Université Montpellier 2

Sciences et Techniques du Languedoc

MÉMOIRE DU STAGE DE RECHERCHE

Master 2 STIC pour la Santé

Spécialité : **Bioinformatique, Connaissances, Données**

Vers la reconstruction de l'histoire de recombinaison modulaire chez les bactériophages : alignement des génomes

---

Étudiante :
Amal **ZINE EL AABIDINE**

Encadrants :
MCF. Sèverine **BÉRARD**
MCF. Annie **CHATEAU**
CR. Krister **SWENSON**

Tuteur pédagogique :
MCF. Alban **MANCHERON**

STAGE RÉALISÉ AU LIRMM
DU 18/02/2014 AU 18/08/2014

Table des matières

Liste des Algorithmes

Table des figures

Liste des tableaux

Liste des codes

1 Remerciements

❀ ❀

Pour réaliser ce document et le travail qu'il présente, j'ai largement bénéficié de l'aide de nombreuses personnes que je tiens à remercier très sincèrement.

J'exprime toute ma reconnaissance à mes deux directrices Sèverine BÉRARD et Annie CHATEAU qui ont dirigé ce travail et qui se sont toujours souciées de m'offrir, de tout point de vue, les meilleures conditions de travail possibles. Je les remercie pour tout ce qu'elles m'ont apporté, pour leurs conseils, leur présence, leur patience, pour m'avoir fait confiance et m'avoir laissé la liberté nécessaire à l'accomplissement de mes travaux, tout en y gardant un œil critique et avisé. Je les remercie également pour les nombreuses relectures de ce rapport.

Je remercie également Krister SWENSON, co-directeur du stage ainsi que Anne BERGERON pour leurs nombreux conseils ainsi que Laurent BREHELIN le chef de l'équipe MAB du LIRMM pour m'avoir acceptée au sein de l'équipe.

Mes remerciements vont également à Alban MANCHERON pour avoir accepté d'être le rapporteur de ce travail. Je le remercie aussi en tant que responsable de Master II BCD.

Merci à Julie, Bastien, Sophie, Raph, Aravind, Manu, Maxime … etc. Merci à tous les permanents, stagiaires et thésards qui m'ont accompagnée pendant ce stage.

Travailler au LIRMM au sein de l'équipe MAB a été très agréable et enrichissant et je tiens à remercier tous ceux qui ont contribué à créer cette atmosphère.

Finalement, un grand Merci chaleureux et de tout mon cœur à mon mari sans qui je n'aurais pas pu entreprendre cette formation. Je le remercie sincèrement pour sa présence, son soutien inconditionnel et constant, pour m'avoir donné du courage et de l'espoir, pour être toujours présent.

Merci à mon fils RYAN qui a subi mon indisponibilité et mon stress tout au long de ces deux années du master BCD.

❀ ❀

RÉSUMÉ/ABSTRACT

2 ABSTRACT/RÉSUMÉ

Abstract

Tracing the evolutionary history of species through algorithms that are based on molecular similarities / genetic distance as phylogenetic approach are frequent. However, they may be unsuitable for the reconstruction of the evolutionary history of bacteriophages. Indeed, they do not take into account the molecular recombination which is much more complex than punctual mutations. A single approach has been recently proposed by Swenson and colleagues [15] to integrate this evolutionary mechanism in the inference of the evolutionary history of phages. This approach takes place in several steps of which the first is the alignment and detection modules. The purpose of this training course is to contribute to the development of a new method for detecting genomic modules based on the results of alignments. For this, we proposed a first approach which was suboptimal and then a second approach more suitable. The algorithm of the retained approach consists of five main stages and has a total complexity of $O(n^4)$. We tested this method on both small simulated data sets and three real data sets of larger sequences containing phages of Staphyloccocus aureus and Lactobacillus lactis. The results were inconclusive, however, improvements are still needed.
***Keywords** : alignements, rearrangements, theory of the graphs, Text Algorithms.*

Résumé

Retracer l'histoire évolutive des espèces à travers des algorithmes qui s'appuient sur les similarités moléculaires/distance génétique est assez fréquente comme approche phylogénétique. Toutefois, ils peuvent être inadaptés pour la reconstruction de l'histoire évolutive des bactériophages. En effet, ils ne prennent pas en compte la recombinaison moléculaire qui est beaucoup plus complexe que les mutations ponctuelles. Une seule et unique approche a été proposé récemment par Swenson et ses collaborateurs [15] pour intégrer ce mécanisme évolutif dans l'inférence de l'histoire évolutive des phages. Cette approche se déroule en plusieurs étapes dont la première est l'alignement et la détection des modules. La finalité de ce stage est de participer au développement d'une nouvelle méthode de détection des modules génomiques en se basant sur les résultats des alignements. Pour cela, on a proposé une première approche qui s'est révélée non optimale puis une deuxième approche plus adaptée. L'algorithme de cette dernière se compose de 5 grandes étapes et a une complexité totale de l'ordre de $O(n^4)$. Nous avons testé cette méthode à la fois sur des petits jeux de données simulées et sur 3 jeux de données réels de taille plus importante contenant des séquences des phages de Staphyloccocus aureus et Lactobacillus lactis. Les résultats étaient concluants toutefois des améliorations restent à faire.
***Mots-clés** : alignements, réarrangements, théorie des graphes, algorithmique du texte.*

I INTRODUCTION

3 Introduction

3.1 Contexte du stage et problématique

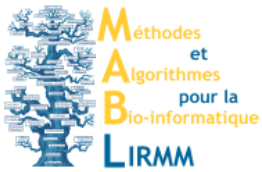
Du 18 février 2014 au 18 août 2014, j'ai effectué mon stage de Master 2 BCD [1] au sein du LIRMM [2] à Montpellier. Ce stage s'inscrit dans le cadre d'une collaboration entre le LIRMM et l'ISEM [3]. L'objectif général de cette collaboration est de mettre au point une approche de phylogénie permettant de mieux comprendre l'histoire évolutive des bactériophages (appelés aussi phages) en analysant les recombinaisons modulaires comme événement évolutif. Le mode de recombinaison phagique a fait l'objet de très nombreuses discussions et des questions autour son mécanisme demeurent encore matière à débat. La théorie de la recombinaison modulaire introduite pour la première fois par Botstein en 1980 [9] est aujourd'hui soutenue par un faisceau d'arguments issus de différentes études. Cette théorie postule que les phages sont des assemblages de modules (on parle d'une structure en mosaïque) codant pour des fonctions biologiques et que deux modules codant la même fonction ne sont pas forcément similaires de point de vue nucléotidiques [9]. Par conséquent, la recombinaison modulaire se définit comme l'échange de modules partageant une même fonction. Ce processus de recombinaison est une stratégie d'adaptation fréquemment observée chez les phages. En effet, la comparaison de génomes phagiques révèle la présence d'alternances de séquences similaires et de séquences divergentes. Cette particularité d'organisation en mosaïque rend inadaptés les outils classiques d'inférence de phylogénie. À titre d'exemple, dans le cas des phages, deux génomes sont dits colinéaires si les modules exécutant la même fonction y apparaissent dans le même ordre. Par conséquent, l'alignement n'est plus un alignement de nucléotides mais un alignement de modules.

Depuis presque un siècle, d'innombrables études leur ont été consacrées mettant en œuvre une large et diverse panoplie de techniques d'analyse. Toutefois, à l'heure actuelle, aucune seule étude s'est penchée sur l'analyse des recombinaisons modulaires pour reconstruire l'histoire évolutive des phages, pour les prendre en compte dans les méthodes de reconstruction des génomes ancestraux.

Ce stage fut l'occasion de mettre en application les nombreuses connaissances enseignées en Master BCD, et plus particulièrement la théorie des graphes et l'analyse des alignements de séquences.

Dans cette section, nous allons présenter les différents partenaires de ce stage : l'équipe MAB [4] du LIRMM et l'équipe Phylogénie et Évolution Moléculaire de l'ISEM. Aussi, nous détaillerons les objectifs de ce stage ainsi que les technologies qui ont été utilisées pour les atteindre.

3.2 L'équipe MAB : Méthodes et Algorithmes pour la Bio-informatique



L'équipe MAB est formée de 20 chercheurs et ingénieurs dont les travaux s'articule autour des questions méthodologiques comme l'algorithmique du texte et des arbres et la modélisation probabiliste dans le souci de répondre à des questions de recherches touchant notamment les domaines de l'évolution,de la génomique comparative, de l'annotation fonctionnelle des gènes et des protéines (source : http://www.lirmm.fr /recherche/equipes/mab).

Les travaux de l'équipe s'intéressent à plusieurs axes de recherches notamment :

- ❒ Algorithmique et combinatoire ;
- ❒ Modélisation probabiliste et statistique ;

1. Bioinformatique, Connaissance, et Données.
2. Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, www.lirmm.fr.
3. Institut des Sciences de l'Évolution de Montpellier, www.isem.fr
4. Méthodes et algorithmes pour la bioinformatique.

- ❒ Analyses de séquences haut-débit ;
- ❒ Phylogénomique ;
- ❒ Annotation des protéomes ;

3.3 L'équipe de Phylogénie et Évolution Moléculaire de l'ISEM

Les travaux de recherches ainsi que les enseignements de l'équipe Phylogénie et Évolution Moléculaire s'articulent autour de 3 principaux axes à savoir (source : http://www.isem.univ-montp2.fr/recherche/equipes/phylogenie-et-evolution-moleculaire/presentation/) :

- ❒ Retracer l'histoire évolutive des mammifères en faisant appel à des méthodes approches probabilistes.
- ❒ Mettre au point des méthodes et des les outils bioinformatiques adéquats. En effet, l'équipe a déjà réussi à créer des bases de données des codes, des logiciels ainsi que des serveurs web p destinés pour l'analyse de données NGS, l'alignement de séquences codantes, la reconstruction d'arbres et super-arbres ainsi que l'exploration des collections de phylogénies.
- ❒ Étudier les "déterminants" de l'évolution moléculaire au niveau des nucléotides et des acides aminées en s'appuyant sur la génomique des populations.

3.4 Objectifs du stage

La principale finalité de ce stage, orienté développement, est de mettre en place une nouvelle méthode permettant de représenter les alignements du BLAST dans un graphe permettant la détection et l'alignement des modules de recombinaison. Pour cela, il était capital d'assimiler et de comprendre les notions de base de la théorie des graphes pour les intégrer dans notre approche de façon optimale. De manière plus détaillée, la mise au point d'une telle approche s'est déroulée en 3 trois grandes étapes :

- ❒ L'analyse des particularités du BLAST.
- ❒ L'analyse des arrangements génomiques.
- ❒ La caractérisation du graphe obtenu.

Durant ce stage, nous nous appuierons d'une part sur des méthodes et des algorithmes fondés sur des principes de la théorie des graphes et d'autre part sur des jeux de données réelles (publiques) et simulées pour juger la pertinence, l'exactitude de notre approche voire la fignoler.

3.5 Technologies utilisées

3.5.1 Python

Les scripts codés lors de ce stage ont été réalisés en utilisant le langage python. Ce langage de programmation est doté d'une syntaxe simple, relativement intuitive et adapté pour des scripts, des petits ou gros projets.

NetworkX

3.5.2 NETWORKX

Networkx [5] est une bibliothèque Python pour l'étude des graphes et des réseaux. C'est un logiciel libre dont la dernière version date du 21 juin 2014 (v 1.9.1). Networkx fournit des structures de données pour les graphiques ainsi que des algorithmes de graphes et des outils de dessin. En effet, le paquet fournit des classes pour les objets graphiques, des générateurs pour créer des graphiques standard, des routines pour avoir l'accès à l'ensemble des données existantes, des algorithmes pour analyser les réseaux résultant et des outils de dessin de base.

Networkx utilise un "dictionnaire de dictionnaires de dictionnaires" comme structure de données de base. Cela permet un accès facile et plus ou moins rapide au graphe sachant que les sommets de G définissent les clés du dictionnaire. Ainsi l'expression G[sommet] retourne un dictionnaire d'adjacence dont les clés sont les voisins directs du sommet dans G et les valeurs sont un dictionnaire des attributs des arcs. L'expression G[sommet1][sommet2] renvoie ainsi le dictionnaire des attributs de l'arc entre sommet1 et sommet2.

La plupart des API de Networkx est assurée par des fonctions qui prennent comme un argument un objet graphique. Les méthodes de l'objet graphique sont limitées à la manipulation de base (graph.nodes(), graph.edges(), graph.successors() ... etc). En effet, Networkx permet entre autres de :

- Générer des classes pour les graphes simples et les graphes orientés.
- Convertir des graphes depuis et vers divers formats.
- Construire des graphes aléatoires ou les construire progressivement.
- Trouver des sous-graphes, cliques, graphe de dégénérescence k.
- Dessiner des réseaux en 2D et en 3D.

3.5.3 GRAPHVIZ

GraphViz (diminutif de Graph Visualization Software) est un ensemble d'outils open source créés par les laboratoires de recherche d'AT&T [6] qui manipulent des graphes définis à l'aide de scripts suivant le langage DOT [7]. Cet ensemble fournit aussi des bibliothèques permettant l'intégration de ces outils dans diverses applications logicielles. Networkx utilise GraphViz pour la génération des graphes.

3.6 ORGANISATION DU STAGE

Cette section présente le planning des tâches effectuées le long de ce stage. La figure 1 montre le diagramme de Gantt (réalisé avec le logiciel GanttProject) qui décrit la division des tâches liées aux différents axes de mon projet.

Description des couleurs :

- ❒ *Gris* : La période du stage.
- ❒ *Vert* : Les phases de conceptions et de définitions de méthodes.
- ❒ *Rouge* : Les phases d'implémentation.
- ❒ *Jaune* : Les phases d'expérimentation.
- ❒ *Violet* : Les phases de caractérisation et de preuve.
- ❒ *Bleu* : Les phases de rédaction.

5. http ://networkx.lanl.gov/.
6. American Telephone Telegraph.
7. Langage de description de graphe dans un format texte.

Il est à noter que des réunions hebdomadaires (tous les mardis) avec les différents encadrants du stage ont été organisées tout au long de la période du stage. Ces réunions avec les tuteurs du stage ont permis d'organiser efficacement le travail en fonction des objectifs fixés. La qualité de l'encadrement a été l'un des piliers de ce travail.

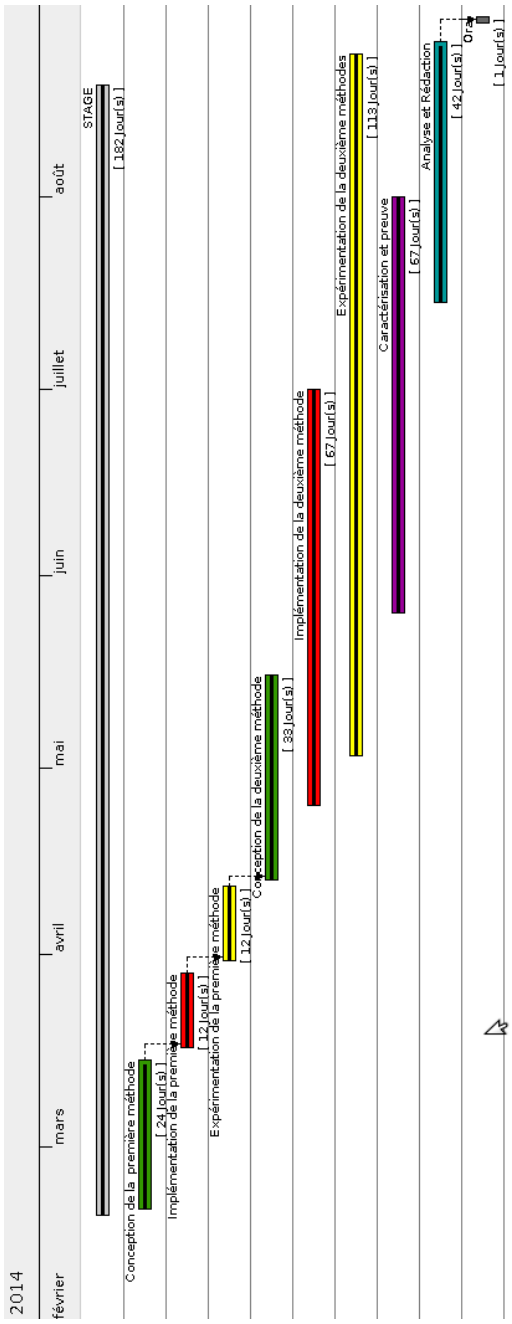


***Fig 1:** Diagramme de Gantt.*

3.7 LIVRABLES

Ci-dessous le recensement des différents éléments produits à l'issu de ce stage :

- ❒ Le présent rapport bibliographique : 28/08/2014.
- ❒ Les différents scripts python codés lors du stage : 28/08/2014.
- ❒ La documentation des codes : 28/08/2014.
- ❒ Les données des séquences utilisées dans les expérimentations ainsi que les résultats : 28/08/2014.
- ❒ La présentation du stage : 28/08/2014.

Pour plus de détails, se référer à la section "description de l'archive 9.6" décrivant le contenu et l'organisation de l'archive rendue.

3.8 ORGANISATION DU RAPPORT

Le rapport est divisé en 8 parties principales. La première partie "état de l'art" permet de définir le cadre biologique et bioinformatique de cette étude (*cf.* section 4). La formalisation de l'ensemble des notions utilisées et qui sont nécessaires à la compréhension de l'algorithme est abordé dans la deuxième partie dite "rappels et définitions" (*cf.* section 5). La troisième partie "réalisations" est dédiée d'une part à la description des différentes étapes de notre méthode ainsi que leur complexité et d'autre part à la caractérisation des différents cas rencontrés et leur preuve (*cf.* section 6). Dans la quatrième partie "expérimentations", nous décrirons les résultats de l'application de notre méthode sur des données réelles afin de mesurer son efficacité (*cf.* section 7). Puis, nous faisons le point sur la pertinence de notre algorithme et les éventuelles améliorations à ajouter dans la partie "conclusions et perspectives" (*cf.* section 8). Finalement nous terminerons par "annexes" et "références bibliographiques".

ÉTAT DE L'ART

4 ÉTAT DE L'ART

4.1 LES BACTÉRIOPHAGES

4.1.1 HISTOIRE DES BACTÉRIOPHAGES

La découverte des bactériophages remonte à presque un siècle quand deux chercheurs se disputaient leur paternité : l'Anglais Frederick Twort (1915) [23] et le canadien Félix d'Hérelle (1917) [12] [11]. Récemment, un article a été publié [2] qui met la lumière sur des articles travaillant sur les bactériophages publiés à des dates antérieures à 1915 (entre 1896 et 1915) dont le plus vieux était celui de Hankin (1996). Les premières recherches sur le phage tentaient de définir la nature des bactériophages. Deux principales théories étaient émises : soit c'était un virus filtrable, soit c'est une enzyme qui s'autoperpétue et dont l'expression cause la destruction de la cellule bactérienne. Quelle que soit la nature exacte du bactériophage, il a été rapidement réalisé que les bactériophages ont le potentiel de tuer les bactéries qui causent de nombreuses maladies infectieuses chez l'homme ainsi que chez les plantes et les animaux. Cette idée est à la base de beaucoup de recherches. Félix d'Hérelle s'intéressait en particulier au potentiel de l'utilisation thérapeutique des phages.

En 1933, d'Hérelle a co-fondé un institut de recherche sur les phages dans la République soviétique de Géorgie avec le microbiologiste géorgien George Eliava. Toutefois, la recherche sur la « thérapie par les phages » a été supplantée par la pénicilline et d'autres antibiotiques découverts à partir des années 1940, mais il y a un regain d'intérêt pour la phagothérapie au cours des dernières années étant donné que la résistance des bactéries aux antibiotiques est devenue une menace pour la santé publique. Pendant ce temps, la recherche sur les bactériophages s'est poursuivie. La nature virale du bactériophage a été clairement établie. Aussi, la composition chimique des virions a été mesurée, et ils se sont révélés être un mélange de protéine et d'ADN. Les premières photographies au microscope électronique de phages, montrant une forme de têtard, ont été obtenues en 1942 par Tom Anderson. Dans les années 1950 et 1960, la recherche sur les phages a joué un rôle dominant dans l'élucidation des notions fondamentales sur les gènes et le mécanisme de leur transcription et leur traduction. Autour des années 70, la recherche biologique a connu une grande révolution «la révolution de l'ADN recombinant» avec laquelle il est devenu possible de modifier efficacement le gène de n'importe quel organisme. La révolution de l'ADN recombinant a produit des changements profonds dans la recherche sur les phages, comme dans tous les autres domaines de la recherche biologique.

4.1.2 GÉNÉRALITÉS SUR LES BACTÉRIOPHAGES

Les bactériophages ou les phages [8] sont des virus dont les hôtes sont des cellules bactériennes. L'organisme responsable de la nomenclature et de la taxonomie des virus est le ICTV [9]. Près de 21 morphologies différentes chez les virus bactériens sont reconnues par l'ICTV. Les bactériophages se classent en 8 ordres taxonomiques à savoir les *Caudovirales* (3 familles), les *Herpesvirales* (3 familles), les *Ligamenvirales* (2 familles), les *Mononegavirales* (4 familles), les *Nidovirales* (4 familles), les *Picornavirales* (5 familles), les *Tymovirales* (4 familles) et les 'non assignés'(71 familles). La plupart des phages (96%) ont une morphologie dite à queue, le reste possède une morphologie soit en polyèdre, filamenteuse ou pléomorphe. Le génome des phages peut être simple brin (ss) ou double brin (ds), linéaire ou circulaire, désoxyribonucléique (ADN) ou ribonucléique (ARN) (figure 2). Une exception est le génome des cystovirus, qui se compose de trois molécules d'ARN. La structure typique d'un phage est une tête creuse contenant l'ADN de phage ou de l'ARN et une queue en forme de tunnel servant à injecter le matériel génétique dans les bactéries. Cependant, d'autres formes morphologiques de phage existent (figure 2).

À l'image des virus qui infectent les eucaryotes, les phages sont constitués d'une enveloppe protéique externe (dite capside) protégeant le matériel génétique. Pour plus de 95% des phages connus, ce matériel est une molécule d'ADN double brin dont la longueur du génome varie de 5 000 à 650 000 pb et dont la taille fluctue de 24 à 200 nm (figure 3).

