si on est en sens *forward*, on va simplement lire la séquence, sinon, on va lire son *reverse-complément* (Figure 2). Un label, tout comme un nœud, peut être associé à des segments situés sur plusieurs génomes. Plus particulièrement, ces génomes sont composés de segments de séquences identiques que l'on peut supposer d'origine ancestrale commune. L'alignement des séquences évoqué plus tôt revient donc à trouver les correspondances entre ces labels, où le partage d'un segment entre plusieurs génomes est synonyme d'un alignement local parfait (au *reverse-complément* près) entre ces génomes.

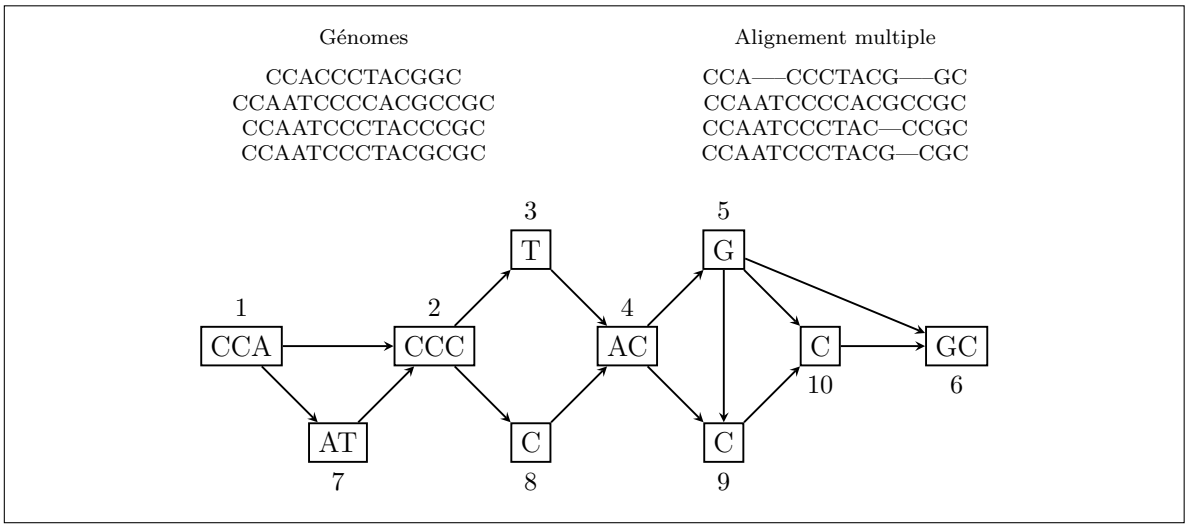


Figure 2 – Graphe de variations et ses génomes le constituant. L'alignement multiple a été ajouté pour faciliter la lecture du graphe, mais il faut garder à l'esprit qu'il n'est qu'une donnée intermédiaire et que plusieurs alignements peuvent être réalisés à partir des mêmes génomes, ce qui donnera alors un graphe différent.

Définition 3 (Graphe) *Un graphe de variations, associé à un ensemble de génomes $g_1, \cdots g_k$ est la donnée d'un graphe $G = (V, E)$, où à chaque nœud $u \in V$ on associe un label qui correspond à la séquence d'un segment d'au moins un des génomes g_i. Deux nœuds u et v sont reliés d'une arête si et seulement si leurs segments associés*

sont des segments contigus dans au moins un des génomes g_i. A chaque chemin du graphe on peut associer un label qui est la concaténation des labels des segments qu'il visite. On peut alors dire que G exprime un génome g_i s'il existe un chemin dont le label est exactement la séquence du génome g_i.

Définition 4 (Graphe complet) *Un graphe complet de pangénome $G = (V, E, P)$ est un graphe tel que défini dans 3, associé à un ensemble de chemins (P) qui décrivent la segmentation des génomes $g_1, \cdots g_k$. (P) est un sous-ensemble des chemins du graphe tel qu'il n'existe pas de chemin dans (P) dont le label n'est pas la séquence d'un génome exprimé par le graphe.*

Ces graphes, qu'ils soient complets ou non, sont nommés graphes de variations car ils représentent l'ensemble des différences observées entre les génomes : un embranchement dans le graphe représente une différence dans la séquence génomique entre les individus du graphe, et dans le cas des graphes complets, l'ensemble (P) donne l'information de la distribution de cette variation. En pratique, la construction de ces graphes repose sur de l'alignement de séquences, et donc sur une heuristique : les labels ne seront pas alignés de la même manière suivant l'algorithme utilisé.

2.2 Manipulation et visualisation de graphes de variations

Afin de pouvoir travailler avec les graphes de variations, j'ai développé un outil utilisable en ligne de commande et une librairie, tous deux écrits en Python.

La librairie, nommée *gfagraphs*, offre une abstraction du format GFA en un objet graphe, lui-même constitué d'objets décrivant chacune des lignes du fichier. Toutes les lignes d'un GFA ayant le même formalisme (champs obligatoires identifiables par leur *position* au sein de la ligne, et champs facultatifs identifiables par un *tag* et un *type*), j'ai créé un objet par type de ligne existant en GFA (*Header*, *Segment*, *Link*, *Walk*, *Path*, *Jump*, *Containment*), et ces objets héritent d'un objet décrivant la structure de base d'une ligne.

Lors de l'initialisation d'un objet graphe, le fichier GFA est lu ligne par ligne. La première lettre de la ligne permet de savoir quel type d'objet on va initialiser, et le constructeur de l'objet se charge de parser les champs obligatoires. Ensuite, on fait appel au constructeur hérité pour les champs facultatifs, qui sont traités de la même manière quelque soit le contenu de la ligne.

Plutôt que de recoder toutes les fonctions classiques de parcours de graphe, j'utilise la librairie *NetworkX*[21] pour parcourir la topologie du graphe. La structure *NetworkX* et le reste de l'objet ne sont pas gérés de manière synchrone : une édition sur le graphe GFA ne sera pas directement appliquée à la structure *NetworkX*. Il faudra que le programmeur utilisant la librairie demande un recalcul du graphe *NetworkX*. Ce choix permet d'éviter le surcoût de recalculer la structure à chaque édition, au prix de la perte d'un peu de facilité d'utilisation de la librairie. J'ai implémenté une fonction de conversion de GFA à *NetworkX*, qui va relier créer les nœuds correspondants aux segments, et les relier grâce à l'information des chemins si celle-ci est disponible, ou à défaut l'information des arêtes. Aussi, lors de l'usage de fonctions classiques de parcours de graphes (voisins d'un nœud, degré d'un nœud, calcul des composantes

connexes...) la librairie *gfagraphs* peut ainsi reposer sur les fonctions de la librairie *NetworkX*. En revanche, j'ai implémenté dans la librairie des fonctions spécifiques aux graphes de variations : recherche par identifiant d'un segment, écriture de fichiers GFA, fusion et division de segments, itération sur les chemins, qui seront utiles aux opérations d'édition de graphes que nous traiterons dans la section 2.3.3.

L'outil en ligne de commande, nommé *pangraphs*, se constitue d'un ensemble de fonctions offrant des opérations sur un ou des graphes, qui utilisent la librairie *gfagraphs* pour traiter les graphes de variations. Un script général contient un parseur de ligne de commande, offrant à chaque module du programme l'ensemble des arguments dont il a besoin, en vérifiant les types et la validité. Tous ces modules sont regroupés sous la commande *pangraphs*, à laquelle on ajoute un argument de module qui permet de choisir l'outil que l'on souhaite utiliser :

- *grapher* crée un fichier interactif pour visualiser la structure d'un graphe
- *length* calcule des métriques sur la distribution de taille des segments
- *reconstruct* renvoie les génomes linéaires décrits par le graphe de variations
- *offset* ajoute une information de position à tous les segments du graphe
- *vcfmatch* permet de comparer les variants d'un VCF aux segments du graphe
- *isolate* et *neighborhood* extraient des portions du graphe
- *convert* gère la conversion entre les différents formats GFA
- *compare* propose la visualisation et le calcul d'une distance entre deux graphes
- *edit* offre une méthode d'édition entre deux graphes de variations

Avec cet outil viennent les fichiers *setup* et *config* permettant d'une simple ligne de commande d'installer l'outil dans un environnement virtuel, qui collecte les fichiers et dépendances nécessaires, et rend possible l'utilisation en ligne de commande. Une aide, à la fois au niveau général et au niveau de chaque module est fournie afin de permettre à l'utilisateur de savoir les arguments nécessaires à l'exécution de la commande. Les fonctions et classes de *gfagraphs* et de *pangraphs* peuvent être importées dans un script Python, et sont accompagnées d'une documentation décrivant leur comportements, types d'entrée et de retour.

2.2.1 Visualisation de la topologie

Pouvoir visualiser la structure avec laquelle on travaille est un enjeu majeur : que ce soit pour diagnostiquer les sorties des outils de construction de graphe ou pour pouvoir capturer d'un coup d'oeil la distribution de la variation sur le graphe, cela permet de pouvoir valider visuellement ce qui est renvoyé par les algorithmes, d'une manière bien plus commode que d'aller chercher les lignes d'intérêt au sein du fichier GFA. J'ai développé un outil de visualisation, mettant en évidence les chemins suivis par les génomes au sein du graphe de variations. Celui-ci est intégré à *pangraphs* sous la commande *grapher*.

Le principe de représentation est le suivant : un nœud correspond à un segment du graphe de variation et une arête donne un sens de progression le long d'un chemin ; le label d'une arête donne l'orientation de lecture de la séquence et de multiples arêtes peuvent joindre deux segments. Enfin, la couleur des arêtes permet de suivre le génome dans le graphe. L'accent est mis sur la topologie du graphe, et non sur la

mise en évidence de la longueur des séquences, comme on peut le voir sur les figures de la section 3.2.

J'ai ensuite étendu cet outil avec la fonctionnalité de pouvoir visualiser plusieurs graphes dans une même interface, ainsi que les liens qu'ils partagent, afin de représenter une comparaison entre graphes, qui sera développée en section 2.3.

2.2.2 Reconstruction des génomes linéaires

Afin de valider le contenu en séquence d'un graphe, c'est-à-dire vérifier qu'il s'agit bien d'un graphe complet, qui contient l'intégralité des séquences d'origine et uniquement celles-ci, et qu'il n'existe pas d'ambiguïté lors de la lecture du graphe, on peut suivre les chemins du graphe et reconstruire, segment après segment, la séquence complète.

Définition 5 (Point de rupture) *Le graphe induit une segmentation de la séquence génomique, qui correspond au découpage en nœuds du chemin associé au génome. Un point de rupture est une position x sur une séquence s telle que $0 < x < |s|$ définissant la position de fin d'une séquence d'un segment σ^i et de début d'une séquence d'un segment σ^{i+1}. La segmentation induite par l'ensemble des points de rupture est une partition de la séquence génomique, qui est constituée de segments contigus dont la concaténation représente la séquence complète.*

Dans le graphe, les génomes sont décomposés en segments : c'est le résultat de l'application des points de rupture sur la séquence complète du génome, et l'enchaînement de ces segments va former des chemins.

Pour vérifier sur des graphes de variations que tous les génomes ayant permis de créer le graphe sont bien exprimés dans ce dernier, j'ai développé un outil qui extrait chaque chemin depuis le GFA, itère à travers les segments de ce chemin pour reconstruire de gauche à droite la séquence, et la compare au génome d'origine. Pour se diriger au sein du graphe, on utilise les informations fournies dans le fichier GFA : les *walks* et les *paths*. En GFA, un chemin est constitué d'une succession de paires segment - orientation. Pour chaque segment référencé dans le chemin, si l'orientation est *forward* on concatène la séquence que l'on est en train de reconstruire et la séquence du segment. Sinon, c'est qu'on est en *reverse* et on concatène la séquence que l'on est en train de reconstruire et le *reverse-complément* de la séquence. En itérant sur chaque segment jusqu'à atteindre le bout du chemin, on obtient la séquence décrite par le chemin.

Cet outil est intégré à *pangraphs*, dans la commande *reconstruct*. Pour être exécuté, il nécessite un fichier GFA avec des chemins. Si à l'issue de la reconstruction, on ne retrouve pas les génomes d'origine, alors on peut en conclure que le graphe de variations n'est pas un graphe complet de pangénome.

2.2.3 Positions dans les chemins

La question des coordonnées au sein d'un graphe est importante pour des tâches telles que l'annotation ou l'extraction de portions du graphe. Afin de pouvoir se repérer plus facilement dans le graphe, j'ai introduit un nouveau *tag* compatible avec la spécification GFA nommé PO (pour *Path Offset*). Ce champ facultatif est une

chaîne JSON, calculée pour chacun des nœuds du graphe, qui se présente sous la forme d'un dictionnaire dont les clés sont les noms de chemins qui traversent le nœud et les valeurs sont un triplet position de début, position de fin et sens de lecture.

Ainsi, **PO:J:{'w1':(333,335,'+'),'w2':(245,247,'-')}** donne l'information que le chemin w1 contient la séquence associée au nœud et qu'elle commence à la position 333 et se termine à la position 335 du génome d'identifiant w1, et le chemin w2 contient cette même séquence qui débute à la position 245 et termine à la position 247 du génome d'identifiant w2, qui est inversée. J'ai implémenté l'outil permettant d'ajouter ce système de coordonnées au GFA, et celui-ci est inclus à *pangraphs* sous le nom *offset*. Il requiert un fichier GFA avec des chemins, et ajoute à chacune des lignes décrivant un nœud le champ PO.

2.2.4 Extraction de sous-graphes

Pouvoir extraire des portions d'un graphe de variations permet de visualiser ou d'étudier des zones réduites de celui-ci. Via les outils *neighborhood* et *isolate*, inclus à *pangraphs*, je propose deux types d'extractions.

Le premier repose sur la topologie des segments du graphe, en récupérant les noms de tous les segments situés autour d'un segment cible, et en se limitant aux n segments les plus proches. D'abord, on récupère les noms des n segments voisins à notre cible avec la fonction bfs^j du package BubbleGun[17]. Ensuite, on copie tous les segments dans un nouveau fichier, puis on copie toutes les arêtes dont le départ et l'arrivée figurent dans notre liste de segments. Enfin on copie les éventuels chemins, et on retire de ces chemins tous les segments ne figurant pas dans la liste. Cette méthode ne nécessite pas d'avoir les chemins dans le GFA.

Dans le second, j'utilise les positions en nucléotides au sein des génomes. On va cibler un intervalle, défini par une position de début et une position de fin, sur une séquence qui va nous servir de référence pour ces coordonnées le temps de l'extraction : on cherche les segments σ^i et σ^j dans ce graphe incluant les extrémités de l'intervalle. Ensuite, on récupère le chemin de la référence entre σ^i et σ^j et pour tous les autres chemins, on cherche le plus long chemin $\sigma^x \cdots \sigma^y$ tel que $\sigma^x, \sigma^y \in \sigma^i \cdots \sigma^j$. On extrait la liste de segments qui est l'union des segments de tous les chemins extraits, et on copie les chemins, tous les segments de la liste et toutes les arêtes dont le départ et l'arrivée figurent dans notre liste de segments dans le GFA de sortie. Cette méthode nécessite la présence des chemins dans le GFA, et que ce GFA soit annoté du *Path Offset* (PO) présenté précédemment.

Dans les deux cas, on récupère en sortie un nouveau fichier GFA ne contenant que les segments cibles et leurs relations, sans altération de la topologie ou du contenu en séquences de la portion du graphe extraite.

2.3 Comparaison de graphes

Ces outils de parcours et de visualisation nous ont permis de mettre en évidence l'existence de divergences entre graphes issus d'outils de création différents représentant la même donnée génomique. Nous avons remarqué la présence de nombreuses

j. Une fonction BFS (Breadth First Search) est une fonction effectuant un parcours en largeur d'un graphe, en retenant l'état (visité ou non) de chacun des nœuds, afin d'éviter de comptabiliser plusieurs fois un même nœud. L'usage ici est de trouver les n plus proches voisins d'un nœud.

portions identiques et de quelques portions divergentes, dispersées dans le graphe. L'objectif que nous nous sommes fixés est de pouvoir identifier et comprendre les différences entre deux graphes représentant le même ensemble de génomes.

S'il n'existe pas de telle méthode spécifique aux graphes de variation, la *distance d'édition* entre graphes est un concept défini[22] : c'est un calcul des points de divergence entre deux graphes, qui décrit comment modifier un graphe pour qu'il corresponde à l'autre. Il n'existe pas une seule, mais bien de multiples manières de calculer une distance d'édition. Si pour des graphes génériques faire correspondre la topologie de deux graphes est un problème complexe[23], ce que l'on cherche à faire correspondre ici n'est pas seulement une topologie, mais surtout les segmentations associées aux génomes, et cela simplifie le problème. Un label n'est pas qu'une chaîne de caractères, c'est aussi un intervalle génomique, ce qui réduit drastiquement l'espace de recherche : on progresse de gauche à droite sur nos deux graphes, et on a juste à considérer les quelques nœuds devant notre position actuelle pour comprendre l'origine de la différence entre les graphes.

Dans la section suivante, je vais présenter un algorithme permettant de calculer une distance d'édition entre deux graphes de variations, basée sur les chemins. Je vais d'abord poser les conditions nécessaires à son application, puis présenter la méthode de calcul de cette distance d'édition, pour enfin montrer comment les modifications du graphe sont effectuées. L'outil est implémenté dans *pangraphs*, sous le nom de commande *compare*. Lorsque l'on considérera deux graphes, l'appartenance d'un segment ou d'un chemin à un graphe sera dénotée par l'indice, permettant ainsi d'identifier σ_x comme un segment compris dans le graphe G_x.

2.3.1 Appariement entre graphes

Par appariement entre graphes, on entend ici le fait d'établir les correspondances entre segments de deux graphes, où une équivalence de segments sera à la fois une identité de séquence entre deux segments et une identité de position au sein des génomes traversant le segment. On sait qu'en GFA, on peut avoir le même graphe et des noms de segments différents, ce qui nous empêche de les utiliser comme base pour se repérer. En revanche, si on considère que nos deux graphes ont été construits depuis le même ensemble de génomes, et que les deux graphes sont des graphes de pangénome complets, alors leurs systèmes de coordonnées sont partagés ; cela permet d'éviter de comparer chaque nœud d'un graphe à tous les nœuds de l'autre. On peut donc suivre les séquences au sein des graphes par leurs positions, réduisant ainsi le calcul de l'édition à savoir quel découpage ont subi les paires de génomes dans chacun des graphes.

Proposition 1 (Chemins homologues) *Soit une paire de chemins ν_1 et ν_2. On dira qu'il s'agit de chemins homologues si la concaténation des séquences des segments formant les deux chemins est identique. On notera alors $\nu_1 \cong \nu_2$.*

La relation d'homologie entre chemins est symétrique : passer d'un chemin à l'autre revient à appliquer une différente série de points de rupture sur un même génome. La concaténation des deux chemins donne la même séquence, la seule chose qui peut différer entre eux est la segmentation.

Proposition 2 (Graphes homologues) *Soient deux graphes G_1 et G_2, possédant tous deux n chemins partagés et ordonnés. Ces graphes sont dits homologues si et seulement si chaque chemin ν_1^i est homologue à ν_2^i.*

$$G_1 \cong G_2 \Leftrightarrow G_2 \cong G_1 \Leftrightarrow \forall i \in [0, n),\ seq(\nu_1^i) = seq(\nu_2^i) \quad (1)$$

Deux graphes de variations sont identiques s'ils sont homologues et que les points de rupture appliqués aux génomes sont identiques. Cela se traduit par une segmentation identique sur le graphe : la comparaison de graphes va alors consister, pour chaque génome g, à comparer sa segmentation au sein des deux graphes.

Pour avoir l'appariement entre graphes, on a besoin de vérifier l'homologie car les opérations d'édition n'ont de sens qu'entre des graphes partageant le même contenu en séquence, et avoir un système de coordonnées pour localiser les points de rupture. Pour l'homologie, je vérifie que le résultat de la concaténation des segments des chemins sont identiques par paires entre les deux graphes, selon la méthode présentée section 2.2.2. Pour les coordonnées, j'ajoute aux deux graphes le *Path Offset*, présenté en section 2.2.3, qui va permettre pour chaque segment de savoir la position exacte qu'il occupe dans chacun des génomes constituant le graphe.

2.3.2 Calcul de la divergence

Après s'être assurés de partager un même système de coordonnées, on peut comparer les graphes. Plutôt que de comparer tout le graphe en une passe, l'algorithme que je présente repose sur l'analyse successive des génomes et de leurs chemins respectifs au sein des deux graphes. Dans la suite de ce développement, on pose que G_1 est le graphe modèle, et G_2 est le graphe que l'on va éditer pour le transformer en G_1.