8. *Phagen* en grec signifie nourriture.
9. International Commitee on Taxonomy of Viruses.

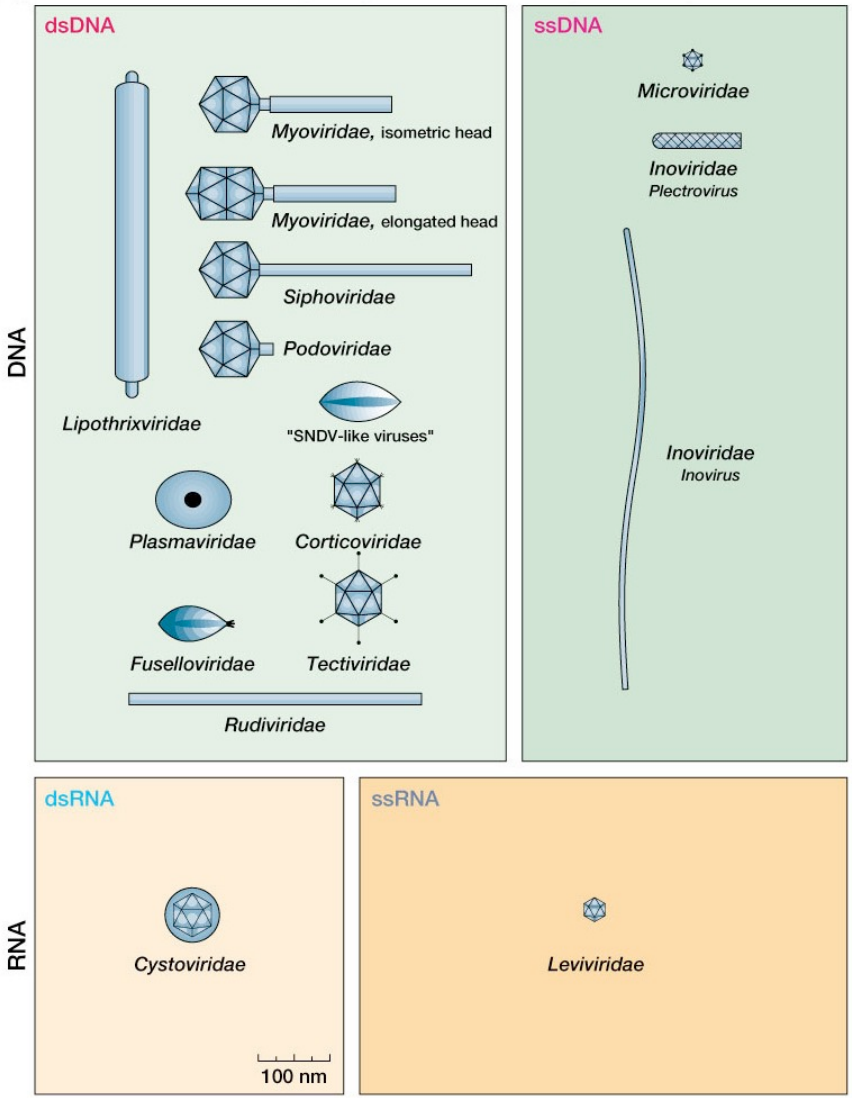


Fig 2: *Classification des phages selon leur morphologie et la nature de leur matériel génétique (source : 210.44.48.210/courseware/05_06/dongwubingduxue/chptereng/2/2.1.htm)*

En raison de leur taille relativement petite et de la simplicité de leur isolement, les bactériophages ont été les premiers génomes complets séquencés, en commençant par les 5 386 bp de l'ADN simple brin du phage *varphi*X174 en 1977. La première séquence complète d'un phage d'ADN double brin (48 502 bp) a été celle du phage λ, et la séquence du phage T7 (39 936 pb) a été obtenue peu de temps après. À l'heure actuelle, selon la base de données des phages [10], près de 577 phages ont été séquencés de façon complète sur un total de 4 111 génomes déposés. Les trois derniers phages séquencés complètement sont OkiRoe (62661bp, 14/11/13), Empty (68 428bp, 13/11/13) et Emerson (60 310 bp, 11/13/2013). La taille des génomes des phages séquencés varie de 41 441 bp à 164 602 avec 50% des phages qui ont un génome de l'ordre de 58 000 bp). Le pourcentage de GC des phages séquencés varie de 50 à 70% avec 50% des phages séquencés qui ont un pourcentage de GC de l'ordre de 64%. Sur un total de 577 phages séquencés, près de 84,20% ont été découverts entre janvier 2008 et novembre 2013 (figure 4).

Les bactériophages sont omniprésents. Ils ont été isolés à partir de multiples sources comme l'eau, le sol, le désert, les sources chaudes, et l'homme. En outre, les phages peuvent être trouvés comme prophages [11] insérés dans les génomes

10. http://phagesdb.org/.
11. Le prophage correspond à la forme du bactériophage lorsqu'il est inséré dans le génome de la bactérie infectée.

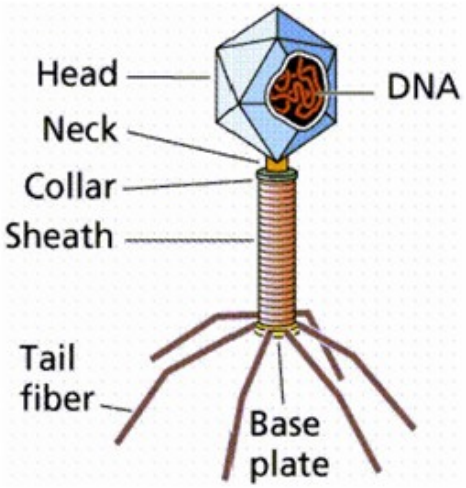


Fig 3: *Structure des virus (source http://www.e-pao.net/).*

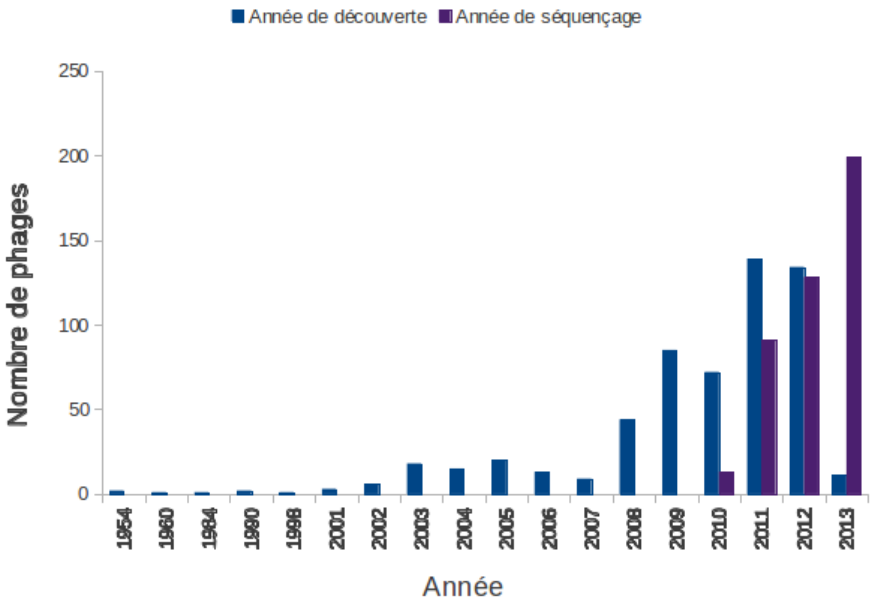


Fig 4: *Évolution du nombre de phages découverts et du nombre de génomes séquencés de manière complète à la date du 22/11/13 (d'après www.phagebd.org).*

bactériens. Probablement, il y a plus de phages dans le monde que n'importe quel autre organisme vivant. Le nombre de phages est proportionnel au nombre de bactéries hôtes présentes dans le milieu. À titre d'exemple, le nombre de phage est estimé à près de $10^6/ml$ et $10^9/ml$ phages à queue dans l'eau de mer et l'eau douce respectivement.

4.1.3 Les groupes de phages

Durant l'infection d'une bactérie par un phage, les étapes précoces sont cruciales et déterminantes en permettant d'orienter le cycle phagique en fonction de l'état physiologique de la cellule bactérienne infectée. En effet, le phage possède deux principaux cycles de vie : le ***cycle lytique*** et le ***cycle lysogène***. Les deux cycles sont initiés par la fixation du phage sur la surface bactérienne qui est habituellement espèce - voire même souche - spécifique. La fixation est suivie de l'injection du matériel génétique phagique dans le cytoplasme bactérien. Alors que tous les phages sont capables d'entrer dans le cycle lytique (phages virulents), certains phages (dits tempérés) peuvent également entrer

dans un cycle lysogène. Ceci nécessite soit l'intégration du génome du phage dans le chromosome de l'hôte bactérien ou le maintien du génome du phage en tant qu'élément extrachromosomique stable. Le cycle lysogénique est une des réponses phagiques possibles et il requiert deux événements majeurs coordonnés : l'établissement de la répression des fonctions lytiques et la recombinaison intégrative conduisant à l'insertion du génome phagique dans le chromosome bactérien. L'induction du prophage implique l'inversion de ce processus : la levée de la répression des fonctions lytiques accompagnées de la répression des fonctions lysogéniques, et la mise en place de la recombinaison excisive qui permettra l'excision de l'ADN viral du chromosome bactérien. Ces deux fonctions étant étroitement liées, les gènes impliqués dans ces fonctions sont regroupés au sein d'un module de lysogénie. Même si plusieurs facteurs ont été identifiés qui régulent ce cycle, le détail de régulation reste mal élucidé. Un phage tempéré dans cet état est appelé *prophage*, indépendamment de savoir s'il est intégré ou pas, il peut rester dans un état de latence pendant plusieurs générations bactériennes.

4.1.4 Description du cycle de vie lytique des phages

Généralement le cycle lytique des phages se déroule en 4 phases fondamentales comme l'illustre la figure 5 chez le phage T4.

❒ **Phase d'absorption :** Durant cette étape, le phage se fixe sur la bactérie grâce à une reconnaissance spécifique entre le récepteur bactérien et celui viral. Toutefois, dans certains cas, l'absorption dépend aussi des cofacteurs libérés par la bactérie qui va être parasitée (ex : T4 se fixe bien si la bactérie libère Tyr et le phage λ se fixe bien si la bactérie libère du Mg^{2+}). Le récepteur, au niveau de la bactérie, varie suivant le phage (ex : T2 = porine ; λ = porine lam B).

❒ **Phase de fixation :** Pour le phage T2, la fixation s'effectue grâce aux filaments caudaux et à la plaque terminale. Ensuite, les enzymes de la plaque caudale s'activent et hydrolysent les enveloppes bactériennes pour pouvoir injecter l'ADN viral par contraction de la gaine externe qui favorise la pénétration du cylindre central.

❒ **Phase d'éclipse :** Après injection, le virion n'existe plus seul mais intégré dans le génome bactérien. Le métabolisme bactérien est entièrement orienté vers la synthèse des constituants du phage assurant ainsi la synthèse de l'ADN viral et sa réplication. L'ADN est synthétisé sous forme de "concatémères" où plusieurs séquences d'ADN sont collées à la suite.

❒ **Phase de maturation et libération :** L'ADN synthétisé sous forme de concatémère est digéré par la suite en fragments ayant à peu près la taille du génome viral avec la présence parfois de quelques erreurs (fragments plus petits ou plus longs que le génome viral). Ces erreurs sont une source de mutation chez les phages. Par la suite, la particule virale ou le virion est assemblé et dès l'accumulation de quelques centaines de virions, la paroi bactérienne éclate suite à la production d'une enzyme dite endolysine [12].

4.2 La recombinaison génétique

La recombinaison génétique est à l'origine de l'apparition de gènes ou de caractères héréditaires dans une association différente de celles présentes dans les génomes parentaux. La recombinaison conduit à la redistribution du génome par échange de matériel génétique entre deux chromosomes voire entre deux fragments d'un même chromosome. Ainsi, la nouvelle réorganisation du contenu génétique conduit à la création de la diversité génétique. Aussi, elle permet la réparation des molécules d'ADN pour assurer la fidélité de la transmission à la descendance, assurant ainsi la survie des cellules. Les mécanismes de recombinaison génétique, essentiels à l'évolution, se classent en 3 familles en fonction du mécanisme utilisé, des facteurs et cofacteurs protéiques impliqués et de la nature de l'ADN recombiné [4] :
❶ La recombinaison homologue.
❷ La recombinaison non homologue dite aussi illégitime.
❸ La recombinaison spécifique de site.

12. L'endolysine est une enzyme des bactériophages qui lyse la paroi des bactéries pour permettre leur libération.

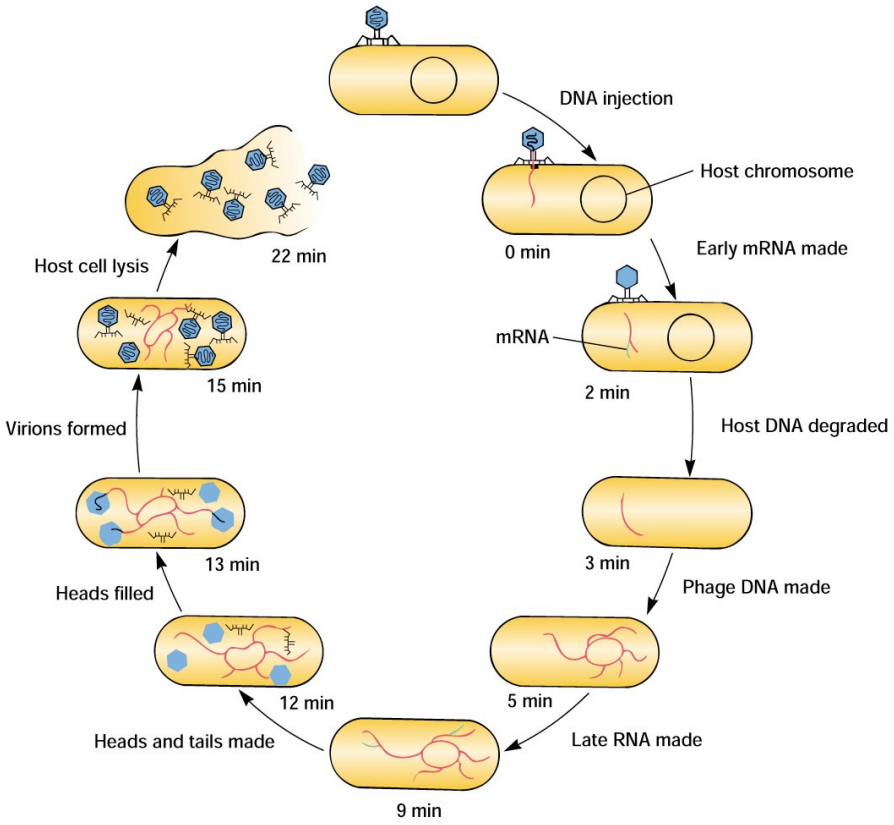


Fig 5: *Cycle de vie du phage T4 (source :http://classroom.sdmesa.edu/eschmid/Lecture9-Microbio.htm).*

La recombinaison non homologue est le processus qui génère le plus de diversité et est considéré comme un mécanisme évolutif puissant vue l'absence de similitude de séquence entre les sites échangés, d'où son caractère très aléatoire.

❶ La recombinaison homologue ou générale :
La recombinaison homologue consiste en l'échange de matériel génétique mettant en jeu des séquences d'ADN homologues sur 2 molécules d'ADN différentes, ou distantes l'une de l'autre sur la même molécule. Le processus global de cette recombinaison consiste en une cassure double brin de deux molécules d'ADN suivie d'une ligature des brins de deux hélices d'ADN avec formation de régions hétéroduplexe étendues. Un des premiers modèles expliquant les recombinaisons a été formulé par Robin Holliday [3]. Le mécanisme général de la recombinaison suivant ce modèle est détaillé dans la figure 6. Les principales caractéristiques du modèle d'Holliday sont la formation d'ADN hétéroduplexe suivie de la création d'un pont en croix (Jonction d'Holliday) puis sa migration le long des deux brins hétéroduplexes (appelée migration de branche), suivie par la réparation des mésappariements et la résolution de la jonction d'Holliday donnant lieu à différents types de molécules recombinantes et à deux sous-types de recombinaison [3] :

La recombinaison non réciproque : résulte de la cassure et la ligature (re-ligature) des deux brins qui se sont croisés. Il n'y a pas d'échange réciproque dans la mesure où un seul des deux brins de l'hélice a été modifié.

La recombinaison réciproque : résulte de la cassure et la ligature des deux brins qui ne se sont pas croisés. Il y a échange réciproque dans la mesure où les deux brins de l'hélice ont été modifiés.

❒ Le clivage enzymatique et la formation d'un ADN hétéroduplex

La première étape de la recombinaison est la cassure d'ADN au même niveau sur les deux hélices et impliquant le même brin (soit 5'$\longrightarrow$ 3' soit 3'$\longrightarrow$ 5'). Sur le schéma 6a, les deux doubles hélices homologues ont été tournées de

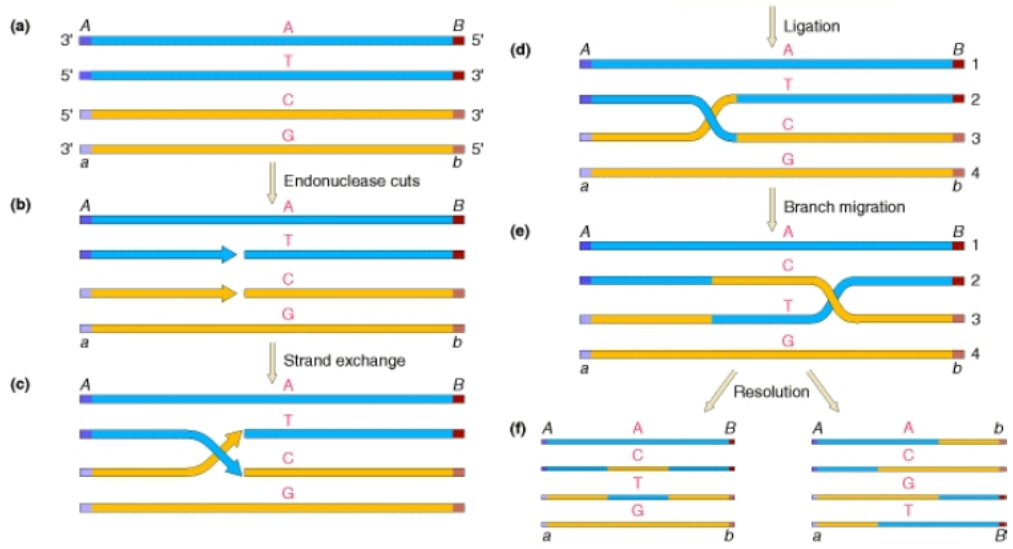


Fig 6: *Processus global de la recombinaison génétique suivant le modèle de Holliday [3].*

telle manière que le brin inférieur de la première hélice ait la même polarité que le brin supérieur de la deuxième hélice (5' ⟶ 3'). Ensuite, une nucléase clive les deux brins qui ont la même polarité (figure 6b). Les extrémités libres quittent leurs brins complémentaires d'origine pour s'apparier grâce à une liaison hydrogène avec les brins complémentaires de l'autre hélice (figure 6c). Leur ligature conduit à la formation de la structure représentée sur la figure 6d. Cette double hélice partiellement hétéroduplexe est un élément crucial intermédiaire dans la recombinaison et qu'on appelle la structure ou la jonction d'Holliday.

❒ La migration des branches

La structure d'Holliday crée un pont en croix qui peut se déplacer ou migrer le long de la région hétéroduplexe (figure 6d et e). Ce phénomène de migration de branche est une propriété distinctive de la structure d'Holliday.

❒ La résolution de la structure d'Holliday

La structure d'Holliday peut être résolue en coupant et en ligaturant soit les deux brins qui sont à l'origine de l'échange de brins (figure 6f, gauche), soit les brins non échangés (figure 6f, à droite). ce qui génère des duplex parentaux avec insertion au milieu d'un bout de séquence de l'autre parent. Si les deux parents possèdent des allèles différents au niveau de l'insertion, on a formation d'un hétéroduplex. La dernière étape de la résolution génère deux duplex qui sont recombinant, avec un brin d'ADN hétéroduplex. Le modèle d'Holliday postule que les éventuelles asymétries d'ADN hétéroduplexe peuvent être réparées par un système de correction enzymatique et excision des bases qui ne correspondent pas à l'un des deux brins. En remplissant les bases excisées, les molécules résultantes porteront soit l'allèle sauvage soit l'allèle mutant suivant quel allèle a été excisé [3].

La figure 7 illustre comment la structure d'Holliday peut être convertie dans des structures recombinantes. La figure 7a,représente la structure de la figure 6e établie dans une forme élargie. Les figures 6e et 7a représentent deux structures équivalentes. Si on fait tourner la partie inférieure de cette structure, comme représenté sur la figure 7b, on peut générer la forme représentée sur la figure 7c. Cette dernière forme peut être convertie en deux doubles hélices non connectées par clivage enzymatique seulement de deux brins. Comme indiqué dans la figure 7c, le clivage peut se produire de l'une des deux manières, dont chacune génère un produit différent (figure 7d). Ces structures clivées peuvent être consultées plus simplement (Figure 7e). La synthèse de réparation produit les molécules recombinantes finales (figure 7f).