Pour chaque paire de chemins homologues (ν_1, ν_2) qui représente le chemin suivi par un même génome g dans deux graphes (G_1, G_2), on calcule la différence entre les points de rupture appliqués sur le génome, qui explique la différence entre les deux segmentations. On va simultanément parcourir de gauche à droite les deux chemins, en retenant la position à laquelle on se trouve au sein du génome.

Si on considère deux segments appartenant chacun à un des deux chemins homologues dont les positions sont telles qu'ils partagent au moins une base, alors ces segments peuvent soit être identiques, soit être décalés l'un par rapport à l'autre, soit être inclus l'un dans l'autre. On nomme *relation* cette correspondance.

Proposition 3 (Relation) *Soient deux segments σ_1^i et σ_2^j, appartenant respectivement aux chemins ν_1 et ν_2. Si σ_1^i et σ_2^j partagent au moins une position du génome g, ils sont liés par une unique relation η. Pour toute relation η entre deux segments σ_1^i et σ_2^j, il existe une unique relation inverse $\bar{\eta}$ qui permet d'expliquer le lien entre σ_2^j et σ_1^i, sous condition $\nu_1 \cong \nu_2$.*

On distingue neuf relations différentes qui couvrent toutes les situations possibles. Pour chacune d'entre elles, à partir de quatre points (points de rupture de début et de fin des deux segments) on définit quel est le recouvrement de l'une par rapport à l'autre. Par exemple, si le segment σ_1 se termine après le début du segment σ_2, tout

en commençant avant la fin de σ_2, alors la relation liant σ_2 et σ_1 est celle d'un *OverlapSuffixPrefix*, abrégé en *OSP* (Figure 3). Les différentes relations sont présentées en annexe (voir annexe A). Ces relations appartiennent à trois groupes : inclusions (un segment est contenu dans l'autre), décalages (il y a chevauchement entre les segments) et équivalences (les segments sont identiques).

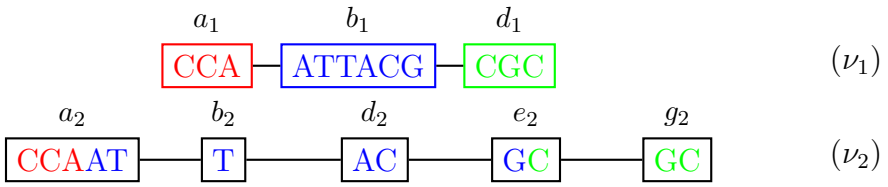


Figure 3 – Exemple de relation entre segments des chemins ν_1 et ν_2, appartenant respectivement aux graphes G_1 et G_2. La relation entre a_2 et b_1 est un *OverlapSuffixPrefix*, tout comme celle entre e_2 et d_1.

Suivant les points de rupture appliqués au génome, un segment du graphe G_1 ne va pas nécessairement correspondre à un unique segment du graphe G_2, et donc à une unique relation. Il va s'agir, pour chaque segment du chemin du génome dans G_1, de projeter sur G_2 ce segment pour calculer toutes les relations qui lient ce segment au(x) segment(s) de G_2. Comme on travaille avec les positions au sein du génome, il va s'agir de retrouver comment sur G_2 les points de rupture s'appliquent sur l'intervalle correspondant au segment de G_1.

Une fois que l'on a les relations associées à chacune des projections des segments de G_1 sur les segments de G_2, on peut répondre à la question des différences entre G_1 et G_2. Pour prouver que l'ensemble des relations que l'on a calculé est correct, on peut modifier le graphe G_2. Si on arrive à retrouver G_1, alors on aura montré que les relations calculées expliquent bien comment passer d'un graphe à l'autre.

2.3.3 Application de l'édition

Le processus d'édition consiste à résoudre, pour chaque génome, les relations conflictuelles entre les deux segmentations, c'est-à-dire toutes celles qui ne décrivent pas une équivalence, c'est-à-dire une relation *String* (STR) dans notre terminologie. Dans la partie précédente, j'ai introduit la notion de relation, et la manière selon laquelle elles étaient identifiées ; si ces relations décrivent fidèlement la correspondance entre segments de deux graphes, elles ne disent pas comment modifier le graphe pour le faire correspondre à l'autre. Comme la seule différence entre deux chemins homologues est la liste des points de rupture qui est appliquée au génome, on a intuitivement besoin de deux opérations : une première pour découper un segment en deux, et une autre pour rassembler deux segments en un.

Définition 6 (Opération élémentaire) *Une opération élémentaire est une fonction appliquée sur les segments d'un chemin, qui modifie la segmentation en ajoutant (division) ou retirant (fusion) un point de rupture, sans altérer la séquence exprimée par le chemin.*

Dans les définitions suivantes, le terme de segment sera utilisé, car on va parler de la division (et de la fusion) de la séquence génomique, mais il convient de garder à l'esprit qu'effectuer une division (ou une fusion) implique l'ajout (respectivement le retrait) d'un nœud au graphe, et la mise à jour des arêtes et des chemins associés.

Définition 7 (Division) *Étant donné un segment σ^i sur un chemin ν d'un graphe G et un point de rupture, la division est l'opération élémentaire qui scinde σ^i en deux segments contigus σ^i, σ' au niveau du point de rupture, et crée une arête entre ces segments. Le nouveau segment va être inséré dans le chemin ν après σ^i, les arêtes entrantes de σ^i vont être conservées, et les arêtes sortantes de σ^i vont être modifiées comme arêtes sortantes de σ'.*

Définition 8 (Fusion) *Étant donné deux segments σ^i et σ^{i+1} n'ayant respectivement qu'une arête sortante et une arête entrante, d'un graphe G où les deux segments sont contigus sur un chemin ν et liés par une arête sortante de σ^i et entrante en σ^{i+1}, la fusion est l'opération élémentaire qui rassemble σ^i et σ^{i+1} en un unique segment σ', et supprime l'arête comprise entre σ^i et σ^{i+1}. L'ancien segment σ^{i+1} va être retiré de ν, les arêtes sortantes de σ^{i+1} vont être modifiées comme arêtes sortantes de σ', et les arêtes entrantes de σ^1 vont être rattachées comme arêtes entrantes de σ'.*

Par la suite, on nomme *édition* l'action de modifier le graphe par une séquence d'opérations élémentaires et on dira que cette édition est optimale si la distance associée est minimale pour le problème observé.

Proposition 4 (Distance d'édition) *Étant donné un ensemble de relations entre chaque segment de chaque chemin de G_1 et les segments de G_2, la distance d'édition $G_2 \to G_1$ correspond à la somme du nombre d'opérations élémentaires — fusions et divisions — qu'il est nécessaire d'appliquer à chaque chemin d'un graphe G_2 pour qu'il soit identique à G_1.*

Pour traduire les relations en éditions, je progresse de gauche à droite le long du chemin ν_1, en regardant pour chaque σ_1 les relations [k] que l'on a calculées à l'étape précédente. Dans un premier temps, j'opère toutes les divisions (Figure 4), chemin par chemin. Il n'y a besoin de découper un segment sur G_2 que s'il dépasse par la droite par rapport au segment observé sur G_1 ; en conséquence, seuls les évènements *OverlapPrefixSuffix* (OPS), *SuperPrefix* (SPP) et *SuperString* (SPR) nécessitent une division. Dans tous les autres cas, on va simplement itérer. La raison est que ces trois évènements sont les seuls qui décrivent un dépassement du segment du graphe que l'on veut modifier par la droite par rapport au segment sur le graphe de référence : comme on traite de gauche à droite, on est assurés que tout ce qui est en amont à gauche a déjà été divisé si nécessaire, et il ne reste qu'à gérer les cas à droite. Le pseudocode de la division est décrit dans l'Annexe B.

k. Décrites dans l'annexe A

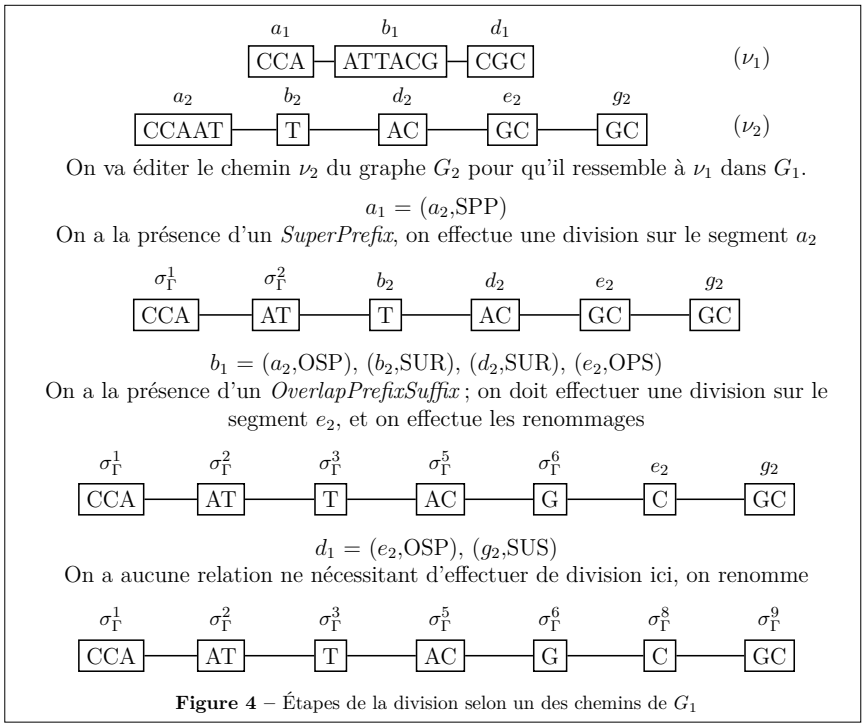
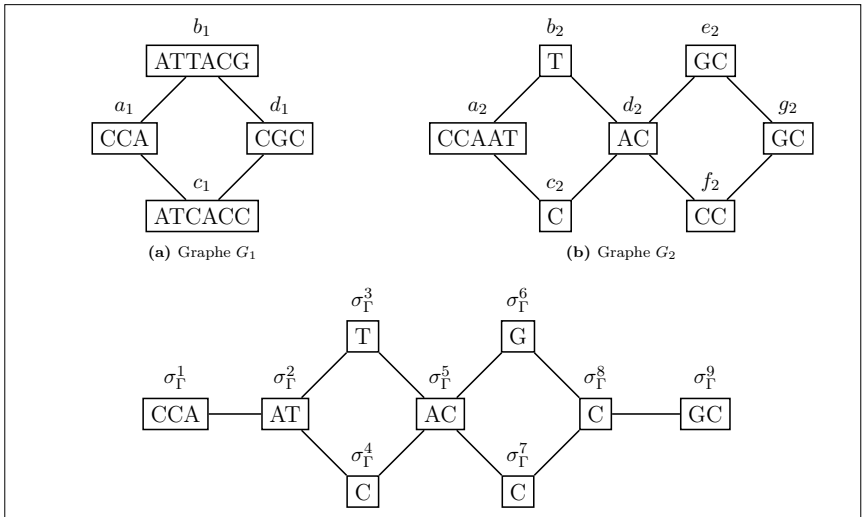


Figure 4 – Étapes de la division selon un des chemins de G_1

A ce stade, une fois que toutes les divisions de tous les chemins ont été appliquées, on dispose d'un graphe intermédiaire Γ décomposé au maximum (Figure 5). Il est constitué de l'union des divisions entre G_1 et G_2. On peut également le considérer comme le graphe consensus entre G_1 et G_2, dans le sens qu'il ne faut qu'effectuer des fusions pour obtenir G_1 ou G_2) partir de Γ.



(a) Graphe G_1 **(b)** Graphe G_2

Figure 5 – Graphe Γ, union des divisions entre G_1 et G_2. On considère ici que G_1 et G_2 sont tous deux construits à partir des génomes « CCAATTACGCGC » et « CCAATCACCCGC ».

Dans un second temps, j'effectue toutes les fusions (Figure 6), chemin par chemin. Le nombre de segments à fusionner sur Γ par segment de G_1 projeté est égal au nombre de relations τ dans la projection. Cette propriété est due à la définition de

la relation : une relation cible un intervalle qui correspond à un unique segment. Le seul cas contraire est celui des segments qui dépasseraient sur le segment suivant, car ceux-ci sont alors partagés entre deux segments de G_1 ; toutefois, comme on a appliqué toutes les divisions précédemment, il est impossible que deux relations sur un même chemin aient pour cible le même segment. Le pseudocode de la division est décrit dans l'Annexe C.

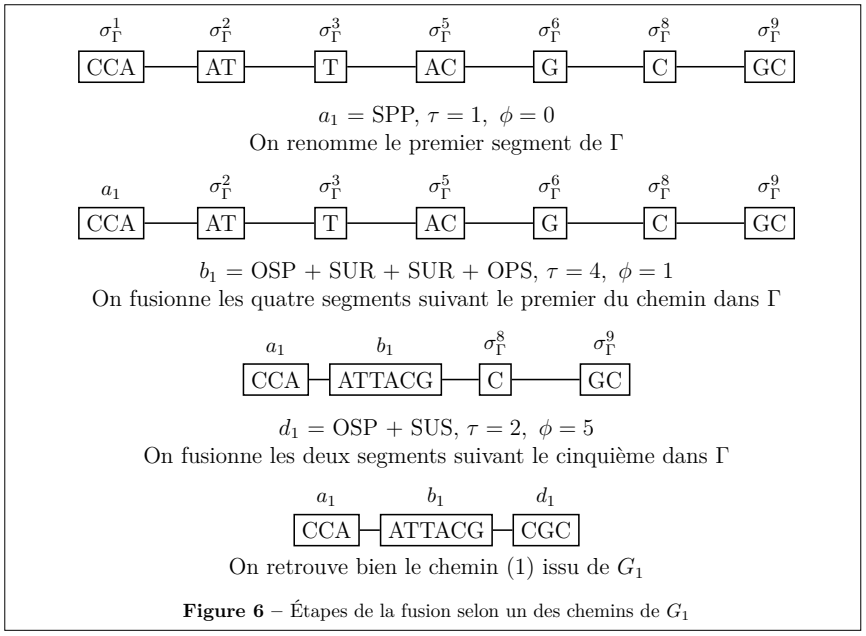


Figure 6 – Étapes de la fusion selon un des chemins de G_1

Dans l'implémentation, pour s'assurer de ne pas créer de conflit avec les autres segments, je copie dans un graphe vide temporaire G_t les nœuds et arêtes qui forment le chemin en cours d'édition, et effectue les modifications sur ce graphe, qui ne contient donc qu'un chemin. Après applications des divisons et des fusions, on dispose du chemin tel qu'il est dans G_1 isolé en un graphe temporaire G_t.

Une fois que l'édition sur un chemin est achevée, j'étends le graphe que l'on va retourner avec ce chemin : si un segment, une arête ou un chemin décrit dans le graphe temporaire G_t n'existe pas dans le graphe final, on l'y ajoute. L'algorithme complet est présenté en annexe (voir Annexe E).

Pour deux graphes G_1 et G_2 homologues, l'application des éditions $G_1 \to G_2$ et $G_2 \to G_1$ nécessitent le même nombre d'opérations, mais le nombre de fusions et divisions sont inversées : cela provient du fait que le nombre de divisions et de fusions découle de la nature des relations, et que si on change de graphe de référence on inverse les relations. La complexité de l'algorithme dépend directement du nombre de points de rupture sur les chemins du graphe. Si on note respectivement n et m la somme du nombre de points de rupture de chacun des chemins des deux graphes, la complexité sera en $O(n + m)$ car pour chaque point de rupture sur le chemin ν_1^i on cherchera à faire correspondre un ou des points de rupture sur ν_2^i. Il faut garder à l'esprit que le nombre de points de rupture sur un chemin dépend du nombre de génomes dans le graphe.

2.4 Données utilisées

Les données employées sont des assemblages complets, issus du projet **SeqOccin** :
— **Bovin** (1 individu) : 1 assemblage consensus + 2 haplotypes résolus (HiFi HiC)
— **Maïs** (4 individus) : 4 assemblages (HiFi HiC)

La construction d'un graphe de variations à partir des assemblages de l'individu bovin, une génisse, a été initialement motivée par l'observation d'une différence de plusieurs mégabases de taille entre l'assemblage que l'on nomme consensus (pour lequel on ne fait pas de différence entre les chromosomes parentaux) et les assemblages avec les haplotypes résolus (une séquence par chromosome parental). L'objectif était donc d'utiliser le graphe de variations construit avec ces trois assemblages pour identifier les différences entre ces assemblages qui correspondent pourtant au génome d'un seul individu. Ce travail de comparaison est en cours, et nos premières analyses sur ces données ont souligné la dépendance du graphe à la méthode de construction, raison pour laquelle nous nous sommes orientés vers cette question de comparaison de graphes de variations.

Dans un second temps, nous avons choisi de travailler également sur des données issues du maïs, et ce pour deux raisons : tout d'abord pour tenter de nous assurer que nos observations n'étaient pas spécifiques aux génomes animaux (plus spécifiquement bovins), et d'autre part pour la disponibilité de lignées dites pures chez le maïs où l'ensemble du génome est homozygote, et se comporte donc comme un génome haploïde. Les assemblages issus de lignées pures simulent des assemblages avec haplotypes résolus, et ils correspondent bien à des chromosomes présents dans la population et non à un mélange des chromosomes parentaux.

Afin de tester les algorithmes proposés, des graphes d'exemple ont été réalisés. Ceux-ci l'ont été de deux manières : soit par construction d'un graphe de grande taille avec les trois outils présentés plus tôt [l] (chromosome entier d'une quarantaine de mégabases) dont on extrait seulement la portion située entre deux positions d'un des chemins, soit par construction d'exemples abstraits, sur papier, permettant de tester l'ensemble des comportements de superposition, rapports préfixe-suffixe... La première approche a été employée pour tester les outils de visualisation, d'annotation, reconstruction ; là où la seconde méthode a été utilisée dans le cas de l'algorithme d'édition.

3 Résultats

Dans cette section, je vais tout d'abord présenter quelques résultats synthétiques sur nos expériences de création de graphes de variations avec les différents outils, et sur des propriétés générales des graphes obtenus. Ensuite, je vais montrer les visualisations que l'outil implémenté peut offrir, ainsi que les capacités des outils de manipulation de graphe implémentés. Enfin, je vais présenter les résultats obtenus avec notre algorithme de comparaison de graphes.

l. *minigraph*, *minigraph-cactus* et *PGGB*.

3.1 Outils de construction de graphe

J'ai comparé les graphes renvoyés par les différents outils sur la donnée bovine. Dans la table 1, les données utilisées sont le chromosome 28 des séquences bovines : un assemblage consensus (46.9Mb) et deux haplotypes résolus (47.1Mb et 42.7 Mb), et tous les outils ont été utilisés avec leurs paramètres recommandés. Lorsqu'une référence était nécessaire (*minigraph* et *minigraph-cactus*), l'assemblage consensus a été choisi comme référence.

	unité	minigraph	minigraph-cactus[1]	pggb[2]
Nombre de nœuds	N	753	259459	297808
Nombre de nœuds $\geq$ 50 bp	N	650	59494	60207
Nombre de nœuds $<$ 50 bp	N	103	199965	237601
Nombre de nœuds dans la référence	N	543	174889	232804
Nombre de nœuds hors référence	N	210	84570	65004
Nombre d'arêtes	N	1073	349608	401596
Taille des séquences reconstruites	bp	N/A[3]	135188912[4]	134677171
Longueur totale des nœuds	bp	47085793	47186793	48066617
Longueur totale des nœuds dans la référence	bp	46911972	46911972	46676047
Longueur totale des nœuds hors référence	bp	173821	274821	1390570
Longueur totale des nœuds $\geq$ 50 bp	bp	47085114	46363682	47087236
Longueur totale des nœuds $<$ 50 bp	bp	679	823111	979381
Format renvoyé	spec	rGFA	GFA1.1	GFA1.0
Temps de construction du graphe[5]	mn	4	11	30
Mémoire maximale utilisée[5]	Go	7	43	76

Table 1 – Comparaison des structures renvoyées par les outils de création de graphe, appliqués sur les données du chromosome 28 des trois assemblages bovins.
[1] Graphe par défaut du pipeline (graphe *clip*).
[2] Graphe après application de *smoothxg*.
[3] Reconstruction impossible sans les chemins, qui sont inexistants en rGFA.
[4] La différence avec la somme des longueurs des séquences est due au retrait des SV$\geq$10kb dans le graphe *clip*. Avec le graphe *full*, on a bien 136722642 bp, ce qui correspond à la somme exacte des longueurs des assemblages.
[5] Tous les calculs ont été exécutés sur 8 coeurs avec 256 gigaoctets de RAM.