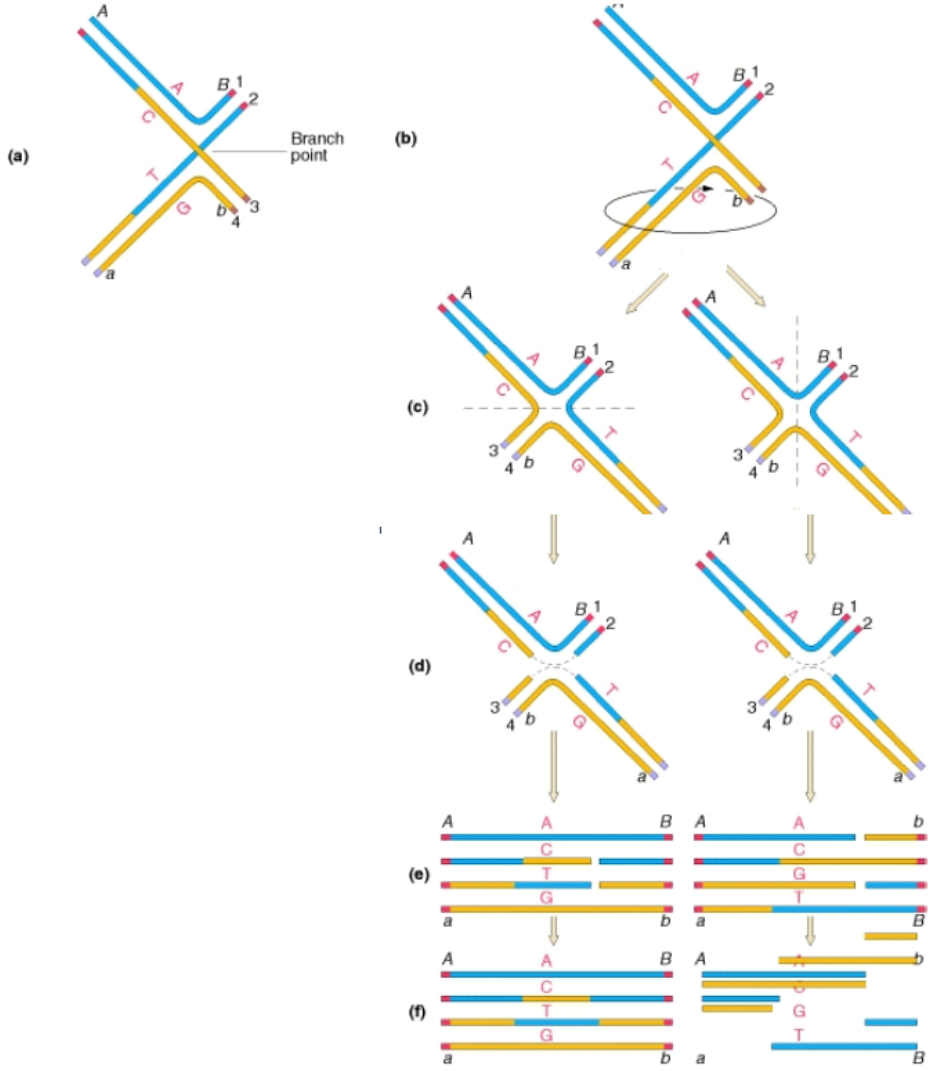


Fig 7: *La structure de Holliday représentée de façon étendue [3].*

❷ La recombinaison non homologue ou illégitimes :
Elle a lieu entre des séquences ne présentant pas ou très peu d'homologie de séquence, et qui conduit à leur appariement. C'est un moteur évolutif important mais qui est aussi impliqué dans les intégrations aléatoires de virus ou de plasmides, l'apparition de délétion ou des duplications dans le génome qui peuvent conduire à certaines maladies génétiques graves.

❸ La recombinaison site-spécifique :
La recombinaison spécifique de site, qualifiée de spécialisée, consiste en l'échange de matériel génétique impliquant de courtes séquences spécifiques présentant des homologies courtes et portées par les deux molécules d'ADN partenaires (site de recombinaison), et reconnues par des recombinases capables de les rapprocher, les couper, les échanger et les réassocier dans une nouvelle combinaison. Ces enzymes agissent de façon unique sur une paire de séquences cibles. Ce processus est largement rencontré lors de l'intégration de génome viral dans un chromosome cellulaire (figure 8), lors de la transposition [17].

Selon l'arrangement initial des deux sites de recombinaison, la recombinaison peut générer trois types d'événements (figure 8) à savoir [20] :

❒ **L'intégration** : résulte de la recombinaison entre deux sites codés par deux séquences d'ADN différentes et

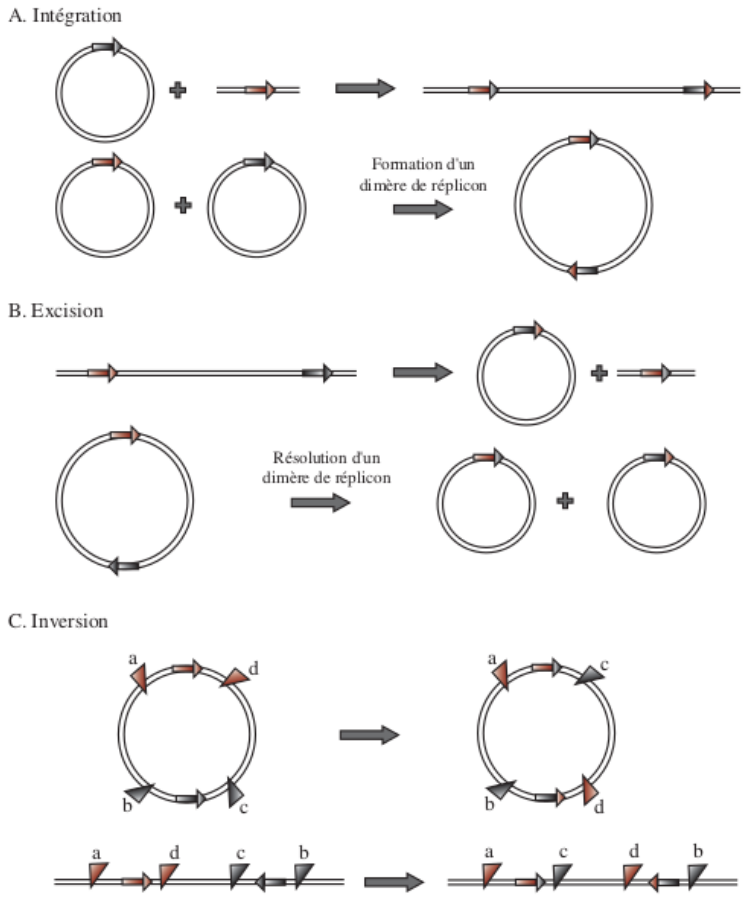


Fig 8: *La recombinaison spécifique de site engendre trois types d'événements différents [20].*

induisant leur fusion dans une orientation prédéfinie.

❒ **L'excision** : résulte de la recombinaison entre deux sites ayant la même orientation et codées par la même molécule d'ADN et conduit à la formation de deux molécules d'ADN à partir d'une seule au départ.

❒ **L'inversion** : résulte de la recombinaison entre deux sites ayant des orientations contraires et codées par la même molécule d'ADN et conduit à l'inversion d'orientation d'un des deux fragments d'ADN.

4.3 La recombinaison modulaire et la reconstruction de l'évolution des phages

4.3.1 La théorie de l'évolution modulaire

Les bactériophages, caractérisés par un génome organisé en mosaïque, ont la capacité de partager entre eux des blocs génomiques. Les phages utilisent souvent la recombinaison comme un processus d'invention radicale et évoluent grâce au transfert horizontal des *modules*. Les modules n'ont potentiellement aucune similitude détectable mais ont la même fonction. Ce phénomène a été observé il y a une trentaine d'années et a été reconnu sous le nom de la *théorie modulaire* [9]. Les mécanismes de la recombinaison modulaire sont encore débattus. Le groupe taxononomique le mieux caractérisé de bactériophages en matière de transfert horizontal de gènes est celui des phages lambdoïdes. Ces fameuses structures en mosaïques ont également été observées chez les phages T4-like [8]. Toutefois, la taille des segments d'ADN modulaire échangés semble être plus petite et le nombre d'allèles différents inférieurs à ceux de phages lambdoïdes, ce qui suggère que la diversité des phages T4-like a été provoquée plutôt par des mutations ponctuelles et des événements de duplication génique occasionnels que par des échanges modulaires [6, 8]. De la même manière, Krupovic et al [16] affirment que les nouvelles architectures phagiques sont choisies parmi un grand nombre de ruptures et de réparations aléatoires. Martinsohn et al [14] suggèrent que l'échange de séquences divergentes entre deux phages fait appel à des séquences homologues flanquantes comme c'est illustré dans la figure 9. Les auteurs donnent

des preuves convaincantes de l'apparition de ce type de recombinaison dans la famille des phages qui infectent les *Staphylococcus aureus* et diverses autres familles permettant de mélanger différents génomes grâce à la recombinaison illégitime. En effet, la comparaison des génomes de deux phages recombinés entre eux montre la présence d'une alternance de séquences similaires et de séquences divergentes. Une étude de génomique comparative, menée par Lucchini et ses collaborateurs [19], corrobore l'implication de la recombinaison modulaire chez les bactériophages spécifiques de *Streptococcus thermophilus*. Chez ces derniers, l'échange de module est suivi par des mutations ponctuelles et de petites délétions/insertions [10].

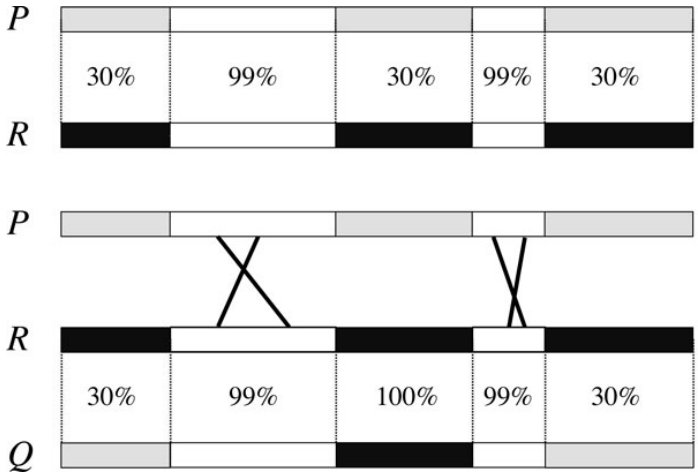


Fig 9: *La recombinaison chez les phages. En haut : comparaison de 2 phages P et R le long de leur séquence et qui révèle un motif alternant des régions hautement similaires et divergentes. En bas : un événement de recombinaison modulaire entre P et R et qui utilise deux paires de régions très similaires de P et R pour produire des phages Q [15].*

4.3.2 La structure des phages en mosaïque modulaire

La théorie de Botstein postule que les phages sont des assemblages de modules qui portent des fonctions biologiques identiques codées éventuellement par des séquences non similaires. Un exemple de ces modules est celui correspondant au gène de l'intégrase permettant l'insertion du génome phagique dans le génome bactérien. La séquence nucléotidique de l'intégrase est fortement spécifique au gène bactérien présent au niveau du site d'insertion. Par exemple *Staphylococcus aureus* présente près de 10 sites différents d'intégration reconnus par différents gènes de l'intégrase. Les différentes implémentations d'un même module sont dites **variantes**.

Le nombre de modules principaux dans un phage varie de 5 à 10 modules alors que le nombre de variantes connues varie en moyenne de 4 à 10. À titre d'exemple, le génome des *Siphovirus* est organisé en 5 modules :

❶ La lysogénie (contient le gène de l'intégrase).

❷ Le métabolisme de l'ADN.

❸ La morphologie de la tête et de l'emballage de l'ADN.

❹ L'assemblage de la queue.

❺ La lyse.

4.3.3 L'alignement des génomes des phages

Dans le but de reconstituer l'histoire des recombinaisons modulaires chez les phages, Swenson et al [15] ont développé un algorithme permettant d'étudier la combinatoire des recombinaisons modulaires en gérant et exploitant l'espace des reconstructions possibles. L'algorithme développé nécessite l'alignement des génomes de phages. En effet, pour pouvoir détecter d'éventuelles recombinaisons modulaires entre plusieurs phages, il est primordial de connaître à

la fois la position de chaque module ainsi que ses différentes variantes. Cette étape d'alignement des phages est particulièrement difficile dans le sens où des variantes d'un même module ne sont pas comparables en matière de composition et/ou de longueur nucléotidique. Cependant, pour pouvoir reconstruire l'évolution des phages, les différentes variantes d'un même module doivent être alignées. Ce paradoxe est résolu par la définition suivante :

Définition 1 *L'alignement d'un ensemble de n séquences colinéaires de phage est l'identification de k positions de chaque séquence de phage, au moins l'une d'elles immédiatement avant le premier gène, en subdivisant chaque séquence en k segments éventuellement vides. L'ensemble des n segments ayant la même position correspond au module de l'alignement et les segments non vides au sein d'un module correspondent à ses variantes.*

La figure 10 montre un exemple d'un fragment d'alignement de 31 génome de phages de *Staphylococcus aureus* et qui couvre le gène codant pour la terminase [module Te], le gène tape [module de Ta] et le module d'ancrage correspondant au gène portal 'P'. Deux cellules de la même couleur (ou numéro) et sur la même colonne sont très semblables alors que les séquences vides sont désignés par 0.

	Phage	Acc.	Te			P								Ta
F0	ETA2	NC_008798	1	1	1	1	1	1	0	0	0	0	0	0	1
F23	80alpha	NC_009526	1	1	1	1	1	1	0	0	0	0	0	0	1
F26	69	NC_007048	8	9	1	1	1	1	0	0	0	0	0	0	1
F39	53	NC_007049	1	1	1	1	1	1	0	0	0	0	0	0	1
F29	11	NC_004615	1	1	1	1	1	1	0	0	0	0	0	0	1
F40	85	NC_007050	1	1	8	1	8	1	0	0	0	0	0	0	1
F27	NM1	NC_008583	1	1	1	1	8	1	0	0	0	0	0	0	1
F41	NM2	DQ530360	1	1	1	1	8	1	0	0	0	0	0	0	1
F2	ETA3	NC_008799	1	2	2	2	2	2	0	0	4	0	2	2	2
F35	NM4	DQ530362	3	2	2	2	2	2	0	0	4	0	2	8	2
F33	ROSA	NC_007058	1	4	4	4	2	4	4	4	4	4	2	8	2
F6	71	NC_007059	3	4	4	4	2	4	4	4	8	0	3	3	3
F5	55	NC_007060	3	2	3	3	3	3	3	4	8	0	3	3	3
F3	ETA	NC_003288	3	2	3	3	3	3	3	4	8	0	3	3	3
F24	80	DQ908929	3	2	3	3	3	3	8	4	8	0	3	8	3
F30	29	NC_007061	1	2	3	3	3	3	8	4	8	0	3	8	3
F28	88	NC_007063	3	8	4	8	3	3	3	4	8	0	3	3	3
F42	92	NC_007064	3	8	4	8	3	3	3	4	8	0	3	3	3
F7	12	NC_004616	5	5	5	5	5	0	0	0	0	0	0	5	5
F11	SLT	NC_002661	0	5	5	5	5	0	0	0	0	0	0	5	5
F19	42E	NC_007052	5	5	5	5	5	0	0	0	0	0	0	5	5
F22	IPLA35	NC_011612	5	5	5	5	5	0	0	0	0	0	0	5	5
F31	tp310-2	NC_009762	5	5	5	5	5	0	0	0	0	0	0	5	5
F12	PVL108	NC_008689	6	6	6	6	6	0	0	0	0	0	0	6	6
F13	PVL	NC_002321	6	6	6	6	6	0	0	0	0	0	0	6	6
F14	tp310-3	NC_009763	6	6	6	6	6	0	0	0	0	0	0	6	6
F15	13	NC_004617	6	6	6	6	6	0	0	0	0	0	0	6	6
F25	PV83	NC_002486	6	6	6	6	6	0	0	0	0	0	0	6	6
F17	N315	NC_004740	6	7	7	7	7	0	0	0	0	0	0	7	7
F34	77	NC_005356	6	7	7	7	7	0	0	0	0	0	0	7	7
F43	P954	NC_013195	7	7	7	7	7	0	0	0	0	0	0	7	7

Fig 10: *Un exemple d'alignement de module [15].*

Cette définition ne dicte pas de règles explicites pour conclure si deux segments correspondent à la même variante ou pas. Le processus d'alignement s'appuie à la fois sur des notions de similarité de séquence et d'annotation fonctionnelle des gènes. Dans le cas où les gènes sont bien annotés, les modules d'alignement correspondent à la réalité biologique des gènes annotés alors que dans le cas contraire, les modules de l'alignement sont essentiellement construits en se basant sur la similarité de séquence.

La procédure d'alignement repose sur deux notions de base : les *blocs* et les *modules d'ancrage*. Un module d'ancrage pour une fonction biologique f est défini comme étant l'ensemble des blocs (chaque bloc correspondant à une variante) annotés avec la fonction f. Sur cette base, on considère que deux blocs sont compatibles si aucune séquence d'un bloc donné n'est une sous-séquence dans un autre.

Aussi, si les génomes d'un ensemble $\mathscr{P}$ sont colinéaires, alors les modules ancres peuvent être ordonnés de façon linéaire de sorte que les premières coordonnées de toutes les sous-séquences correspondantes dans un phage $\mathscr{P}$ sont en augmentation. Pour deux constantes m et M, un bloc $\mathscr{B}$ est un alignement des l sous-séquences b_1, b_2,..., b_l à partir d'un ensemble de phages $\mathscr{P}$ tels que :

❶ Chaque phage dans $\mathscr{P}$ a au plus une sous-séquence dans B.

❷ L'identité entre deux séquences en $\mathscr{B}$ est supérieur à $M\%$.

❸ L'identité entre une séquence de $\mathscr{B}$ et n'importe quelle autre sous-séquence en dehors de $\mathscr{B}$ (mais en $\mathscr{P}$) est inférieur à $m\%$.

Dans la pratique, on utilise des valeurs de M supérieures à 90 et des valeurs de m inférieures à 70 [13]. Il est à noter que la qualité d'un module d'ancrage se traduit par la différence $M - m$ calculée à partir de chacun de ses blocs (*cf.* exemple tableau 1).

Tab. 1: *Exemple de qualité du module du gène portal, analysé chez 31 phages de Staphylococcus aureus.*

Variant	1	2	3	4	5	6	7	8
M (%)	94	99	96	99	98	99	98	100
m (%)	54	51	64	76	54	51	61	76

4.3.4 La reconstruction de l'histoire de recombinaison

Les recombinaisons observables Un phage est représenté par une liste de nombres entiers a_1, a_2,..., a_k où chaque entier $a_i \geq 1$ représente une variante du module i et où le nombre de modules k est fixe. Il est à noter que la liste est circulaire de telle sorte que le module k est adjacent au module 1, le nombre de variantes de chaque module est constant et le nombre de variantes diffère potentiellement d'un module à l'autre. Lorsqu'un module est absent dans un phage donné, son numéro de variante est noté 0 (signifie la présence de délétions). La recombinaison modulaire entre deux phages $a_1...a_i...a_j...a_k$ et $b_1...b_i...b_j...b_k$ mettant en jeu les segments $a_i...a_j$ et $b_i...b_j$ implique que $a_i = b_i$, que $a_j = b_j$ et que les deux ne valent pas 0. Les modules correspondants à a_i et a_j sont appelés les modules flanquant de la recombinaison.

Par exemple, la recombinaison entre les phages A et B, ayant 6 modules, avec i= 1 et j=3 donnerait naissance au phage C et alternativement la recombinaison entre les phages A et D donnerait aussi lieu au phage C à savoir :

A = **2 5 3 3** 5 5
B = **2** 6 4 **3 2 1**
C = 2 5 3 3 2 1

A = **2 5 3 3** 5 5
D = **2** 7 2 **3 2 1**
C = 2 5 3 3 2 1

De manière conventionnelle, la relation parent/enfant est décrite par une flèche de sorte que $A \rightarrow B$ implique que A est un parent éventuel de B. Le phage C est un enfant potentiel de A et B se traduit $A \rightarrow C \leftarrow B$. L'histoire de recombinaison est décrite tout simplement par un ensemble de recombinaisons.

Le premier problème (Problème 1) lié à la reconstruction de l'histoire de recombinaison se résume comme suit : *étant donné un ensemble de phages $\mathscr{P}$, reconstruire l'histoire de recombinaison de telle sorte qu'un nombre maximum de phages de $\mathscr{P}$ est issu de la recombinaison des éléments de $\mathscr{P}$.*

La complexité de ce problème de reconstruction de l'histoire de recombinaison reste indéfinie et se voit davantage compliquée notamment à cause de la présence de données manquantes (le nombre de phages répertoriés et séquencés). Du problème 1 découle le second problème (Problème 2) qui est de mettre en place des techniques pour résoudre ce qui suit : *étant donné un ensemble $\mathscr{P}$ de phages, trouver un ensemble minimal des parents manquants $\mathscr{Q}$ de telle façon qu'un nombre maximum de phages de $\mathscr{P}$ sont produits par la recombinaison des éléments de ($\mathscr{P} \cup \mathscr{Q}$) .*

La première étape de la reconstruction de l'histoire de recombinaison consiste en l'identification de toutes les recombinaisons observables composées de deux parents et leur enfant potentiel : $P \rightarrow Q \leftarrow R$. Étant donné un ensemble $\mathscr{P}$ de phages, l'ensemble $\mathscr{C}$ des enfants potentiels est le sous-ensemble de phages de $\mathscr{P}$ qui peuvent être produits par au moins une recombinaison de deux phages de $\mathscr{P}$ et qu'on appelle *recombinaisons observables* . Ces recombinaisons sont représentées dans un graphe orienté, éventuellement cyclique ayant P sommets, dit le **graphe des parents** . Chacune des recombinaisons observables est traduite par deux arêtes partant des parents vers l'enfant. La figure 11 représente un graphe de parents. Le **graphe de recombinaison** est un sous-graphe du graphe des parents, qui est acyclique, et où chaque nœud a soit deux arêtes entrantes (de ses parents) soit aucune. Un **graphe complet** est un graphe de recombinaison dans lequel aucune recombinaison ne peut être ajoutée sans violer la condition acyclique et un **graphe de recombinaison maximal** est un graphe complet qui contient un nombre maximum de recombinaisons. Les arêtes pleines de la figure 12 représentent un graphe de recombinaison maximal.

Dans un graphe de recombinaison, le sous-ensemble des ancêtres $\mathscr{F}$ est l'ensemble des sommets qui ont une ou plusieurs arêtes sortantes, mais qui n'a pas d'arêtes entrantes tandis que le sous-ensemble de descendants $\mathscr{D}$ est l'ensemble de sommets qui ont deux arêtes entrantes.
Il est simple de reconstruire un graphe de recombinaison complet (algorithme 1) avec la contrainte que l'ensemble de ses ancêtres $\mathscr{F}$ est inclut dans un sous-ensemble fixe $\mathscr{S} \subseteq \mathscr{P}$.