3.1.1 *Minigraph*

L'outil *minigraph*, implémenté en C, permet de créer des graphes au format rGFA (un sous-format du GFA1.0 qui ne conserve pas les chemins). Il permet aussi d'aligner des séquences sur le graphe après construction. *Minigraph* retire les petites variations (de taille inférieure à 50 bases) au profit de la référence. Cela empêche les génomes d'entrée d'être intégrés sans perte au graphe.

3.1.2 Pipeline *minigraph-cactus*

L'implémentation de *minigraph-cactus* se présente sous la forme d'un pipeline Toil, qui peut soit être exécuté d'une seule commande, soit dont chacune des étapes peut être exécutée séparément. Ce pipeline requiert en entrée un fichier texte donnant les adresses des fichiers FASTA, ainsi que les arguments indiquant les sorties demandées

(GFA, HAL, PAF, VCF...) Il est également possible de donner un fichier JSON de paramètres en entrée, pour changer les options de l'aligneur par exemple. Pour être exécuté, le pipeline requiert un processeur compatible AVX-2.

Suivant les paramètres spécifiés, le pipeline renvoie jusqu'à quatre graphes en sortie : le graphe *minigraph*, un graphe identique au graphe que *minigraph* renverrait avec les paramètres par défaut, au format rGFA ; le graphe **full** (.full.gfa), qui est un graphe normalisé sans séquences retirées au format GFA1.1 ; le graphe **clip** (.gfa), avec séquences de taille supérieure à 10 kilobases non alignés retirées, au format GFA1.1, et enfin le graphe **filter** (.d2.gfa), obtenu par retrait des nœuds couvrant moins de deux haplotypes depuis le **clip**, aussi au format GFA1.1.

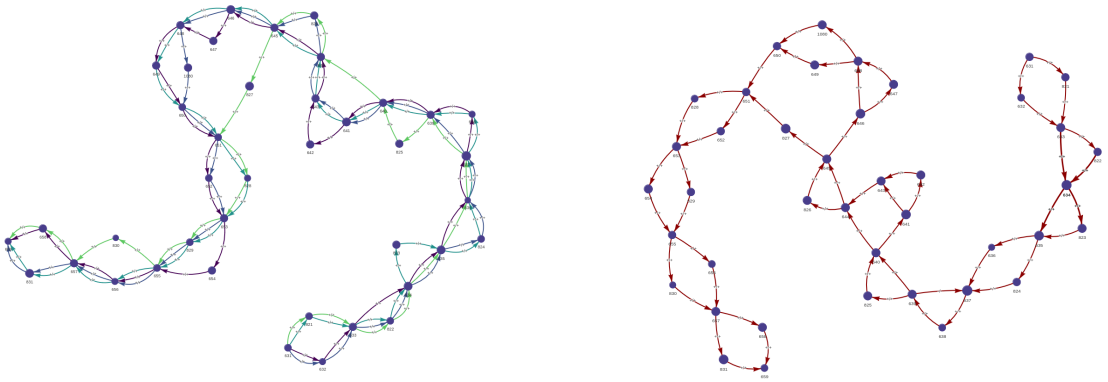
3.1.3 Pipeline *PGGB*

PGGB est un pipeline bash, qui s'exécute d'une simple commande et dont les différentes étapes ne peuvent pas être exécutées séparément sans modification du pipeline. Les entrées sont composées des séquences qui doivent être en un unique fichier FASTA, ainsi que d'un ensemble de paramètres détaillés dans l'aide du pipeline.

Notre compréhension actuelle des sorties de l'outil et de la méthode ne nous a pas permis de véritablement exploiter ces graphes. J'ai pu observer que des portions des génomes d'entrée disparaissaient du graphe final, sans que cela ne soit justifié comme dans le cas de *minigraph*. Le pipeline renvoie cinq fichiers GFA, correspondant à ses différentes étapes ; le preprint de l'outil[11], disponible sur biorxiv et datant du 6 avril 2023 n'est pas très détaillé quant aux sorties, et l'outil reste en phase active de développement.

3.2 Visualisation de graphes

L'outil que j'ai développé permet de visualiser, par l'intermédiaire de fichiers HTML interactifs, des portions de graphes de variation (Figure 7). Il est possible de zoomer, de se déplacer dans le graphe, de bouger des nœuds, et de visualiser sur un bandeau latéral des métriques sur le graphe.



(a) Graphe avec chemins **(b)** Graphe sans chemins

Figure 7 – Visualisation d'une portion de 10 kb d'un graphe de variations issu du pipeline *minigraph-cactus* sur 4 assemblages de maïs avec l'outil *pangraphs*. Si on dispose de l'information des chemins, ceux-ci seront mis en évidence au sein du graphe, sinon, une arête simple rouge reliera les nœuds du graphe.

Il permet également de mettre en évidence des appariements entre graphes de variations (Figure 8). Par souci de lisibilité sur de grands graphes, j'ai choisi sur cette représentation de ne pas faire figurer toutes les relations décrites en annexe A, mais seulement les trois groupes auxquels elles appartiennent : inclusions, décalages et équivalences.

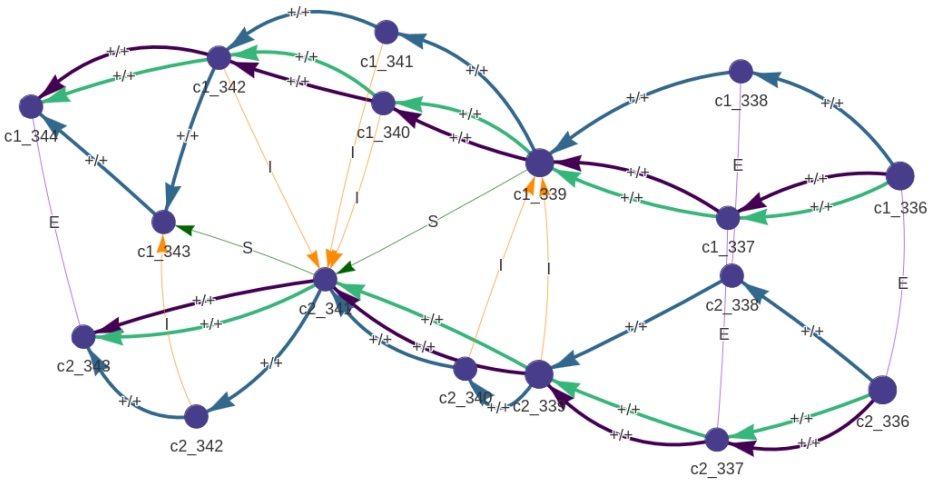


Figure 8 – Visualisation d'un appariement entre une même portion de 10 kb de deux graphes de variations issu du pipeline *minigraph-cactus* sur les données bovines, avec des paramètres d'aligneur différents, grâce à l'outil *pangraphs*. Les graphes sont identifiables grâce au préfixe du nom des nœuds (*c1* et *c2*) Les liens entre les deux graphes montrent ce qui unit les nœuds, regroupées par types : les inclusions (I), les décalages (S) et les équivalences (E).

3.3 Manipulation de graphes de variation

3.3.1 Reconstruction des génomes linéaires

La reconstruction des génomes permet de vérifier que le graphe est bien un graphe complet de pangénome. Sur le graphe du chromosome 28 complet des données bovines (46 mégabases de long, 3 assemblages), il faut 23 minutes (1 thread à 4.4Ghz, pic mémoire à 800Mo) pour reconstruire les 3 génomes linéaires que le graphe comporte.

3.3.2 Positions dans le graphe

Cette fonctionnalité permet de lier un chemin dans le graphe et l'intervalle génomique correspondant, en associant ses positions et sa séquence. Il permet de retrouver à partir d'un intervalle génomique la région du graphe correspondante. Sur le graphe du chromosome 28 complet des données bovines (46 mégabases de long, 3 assemblages), il faut 18 minutes (1 thread à 4.4Ghz, pic mémoire à 900Mo) pour calculer les coordonnées des 259459 nœuds du graphe, selon les trois chemins, et les insérer dans le fichier GFA.

3.3.3 Extraction de sous-graphes

Grâce à cet outil, on peut extraire par position dans les génomes ou par nombre de nœuds voisins une portion d'un graphe de pangénome. On peut perdre une partie des nœuds lors de l'usage de ces outils : si des segments sont des variations par rapport à

ce que l'on extrait, mais que notre fenêtre d'extraction n'est pas suffisamment grande pour englober les points où cette variation forment son embranchement [m], alors l'intégralité du chemin emprunté par la variation sera perdu. Il convient donc d'observer le graphe — visuellement, ou *via* d'autres outils — avant d'effectuer l'extraction d'une portion du graphe, pour s'assurer de récupérer tous les segments entre une *source* et un *puits*.

3.4 Comparaison de graphes

Les observations ci-dessus nous ont conduits à aborder le problème suivant : étant donné deux graphes de variations construits à partir du même jeu de génomes, comment comparer les deux graphes pour identifier les parties communes et celles qui diffèrent ? La différence entre ces deux graphes pouvant provenir de la méthode de construction (logiciels différents qui ont été utilisés), ou que l'un ait été obtenu comme un sous-graphe d'un graphe contenant plus de génomes, ou bien encore que l'un des deux ait été construit localement seulement, c'est à dire en se limitant à une région chromosomique. Notre approche de comparaison de graphe, décrite en section 2.3 permet de proposer une distance d'édition sous deux formes : celle d'un nombre de relations, et celle d'un nombre d'opérations élémentaires. Cela nous permet d'identifier les parties communes et les parties divergentes en nous limitant ici aux graphes complets de pangénomes. Nous avons appliqué cet algorithme aux données bovines, et la Table 2 présente une synthèse des relations et opérations élémentaires entre graphes comparés.

		toy_example[1]	chr_28_bovin[2]
Equivalence	String (STR)	0	508145
Inclusion	SubString (SUR)	4	1999
	SuperString (SPR)	0	851
Décalage	SubPrefix (SUP)	0	670
	SuperPrefix (SPP)	2	653
	SubSuffix (SUS)	2	554
	SuperSuffix (SPS)	0	769
	OverlapSuffixPrefix (OSP)	4	538
	OverlapPrefixSuffix (OPS)	2	654
Opérations élémentaires	Fusions	8	3207
	Divisions	4	2158
	Temps de calcul (h)[3]	-	7.51

Table 2 – Comparaison des nombres de relations et d'éditions nécessaires sur différents graphes. Le choix d'un graphe de référence a ici été arbitraire.
[1] Exemple présenté dans la section 2.3
[2] 46 mégabases, 259383 nœuds, 3 assemblages. Les graphes diffèrent par les paramètres de l'aligneur
[3] Tous les calculs ont été exécutés sur 8 coeurs avec 80 gigaoctets de RAM.

Les portions divergentes sont assez isolées dans le graphe. Avec l'outil *pangraphs compare*, on peut observer les zones de différences après extraction (Figure 9).

m. Si la *bubble* complète n'est pas comprise dans la fenêtre d'extraction, alors au moins une branche de la *bubble* sera perdue

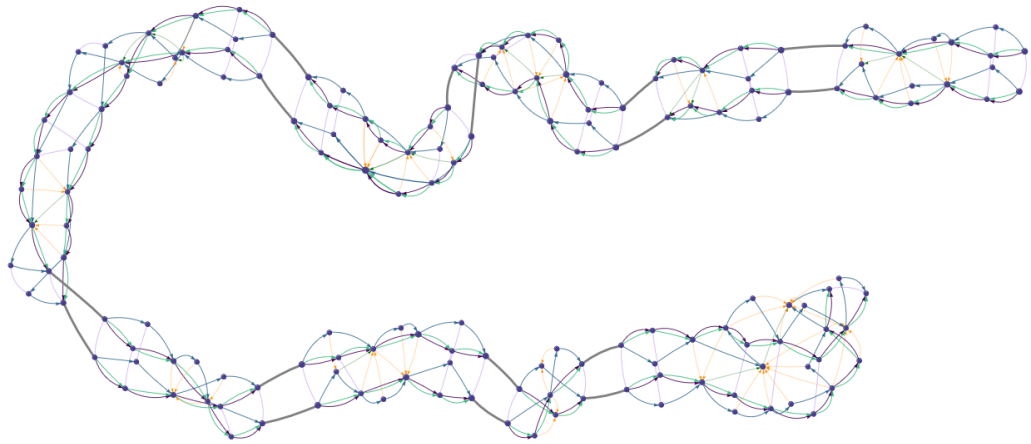


Figure 9 – Visualisation des zones de divergence sur les 2 premières mégabases des graphes du chromosome 28 des 3 assemblages bovins. Sur les 10118 liens mis en évidence, 10006 équivalences sur 10045 ont été substituées par les épaisses arêtes grises présentes entre les zones de divergence, se trouvant hors de zones d'édition.

3.5 Disponibilité des outils développés

L'ensemble des outils est implémenté sous la forme de deux packages Python :

— La librairie, *gfagraphs*, dédiée à la lecture, écriture et édition de graphes, disponible sur PyPi : `https://pypi.org/project/gfagraphs`

— Le programme en ligne de commande, *pangraphs*, rassemblant les outils de manipulation et visualisation de graphes, disponible sur GitHub : `https://github.com/Tharos-ux/pangraphs`

4 Discussion

4.1 Retour d'expérience sur les outils

Ce stage a été l'occasion d'éprouver les outils de construction de graphes de variations. Mes observations en Table 1 sont en accord avec des résultats publiés précédemment[1,20] : parmi les outils, *minigraph* et *minigraph-cactus* renvoient des graphes comparables en taille de séquences mais pas en terme de nombre de nœuds. Cela est dû au fait que minigraph ne conserve que les variants de taille supérieure ou égale à 50 bases. *PGGB* reste proche de *minigraph-cactus* quant au nombre de nœuds ; en revanche, nous n'avions pas anticipé que *PGGB* ne conserve pas l'intégralité des séquences, ce qui semble presque en contradiction avec la publication de l'outil[9]. Notre hypothèse est que la dernière étape du pipeline (*smoothxg*) est responsable de cette perte de séquences, en voulant simplifier le graphe. Toutefois, il faut attendre la publication de *smoothxg* pour pouvoir tirer des conclusions.

De mon expérience, *minigraph* est l'outil le plus simple d'utilisation. Il ne faut qu'une ligne de commande, et les rapports d'erreurs sont très clairs ; c'est toutefois aux dépends de peu de paramètres utilisateur. Le pipeline *minigraph-cactus* est très

modulaire, on peut intégrer des données issues de d'autres sources (fichier d'alignements par exemple) pendant une étape, que l'on peut contrôler indépendamment du reste du pipeline. En revanche, les rapports d'erreurs sont saturés par les sorties de chaque étape du pipeline, qui en cas d'erreur renvoie un code d'erreur assez peu explicite. Il existe un ensemble de points à respecter scrupuleusement (pas d'espace dans les *headers* FASTA, formatage strict du fichier décrivant les séquences...). Enfin, *PGGB* présente les mêmes problèmes quant aux rapports d'erreurs que *minigraph-cactus*, et s'il est possible d'injecter des fichiers externes dans le pipeline, le formatage de ceux-ci doit strictement correspondre aux standards internes de *PGGB*. Enfin, cet outil présente des pics d'utilisation de mémoire très élevés (sur les quatre assemblages de maïs, un pic mémoire à 310 Go a été atteint)

4.2 Visualisation de graphes

L'outil de visualisation que je propose peut être utilisé sur n'importe quel graphe au format GFA. Toutefois, comme la visualisation est basée sur une représentation des segments sous forme de nœuds, et que les positions des nœuds sont calculées pendant l'affichage, le seul facteur limitant pour son usage est le nombre de nœuds dans le graphe. L'outil a été testé sur une machine avec 4 coeurs et 8 gigaoctets de RAM, et affiche avec succès jusqu'à 10000 nœuds simultanément. Pour des graphes plus volumineux, il sera nécessaire d'extraire un sous-graphe, ou utiliser d'autres outils de visualisation.

Il est assez peu informatif de se comparer ici aux autres méthodes existantes, puisque *VG*[8] et *ODGI*[16] renvoient des images fixes du graphe, et que *Bandage*[19] ne met pas en évidence les chemins dans ses visualisations. De plus aucun de ces outils ne permet l'affichage simultané de plusieurs graphes et des liens qu'ils partagent.

4.3 Développement logiciel

J'ai fait le choix de proposer mon implémentation en deux structures distinctes, avec un outil d'une part et une librairie d'autre part. L'objectif est de pouvoir maintenir la librairie, et optimiser ses fonctions, sans avoir à refactoriser le code de l'édition ou de la visualisation. Par exemple, la fonction de recherche de nœud actuellement implémentée dans la librairie est très lente au regard du nombre de fois où celle-ci est appelée, et doit subséquemment être améliorée, et pourra l'être sans impact sur le reste du code. Travailler avec une librairie dédiée me permet d'avoir une implémentation légère — seulement 9 classes, définies sur ≈800 lignes de code — qui soit distribuable pour être utilisée dans d'autres projets ayant besoin d'interpréter des graphes GFA et de les éditer, et qui ne nécessite qu'une unique dépendance : *NetworkX*[21]. D'autres librairies manipulant des fichiers GFA existent : *BubbleGun*[17] et GFApy[18], d'ailleurs elles aussi écrites en Python ainsi que *bdsg*[24] et *ODGI*[16] ayant une interface compatible Python permettent de naviguer dans de tels graphes ; l'originalité de ma proposition est qu'elle ne vise pas à offrir une nouvelle manière de parcourir le graphe, mais un moyen d'effectuer des modifications de structure sur celui-ci, tout en préservant les données qu'il contient.

D'autre part, j'ai rassemblé l'intégralité des outils que j'ai pu développer, qui exploitent des fonctionnalités de cette librairie, en un unique programme. L'idée est de

proposer à l'utilisateur des lignes de commande simples et documentées, afin d'effectuer des analyses sur ses propres graphes. Le but est d'offrir l'expérience la plus simple possible, sans avoir à manipuler ou modifier de multiples scripts. Un module répertorie l'intégralité des commandes et des arguments qui leur sont liées, et appelle les fonctions correspondantes en ayant vérifié le type et la validité des arguments.

Si la librairie et l'outil sont stables, le développement de certaines fonctionnalités n'est pas allé à son terme : j'ai commencé à implémenter un convertisseur GFA pour, depuis n'importe quel fichier GFA, obtenir un fichier de sortie dans le sous-format demandé par l'utilisateur. Toutefois, si cet outil fonctionne entre GFA1.0, GFA1.1, GFA1.2 et GFA2.0 vers n'importe quel format, il ne fonctionne pas pour du rGFA (*minigraph*) vers n'importe quel format, cela à cause de l'absence des chemins qui nécessite un réalignement pour les retrouver, et qui ne peut pas être calculé directement depuis le graphe. J'avais commencé à développer un outil pour créer un fichier VCF à partir du graphe. Le développement a été stoppé, car de tels outils existent déjà (la commande *call* de *Variation Graph toolkit*) et car le VCF est adapté à l'analyse de la variation par rapport à une référence, et non par rapport à un pangénome.

Avant la version actuelle, deux autres versions de distance ont été proposées. La première calculait une distance d'édition entre les labels à comparer. Le temps de calcul était trop long sur des graphes dépassant les quelques dizaines de nœuds, et ne donnait aucune information sur la différence de découpe des génomes entre les graphes. La seconde ne vérifiait pas l'homologie entre graphes mais utilisait déjà la position au sein des chemins comme un guide pour éviter de comparer les nœuds tous contre tous. Elle repose sur le même principe que l'algorithme détaillé en section 2.3 mais ne distinguait que trois cas : inclusions, décalages et équivalences.

4.4 Comparaison de graphes

L'algorithme d'édition rend compte, chemin par chemin, des relations entre segments des chemins homologues, et les opérations élémentaires qu'il faut effectuer pour passer d'un chemin à l'autre. A notre connaissance, cet algorithme est une contribution originale : la littérature ne référence pas l'existence d'un algorithme de comparaison de graphes pour les graphes de pangénome. Suite à ce premier travail, plusieurs questions se posent. Comment choisir une métrique pour mesurer la distance d'édition ? Peut-on améliorer la complexité de l'algorithme ?

J'ai commencé à élaborer une fonction de score pour l'édition, dont une première version a été implémentée : l'idée est d'associer à une métrique la divergence, à une échelle locale, entre les graphes. Pour le moment, la fonction de score est une fonction affine avec une pénalité différente liée aux différentes relations entre nœuds du graphe, qui peut être calculée en fonction du nombre de bases concernées ou seulement en fonction des relations (nombre de nucléotides manquants dans une relation de décalage, par exemple). Toutefois, pour que ce score soit représentatif, il faut encore trouver les bons coûts pour les différentes opérations, et cette tâche pourra faire l'objet d'un développement futur.