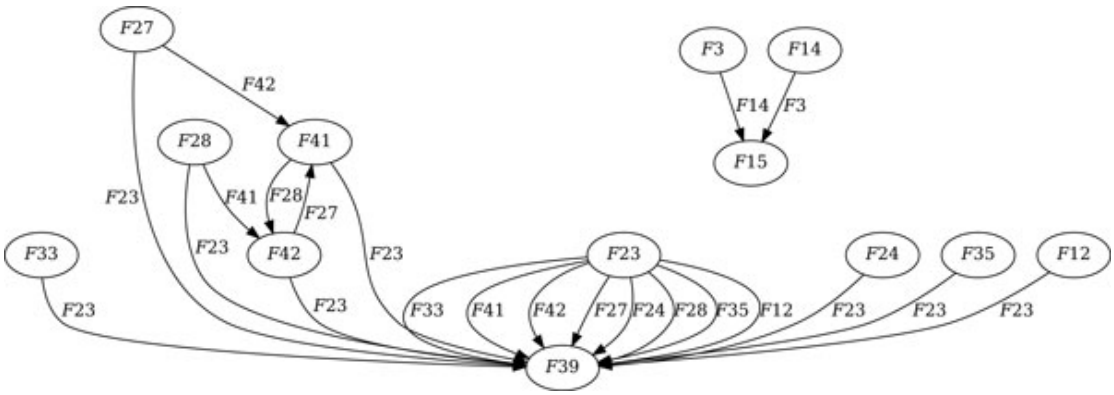


Fig 11: *Un exemple de graphe des parents [15].*

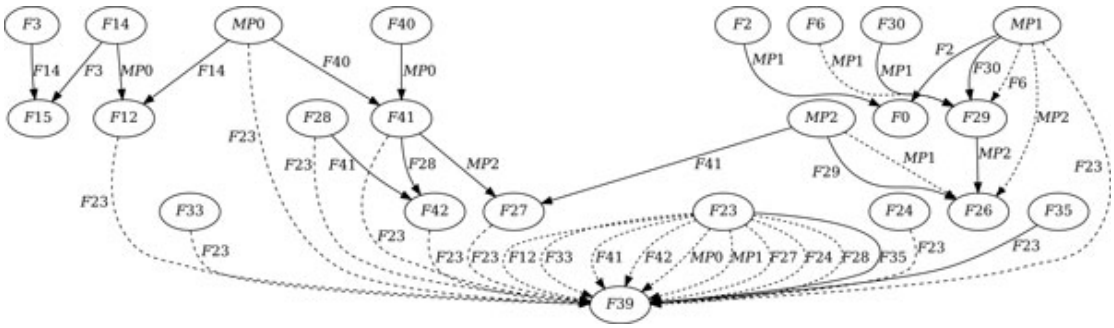


Fig 12: *Un exemple de supergraphe du graphe de recombinaison maximale [15].*

Définition 2 *Soit l'ensemble de phages $\mathscr{P}$ ayant $\mathscr{C}$ comme enfants potentiels. Le sous-ensemble $\mathscr{E}$ de $\mathscr{C}$ est* ***réalisable*** *s'il existe un graphe de recombinaison ayant comme ancêtres $\mathscr{F} \subseteq (\mathscr{P} \setminus \mathscr{C})$ tel que $\mathscr{E} \subseteq \mathscr{D}$.*

Cet algorithme calcule une solution maximale pour le problème 1 avec la contrainte que les ancêtres sont dans $\mathscr{P} \setminus \mathscr{C}$. Tout graphe complet de recombinaison dont les ancêtres sont dans $\mathscr{P} \setminus \mathscr{C}$ est soit un graphe de recombinaison maximale soit il peut le devenir en ajoutant des recombinaisons supplémentaires.

Algorithme 1 : Construire un graphe complet avec ancêtres dans $\mathscr{S}$.

Entrées : la liste de recombinaisons observables $\mathscr{L}$ et l' ensemble des ancêtres potentiels S
Sorties : le graphe de recombinaison complet $\mathscr{G}$ dont les ancêtres sont dans S.
$\mathscr{L}' \leftarrow$ L
répéter
  pour chaque *chaque recombinaison* $P \rightarrow Q \leftarrow R$ *dans* $\mathscr{L}$ **faire**
    $P, R \in S'$ et $Q \notin S'$ Ajouter $P \rightarrow Q \leftarrow R$ à $\mathscr{G}$
    Ajouter Q à $\mathscr{L}'$
jusqu'à *Plus rien n'est ajouté à* $\mathscr{G}$
;

En effet, si on a un graphe complet $\mathscr{G}$ avec des descendants $\mathscr{D}$ et dont les ancêtres $\mathscr{F}$. sont dans $\mathscr{P} \setminus \mathscr{C}$, et soit $\mathscr{M}$ un graphe maximal de descendants $\mathscr{D}'$ et ancêtres $\mathscr{F}'$. Considérons le graphe acyclique de $\mathscr{M}'$ de $\mathscr{M}$ induit par les ancêtres $\mathscr{F}' \setminus \mathscr{F}$, et les sommets $\mathscr{D}' \setminus \mathscr{D}$. Par construction, les sommets et les arêtes de $\mathscr{G}$ et $\mathscr{M}'$ sont disjoints. Tout sommet de $\mathscr{D}' \setminus \mathscr{D}$ est connecté, dans $\mathscr{M}'$, à au moins un parent, sinon ce serait en contradiction avec le caractère complet de $\mathscr{G}$. Pour les sommets ayant un seul parent en $\mathscr{M}'$, l'autre parent doit être dans $\mathscr{G}$. Par conséquent, tous les arcs manquants allant des sommets de $\mathscr{G}$ aux sommets de $\mathscr{M}$, peuvent être ajoutés sans créer de cycles.

En combinant ces résultats, et sachant que $\mathscr{B}$ est le sous-ensemble des enfants potentiels $\mathscr{C}$ qui ne sont pas dans $\mathscr{G}$, le problème 1 peut être résolu en 2 étapes :

❶ Calculer un graphe de recombinaison complet $\mathscr{G}$ en utilisant l'algorithme 1 et l'ensemble $\mathscr{P} \setminus \mathscr{C}$.

❷ Trouver un sous-ensemble minimal $\mathscr{F}'$ de $\mathscr{B}$ tel que tous les éléments de $\mathscr{B} \setminus \mathscr{F}'$ ont des parents en $\mathscr{B}$, $\mathscr{G}$ ou $\mathscr{P} \setminus \mathscr{C}$.

Les parents manquants Swenson et al [15] proposent une approche de reconstruction des parents manquants qui est basée sur l'hypothèse que les enfants de la recombinaison des phages P et Q partagent au moins trois modules consécutifs de chaque parent.

Supposons deux phages P et Q partageant au moins trois modules consécutifs, alors le phage P peut être un parent du phage Q et inversement. Ces deux hypothèses seront désignées $P \rightarrow Q$ et $Q \rightarrow P$, ou tout simplement $P \leftrightarrow Q$. La reconstruction de l'histoire de la recombinaison doit permettre de décider, pour chaque paire de phages qui partagent au moins trois modules consécutifs, si $P \rightarrow Q$, $Q \rightarrow P$, ou aucun des deux.

Si $P \rightarrow Q$, il est possible d'identifier partiellement le parent manquant . Par exemple, considérons l'ensemble suivant des phages, qui ne contient qu'une seule recombinaison observable $A \rightarrow C \leftarrow D$:

A = 2 1 4 3 2 1
C = 2 2 4 3 2 1
D = 2 2 4 2 2 1
G = 1 2 3 1 1 2
H = 2 2 3 1 1 2

Nous avons : $A \leftrightarrow C$, $A \leftrightarrow D$, $C \leftrightarrow D$ et $G \leftrightarrow H$. Ces relations donnent 8 modèles de parents manquants. Les deux modèles générés de la relation $G \leftrightarrow H$ sont :

$$\mathbf{G} \rightarrow \mathbf{H} \leftarrow G_H = \mathbf{2\ 1\ *\ *\ *\ 1}$$
$$\mathbf{H} \rightarrow \mathbf{G} \leftarrow H_G = \mathbf{1\ 2\ *\ *\ *\ 2}$$

La notation G_H désigne le parent manquant sachant que G est le premier parent de H alors que H_G désigne la parent manquant observable sachant que H est le premier parent de G. Deux ou plusieurs modèles sont qualifiés de **compatibles**

si l'l'ensemble de phages qu'ils représentent n'est pas vide. Le graphe des parents manquants est donc le graphe de la relation de compatibilité sur un ensemble de phages et de tous les modèles des parents manquants déduits de l'ensemble. La figure 13 montre le graphe correspondant à l'exemple ci-dessus. Dans cet exemple, il existe une arête entre les modèles $C_A = 214***$ et $D_C = **432*$ car ils sont compatibles (ils ne présentent pas de variantes contradictoires pour n'importe quel module : 2 et $'*'$ sont compatibles, 1 et $'*'$ sont compatibles, 4 et 4 sont compatibles, et ainsi de suite).

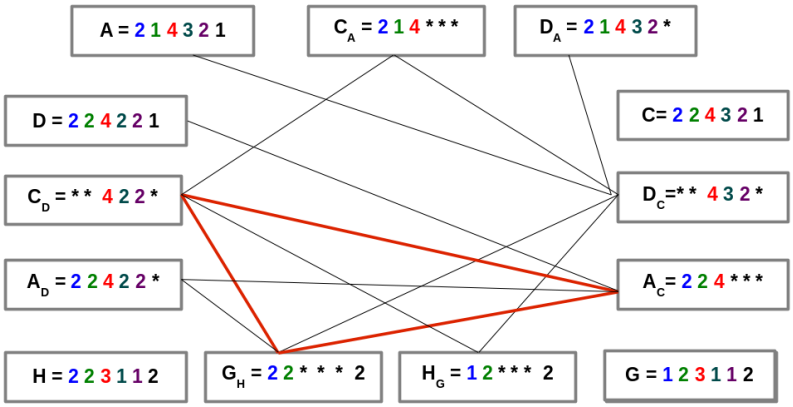


Fig 13: *Le graphe des parents manquants.*

Les cliques contenant un phage observé correspondent à des événements de recombinaison observables. Les cliques les plus intéressantes sont celles dont les éléments sont tous des modèles car ils décrivent les parents manquants. Par exemple, sur la figure 13, la clique qui contient $G_H = 22***2$, $C_D = **422*$ et $A_C = 224***$ définit le phage 224222, qui est le seul phage qui appartient à l'ensemble des trois sommets de la clique correspondant à des modèles de parents manquants.

Choisir cette solution permet de déduire le phage manquant $B = 224222$, et trois événements de recombinaison : $A \rightarrow C \leftarrow B$, $C \rightarrow D \leftarrow B$ et $G \rightarrow H \leftarrow B$.

Comment ajouter les parents manquants ?

L'ajout des parents manquants doit être effectué avec soin afin de s'assurer que la procédure converge si elle est appliquée de manière récursive. De manière générale :

❶ Un parent manquant ne doit pas être un enfant potentiel de l'ensemble des phages à laquelle il est ajouté.

❷ Il faut augmenter le nombre de recombinaisons par deux au plus.

❸ Les parents manquants des membres de $\mathscr{C}$ qui n'ont pas de parents dans le graphe de recombinaison ont la priorité.

La complexité de l'algorithme présenté ici pour résoudre le problème lié à la reconstruction de l'histoire évolutive modulaire chez les phages reste indéfinie. Il s'agit d'une *heuristique* permettant de trouver une *solution réalisable* mais qui n'est pas nécessairement optimale ou exacte. Le *problème exact* n'étant pas bien caractérisé pour garantir que l'on obtient la *solution optimale*.

I RAPPELS & DÉFINITIONS

5 Rappels et Définitions

Avant de développer la méthode mise au point pour répondre à cette problématique, il nous semble judicieux de rappeler d'une part quelques notions et notations générales relatives à la théorie des graphes et d'autre part les notions développées dans le contexte de cette étude et que j'utiliserai dans la suite du document.

Rappel de quelques notions générales de la théorie des graphes et de l'algorithmique du texte La figure 14 sera utilisée tout au long de cette partie pour illustrer certaines définitions en relation avec la théorie des graphes.

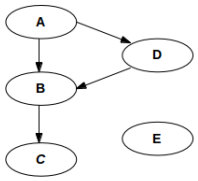


Fig 14: *Un graphe orienté acyclique G.*

Définition 3 (Graphe orienté) *: un graphe orienté G est la donnée d'un couple $G = (V,A)$ tel que :*

- ❒ *V est un ensemble fini de sommets.*
- ❒ *A est un ensemble de couples ordonnés de sommets $(x,y) \in V^2$.* Ex : *on note le graphe orienté de la figure 14 G = (V', A') avec V′ est l'ensemble alphabétique compris entre les lettres A et E et A'= $\{(A,B),(A,D),(B,C),(D,B)\}$.*

Définition 4 (Notions générales relatives aux graphes acycliques orientés (DAG)) *: un graphe orienté G est la donnée d'un couple $G = (V,A)$ tel que :*

- ❒ *Un couple (x, y) est appelé un **arc**, et est représenté graphiquement par $x \to y$, x et y sont respectivement l'**origine** et l'**extrémité** de l'arc (x, y). L'arc (x,y) est dit **sortant** en x et **entrant** en y.* Ex : *l'arc (A, B) a pour origine A et pour extrémité B.*
- ❒ *Le nombre d'arcs entrants en x dans G est appelé **degré entrant**, noté $d_-(x)$.* Ex : $d_-(A) = 0$.
- ❒ *Le nombre d'arcs sortants en x dans G est appelé **degré sortant**, noté $d_+(x)$.* Ex : $d_+(A) = 2$.
- ❒ *$x \in V$ est dit **source** de G si Pred(x) = $\emptyset$.* Ex : *G admet deux sources A et E.*
- ❒ *$x \in V$ est dit **puits** de G si Succ(x) = $\emptyset$.* Ex : *G admet deux puits C et E.*
- ❒ *Tout graphe orienté et sans circuit (DAG) admet au moins un sommet **entrant** de **degré nul** dit **source**.*
- ❒ *Le degré d'un sommet x dans G est le nombre d'arcs incidents à x (noté $d(x)$) défini par $d(x) = d_+(x) + d_-(x)$.*
- ❒ *Une **marche orientée** dans G est une suite $(x_0, \ldots, x_h)$ de sommets de G telle que $h \geq 0$ et $\forall i$, $0 \leq i < h$, $(x_i, x_{i+1}) \in A$. x_0 est l'origine de la marche orientée, x_h son extrémité, h sa longueur. On note x_0x_h–marche orientée, la marche orientée d'origine x_0 et d'extrémité x_h.* Ex : *c1 = (A, D, B, C) est une AC-marche orientée, de longueur 4 (elle n'est pas de longueur minimale). c2 = (A, B, C) est une AC–marche orientée extraite de c1 et de longueur 3.*
- ❒ *Deux sommets reliés par un arc sont qualifiés de **voisins**. On note $\upsilon(x,G)$ les voisins de x dans G.*
- ❒ *Les **successeurs** d'un sommet x dans G, noté Succ(x), est l'ensemble des $y \in V$ tels que $(x, y) \in A$.* Ex : $Succ(A) = \{B,D\}$, $Pred(D) = \{A\}$.
- ❒ *Les **prédécesseurs** d'un sommet x dans G, noté Pred(x), est l'ensemble des $y \in V$ tels que $(y, x) \in A$.* Ex : $Pred(D) = \{A\}$.
- ❒ *$x \in V$ est dit sommet **isolé** de G si Succ(x) = Pred(x) = $\emptyset$.* Ex : *E est le seul sommet isolé de G.*
- ❒ *L'ensemble des **descendants** d'un sommet x de G, noté DescG (x), est l'ensemble des sommets y pour lesquels il existe un xy-marche.* Ex : $DescG(A) = \{B,C,D\}$ *et* $AscG(A) = \emptyset$

- ❒ *L'ensemble des **ascendants** d'un sommet x de G, noté AscG (x), est l'ensemble des sommets y pour lesquels il existe une yx-marche.*
- ❒ *Une **boucle** est une arête d'un graphe ayant pour extrémités le même sommet. L'arc (x, x) est une boucle.*
- ❒ *Un circuit est une marche orientée dont le sommet d'origine et d'extrémité est le même. Dans les graphes non orientés, la notion équivalente est celle de cycle. Un cycle est une suite d'arêtes consécutives (chaîne) dont les deux sommets extrémités sont identiques.*
- ❒ *On dit qu'un graphe possède un **circuit hamiltonien** s'il existe un circuit passant une, et une seule fois, par chaque sommet.*
- ❒ *Un **circuit eulérien** est un circuit passant une et une seule fois par tous les arcs de G.*
- ❒ *Un graphe orienté est dit **acyclique** s'il ne contient pas de boucle élémentaire ni de circuit. On appelle DAG un tel graphe.*
- ❒ *Une **racine** r de G est un sommet tel que DescG (r) = V et AscG (r) = ∅.*
- ❒ *Le **tri topologique** de G est un ordre linéaire des sommets de G tel que si G contient l'arc (x,y), x apparaît avant y. Le tri topologique d'un graphe peut être vu comme un alignement de ses sommets le long d'une ligne horizontale tel que toutes les arcs soient orientés de gauche à droite. Tri topologique de G est un **ordre total des sommets** de G. Tout DAG admet un tri topologique qui n'est pas forcément unique. Il est clair que si le graphe G contient un circuit alors il n'est pas possible de trouver un tri topologique de ce graphe.*
- ❒ *Un graphe orienté $G = (S,A)$ est **connexe** si le graphe non orienté associé est connexe $G' = (S,A')$. G' est connexe si quelque soient les sommets x et y de S, il existe existe une suite finie d'arêtes consécutives reliant x à y.*
- ❒ *Un arc de G est un **isthme** ou un **pont** si et seulement si son élimination rend le graphe non connexe. De façon similaire,un arc est un pont si et seulement si il ne fait pas partie d'un circuit.* Ex : *l'arc (B,C) est un isthme car sa suppression rend G non connexe.*

Définition 5 (Forte connexité) *: soit G = (V, A) un graphe orienté. On définit la relation $\underset{fc}{\approx}$ sur V par : $x \underset{fc}{\approx} y$ si et seulement si il existe une xy-marche orientée et une yx-marche orientée dans G. La relation $\underset{fc}{\approx}$ est une relation d'équivalence.*

***Fig 15:** Deux graphes orientés, un fortement connexe (trait pointillé) et l'autre non fortement connexe (trait plein). Le graphe non fortement connexe possède quatre composantes fortement connexes : les sous-graphes possédant un seul sommet.*

Définition 6 (Composante fortement connexe) *: les classes d'équivalence $\underset{fc}{\approx}$ sont appelées composantes fortement connexes. Un graphe est fortement connexe si pour tout couple de sommets distincts $(x,y) \in A^2$, il existe un chemin de x à y et un chemin de y à x.*

Définition 7 (Graphe réduit) *: soit G = (V,A) un graphe orienté. On appelle graphe réduit de G le graphe quotient de G par la partition des sommets induite par $\underset{fc}{\approx}$. On appelle ce graphe réduit G_r. Ses sommets $c_1,\ldots,c_p$ sont les composantes fortement connexes de G, et il existe un arc entre c_i et c_j dans G_r si et seulement s'il existe au moins un arc entre un sommet de c_i et un sommet de c_j dans le graphe G.*
Propriété : *Un graphe réduit est toujours sans circuit : c'est un DAG.*

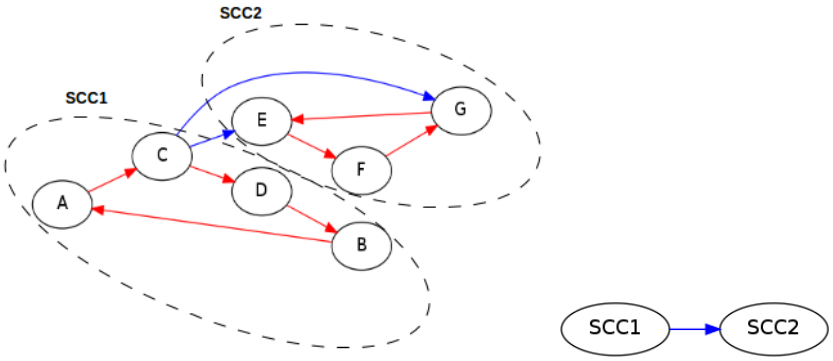


Fig 16: *Un graphe orienté G, ses composantes fortement connexes (entourées en pointillés) et son graphe réduit. Le graphe G a pour composantes connexes les sous-graphes SCC1, ayant pour sommets A, B, C, D, et SCC2, ayant pour sommets E, F, G. Il existe un arc entre un sommet de SCC1 et un sommet de SCC2 dans G d'où la présence d'un arc entre SCC1 et SCC2 dans le graphe réduit.*

Définition 8 (Réduction transitive d'un DAG) *: la réduction transitive G' d'un DAG G = (V, A) est le sous-graphe de G G'= (V, A') possédant le moins d'arcs possibles pour représenter la même relation d'accessibilité que le graphe original. S'il existe un chemin d'un sommet x à un sommet y dans le graphe G, alors il existe un chemin de x à y dans G' et vice versa.*

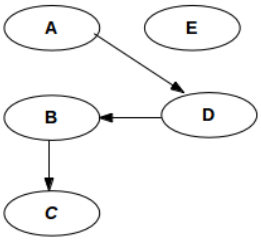


Fig 17: *Réduction transitive du graphe G de la figure 14.*

Définition 9 (Décomposition par niveaux) *: soit $G = (V,A)$ un DAG. On note S(G) l'ensemble des sources de G. La décomposition en niveaux de G (ou S-séquence) est la suite de parties non vides de $X(S_0,S_1,\ldots,S_k)$ telle que* (cf. *figure 18) :*
$S_0 = S(G)$
$S_1 = S(G(X \backslash S_0))$
$S_2 = S(G(X \backslash (S_0 \cup S_1)))$
$S_i =$ *est l'ensemble des sources du sous-graphe de G induit par l'ensemble de sommets* $X \backslash (S_0 \cup \ldots \cup S_{i-1})$
$S_{K+1} = \emptyset$

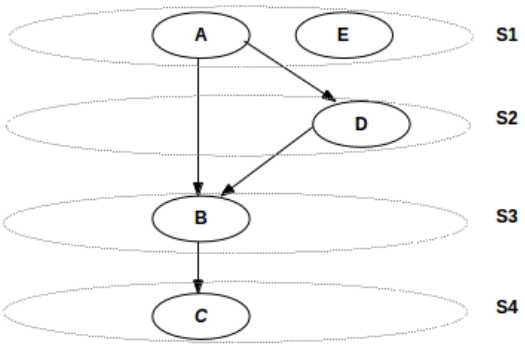


Fig 18: *La décomposition par niveau du graphe orienté de la figure 14.*

Jusqu'à ici nous avons introduit les différentes notions de la théorie des graphes qui seront utilisées dans le reste du document. Toutefois, quelques notions de l'algorithmique du texte restent à définir de manière formelle à savoir :

Définition 10 (Mot/alphabet) *: un* ***alphabet*** Σ *est un ensemble, non vide, fini ou infini de symboles. Un* ***mot*** *(ou chaîne de caractères) de l'alphabet m est une liste de symboles issus de* Σ*.*

Définition 11 (Facteur) *: on dit qu'une chaîne de caractère x est un* ***facteur*** *d'une chaîne de caractère w s'il existe des mots y et z tels que* $w = yxz$*. Un synonyme de facteur est sous-mot.*

Définition 12 (Génome) *: un* ***génome*** *est un mot de l'alphabet défini par l'ensemble* $\{A, C, G, T\}$*.*

RÉALISATIONS

6 RÉALISATIONS

Cette section détaille l'essentiel du travail entrepris au cours du stage. Le but de ce stage est de formaliser la première partie de l'algorithme permettant de reconstruire l'histoire évolutive des bactériophages sur la base des recombinaisons modulaires observables à savoir l'étape de l'alignement des modules sur les génomes de phages. Cette étape s'appuie sur le résultat du BLAST [13]. L'étude menée dans le cadre de ce stage vise à développer une méthode permettant de contribuer à la détection des modules fonctionnels. C'est une étape clé dans l'approche de reconstitution de l'histoire évolutive des bactériophages développée par Swenson et al. 2013. Les résultats obtenus avec cette approche étaient très satisfaisants. Toutefois, Swenson et al. (2013) avaient procédé à la détection des modules manuellement ce qui est à la fois lent, lourd, impossible à appliquer sur les gros jeux de données et peut constituer une source d'erreurs. Ici, nous aborderons la problématique de la détection des modules à partir des résultats du BLAST deux à deux d'une part et d'autre part la formalisation du résultat attendu.