Suivant l'échelle à laquelle on se place (graphe ou chemin), le nombre d'éditions que l'on applique n'est pas forcément minimal. Si on considère chemin par chemin, on comptabilise et réalise bien le nombre minimal d'opérations ; en revanche, si on observe

l'édition à l'échelle du graphe, il est possible d'éviter certaines opérations (un segment qui doit être divisé en trois selon un chemin, et en deux selon un autre, avant d'être fusionné en le même segment par exemple). On pourrait modifier l'algorithme pour n'éditer qu'une seule fois les nœuds du graphe. Cela soulève deux questions : comment s'assurer d'appliquer les éditions sans collisions ou perte de fragments de séquence, et quelle information obtient-on si la distance d'édition est calculée à l'échelle du graphe plutôt qu'à l'échelle du chemin ?

Si on calcule une édition au niveau du chemin, celle-ci retranscrit la segmentation du génome au sein du graphe, tandis qu'à l'échelle du graphe celle-ci pourrait être indicatrice de la différence de segmentation des variants. Dans tous les cas, il est nécessaire de tester davantage cette méthode, et la mettre en relation avec des résultats de *variant calling* et d'*alignement* afin de pouvoir en faire un outil de contrôle qualité du graphe. On peut imaginer une extension de l'algorithme où des parties d'un des génomes ne soit pas présente dans un des deux graphes et que cela soit intégré dans le calcul de la distance d'édition, ce qui permettrait à terme de ne plus se limiter aux graphes complets de pangénome.

Dans la version actuelle de l'algorithme, on compare le graphe en prenant en compte tous les types de variations, sans regard sur leur taille. Or, si on veut se concentrer sur les grandes variations, il faudrait pouvoir faire abstraction des SNPs et INDELs. Pour cela, une perspective est de lisser le graphe, en éditant le graphe de manière à ce que tous les segments de taille inférieure à un seuil soient gommés et rattachés à d'autres segments, puis calculer la distance d'édition sur les variants restants.

Grâce au calcul d'édition entre graphes, on peut mettre en évidence des zones où l'alignement ou bien les outils ont mené à des choix différents. Cela permet de mettre en évidence les incertitudes d'un graphe, et comme on peut associer cette divergence locale à un score, on pourrait s'en servir comme d'une métrique de confiance sur les variants que l'on décrirait depuis le graphe. On pourrait aussi mettre en relation les différences observées entre les graphes et les différences entre les variants issus de ces deux graphes.

5 Conclusion

Au cours de ce stage, j'ai pu proposer un ensemble d'outils de visualisation, parcours, comparaison et de modification de graphes de variations, implémenté en une librairie et un outil, ainsi qu'une formalisation d'une distance et d'une édition entre graphes de séquences, et ai ainsi pu contribuer à la compréhension de la structure que sont les graphes de variations. Le caractère récent de ce domaine de recherche le rend très mobile : de nombreux projets sont menés, et des publications et avancées fréquentes en ressortent ; dans ce contexte, il est nécessaire de faire des choix sur les pistes de recherche à creuser, et de prendre du recul sur les outils permettant de produire les graphes. Au gré des mises à jour, les sorties de ces outils changent : il est fondamental de pouvoir proposer des algorithmes qui permettent d'évaluer les structures que l'on manipule avant de faire le choix de les utiliser.

Références

1. Liao, W.-W. *et al.* A draft human pangenome reference. en. *Nature* **617,** 312-324. issn : 1476-4687. https://www.nature.com/articles/s41586-023-05896-x (2023) (mai 2023).
2. Eisenstein, M. Every base everywhere all at once : pangenomics comes of age. en. *Nature* **616,** 618-620. https://www.nature.com/articles/d41586-023-01300-w (2023) (avr. 2023).
3. Ebert, P. *et al.* Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science* **372,** eabf7117. https://www.science.org/doi/10.1126/science.abf7117 (2023) (avr. 2021).
4. Baaijens, J. A. *et al.* Computational graph pangenomics : a tutorial on data structures and their applications. en. *Natural Computing* **21,** 81-108. issn : 1572-9796. https://doi.org/10.1007/s11047-022-09882-6 (2023) (mars 2022).
5. Hickey, G. *et al.* Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome Biology* **21,** 35. issn : 1474-760X. https://doi.org/10.1186/s13059-020-1941-7 (2023) (fév. 2020).
6. Sirén, J. *et al.* Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. eng. *Science (New York, N.Y.)* **374,** abg8871. issn : 1095-9203 (déc. 2021).
7. Abel, H. J. *et al.* Mapping and characterization of structural variation in 17,795 human genomes. eng. *Nature* **583,** 83-89. issn : 1476-4687 (juill. 2020).
8. Garrison, E. *et al.* Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology* **36,** 875-879. issn : 1087-0156. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6126949/ (2023) (oct. 2018).
9. Garrison, E. *et al. Building pangenome graphs* en. Avr. 2023. https://www.biorxiv.org/content/10.1101/2023.04.05.535718v1 (2023).
10. Li, H., Feng, X. & Chu, C. The design and construction of reference pangenome graphs with minigraph. *Genome Biology* **21,** 265. issn : 1474-760X. https://doi.org/10.1186/s13059-020-02168-z (2023) (oct. 2020).
11. Garrison, E. & Guarracino, A. Unbiased pangenome graphs. *Bioinformatics* **39,** btac743. issn : 1367-4811. https://doi.org/10.1093/bioinformatics/btac743 (2023) (jan. 2023).
12. Hickey, G. *et al. Pangenome Graph Construction from Genome Alignment with Minigraph-Cactus* en. Avr. 2023. https://www.biorxiv.org/content/10.1101/2022.10.06.511217v3 (2023).
13. Guarracino, A., Mwaniki, N., Marco-Sola, S. & Garrison, E. *wfmash : whole-chromosome pairwise alignment using the hierarchical wavefront algorithm* avr. 2023. https://zenodo.org/record/7846228 (2023).

14. Group, T. G. F. S. W. *Graphical Fragment Assembly (GFA) Format Specification* en-US. Juin 2022. http://gfa-spec.github.io/GFA-spec/GFA1.html (2023).

15. Li, H. *A proposal of the Grapical Fragment Assembly format* 2014. http://lh3.github.io/2014/07/19/a-proposal-of-the-grapical-fragment-assembly-format (2023).

16. Guarracino, A., Heumos, S., Nahnsen, S., Prins, P. & Garrison, E. ODGI : understanding pangenome graphs. *Bioinformatics* **38,** 3319-3326. issn : 1367-4803. https://doi.org/10.1093/bioinformatics/btac308 (2023) (juill. 2022).

17. Dabbaghie, F., Ebler, J. & Marschall, T. BubbleGun : enumerating bubbles and superbubbles in genome graphs. *Bioinformatics* **38,** 4217-4219. issn : 1367-4803. https://doi.org/10.1093/bioinformatics/btac448 (2023) (sept. 2022).

18. Gonnella, G. & Kurtz, S. GfaPy : a flexible and extensible software library for handling sequence graphs in Python. *Bioinformatics* **33,** 3094-3095. issn : 1367-4803. https://doi.org/10.1093/bioinformatics/btx398 (2023) (oct. 2017).

19. Wick, R. R., Schultz, M. B., Zobel, J. & Holt, K. E. Bandage : interactive visualization of de novo genome assemblies. *Bioinformatics* **31,** 3350-3352. issn : 1367-4803. https://doi.org/10.1093/bioinformatics/btv383 (2023) (oct. 2015).

20. Leonard, A. S., Crysnanto, D., Mapel, X. M., Bhati, M. & Pausch, H. *Graph construction method impacts variation representation and analyses in a bovine super-pangenome* en. Sept. 2022. https://www.biorxiv.org/content/10.1101/2022.09.17.508368v1 (2023).

21. Hagberg, A., Schult, D. & Swart, P. *Proceedings of the Python in Science Conference (SciPy) : Exploring Network Structure, Dynamics, and Function using NetworkX* https://conference.scipy.org/proceedings/SciPy2008/paper_2/ (2023).

22. Riesen, K. & Bunke, H. Approximate graph edit distance computation by means of bipartite graph matching. en. *Image and Vision Computing. 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007)* **27,** 950-959. issn : 0262-8856. https://www.sciencedirect.com/science/article/pii/S026288560800084X (2023) (juin 2009).

23. Conte, D., Foggia, P., Sansone, C. & Vento, M. Thirty Years Of Graph Matching In Pattern Recognition. *IJPRAI* **18,** 265-298 (mai 2004).

24. Eizenga, J. M. *et al.* Efficient dynamic variation graphs. *Bioinformatics* **36,** 5139-5144. issn : 1367-4803. https://doi.org/10.1093/bioinformatics/btaa640 (2023) (nov. 2020).

A Relations entre deux segments

String (STR)	Type	Équivalence entre deux segments
	Condition	$s_1 = s_2$ et $e_1 = e_2$
	Implique	$o_1 = o_1 + (e_1 - s_1)$ et $o_2 = o_2 + (e_2 - s_2)$
	Inverse	String
SubString (SUR)	Type	Inclusion de σ_2 dans σ_1
	Condition	$s_2 > s_1$ et $e_1 > e_2$
	Implique	$o_1 = o_1 + (e_2 - s_2)$ et $o_2 = o_2 + (e_2 - s_2)$
	Inverse	SuperString
SuperString (SPR)	Type	Inclusion de σ_1 dans σ_2
	Condition	$s_1 > s_2$ et $e_2 > e_1$
	Implique	$o_1 = o_1 + (e_1 - s_1)$ et $o_2 = o_2 + (e_1 - s_1)$
	Inverse	SubString
SubPrefix (SUP)	Type	Cas particulier d'inclusion de σ_2 dans σ_1
	Condition	$s_2 = s_1$ et $e_1 > e_2$
	Implique	$o_1 = o_1 + (e_2 - s_2)$ et $o_2 = o_2 + (e_2 - s_2)$
	Inverse	SuperPrefix
SuperPrefix (SPP)	Type	Cas particulier d'inclusion de σ_1 dans σ_2
	Condition	$s_1 = s_2$ et $e_2 > e_1$
	Implique	$o_1 = o_1 + (e_1 - s_1)$ et $o_2 = o_2 + (e_1 - s_1)$
	Inverse	SubPrefix
SubSuffix (SUS)	Type	Cas particulier d'inclusion de σ_2 dans σ_1
	Condition	$s_2 > s_1$ et $e_1 = e_2$
	Implique	$o_1 = o_1 + (e_2 - s_2)$ et $o_2 = o_2 + (e_2 - s_2)$
	Inverse	SuperSuffix
SuperSuffix (SPS)	Type	Cas particulier d'inclusion de σ_1 dans σ_2
	Condition	$s_1 > s_2$ et $e_2 = e_1$
	Implique	$o_1 = o_1 + (e_1 - s_1)$ et $o_2 = o_2 + (e_1 - s_1)$
	Inverse	SubSuffix
OverlapSuffixPrefix (OSP)	Type	Chevauchement suffixe σ_1 et préfixe σ_2
	Condition	$s_1 > s_2$ et $e_1 > e_2$
	Implique	$o_1 = o_1 + (e_2 - s_1)$ et $o_2 = o_2 + (e_2 - s_1)$
	Inverse	OverlapPrefixSuffix
OverlapPrefixSuffix (OPS)	Type	Chevauchement préfixe σ_1 et suffixe σ_2
	Condition	$s_2 > s_1$ et $e_2 > e_1$
	Implique	$o_1 = o_1 + (e_1 - s_2)$ et $o_2 = o_2 + (e_1 - s_2)$
	Inverse	OverlapSuffixPrefix