6.1 DÉFINITIONS DES NOTIONS DÉVELOPPÉES LORS DE CE STAGE

Les définitions introduites ici sont basées sur les alignements deux à deux d'un ensemble de génomes de phages P. Ces alignements définissent des sites de début et de fin d'appariement (*cf.* définition 13 et figure 19). En effet, pour tout appariement entre deux phages P_x et P_y correspond 4 sites dont 2 appartenant à chacun des phages (*cf.* définition de site 14). On note, à titre d'exemple, $s = (P_x, b_1, b_1)$ le site s défini par un alignement impliquant le phage P_x où b_1 est une extrémité d'appariement sur P_x. On dit que *s appartient* à P_x (on note $s \in P_x$) et que P_x est *représenté* par le site s.
Pour des raisons pratiques, on a fait le choix de représenter un site par le nom du phage auquel il appartient mais également par deux indices (position nucléotidique sur le génome) qui sont identiques lors de l'initialisation des sites. Ce choix trouve sa justification lors du traitement et de la manipulation des sites, comme la *fusion*, et que j'aborderai un peu plus loin dans le document (*cf.* la partie 6.2.3).

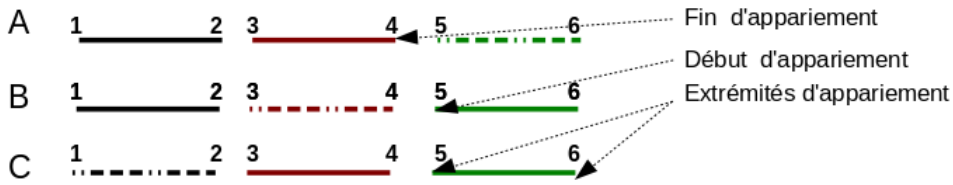


Fig 19: *Alignement entre 3 génomes A, B et C. Deux facteurs identiques entre deux génomes sont représentés par des traits du même style et de la même couleur.*

Définition 13 (Appariement) *: est une paire de facteurs similaires entre deux génomes. Chacun des facteurs appariés A est caractérisé par* un site de début d'appariement s_d *et un* site de fin d'appariement s_f *indiquant respectivement le début et la fin de la similarité et qu'on note par le couple* $A = (s_d, s_f)$*. De manière plus générale, on appelle* s_d *et* s_f extrémités *du facteur apparié A.*

Définition 14 (Site/Collection de sites) *: on définit un* site s *sur un génome* P_x *par le triplet* $s = (P_x, b_x, b_y)$ *où* b_x *correspond à la position d'une des 2 extrémités de l'appariement de* P_x *avec phage* P_y*. On dit que s appartient à* P_x *(on note* $s \in P_x$*) et que* P_x *est représenté par le site s. Chaque site appartient à un seul et unique phage. Une collection de sites est un ensemble de sites appartenant au même phage.*

Définition 15 (Facteur ancre) *: est un facteur à occurrence unique fortement conservé chez tous les génomes comparés/alignés.*

13. The Basic Local Alignment Search Tool.

Définition 16 (Ordre sur les appariements) *: soient deux appariements appartenant au même phage $A_1 = (s_{1d}, s_{1f})$ et $A_2 = (s_{2d}, s_{2f})$. On dit que :*

- *A_1 est* plus petit *que A_2 si et seulement si $s_{1d} \leq s_{2d}$ et $s_{1f} \leq s_{2f}$. On note $A_1 \leq A_2$.*
- *A_1 recouvre A_2 si et seulement si $s_{1d} \leq s_{2d}$ et $s_{1f} > s_{2f}$. On note $A_1 \supset A_2$.*

Définition 17 (Chevauchement/appariements chevauchants) *: soient deux appariements $A_1 = (s_{1d}, s_{1f})$ et $A_2 = (s_{2d}, s_{2f})$ définis sur le même génome. A_1 et A_2 sont dits chevauchants dans 2 cas :*

- *cas 1 : si $A_1 \supset A_2$.*
- *cas 2 : si $A_1 < A_2$ (sans perte de généralité), et si $s_{2d} \leq s_{1f}$.*

Proposition 1 (Colinéarité de deux génomes) *: deux génomes P_x et P_y sont colinéaires si l'ensemble des mots/facteurs/appariements communs avec une occurrence unique sur chaque génome est ordonné de la même façon sur chacun des génomes. La figure 20 montre deux génomes P_x et P_y dont les facteurs en communs ne conservent pas le même ordre sur les deux génomes.*
Autrement dit, on a une contradiction de colinéarité dans $\mathscr{P}$ si et seulement s'il existe une séquence de facteurs $f_0, f_1, \ldots, f_k$ avec $f_0 = f_k$ et que pour chaque paire de facteurs f_i, f_{i+1} correspond l'un des deux cas :

① *f_i et f_{i+1} avec f_i est positionné quelque part avant f_{i+1} sur le même génome.*
② *f_i est une occurrence du mot f_{i+1} dans un phage $P_y \in P \setminus \{P_x\}$.*

Il faut qu'on est au moins une paire f_i, f_{i+1} qui satisfait la condition 1 .

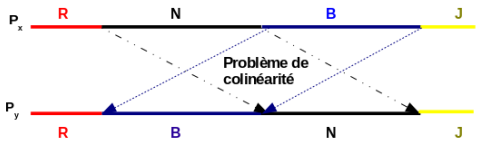


Fig 20: *Absence de colinéarité entre deux phages P_x et P_y. Les deux facteurs N et B n'ont pas le même ordre sur P_x et P_y.*

Proposition 2 (Colinéarité de plusieurs génomes) *: un ensemble de génomes est colinéaire si les facteurs communs avec une occurrence unique sur chaque génome ne forment pas de cycle sur l'ensemble des génomes.*

Remarque 1 *: considérons les phages P_x, P_y, P_z ayant respectivement les génomes ci-dessous. Les 3 phages sont colinéaires 2 à 2 mais non colinéaires à trois* (cf. *figure 21).*

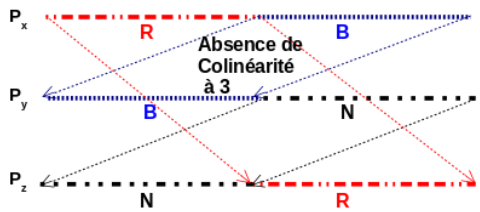


Fig 21: *Génomes colinéaires 2 à 2 mais non colinéaires ensemble.*

Définition 18 (Insertion) *: soient deux génomes colinéaires P_x, P_y, s'il existe un site $s_x = (P_x, b_x, e_x)$ dans P_x correspondant à deux sites $s_{y1} = (P_y, b_{y1}, e_{y1})$ et $s_{y2} = (P_y, b_{y2}, e_{y2})$ dans P_y avec s_{y1} et s_{y2} ($s_{y1} < s_{y2}$) sont deux positions successives sur P_y alors on dit qu'il y a insertion du facteur b_{y1} à e_{y2} dans P_y en comparaison avec P_x* (cf. *figure 22). On distingue deux types d'insertion :*

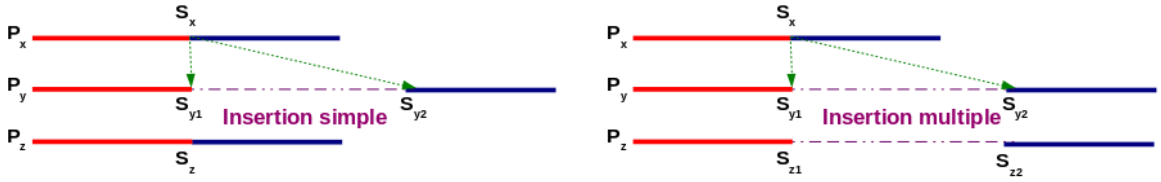


Fig 22: *Exemple d'insertions simple et multiple : insertion simple du facteur $S_{y1}S_{y2}$ (trait en pointillés) dans le génome P_y en comparaison avec P_x et P_z et insertion multiple des facteurs $S_{y1}S_{y2}$ et $S_{z1}S_{z2}$ dans respectivement P_y et P_z en comparaison avec P_x.*

Insertion simple *: présence d'insertion chez un seul génome parmi ceux analysés.*
Insertion multiple *: présence d'insertion chez au moins deux génomes parmi ceux analysés.*

Définition 19 (Relation de correspondance des sites) *: on définit la relation de correspondance α entre deux sites appartenant à deux phages différents de la façon suivante : $(s_x)\ \alpha\ (s_y)$ avec $s_x = (P_x, b_x, e_x)$ et $s_y = (P_y, b_y, e_y)$ si et seulement si les positions b_x, b_y définissent un même site de début ou de fin de similarité entre les génomes P_x et P_y. La relation $(s_x)\ \alpha\ (s_y)$ est symétrique.*

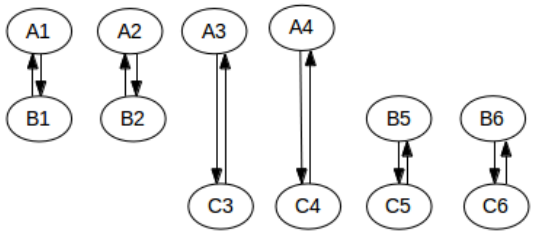


Fig 23: *Graphe non connexe de correspondance des sites issu de l'alignement représenté dans la figure 19. À titre d'exemple les sommets A1 et B1 sont en relation de correspondance et les arcs les reliant sont appelés des arcs de correspondance.*

Définition 20 (Relation de correspondance des appariements) *: on définit la relation de correspondance α' entre deux appariements $A_x = (s_{dx}, s_{fx})$ et $A_y = (s_{dy}, s_{fy})$ appartenant respectivement à deux phages P_x et P_y (avec $A_x \neq A_y$) de la manière suivante : $(A_x)\ \alpha'\ (A_y)$. Deux appariements sont en relation de correspondance si et seulement si $s_{dx}\ \alpha\ s_{dy}$ et $s_{fx}\ \alpha\ s_{fy}$ sont vraies.*

Définition 21 (Relation d'ordre sur les sites) *: on définit la relation d'ordre β entre deux sites $s_x = (P_x, b_x, e_x)$ et $s_y = (P_y, b_y, e_y)$ appartenant à un même phage de la façon suivante : $(P_x, b_x, e_x)\ \beta\ (P_y, b_y, e_y)$ si et seulement si $P_x = P_y$ et b_x est positionné avant b_y dans P_x. La relation β est anti-symétrique.*

Définition 22 (Fusion) *: Soient deux sites définis chez le même phage $s_x = (P_x, P_{xd}, P_{xf})$ et $s_{x'} = (P_x, P_{xd'}, P_{xf'})$ (avec $s_x < s_{x'}$). La fusion de s_x et $s_{x'}$ donne lieu au site $s_{x''} = (P_x, min(P_{xd}, P_{xd'}), max(P_{xf}, P_{xf'}))$.*

Définition 23 (Graphe des sites) *: les deux relations binaires définies dans 19 et 21 permettent d'avoir un graphe connexe dit le graphe des sites $Gs = (V, E)$ sur l'ensemble des sites $\mathscr{S}$ avec $V = S$ et $E = \alpha \cup \beta$.*
*Les arcs de la relation α sont symétriques et sont appelées **arcs de correspondance** (arcs interphages). Ces arcs sont exclusivement entre deux positions dans deux génomes différents tandis que les arcs de la relation β sont anti-symétriques et sont appelées les **arcs d'ordre** (arcs intraphages) et ne lient que des sites d'un même génome.*

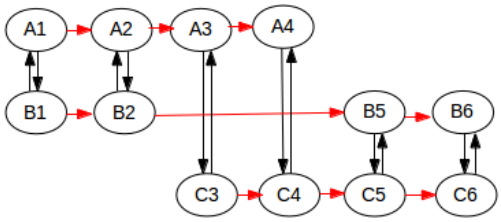


Fig 24: *Le graphe d'ordre des sites obtenu à partir du graphe des correspondances des sites de la figure 23. Les arcs intraphages (rouge) sont des arc d'ordre et les arcs interphages (noir) sont des arcs de correspondance.*

Définition 24 (Région) *: on appelle région dans le graphe des sites une composante fortement connexe du graphe.*

Remarque 2 *La construction du graphe réduit à partir du graphe des sites, on obtient un DAG, qui indique l'ordre partiel dans lequel apparaissent les composantes fortement connexes et donc les régions.*

Définition 25 (Graphe des régions) *On appelle graphe des régions Gr la réduction transitive du graphe réduit du graphe des sites Gs par la relation de forte connexité.*

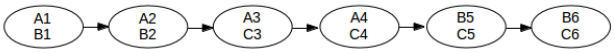


Fig 25: *Le graphe des régions obtenu par condensation des composantes fortement connexes présentes dans le graphe de la figure 24 .*

6.2 Méthodes développées

Dans ce qui suit, nous travaillerons sur un ensemble de génomes de phages $\{F_1, \ldots, F_n\}$, pour lesquels on dispose des extrémités d'appariement deux à deux issues du BLAST. Le but est de représenter les résultats du BLAST dans un graphe où les facteurs compris entre deux régions, connectées par un arc, assurent la même fonction chez les phages représentés par les sites présents dans les deux régions en question. Dans cette partie nous formaliserons notre problématique ainsi que le résultat attendu puis nous détaillerons l'algorithme développé.

6.2.1 Problématique et formalisation du résultat attendu

Problématique : On cherche à construire un algorithme qui prend en entrée une liste de sites définis à partir des résultats du BLAST d'un ensemble de génomes de phages 2 à 2, et qui retourne un DAG (le graphe des régions) dont les sommets sont des régions (*cf.* définition 24) où idéalement chaque phage est présent une fois au maximum par sommet.

Limitation : Nous travaillerons dans un premier temps sur des portions du génome délimitées par deux facteurs '**ancres**' pour réduire le nombre de sommets dans le graphe et donc faciliter son interprétation (*cf.* définition 15).

Pour répondre à notre problématique, nous avons développé, dans un premier temps, une méthode qui après implémentation sous python s'est avérée inadaptée et que nous avons abandonnée. Puis, dans un deuxième temps, nous avons mis au point une seconde approche qui répond parfaitement à notre problématique. Ici, nous évoquerons l'algorithme de la première approche sans trop rentrer dans le détail puis nous détaillerons les différentes étapes du deuxième algorithme ainsi que leur complexité.

6.2.2 Première approche

① **Définitions de notions relatives à la première approche** Avant de décrire l'algorithme développé, il est important de définir les notions utilisées à savoir :

Définition 26 (Position) *: on définit une* position *sur l'ensemble des génomes* $\{F_1,\ldots,F_n\}$ *comme un* $n-$ *uplet de paires d'indices sur respectivement les génomes* $F_1,\ldots,F_n$*. On note* $P[k]$ *la paire d'indice de la position P qui correspond au génome* F_k*.*

Définition 27 (Ordre sur les positions) *: on définit sur l'ensemble des positions sur les génomes* $\{F_1,\ldots,F_n\}$ *la relation binaire suivante : si* P_1 *et* P_2 *sont deux positions, on a* $P_1 \leq P_2$ *si* $\exists k \in \{1,\ldots,n\}$ *tel que* $P_1[k]$ *et* $P_2[k]$ *sont définis et* $P_1[k] \leq P_2[k]$*.*

Proposition 3 *: si les génomes de l'ensemble* $\{F_1,\ldots,F_n\}$ *sont colinéaires, la relation binaire précédente définit un ordre partiel sur l'ensemble des positions issues des extrémités d'alignement. D'après cette proposition, l'ensemble des positions issues des extrémités d'alignements deux à deux donne une structure de DAG connexe.*

Définition 28 (Sous-graphe des positions) *: soient* P_1 *et* P_2 *deux positions comparables de G, avec* $P_1 \leq P_2$*. On note* sous-graphe entre P_1 et P_2*, et on note* $A(P_1,P_2)$ *le sous-graphe composé des positions qui sont entre* P_1 *et* P_2 *dans G. NB : c'est également un DAG, connexe.*

② **Description de l'algorithme**

La première méthode d'insertion des sites d'appariement dans un graphe orienté prend en entrée la liste des paires de sites en correspondance et renvoie un graphe orienté qu'on appelle le graphe des positions (*cf.* algorithme 2). L'algorithme permet en une seule opération d'insérer 2 sites d'appariements $(s_1 = (P_x, b_1, e_1); s_2 = (P_y, b_2, e_2))$ toute en les ordonnant selon b_1 et b_2 et de conclure à la présence de colinéarité/chevauchement dans le cas de présence de contradiction d'ordre détectée lors l'ordonnancement des deux sites. Toutefois, cet algorithme gère mal les insertions. Il crée des sites artéfactuels dans les sommets des régions en fonction de l'ordre du traitement des sites dans le graphe. En effet, on s'est rendu compte de la présence de ce biais lors du test du code sur différents jeux de données. On l'a testé progressivement sur des résultats d'alignement deux à deux issus de jeux de données composés de 2, 4, 6, 7, 10 et 11 phages (*cf.* tableau 2). En absence de réarrangement de type "insertion", le graphe généré traduit fidèlement les résultats du BLAST (résultats non présentés ici). Dans le cas contraire, il crée des sites artéfactuels.

6.2.3 Deuxième approche

Pour pallier les points faibles du premier algorithme, on a proposé un deuxième algorithme répondant à notre problématique mais où le traitement et la gestion des insertions /inversions n'intervient qu'après la construction du graphe. Rappelant que le but est de générer un graphe de régions résumant les différents sites issus du BLAST, notre méthode utilisera la relation de correspondance sur les sites. Un site s est défini par le triplet $s = (F, b_1, e_1)$ avec la paire (b_1, e_1) définissant une position (b_1 et e_1 sont des entiers positifs et $b_1 \leq e_1$) sur le génome F. Au début de l'algorithme, l'intervalle b_1, e_1 est nul. Dans le graphe des sites, les composantes fortement connexes sont séparées par des arcs d'ordre nécessairement puisque les arcs de correspondance sont symétriques. Chacune des composantes fortement connexes est constituée par un ensemble de sites (au moins deux) correspondant aux extrémités des alignements et donc définissant ainsi les régions.

L'algorithme mis en place se découpe en 5 étapes comme l'illustre la figure 31. Il prend en entrée une liste de paires de sites L où chaque paire correspondent à deux extrémités d'appariement en correspondance entre deux phages (*cf.* définition de site 14) et retourne le graphe des régions.

Algorithme 2 : Insertion d'une extrémité d'appariement

Entrées : Deux sites en relation de correspondance $(s_1 = (P_x, b_1, e_1); s_2 = (P_y, b_2, e_2))$
si *On trouve s_1 (resp. s_2) dans une position P et s_2 (resp. s_1) dans aucune position* **alors**
  On rajoute s_2 (resp. s_1) dans P
si *On ne trouve ni s_1 ni s_2* **alors**
  On crée une position P avec juste s_1 et s_2;
  On considère P_{b_1} et P_{e_1} (resp. P_{b_2} et P_{e_2}) tels que $P_{b_1}[1] < b_1 < e_1 < P_{e_1}$ et il n'y a personne entre (couverture);
  On regarde le nombre d'arcs de $A(P_{b_1}, P_{e_1}) \cap A(P_{b_2}, P_{e_2})$;
  sinon
    il est égal à 0
  Message d'alerte : Problème de colinéarité;
  sinon
    il est égal à 1
  On insère sur cet arc la position P;
  On insère P entre la source et le puits de l'intersection;
si *On trouve s_1 dans P_1 et (P_y, b_2, e_2) dans P_2 avec $P_1 \neq P_2$* **alors**
  sinon
    P_1 et P_2 ne sont pas comparables
  On fusionne P_1 et P_2;
  (On suppose sans perte de généralité $P_1 \leq P_2$) on complète les positions de $A(P_1, P_2)$ avec les intervalles s_1 et (P_y, b_2, e_2);

Tab. 2: *Présence/absence des insertions dans les jeux de données testés.*

Nombre de phages	**2**	**4**	**5**	**7**	**10**	**11**
Présence d'insertion	non	non	non	non	non	oui

① Étape 1 : Initialisation d'un graphe orienté G et détermination des classes d'équivalence

Cette première étape permet de générer un graphe orienté G où chaque paire de sites (s_i,s_j) équivalents (ou classe d'équivalence) est représentée par deux sommets respectivement s_i et s_j connectés par deux arcs de correspondance, à savoir $s_i \rightarrow s_j$ et $s_j \rightarrow s_i$, dans le graphe G. Le nombre de sommets dans le graphe final G est inférieur ou égal à deux fois le nombre de sites donnés en entrée. Le graphe résultant est appelé le graphe de **correspondance des sites** connectant les différentes classes d'équivalence (figure 27). L'algorithme permettant de générer les classes d'équivalence à partir de la liste des sites équivalents est décrit dans l'algorithme 3. Le module Networkx utilise une structure en dictionnaire des dictionnaires des dictionnaires pour créer l'objet graphe (*cf.* code 1 dans la section "annexes") et dont la complexité est linéaire : $O(n)$. L'ajout d'un arc dans le graphe (*cf.* code 2 dans la section "annexes") ainsi que l'ajout d'un sommet (*cf.* code 4 dans la section "annexes") ont une complexité de l'ordre de $O(n)$. La complexité totale de cette étape est de $O(n^2)$ avec n le nombre de sommets dans le graphe.

② Étape 2 : fusion et ordonnancement des sites dans le graphe de correspondance des sites G

Cette 2^{eme} étape permet d'ordonner dans le graphe de correspondance des sites, pris en entrée, l'ensemble des sites d'un **même génome/phage** selon la **valeur numérique** du début de chaque site b_i. Par exemple, chez le phage P_x, l'ordonnancement de 3 sites s_1, s_2 et s_3 sachant que $s_1 = (P_x, b_1 = 5, e_1 = 6)$, $s_2 = (P_x, b_2 = 15, e_2 = 16)$, $s_3 = (P_x, b_3 = 25, e_3 = 26)$ résulte en l'ajout des arcs $s_1 \longrightarrow s_2$ et $s_2 \longrightarrow s_3$.
L'ordonnancement des sites d'un même phage permet de réduire le nombre des composantes du graphe de correspondance des sites pris en entrée. Pour un phage donné P, représenté par n sites/sommets dans G, l'ordonnancement de ses sites résulte en l'ajout de $n-1$ arcs d'ordre dans G. Le graphe renvoyé à la fin de cette étape est dit le graphe d'ordre des sites (figure 28).

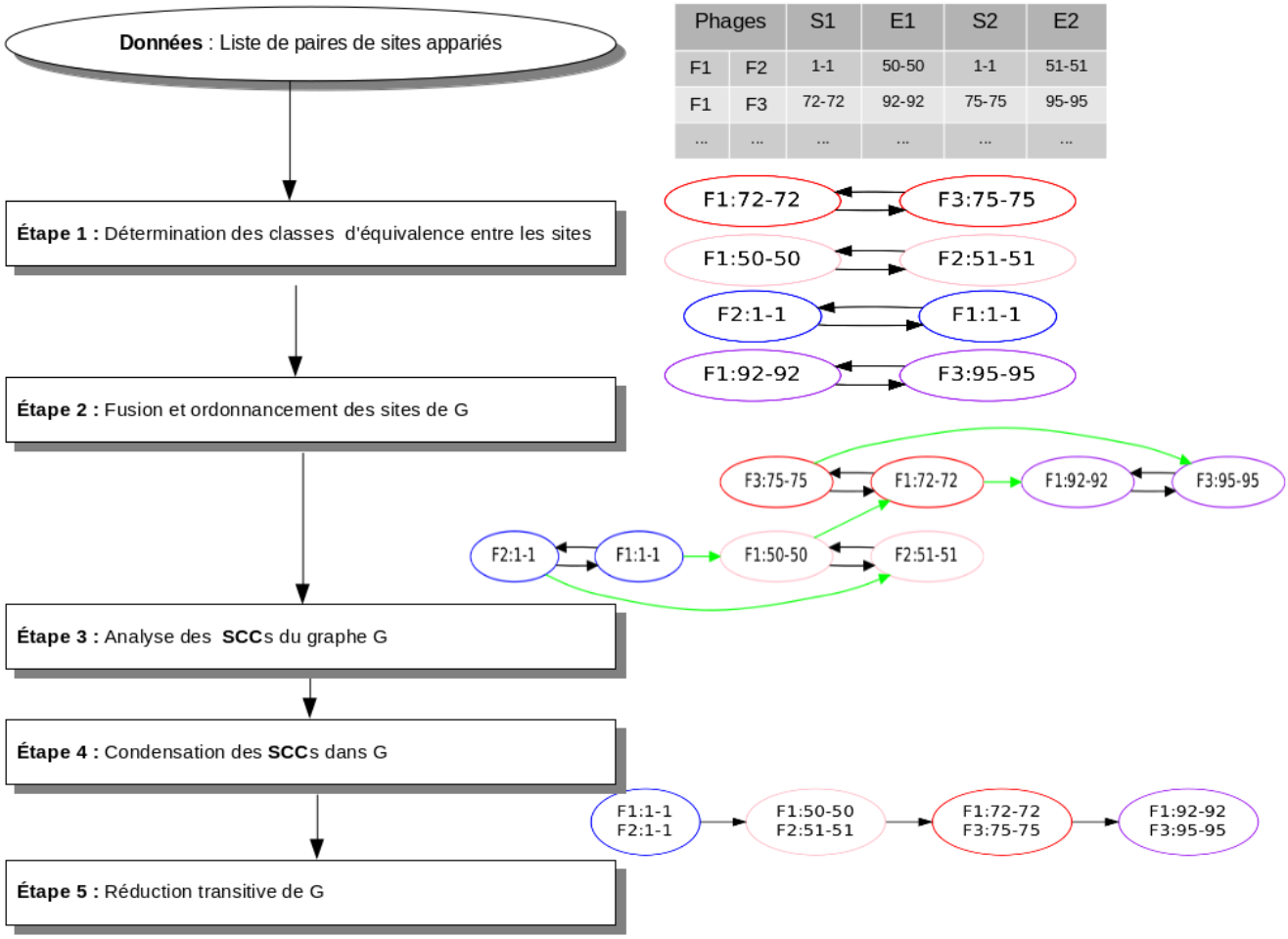


Fig 26: *Les cinq modules/étapes résumant la méthode de génération du graphe des régions à partir de la liste de sites.* ***S1****,* ***E1****,* ***S2****,* ***E2*** *: positions de début (S pour start) et de fin (E pour end) chez F1 et F2 respectivement,* ***SCC*** *: composante fortement connexe.*

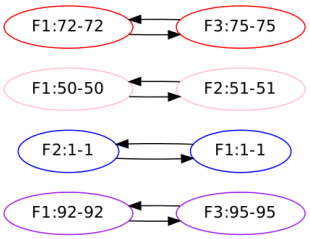


Fig 27: *Exemple de graphe SC des sites de correspondance. Les arcs sont qualifiés de correspondance.*

Il est à signaler que lors de l'ordonnancement des sites de correspondance, on procède à la fusion des extrémités des sites d'un même phage dont écart inférieur ou égal à un seuil fixé par l'utilisateur (quelques paires de bases) pour rectifier d'éventuelles "erreurs ou biais" dues au BLAST. La figure 29 représente un exemple de fusion où deux sites du phage F2, $F2:1-1$ et $F2:2-2$, ont été fusionnés pour donner lieu au site $F2:1-2$. L'étape 2 fait appel à deux nouvelles méthodes implémentées dans Networkx permettant d'extraire les successeurs et les prédécesseurs d'un nœud donné dans le graphe à savoir graph.successors(node) et graph.predecessors(node) (*cf.* code 10, 11, 12, 13 dans la section "annexes"). Ces deux fonctions ont chacune une complexité de l'ordre de $O(n)$. L'étape 2 a été programmé avec une complexité totale de $O(p)$ **x** $(llog(l)$ **+** l^2 **x** $(7O(n)$ **+**$2l) \approx O(nl^2)$ où n le nombre de sommet dans le graphe, l le

Algorithme 3 : Initiation d'un DAG (G) et détermination des classes d'équivalence (complexité : $O(n^2)$)

Entrées : Liste des paires des extrémités d'appariements L
Sorties : Graphe orienté G de correspondance des sites
Initialisation de G à un DAG vide
pour chaque p (s_i, s_j) *dans* L **faire**
  si s_i *n'est pas dans* G **alors**
    Ajouter le sommet s_i dans G ;
  si s_j *n'est pas dans* G **alors**
    Ajouter le sommet s_j dans G ;
  Ajouter l'arc $s_i \longrightarrow s_j$ dans G;
  Ajouter l'arc $s_j \longrightarrow s_i$ dans G
Retourner G

nombre maximal des sites par phages et p le nombre total de phages analysés.

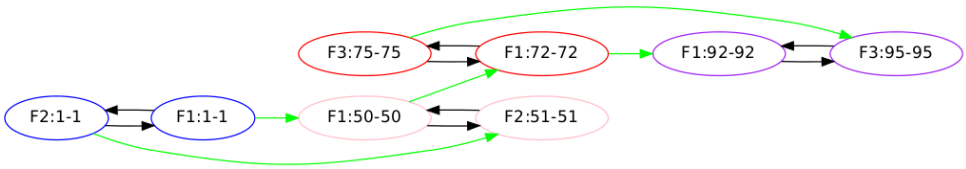


Fig 28: *Exemple de graphe d'ordre des sites généré à partir du graphe de correspondance des sites représenté dans la figure 27. Les arcs d'ordre sont en vert et ceux de correspondance sont en noir.*

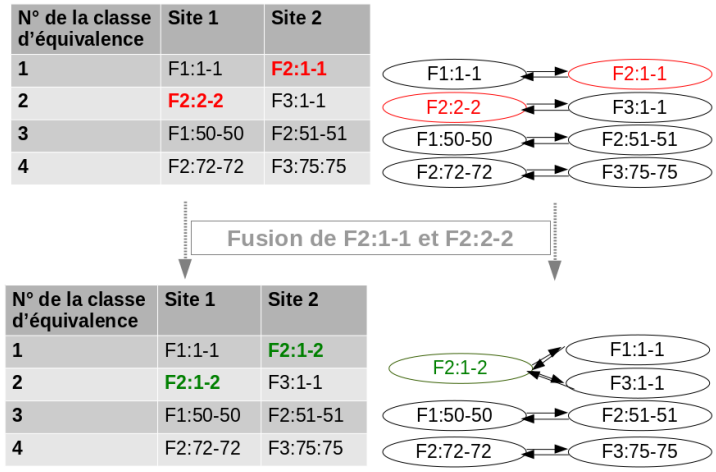


Fig 29: *Exemple de fusion de sites. La fusion des deux nœuds en rouge résulte en un seul nœud vert dont la position minimale et maximale sont respectivement la plus petite et la plus grande des positions des deux nœuds fusionnés (nœuds rouges). Le nœud résultant de la fusion (nœud vert) a comme successeurs et prédécesseurs ceux des deux nœuds fusionnés.*

③ Étape 3 : analyse des composantes connexes présentes dans le graphe d'ordre des sites G

Cette étape permet l'analyse, un par un, des différents sous-graphes des composantes fortement connexes (SCC) de G pris en entrée. L'analyse se déroule en plusieurs étapes successives (*cf.* figure 30 et algorithme 5). Pour chacune

Algorithme 4 : Fusion et ordonnancement des sites dans le graphe des correspondances des sites G (complexité : $O(nl^2)$)

Entrées : Liste des paires de sites d'appariement L, graphe de correspondance des sites G
Sorties : Graphe orienté d'ordre des sites G
pour chaque *phage P dans l'ensemble F* **faire**
  Lpos ← liste ordonnée des positions de f présents dans G;
  pour $i \leftarrow 0$ *;* $i \leq (Lpos.size() - 2)$ *;* $i++$ **faire**
    si $Lpos[i+1] - Lpos[i] <$ *seuil_fusion* **alors**
      $s \leftarrow$ créer_site(P, $Lpos[i]_1, Lpos[i+1]_2$) ;
      nouvelle_position ←créer_position($Lpos[i]_1, Lpos[i+1]_2$) ;
      $Sommet\_fusionne_i \leftarrow$ créer_sommet(P, $Lpos[i]$);
      $Sommet\_fusionne_{i+1} \leftarrow$ créer_sommet(P, $Lpos[i+1]$);
      Ajouter le sommet s dans le graphe G;
      successeurs $\leftarrow (successeurs(Sommet\_fusionne_i) \cup successeurs(Sommet\_fusionne_{i+1})) \setminus set(Sommet\_fusionne_i, Sommet\_fusionne_i)$;
      prédécesseurs $\leftarrow (predecesseurs(Sommet\_fusionne_i) \cup predecesseurs(Sommet\_fusionne_{i+1})) \setminus set(Sommet\_fusionne_i, Sommet\_fusionne_{i+1})$;
      pour chaque *successeur t dans successeurs* **faire**
        Ajouter l'arc $s \longrightarrow$ t dans le graphe G;
      pour chaque *prédécesseurs p dans prédécesseurs* **faire**
        Ajouter l'arc p $\longrightarrow s$ dans le graphe G;
      Supprimer $Lpos[i]$ de la liste $Lpos$;
      Supprimer $Sommet\_fusionne_i$ de G;
      Supprimer $Sommet\_fusionne_{i+1}$ de G;
      $L[i] \leftarrow$ nouvelle_position i=i-1

des SCC, si chacun des génomes présents dans la SCC en question est représenté par un seul et unique site, alors la composante est renvoyée sans traitement supplémentaire et déclarée sans problème (indiqué dans le graphe en vert). Dans le cas contraire, on procède à une analyse plus approfondie de la SCC en réalisant une opération de duplication permettant de caractériser les cas d'insertion par l'absence de *cycle* dans le graphe. Si il n'y a aucun cycle alors la composante a probablement un problème d'insertion sinon on fusionne les sites selon un deuxième seuil de fusion donné en paramètre à l'algorithme. Si la fusion ne modifie pas le graphe alors on conclut à la présence d'un problème de colinéarité et/ou de duplication en tandem et/ou de chevauchement plus grand que le seuil de fusion donnée. Dans le cas où cette fusion modifie le graphe, alors, on procède à la détection de cycle dans la duplication du graphe résultant de la fusion. Si on n'a pas de cycle, on conclut à la présence de problème d'insertion sinon c'est un problème de colinéarité et/ou de duplication en tandem et/ou de chevauchement.

Il est à noter que la duplication consiste en la duplication des arcs de correspondance présentes dans la composante SCC selon la fonction décrite dans l'algorithme 6. Cette transformation permet de remplacer chaque extrémité d'appariements (sommet) v par deux sommets v_g et v_d connectés par un arc (v_g, v_d). Les arcs entrant dans v seront remplacés par des arcs entrant dans v_d et inversement ceux sortant de v seront remplacés par des arcs sortant de v_g. Cette transformation est cruciale pour pouvoir analyser la présence/absence de cycle dans les SCC.

Le module s'appuie sur la fonction de duplication codé en $O(n^2)$ et sur deux autres fonctions de Networkx à savoir :

- ❒ La fonction strongly_connected_components() (*cf.* code 7) permettant de lister les composantes fortement connexes du graphe et qui est implémentée suivant l'algorithme de Tarjan (1972, [22]) mais intégrant des petites modifications proposées par Nuutila (1994, [18]. Cette méthode a une complexité constante de l'ordre de $O(n+a)$.
- ❒ La fonction is_directed() ((*cf.* code 8) qui permet de savoir si oui ou non le graphe contient un cycle. Cette

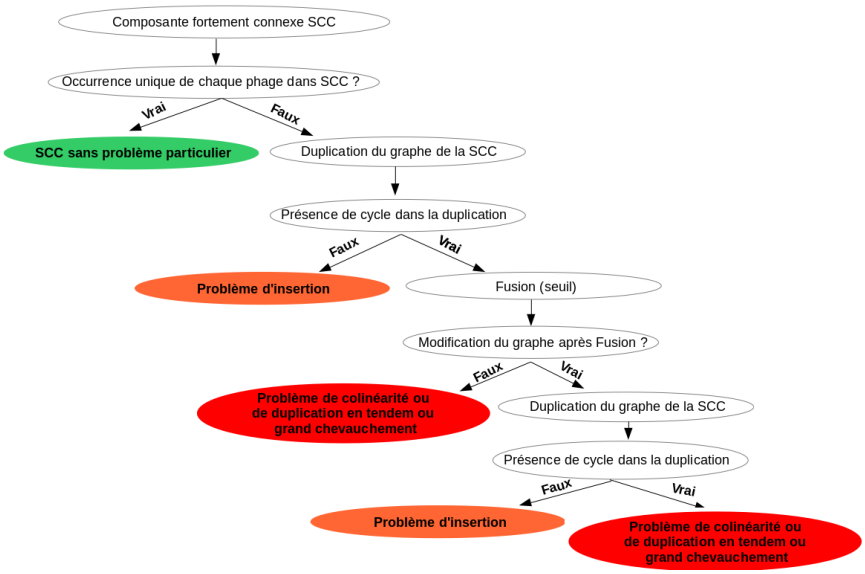


Fig 30: *Les différentes étapes du déroulement de l'étape 3.*

fonction fait appel à la fonction topological_sort() (*cf.* code 9) qui n'est possible que si et seulement si le graphe ne contient pas de cycle et qui se déroule en deux étapes. La première consiste en un parcours en profondeur du graphe ($O(n+a)$) et la deuxième en un tri des sommets du graphe ($O(nlog(n))$) avec n et a sont respectivement le nombre de sommets et d'arcs dans le graphe soit une complexité totale du tri topologique $\approx (O(nlog(n)+a))$. La complexité totale de cette étape est de l'ordre de $O(nlog(n)+n^2)$.

④ Étape 4 : Condensation du graphe G

Cette étape permet de déterminer les régions en contractant chacune des composantes fortement connexes définies par les sites de correspondance en un seul sommet. Le graphe résultant dit des régions est un graphe orienté acyclique. Pour pouvoir intervenir sur le choix des noms des nœuds, nous avons implémenté notre propre méthode de condensation au lieu d'utiliser celle implémentée dans Networkx. L'algorithme 7 donne le détail de l'algorithme utilisé et qui prend en entrée le graphe d'ordre des sites de correspondance SC après analyse de ses composantes fortement connexes. Cette étape a une complexité quadratique ($O(n^2)$) avec n est le nombre de sommet du graphe.

⑤ Étape 5 : Réduction transitive du graphe C

Comme son nom l'indique, cette étape consiste à enlever les arcs de transitivité du graphe obtenu à l'étape précédente et de générer le graphe final des régions (*cf.* algorithme 8. Pour permettre une première lecture intuitive du graphe obtenu, les sommets sont colorés en fonction du résultat de l'analyse des SCCs entreprise lors de la troisième étape (*cf.* paragraphe 6.2.3). On a fait le choix de représenter les sommets :

- ❒ ayant une insertion en orange.
- ❒ ayant un problème de colinéarité /chevauchement (présence de cycle) en rouge.
- ❒ n'ayant aucun problème en vert.

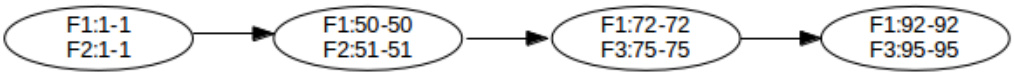


Fig 31: *Exemple de graphe réduit généré à partir du graphe d'ordre présenté dans la figure 28.*

Algorithme 5 : Analyse des composantes connexes du graphe G (complexité : $O(n^2)$)

Entrées : Graphe d'ordre sur les sites d'équivalence, seuil-fusion
Sorties : Graphe G connecté
$Liste_{cg} \leftarrow$ liste des sous_graphes des composantes fortement connexes dans G;
pour chaque *sous_graphe cg dans* $Liste_{cg}$ **faire**
    $D_i \leftarrow$ dictionnaire dont les clés sont les phages et les valeurs sont les positions présents dans cg;
    boolean←occurrence_multiple_phage(D_i);
    si *boolean = False* **alors**
        Initialisation de h à un DAG vide;
        $h \leftarrow$ duplication(cg) ;
        si *il* $\exists$ *un cycle dans h* **alors**
            $f \leftarrow$ copier(cg);
            cg $\leftarrow$ fusion(G, cg, seuil_fusion) ;
            si $cg = f$ **alors**
            Problème de colinéarité, de grand chevauchement ou de duplication;
            sinon si $cg \neq f$ **alors**
                $D_{aprs\_Fusion} \leftarrow$ dictionnaire dont les clés sont les phages et les valeurs sont les positions présents dans cg ;
                boolean $\leftarrow$ occurrence_multiple_phage(D_{aprs_Fusion}) ;
                si *boolean == False* **alors**
                    Initialisation de h à un DAG vide;
                    $h \leftarrow$ duplication(f) ;
                    si *il* $\exists$ *un cycle dans h* **alors**
                        Problème de colinéarité, de grand chevauchement ou de duplication;
                    sinon
                        Problème d'insertion ou de duplication simple ;
                        caractérisation_problème(cg);
                sinon
                    Problème de chevauchement inférieur au seuil de fusion réglé ;
            sinon
                Problème d'insertion ou de duplication simple ;
                caractérisation_problème(cg);
        sinon
            Absence de problème;

6.3 Complexité de l'algorithme

L'algorithme dans sa version actuelle a une complexité polynomiale de l'ordre de $O(n^4)$ comme le résume le tableau 3.