Table 3 – Relations possibles entre deux paires de points de rupture situées sur deux graphes, lues depuis G_1 vers G_2. Les indices indiquent le graphe sur lequel on se trouve. La condition décrit comment ces points de rupture doivent être, avec s position de début du segment et e position de fin du segment. L'implication décrit comment la position actuelle au sein de chacun des chemins pour chaque graphe (o_1 et o_2) doit être mise à jour. Enfin, la relation inverse décrit quelle relation si on part de G_2 pour aller vers G_1 avec le même intervalle.

B Pseudocode de la division de segment

A partir d'un segment σ appartenant au chemin ν d'un graphe G, et d'une liste $x = x_0 \cdots x_n$ de points de division (positions au sein de seq(σ)), la fonction *split_segments* gère la division de σ en $n+1$ segments. La fonction va également créer les arêtes entre les segments ainsi formés, actualiser les arêtes présentes dans le graphe, et mettre à jour les chemins présents dans le graphe. Cette fonction effectue une division élémentaire uniquement si on lui donne en entrée un seul point de rupture.

Algorithm 1 *split_segments*

Input G, σ, $x_0 \cdots x_n$
Init $s \leftarrow seq(\sigma)$
$seq(\sigma) \leftarrow s[0, x_0)$
$x \leftarrow x + |seq(\sigma)|$
Rename σ to σ^0
for i in range of $[0, n)$ **do**
    Init new segment σ^{i+1} with $seq(\sigma^{i+1}) = s[x_i, x_{i+1})$
    Init new edge between σ^i and σ^{i+1}
end for
for each path $\nu \in G$ **do**
    if $\sigma \in \nu$ **then**
        Replace σ by $\sigma^0 \cdots \sigma^n$ in ν
    end if
end for
for each σ entering edge λ **do**
    Set λ target to σ^0
end for
for each σ exiting edge λ **do**
    Set λ source to σ^n
end for
return

C Pseudocode de la fusion de segment

A partir d'une série $\sigma^0 \ \cdots \ \sigma^n$ de segments au sein d'un graphe G sur un chemin ν, la fonction *merge_segments* gère la fusion de ces segments en un unique segment σ par la concaténation des séquences des segments de la série, lus de gauche à droite, par rapport au sens de lecture des séquences (*forward* ou *reverse*). La fonction va également supprimer les arêtes entre les nœuds associés aux segments listés, actualiser les arêtes aux extrémités, et mettre à jour les chemins de G. Cette fonction effectue une fusion élémentaire uniquement si on lui donne en entrée deux segments contigus.

Algorithm 2 *merge_segments*

Input G, $\sigma^0 \ \cdots \ \sigma^n$
$seq(\sigma^0) \leftarrow \Big\|_{i=0}^{n} seq(\sigma^i)$
for i in range of $(0, n]$ **do**
    Remove segment σ^i
    Remove edge between σ^i and σ^{i+1}
end for
for each path $\nu \in G$ **do**
    if $\sigma^0 \ \cdots \ \sigma^n \in \nu$ **then**
        Replace $\sigma^0 \ \cdots \ \sigma^n$ by σ in ν
    end if
end for
for each σ^0 entering edge λ **do**
    Set λ target to σ
end for
for each σ^n exiting edge λ **do**
    Set λ source to σ
end for
Rename σ^0 in σ
return

D Pseudocode du calcul des relations

Pour calculer les relations, on dispose de quatre points de rupture (deux sur chaque chemin), dont chaque paire décrit un segment du chemin. Notons $f(s_1, s_2, e_1, e_2) = \eta$ la fonction qui à l'ensemble des quatre points associe une relation, et notons η cette relation. On notera alors $\mathit{offset}(\eta)_1^i$ le nombre de bases à ajouter à *offset* selon ν_1^i et $\mathit{offset}(\eta)_2^i$ à l'*offset* selon ν_2^i. Ces propriétés sont décrites en annexe A.

Algorithm 3 *compute_edition*

Input ν_1, ν_2 with $\nu_1 \cong \nu_2$
Init $E \leftarrow \emptyset$
▷ i and j are the curent positions (in segments) on ν_1 and ν_2
Init o_1, o_2, i, $j \leftarrow 0$
▷ We iterate while we're not at the end of ν_1
while $|\nu_1| > i$ **do**
    Init $\rho \leftarrow i$
    Init $q_1 \leftarrow \sigma_1^i$'s end offset
    Init $E(\sigma_1^x) \leftarrow \emptyset$
    while $i = \rho$ **do**
        Init $q_2 \leftarrow \sigma_2^j$'s end offset
        $\eta \leftarrow f(s_1, s_2, e_1, e_2)$
        $E(\sigma_1^i) \leftarrow E(\sigma_1^i) + \eta$
        $o_1 \leftarrow o_1 + \mathit{offset}(\eta)_1^i$
        $o_2 \leftarrow o_2 + \mathit{offset}(\eta)_2^i$
        if $o_2 \geq q_2$ **then**
            $i \leftarrow i + 1$
        end if
        if $o_1 \geq q_1$ **then**
            $j \leftarrow j + 1$
        end if
    end while
end while
return E^i

E Pseudocode de l'édition

Dans l'algorithme suivant, on note PE la structure de données qui contient tous les E^i, dictionnaires de relations de chacun des segments du chemin i. Pour chaque chemin, on initialise un graphe vide G_t dans lequel on fait l'édition qui concerne ce chemin, puis on récupère les éléments (nœuds, arêtes et chemin) que l'on a pas encore rencontrés dans le graphe de retour G_r que l'on construit. C'est l'union des chemins édités qui forme le graphe de retour.

Algorithm 4 *edit_by_paths*

Input G_2, PE
Init $G_r \leftarrow$ empty graph
for each path edition E^i in PE **do**
    Init $G_t \leftarrow$ empty graph
    Init $\phi \leftarrow 0$
    Copy all segments and edges in ν_2^i to G_t
    for each σ_1^i in E^i **do**
        Init $\tau \leftarrow |E^i(\sigma_1^i)|$
        for each edit η in $E^i(\sigma_1^i)$ **do**
            if η is a OSP, SPP or SPR **then**
                Split target segment of η in two at target position
            end if
        end for
        if $\tau > 1$ **then**
            Merge segments in path at positions ranging ϕ to $\phi + \tau$
        end if
        Rename segment as σ_1^i
        $\phi \leftarrow \phi + 1$
    end for
    Copy path ν_t^i from G_t to G_r
    for each segment σ_t in G_t **do**
        if $\sigma_t \notin G_r$ **then**
            Copy σ_t to G_r
        end if
    end for
    for each edge λ_t in G_t **do**
        if $\lambda_t \notin G_r$ **then**
            Copy λ_t to G_r
        end if
    end for
end for
return G_r

Master de Bioinformatique de l'Université de Rennes
Année universitaire 2022 - 2023
Siegfried Dubois

Résumé

Les *graphes de variations* sont des structures de données visant à exprimer l'information de plusieurs génomes choisis au sein de graphes dirigés, dont l'information en séquence se trouve fragmentée dans les nœuds, et le sens de lecture est donné par les arêtes. Ces structures, dont les premières occurrences remontent à 2018, sont construites à base d'un alignement entre de multiples assemblages ; un embranchement témoigne d'une discordance entre les génomes : une *variation*. L'étude des variations est une analyse classique en bioinformatique : celle-ci est habituellement effectuée en comparant une séquence dite de *référence* à une autre séquence linéaire, et non en parcourant un graphe. Le caractère récent de ces structures pose un double problème : il y a un manque d'outils pour analyser les graphes de variations, et une nécessité de prise de recul par rapport aux outils permettant de les créer. A partir de données séquencées et assemblées, nous avons construit des graphes de variations avec différentes outils. Cela nous a permis de montrer que suivant les outils de construction, les graphes retournés pour un même jeu de génomes étaient différents. Nous proposons en réponse un algorithme et une implémentation d'une méthode de comparaison de graphes de variations, permettant de quantifier, qualifier et visualiser localement les différences entre graphes, et avons appliqué ces méthodes sur des graphes de variations construits avec différents paramètres, mettant ainsi en évidence des zones restreintes de divergence entre graphes.

Mots-clés — Pangénomique · Graphes de variations · Distance d'édition · Visualisation · Algorithmique.

Variation graphs are data structures which seeks to capture multiple selected genomes' information within a directed graph, where the sequences are splitted into nodes, and the reading direction is given by the edges. Those structures, which first occurences dates back to 2018, are built from alignment between multiple assemblies; a branch testifies some sort of discordance between genomes: namely, a *variation*. Variation study is a common analysis in bioinformatics: usually done by comparing a sequence identified as *reference* to another linear sequence, and not by browsing through a graph. The novelty of these structures places two problems: there is a lack of tools to perform variation graph analysis, and a need to step back from tools that can build them. From sequenced and assembled data, we build variation graphs with different tools. This allows us to depict that, given the construction tool and from a same set of genomes, graphs are different. As a partial answer, we propose an algorithm and an implementation of a variation graph comparison method which allows for the quantification, qualification and visualisation of local differences between graphs, and we applied these methods on variation graphs built with different parameters, pinpointing divergence between graphs in small areas.

Keywords — Pangenomics · Variation graphs · Edit distance · Visualisation · Algorithmics.

Code source — Le code est disponible sous licence MIT aux adresses `https://pypi.org/project/gfagraphs` et `https://github.com/Tharos-ux/pangraphs`