Algorithme 6 : La fonction de duplication(cg) (complexité : $O(nlog(n)+n^2)$)

```
/*génère à partir d'un graphe d'une SCC le graphe correspondant (T) où les nœuds sont dupliqués G et D*/
Initialisation de h à un DAG vide
pour chaque node dans cg.nodes() faire
    nodeG← node+"G"
    nodeD← node+"D"
    Ajouter le sommet nodeD dans h;
    Ajouter le sommet nodeG dans h;
    Ajouter l'arc nodeG ← nodeG dans h
    successeurs ← cg.successeurs(node);
    prédécesseurs ← cg.prédécesseurs(node);
     pour chaque successeur t dans successeurs faire
        Ajouter nodeG ⟶ t dans le graphe cg;
    pour chaque prédécesseur p dans prédécesseurs faire
        Ajouter p ⟶ nodeD dans le graphe cg;
    retourner h
```

Algorithme 7 : Condensation du graphe G (complexité : $O(n^2)$)

```
Entrées : un graphe d'ordre des sites G connecté
Sorties : le graphe G réduit
SCC← nx.strongly_connected_components(G)
Initialisation de C à un DAG vide ;
Initialisation de liste_sommet à liste vide;
Initialisation de mapping à une table de hachage vide;
pour chaque composante_fortement_connexe scc dans SCC faire
    Trier scc par ordre alphabétique;
    classe ← la jointure de scc ;
    Ajouter le sommet classe dans C ;
    pour chaque sommet s dans scc faire
        mapping[s] ← classe;
pour chaque sommet u, v dans les arcs de G faire
    si mapping[u]! = mapping[v] alors
        Ajouter l'arc mapping[u] → mapping[v] dans C;
retourner C
```

6.4 Caractérisation des différents types de sommet dans le graphe des régions

Dans cette section, nous caractérisons quelques types de sommets en se basant sur les propriétés de leur sous-graphe. Dans certains cas, il est difficile déduire avec certitude le type de réarrangements d'un sommet en s'appuyant uniquement sur le graphe. Il s'avère des fois utile et essentiel de consulter les données du BLAST pour caractériser un sommet avec certitude.

À chaque sommet dans le graphe des régions R correspond une composante fortement connexe dans le graphe des sites de correspondance SC. Nous travaillerons, tout au long de cette section, sur les sous-graphes SCg(S,A) des sommets de R. Certains de ses sommets peuvent correspondre à des régions d'insertion ou de duplication ou de chevauchement ou de non colinéarité ; qu'on qualifie de sommets *particuliers*. Les autres sommets sont dits sommets *généraux*.

Notations générales : Ci-dessous quelques notations générales qui seront utilisées dans cette section :

Algorithme 8 : Réduction transitive du graphe C (complexité : $O(n^4)$)

Entrées : un graphe acyclique orienté C et condensé
Sorties : le graphe C réduit
tant que *H n'est pas vide* **faire**
  source0 $\leftarrow S(H)$ (reçoit les sources de H);
  pour chaque *sommet* s_0 *dans source1* **faire**
    Supprimer s_0 du grahe H;
  source1 $\leftarrow S(H)$ (reçoit les sources de H);
  liste_prédécesseurs $\leftarrow$ Pred(s) dans C
  liste_descendants $\leftarrow$ Desc(s) dans C
  pour chaque *sommet s dans source1* **faire**
    liste_prédécesseurs $\leftarrow$ Pred(s) dans C;
    liste_descendants $\leftarrow$ Desc(s) dans C;
    pour chaque *Prédécesseur Pred dans liste_prédécesseurs* **faire**
      pour chaque *Successeur Succ dans liste_descendants* **faire**
        si *il* $\exists$ *un arc succ* $\rightarrow$ *pred dans C* **alors**
          Supprimer l'arc *succ* $\rightarrow$ *pred* dans C;
retourner C

Tab. 3: *La complexité en temps, dans les pires des cas, des différentes étapes de notre algorithme.*
l : nombre maximal de sites par phage, n : le nombre de nœuds dans le graphe pris en entrée à chaque étape sauf pour la première étape où n est la taille de la liste des sites prise en entrée.

Étapes	Complexité en temps
Initialisation d'un DAG et détermination des classes d'équivalence	$O(n^2)$
Fusion et ordonnancement des sites de correspondance dans G	$O(nl^2)$
Analyse des composantes connexes	$O(nlog(n)+n^2)$
Condensation du graphe G	$O(n^2)$
Réduction transitive du graphe C	$O(n^4)$

- n et m sont respectivement le nombre de sommet et d'arc dans SCg.
- $\mathscr{F}$ est l'ensemble des phages représentés par les sommets ($\equiv$ sites) de **tout** les SCg du graphe des régions R.
- F est l'ensemble des phages représentés par les sommets ($\equiv$ sites) d'**un** SCg.

6.4.1 Caractérisation d'un sommet général

Soit F l'ensemble des phages présent dans le sommet S. S est déclaré sans problème s'il vérifie la propriété suivante :
- $\forall P_x \in F$, P_x est représenté par un seul site (position) $s \in S$.

Propriétés : Étant donné que pour chaque arc de correspondance (x,y) dans SCg, il existe un arc (y,x) et que SCg est connexe alors ce dernier est fortement connexe.
- Le nombre d'arcs m présents dans SCg est : $2(n-1) \leq m \leq n(n-1)$.
- $\forall s \in S$, $d_+(x) = d_-(x)$.
- $\forall s \in S$, $2 \leq d(s) \leq 2n-2$. $d(s)$ est paire.
- Le graphe de la composante connexe est forcément Eulérien. En effet, étant donné que $\forall s \in S$, $d(s)$ est paire, alors il n'existe pas d'arc pont (isthme) dans SCg.

Remarque 3 *:*
Dans le cas où les sites sont définis à partir d'appariements fortement homologues (très similaires) alors :

- ❒ $\forall\, s \in S$, *s est le centre du graphe SCg.*
- ❒ *Le graphe SCg admet une seule clique maximale qui est le graphe lui même.*
- ❒ *SCg est hamiltonien (contient un cycle hamiltonien).*

6.4.2 Caractérisation d'un sommet particulier avec une insertion simple :

Propriétés : Soit F l'ensemble des phages présent dans SEg, on dit que SEg correspond à une région ayant une insertion si et seulement si :

- ❒ il existe un phage $P_x \in F$ tel que il existe deux sites s_1 et s_2 avec $s_1 \in P_x$ et $s_2 \in P_x$.
- ❒ $\forall\, P_y \in F \setminus \{P_x\}$, P_y est représenté par un seul site.
- ❒ s_1 et s_2 ont au moins un voisin en commun s ($s \in \upsilon(s_1, SEg) \bigcap \upsilon(s_2, SEg)$) avec $s \notin P_x$.

Sans perte de généralité, nous ne considérons que le cas où $s_1 < s_2$ (*cf.* la définition 21 de la relation d'ordre sur les sites). Si, les 3 conditions sont réunies alors on conclut la présence d'insertion du facteur compris entre s_1 et s_2 dans P_x et respectivement une délétion dans l'ensemble des phages défini par $F \setminus \{P_x\}$ (c. figure 32 illustrant l'insertion à différentes étapes de notre méthode).

Remarque 4 *: Dans le cas où seules la 1^{ere} et la 2^{eme} condition sont réunies et si le seul et l'unique chemin eulérien ayant pour source s_1 a comme extrémité terminale s_2 on parle alors de* ***fausse insertion induite par transitivité****.*

Lemme 1 *Si $s_1 \in P_x$ et $s_2 \in P_x$ ont au moins un voisin en commun $s \notin P_x$, il y a insertion chez P_x*

Preuve 1 *Soient deux phages P_x et P_y avec P_x est représenté par deux sites s_1 et s_2 (avec $s_1 < s_2$; sans perte de généralité) et P_y est représenté par un seul site s.*

Nous avons :

① *s et s_1 sont en relation de correspondance puisqu'ils appartiennent à deux phages différents P_y et P_y respectivement. Donc, les deux sites correspondent aux sites de début ou de fin d'un appariement entre leur phages respectifs.*

② *De la même façon, s et s_2 sont en relation de correspondance puisqu'ils appartiennent à deux phages différents P_y et P_y respectivement. Donc, les deux sites correspondent aux sites de début ou de fin d'un appariement entre leur phages respectifs.*

Sachant ① et ② et que $s_1 < s_2$, on déduit que :

③ *s et s_1 correspondent aux sites de fin d'un appariement entre P_x et P_y.*

④ *s et s_2 correspondent aux sites de début d'un appariement entre P_x et P_y.*

De ③ et ④, on déduit qu'il existe sur P_x un facteur de longueur s_2-$s_1 \neq 0$ qui n'est pas présent chez P_y. On parle d'insertion simple ce facteur entre s_1 et s_2 chez le phage P_x.

6.4.3 Caractérisation d'un sommet particulier avec présence de contradiction de colinéarité

① Cas simple de la présence de réarrangement de type inversion : vraie contradiction de colinéarité :
Soit $\mathscr{L}$ la liste des ensembles des paires de sites des extrémités d'appariements dans $\mathscr{S}$ entre deux génomes P_x et P_y. On dit qu'il y a un problème de colinéarité 2 à 2 si le tri numérique croissant de $\mathscr{L}$ selon le premier élément des sites dans $\mathscr{S}$ ne correspond pas au tri numérique de la liste $\mathscr{L}$ selon le deuxième élément de paires des sites dans $\mathscr{S}$.

On note $\mathscr{L} = [(P_{x1}, P_{y1}), (P_{x2}, P_{y2}), \ldots, (P_{xn}, P_{yn})]$. P_{xi} est un entier correspondant à la position de l'extrémité d'un appariement chez P_x et (P_x, P_y) est la paire d'entier défini par l'une des extrémités (soit début soit fin) d'un appariement entre 2 phages P_x et P_y.

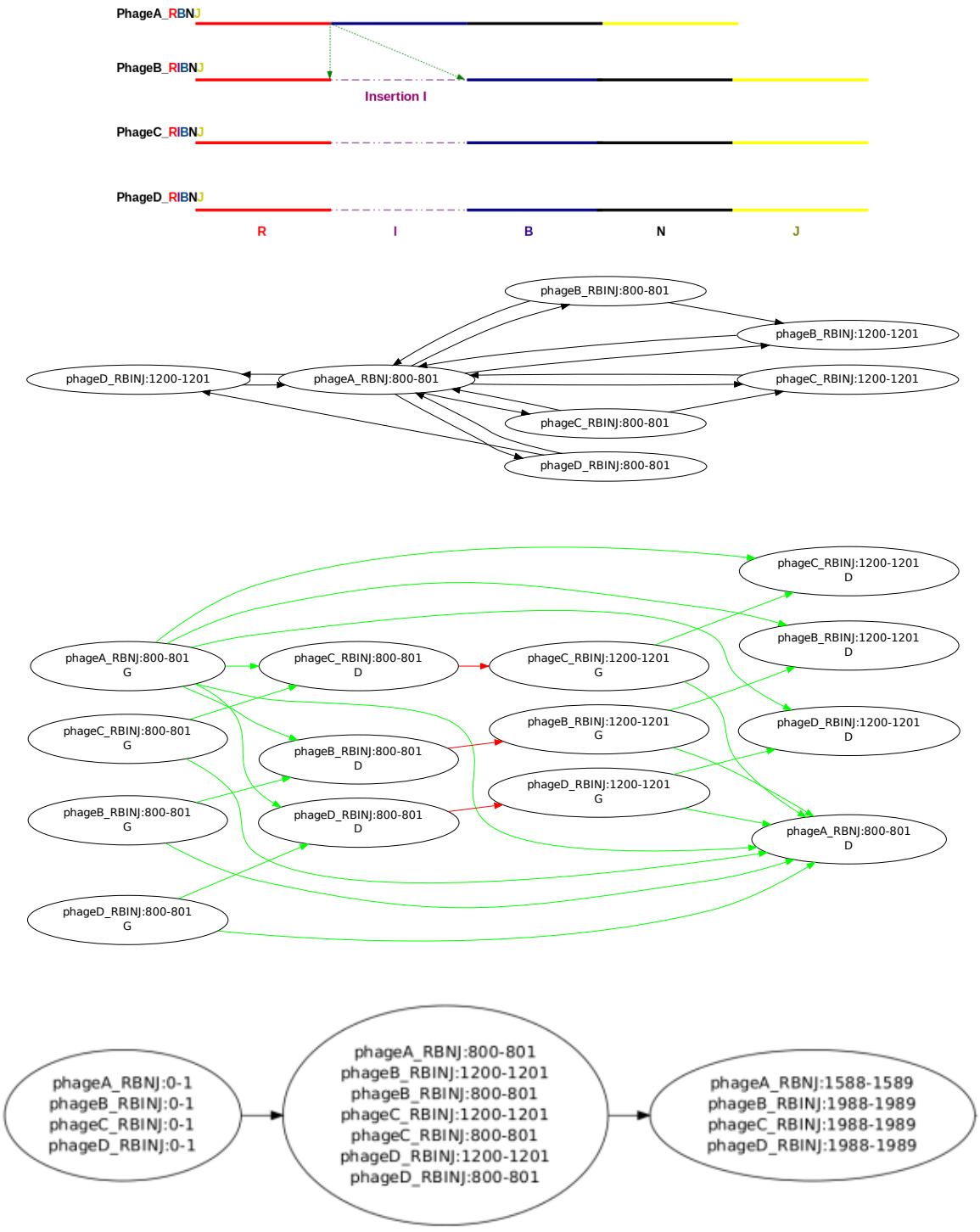


Fig 32: *Illustration de la traduction de la présence d'insertion aux différentes étapes de notre méthode. Graphes du haut en bas 1- Présence d'insertion simple au niveau de l'alignement des séquences. 2- La composante fortement connexe scc correspondant à la zone d'insertion. 3- La duplication de la scc et 4- le graphe des régions correspondant à l'alignement des 4 génomes.*

S'il existe un couple de phages (P_x, P_y) $(P_x \in F, P_y \in F)$ en relation de correspondance avec chacun est représenté par au moins 2 sites dans le graphe. S'il n'y a pas de cycle eulérien passant par l'ensemble et uniquement l'ensemble des sites issus de l'appariement entre P_x et P_y (cf figure

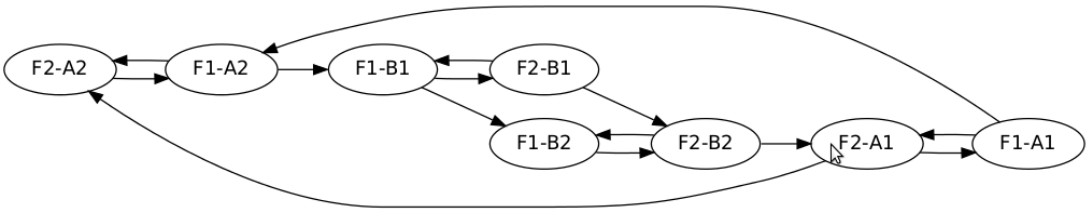


Fig 33: *Exemple de scc issu d'un appariement entre deux phages avec présence de contradiction de colinéarité (absence de cycle eulérien.*

② **Cas de présence de contradiction de colinéarité avec ou sans inversion** :
On considère un ensemble de sites $\mathscr{S}$ associés aux facteurs $\mathscr{F}$. Les sites dans chaque phage sont ordonnés selon leur position sur le génome. On définit le graphe G(V,A) tel que à chaque site dans $\mathscr{S}$ correspond 2 arc $v, v' \in V$. G contient 3 types d'arcs à savoir :

- ❒ **Arc d'ordre ou d'adjacence** : $(v', u) \in A$ si le site v est le site positionné immédiatement avant le site u dans un phage donné P_x,
- ❒ **Arc intra-site** : $(v', u) \in A$ pour les arcs correspondant au site v,
- ❒ **Arc interphage ou arc de correspondance** : $(v, u'), (u, v') \in A$ si v est le site dans P_x correspondant au site u dans P_y.

Lemme 2 *La présence de cycle dans le graphe traduit l'absence de colinéarité.*

Preuve 2 *[par contradiction] Elle découle directement de la définition de G selon laquelle une séquence cyclique de facteurs (contradiction de colinéarité) implique un cycle dans G.*
Supposons qu'on a un cycle c = $v_0, v_1, \ldots, v_\ell$ avec $v_0 = v_\ell$). Nous transformons le cycle c en une séquence de facteurs correspondant à la définition de la contradiction de colinéarité. Transformer le cycle c en une séquence cyclique de sous-facteurs $f_0, f_1, \ldots, f_k$ ($f_0 = f_k$) en procédant comme suit :
1- On remplace chacun des arcs de correspondance (v_i, v_{i+1}) par le/les facteurs auxquels ils appartiennent.
2- On supprime les autres v_i.
On appelle m *la fonction de mise en correspondance des arcs aux facteurs impliquées dans la transformation. m_d^{-1} et m_f^{-1} permettent la mise en correspondance des facteurs respectivement à leur sites de début et de fin.*

*Le cycle ainsi transformé contient une contradiction de colinéarité. Chaque paire de facteurs obtenus f_i, f_{i+1} appartenant à deux phages différents correspond à l'apparition d'un même facteur dans ces 2 phages puisqu'ils résultent d'un arc de correspondance. Si on admet qu'il n'existe pas de f_i, f_{i+1} positionnés sur le même génome, alors, nous avons obligatoirement $v_i = m_f^{-1}(f_i)$ et $v'_{i+1} = m_f^{-1}(f_{i+1})$. Toutefois, ceci implique $v_{i+2} = m_d^{-1}(f_{i+2})$, puisque (v'_{i+1}, v_{i+2}) doit correspondre à un arc de correspondance, ce qui signifie que f_{i+1} et f_{i+2} appartiennent au même génome et donc une contradiction (*cf. *figure 35).*

6.4.4 Caractérisation d'un sommet particulier avec problème de chevauchement

Propriétés : Il est important de signaler que le chevauchement est un cas particulier de contradiction de colinéarité. Un sommet a un problème de chevauchement :
- S'il existe au moins 2 phages $P_x \in F$ et $P_y \in F$ tels que il existe au moins 4 sites s_1, s_2, s_3 et s_4 avec $s_1 \in P_x$, $s_2 \in P_x$, $s_3 \in P_y$ et $s_4 \in P_y$.
- S'il existe un circuit passant uniquement par les 4 sites.

Preuve 3 *Sans perte de généralité, admettons que $s_1 < s_2$ et $s_3 < s_4$.*
Alors il existe deux arcs d'ordre dans le graphe $s_1 \to s_2$ et $s_3 \to s_4$.

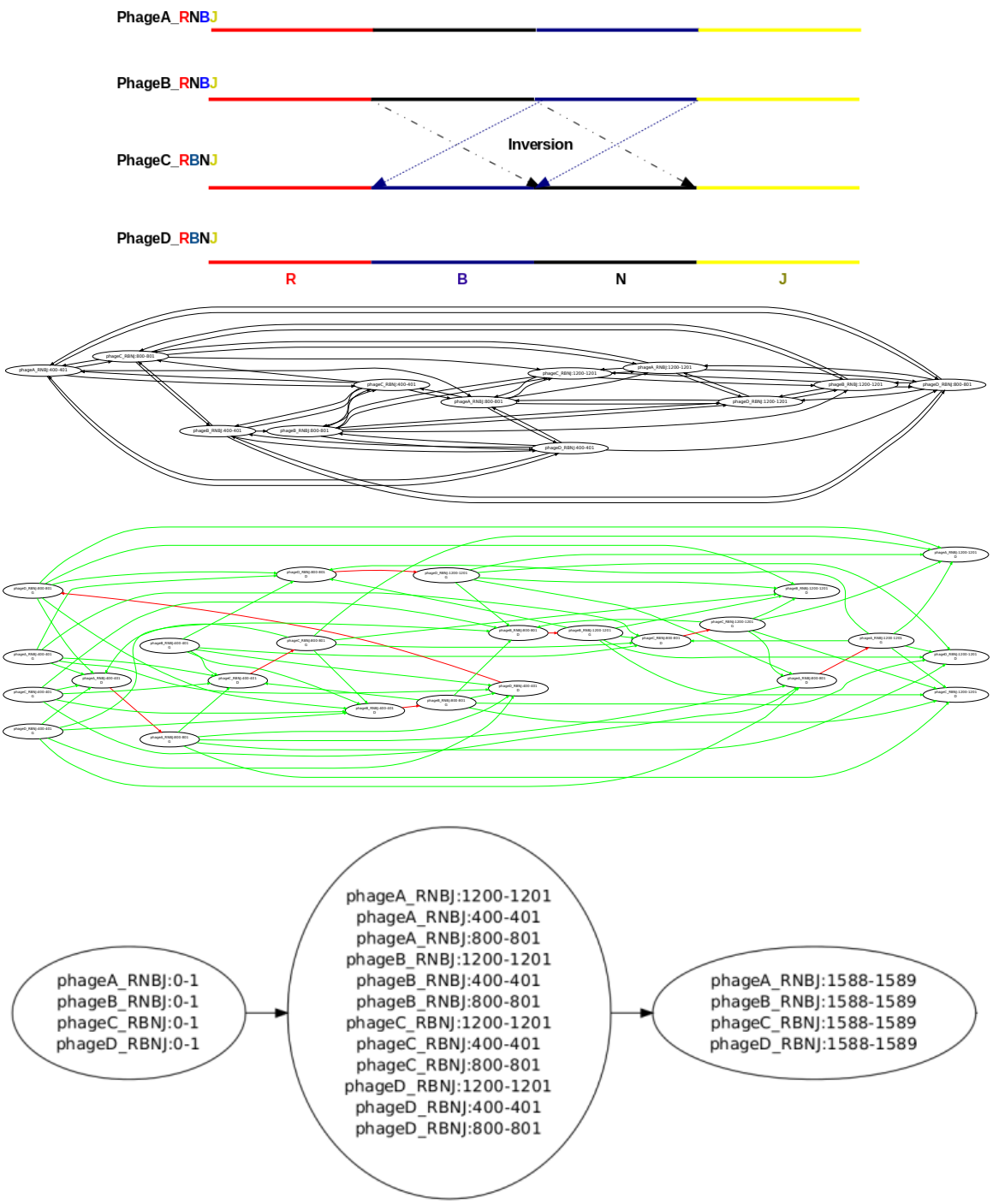


Fig 34: *Illustration de la traduction de la présence d'inversion à différentes étapes de notre méthode. Graphes du haut en bas 1- Présence d'inversion simple au niveau de l'alignement des séquences. 2- La composante fortement connexe scc correspondant à la zone d'inversion. 3- La duplication de la scc et 4- le graphe des régions correspondant à l'alignement des 4 génomes.*

Ainsi, on ne peut avoir un cycle passant uniquement par les quatre sites que si et seulement si on a les arcs $s_4 \rightarrow s_1$ et $s_2 \rightarrow s_3$ dans G.
Étant donné que $s_1 \in P_x$, $s_2 \in P_x$, $s_3 \in P_y$ et $s_4 \in P_y$, on conclut que s_4 et s_1 sont en relation de correspondance de même que s_2 et s_3.
Sachant que $s_1 < s_2$ et $s_3 < s_4$ alors il y a chevauchement entre deux appariements A_1 et A_2

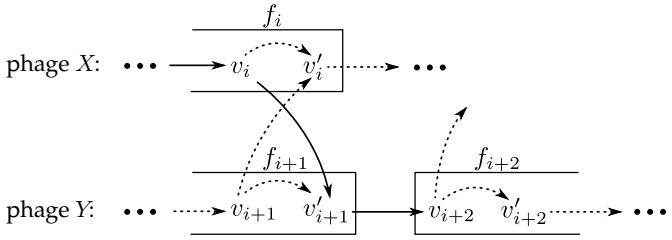


Fig 35: *Un cycle dans le graphe G implique l'existence d'au moins deux facteurs à partir du même phage (*cf. *la preuve 2 pour plus d'explication.*

ÉXPERIMENTATIONS

7 EXPÉRIMENTATIONS

7.1 PRÉTRAITEMENT DES DONNÉES

Lors de ce stage, nous avons testé notre méthode sur 3 jeux de données à savoir 1 et 2 jeux de données composés respectivement de phages de *Lactobacillus lactis* et de *Staphylococcus aureus*. L'annexe 9.3 détailles les différentes séquences utilisées dans les 3 jeux de données. La totalité des séquences ainsi que leur annotation ont été extraites de la base NCBI [14].

Dans cette partie, nous mettrons particulièrement l'accent sur les résultats de l'étude de phages lytiques de *L. lactis*. Cette dernière est utilisée généralement dans l'industrie fromagère notamment pour la fermentation du lactose en acide lactique. Les dégâts de l'infection de ce type de production par des phages de *L. lactis* sont particulièrement désastreuses pouvant aller jusqu'à l'arrêt de la fermentation d'autant plus que ces phages se trouvent dans le lait cru et résistent à la pasteurisation [7]. Les phages de *L. lactis* appartiennent à l'ordre des *Caudovirales* et sont très diverses à la fois génétiquement et morphologiquement. Cet ordre est représenté par 3 familles [7] :

- ❒ *Myoviridae* : caractérisée par de longues queues contractiles.
- ❒ *Siphoviridae* : caractérisée par de longues queues non contractiles.
- ❒ *Podoviridae* : caractérisée par de courtes queues.

Avant de tester notre algorithme, il était important de pré-traiter les séquences des phages avant de procéder à leur alignement deux à deux. En effet, sous l'hypothèse de la colinéarité des génomes analysés et étant donnée qu'ils sont circulaires, nous avons décidé de travailler avec des séquences génomiques comprises entre deux séquences/modules d'ancrage à savoir le gène **portal** et le gène de la **lysine**. La taille des séquences analysées varie de 14333 à 15198 bp avec une moyenne de 14778 ± 370 bp ce qui représente près de 45% de la taille initiale des génomes étudiées. Le tableau 9 récapitule les 27 séquences analysées à savoir : leur numéros d'accession, leur nom, ainsi que les positions de début et de fin des séquences extraites sur le génome complet.

7.2 DESCRIPTION DES RÉSULTATS DE L.LACTIS

7.2.1 ALIGNEMENT DES SÉQUENCES

La première étape de notre algorithme consiste en l'alignement 2 à 2 des séquences extraites en utilisant BLAST [15] en local (-version :BLASTN 2.2.25+) . Pour cela, il était nécessaire de créer une base de données locale refermant les 27 séquences en utilisant la commande makeBLASTdb (*cf.* création d'une base de donnée locale avec makeblastdb de blast+, annexe 9.3.1). 4654 appariements ont été trouvé et ayant une longueur variant de 29 à 15150 bp et une moyenne de 1952 ± 3114 bp (*cf.* figure 36). 50% et 80% des appariements ont respectivement une longueur inférieure à 1000 bp et à 3000 bp.

7.2.2 EFFET DE LA FUSION

Dans le but de tester l'effet de l'étape de la fusion, nous avons tester notre algorithme avec (seuil de fusion = 5 pb) et sans fusion (seuil de fusion = 0 pb). Les résultats obtenu avec les différents jeux de données sont résumés dans le tableau 5 et révèlent qu'un seuil de fusion de l'ordre de 5bp ne permet pas de régler le problème des nœuds "rouges" ce qui signifie qu'ils résultent d'un phénomène biologique (ex : non colinéarité) plutôt qu'un biais lié au BLAST. Toutefois, l'un des 3 sommets "rouges" présents dans le graphe des régions obtenu avec le deuxième jeu de données de*S.aureus* est déclaré en rouge suite à la présence de chevauchement entre deux appariements de longueur supérieure au seuil de fusion choisi (28bp, donnée non présentée ici). Il est à noter que nous avons choisi d'utiliser un faible seuil de fusion pour ne pas modifier les données initiales.

14. National Center for Biotechnology Information :www.ncbi.nlm.nih.gov
15. Basic Local Alignment Search Tool

Tab. 4: *Récapitulatif détaillant les 27 séquences de L.lactis analysées.*

Accession	Nom du phage	Position de début	Position de début
JQ740787.1|	Phage_L1	2924	17648
JQ740788.1|	Phage_L2	2913	17648
JQ740789.1|	Phage_L3	3065	18262
JQ740790.1|	Phage_L4	3065	18262
JQ740791.1|	Phage_L5	2913	17648
JQ740793.1|	Phage_L6	2913	17342
JQ740794.1|	Phage_L7	2913	17342
JQ740795.1|	Phage_L8	3065	18262
JQ740796.1|	Phage_L90	3065	18263
JQ740797.1|	Phage_L101	3065	18262
JQ740798.1|	Phage_L11	2913	17342
JQ740799.1|	Phage_L12	2913	17306
JQ740800.1|	Phage_L13	2913	17306
JQ740801.1|	Phage_L14	2913	17246
JQ740802.1|	Phage_L15	2913	17648
JQ740803.1|	Phage_L16	2913	17306
JQ740804.1|	Phage_L17	3087	18284
JQ740805.1|	Phage_L18	3087	18285
JQ740806.1|	Phage_L19	2913	17306
JQ740807.1|	Phage_L20	3087	18284
JQ740808.1|	Phage_L21	3087	18284
JQ740809.1|	Phage_L22	2913	17342
JQ740810.1|	Phage_L23	2923	17316
JQ740811.1|	Phage_L24	2913	17411
JQ740812.1|	Phage_L25	3087	18284
JQ740813.1|	Phage_L26	3087	18284
JQ740814.1|	Phage_L27	2913	17342

Tab. 5: *Récapitulatif des résultats obtenus avec les différents jeux de données.*

Jeux de données	$1-S.Aureus^{*}$		$2-S.Aureus^{**}$		$3-Lactobacillus^{***}$	
Seuil de fusion (bp)	F=0	F=5	F=0	F=5	F=0	F=5
Nombre de génomes	31				27	
Statistiques						
Nombre de site de correspondance	342		1813		1031	
Nombre des classes d'équivalence	61		252		60	
Nombre d'arc dans Gr	57	57	272	272	60	60
Nombre de sommet dans Gr	61	61	252	252	57	57
Nombre de sommets verts dans Gr	59	59	249	249	55	55
Nombre de sommets rouges dans Gr	2	2	3	3	2	2
Nombre de sommets oranges dans Gr	0	0	0	0	0	0
Légende : *portal-tape measure, **holin-RinA, ***portal-lysine						

7.2.3 TEMPS D'EXÉCUTION

Malgré une complexité avoisinante $O(n^4)$, notre programme s'exécute en un temps global de l'ordre de quelques seconde (2 à 3 secondes) comme le détaille le tableau 6.

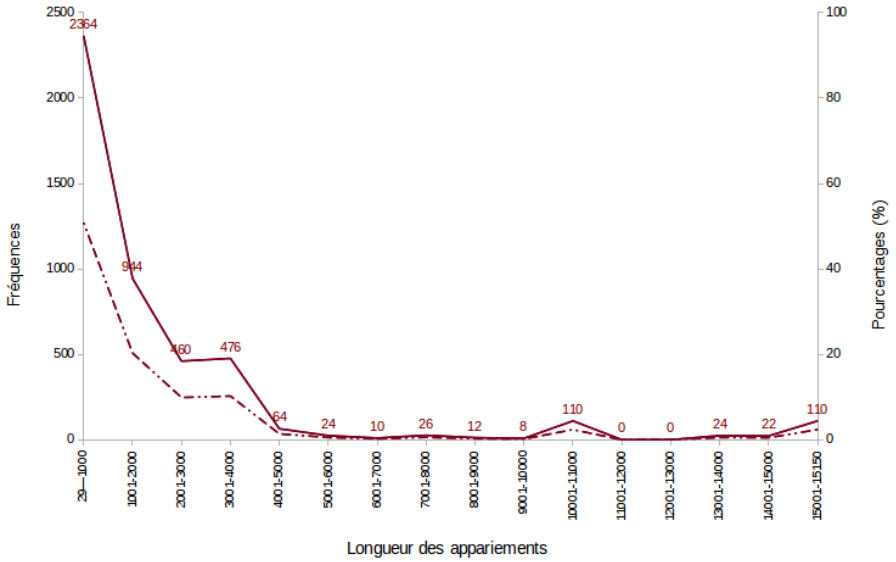


Fig 36: *Distribution de la fréquence des longueurs (en ligne continue) des appariements ainsi que leur pourcentage (en ligne pointillé).*

Tab. 6: *Exemple de temps en secondes des différentes étapes de notre algorithme.*

Étape	**Temps (s)**
Alignements 2 à 2 des génomes et extraction des sites de correspondance	0.660591
Initialisation d'un DAG et détermination des classes d'équivalence	0.619580
Fusion et ordonnancement des sites de correspondance dans G	0.024228
Analyse des composantes connexes	0.534617
Condensation du graphe G	0.321483
Réduction transitive du graphe C	0.534963
Temps total	**2.695462**

7.2.4 Description du graphe des régions

Le graphe de régions est composé de 57 sommets (*cf.* figure 37) définis en moyenne par 18 ± 10 sites. La figure 38 montre la distribution du nombre de sites par sommet. Les sommets définis par un nombre de sites supérieur au nombre de génomes analysés sont obligatoirement des sommets particuliers tandis que l'inverse n'est pas vrai. En effet, les sommets définis par un nombre de sites inférieur ou égal au nombre de génomes analysés ne sont obligatoirement des sommets généraux. Deux sommets particuliers (rouge) SP1 et SP2 composés respectivement de 63 et 55 génomes ont été trouvés. POur plus de détails, les résultats obtenus avec les 3 jeux de données sont résumés dans le tableau 5.

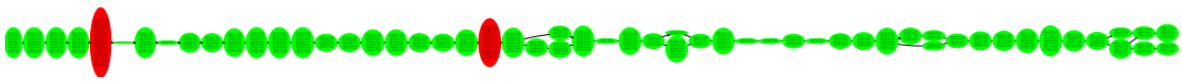


Fig 37: *Aperçu du graphe des régions obtenu avec le jeu de données de* L.lactis.

Zoom sur le sommet particulier SP1 :
SP1 correspond à une zone de non colinéarité entre les appariements. En effet, l'analyse des appariements impliquent les sites définissant ce sommet révèle la présence de deux groupes de génomes à savoir le premier à une configuration *ab* et le deuxième à une configuration *ba* avec *a* et *b* sont deux facteurs différents. Le premier groupe renferme 11 génomes tandis que le deuxième groupe est défini par 16 génome (*cf.* figure 39).

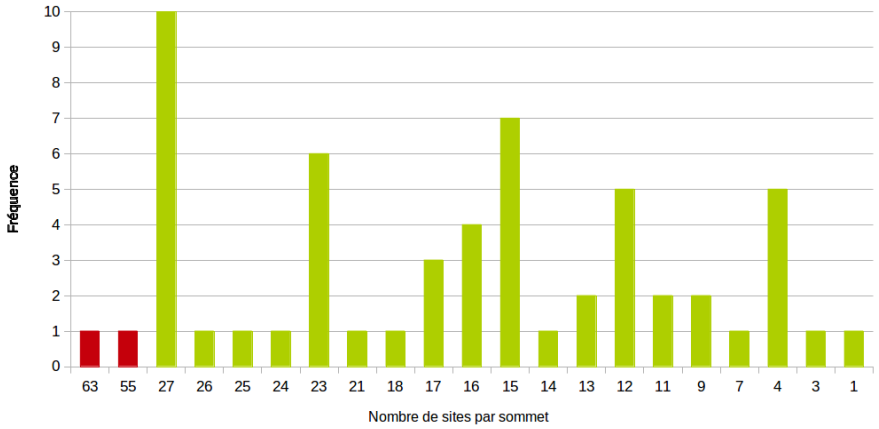


Fig 38: *Distribution des fréquences de nombre de sites par sommet dans le graphe des régions. En rouge les sommets de type particulier et en vert les sommets de type général.*

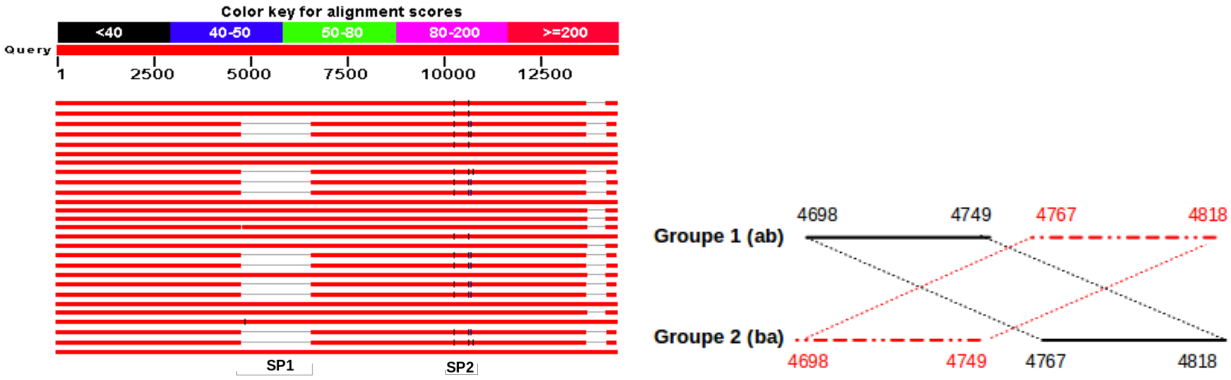


Fig 39: *À gauche, le résultat du BLAST des 27 séquences de L.lactis mettant en évidence les deux zones SP1 et SP2 et à droite le détail des appariements situés dans la zone SP1.*

Zoom sur le sommet particulier SP2 :

SP1 correspond à une zone d'appariements très chevauchants avec des chevauchements de longueurs supérieures 5bp (*cf.* figure 39).

I CONCLUSIONS & PERSPECTIVES

8 Conclusions et perspectives

8.1 Conclusions

À l'issue de ce stage de recherche autour de la détection des modules à partir des résultats des alignements, nous avons proposé et formalisé une méthode qui permet de générer un graphe des régions résumant les données d'appariements deux à deux tout en essayant de caractériser les éventuels réarrangements génétiques dans les différentes régions et dans la mesure du possible. En effet, prendre en considération les innombrables cas possibles de réarrangement était juste impossible. On s'est limité à la caractérisation des cas les plus rencontrés biologiquement et les plus probables. Par ailleurs, Il était nécessaire, pour juger des performances de notre algorithme, de le faire fonctionner dans des conditions réelles mais aussi sur des données simulées. À cet effet, nous avons donc mis au point des scripts permettant de simuler les différents cas de réarrangements génomiques.

Par ailleurs et sur le plan personnel, six mois de stage purement théorique au sein du LIRMM m'a était bénéfique sur le plan professionnel. L'approche d'un tel aspect de la bioinformatique a fortement amélioré mes capacités d'une part d'adaptaion et d'autre part m'a initié à la bioinformatique théorique. En effet, ce stage m'a permis de me former en matière de théorie des graphes et de l'appliquer à une problématique biologique qui est l'alignement de séquences. C'est vrai que parfois, ma motivation personnelle a été mise à mal notamment lors des différentes étapes d'adaptation et d'amélioration de notre méthode et de caractérisation des différentes réarrangements génétiques. Néanmoins les qualités scientifique et humaine de l'encadrement ont fortement contribué à ma « remotivation ».

8.2 Perspectives

8.2.1 Perspectives à court terme

À court terme, il est prévu de/d' :

- ➔ Proposer une approche d'annotation des variants des modules (en se basant sur le pourcentage de similarité ainsi que la phylogénie des modules).
- ➔ Comparer les résultats de la détection manuelle à ceux de notre approche.
- ➔ Intégrer cette méthode dans le processus de reconstruction de l'histoire évolutive des phages.

8.2.2 Perspectives à long terme

De nombreuses perspectives à long terme peuvent découler suite à ce stage notamment :

- ➔ Il serait intéressant à long terme d'implémenter notre méthode en C^{++} d'une part pour simplifier le code et d'autre part pour diminuer éventuellement sa complexité. en effet, En effet, Networkx s'est révélé incomplet et ne permettait pas de manipuler les graphes de manière simple. On a fait le choix de travailler avec python étant donné que les différents script développés précédemment dans l'équipe pour la reconstruction de l'histoire évolutive des phages ont été implémentés sous python. Ceci permettait d'avoir une certaine uniformité du langage utilisé toute au long de l'approche de reconstruction de l'histoire des phages.
- ➔ Proposer un outil de visualisation des modules/variants de modules.
- ➔ Intégrer cette méthode dans un processus de correction des annotations existantes voire comme outils complémentaire d'annotation.

ANNEXES

9 ANNEXES

9.1 CODES DES FONCTIONS IMPLÉMENTÉES DANS NETWORKX

Dans cette partie, je répertorie les codes des fonctions de Networkx que j'ai utilisé durant ce stage à savoir :

- ❒ La fonction de création de l'objet DAG (def __init__(self, data=None, **attr) ; *cf.* code 1).
- ❒ La fonction d'ajout d'arc (add_edge(self, u, v, attr_dict=None, **attr) ; *cf.* code 2).
- ❒ La fonction de suppression d'arc (remove_edge(self, u, v) ; *cf.* code 3).
- ❒ La fonction d'ajout de sommet (add_node(self, n, attr_dict=None, **attr) ; *cf.* code 4).
- ❒ La fonction de suppression de sommet (remove_node(self, n) ; *cf.* code 5).
- ❒ La fonction de listage des sous-graphes des composantes fortement connexes (strongly_connected_component_subgraphs copy=True) ; *cf.* code 6).
- ❒ La fonction de listage des composantes fortement connexes (strongly_connected_components(G) ; *cf.* code 7).
- ❒ La fonction de vérification de l'acyclicité d'un graphe (is_directed_acyclic_graph(G) ; *cf.* code 8).
- ❒ La fonction du tri topologique (topological_sort(G, nbunch=None, reverse=False) ; *cf.* code 9).
- ❒ La fonction de listage des prédécesseurs d'un sommet (predecessors(self, n) ; *cf.* code 10).
- ❒ La fonction de listage des successeurs d'un sommet (successors(self, n) ; *cf.* code 11).
- ❒ La fonction d'itération des prédécesseurs d'un sommet (predecessors_iter(self,n) ; *cf.* code 12).
- ❒ La fonction d'itération sur les successeurs d'un sommet (successors_iter(self,n) ; *cf.* code 13).
- ❒ La fonction d'itération sur les sommets de G (edges_iter(self, nbunch=None, data=False) ; *cf.* code 14).
- ❒ La fonction de copie de graphe (copy(self) ; *cf.* code 15).
- ❒ La fonction "shortest_path_length" (shortest_path_length(G, source=None, target=None, weight=None) ; *cf.* code 16).
- ❒ La fonction single_source_dijkstra_path_length (single_source_dijkstra_path_length(G, source, cutoff=None, weight='weight') ; *cf.* code 17).
- ❒ La fonction d'itération sur les degrés des nœuds d'un graphe (degree_iter(self, nbunch=None, weight=None) ; *cf.* code 18).
- ❒ La fonction de listage des descendants d'un sommet (descendants(G, source) ; cf.code 19).
- ❒ La fonction de listage des ancêtres d'un sommet (ancestors(G, source) ; *cf.* code 20).

```
def __init__(self, data=None, **attr):

        self.graph = {} # dictionary for graph attributes
        self.node = {} # dictionary for node attributes
        # We store two adjacency lists:
        # the  predecessors of node n are stored in the dict self.pred
        # the successors of node n are stored in the dict self.succ=self.adj
        self.adj = {}  # empty adjacency dictionary
        self.pred = {}  # predecessor
        self.succ = self.adj  # successor

        # attempt to load graph with data
        if data is not None:
            convert.to_networkx_graph(data,create_using=self)
        # load graph attributes (must be after convert)
        self.graph.update(attr)
        self.edge=self.adj
```

Code 1: *La fonction de création de l'objet DAG implémentée dans Networkx*

```
def add_edge(self, u, v, attr_dict=None, **attr):
        if attr_dict is None:
            attr_dict=attr
        else:
            try:
                attr_dict.update(attr)
```

```
            except AttributeError:
                raise NetworkxError("The attr_dict argument must be a dictionary.")
        # add nodes
        if u not in self.succ:
            self.succ[u]={}
            self.pred[u]={}
            self.node[u] = {}
        if v not in self.succ:
            self.succ[v]={}
            self.pred[v]={}
            self.node[v] = {}
        # add the edge
        datadict=self.adj[u].get(v,{})
        datadict.update(attr_dict)
        self.succ[u][v]=datadict
        self.pred[v][u]=datadict
```

Code 2: *La fonction d'ajout d'arc implémentée dans Networkx*

```
def remove_edge(self, u, v):
        try:
            del self.succ[u][v]
            del self.pred[v][u]
        except KeyError:
            raise NetworkxError("The edge %s-%s not in graph."%(u,v))
```

Code 3: *La fonction de suppression d'arc implémentée dans Networkx*

```
  def add_node(self, n, attr_dict=None, **attr):
        if attr_dict is None:
            attr_dict=attr
        else:
            try:
                attr_dict.update(attr)
            except AttributeError:
                raise NetworkxError("The attr_dict argument must be a dictionary.")
        if n not in self.succ:
            self.succ[n] = {}
            self.pred[n] = {}
            self.node[n] = attr_dict
        else: # update attr even if node already exists
            self.node[n].update(attr_dict)
```

Code 4: *La fonction d'ajout de sommet implémentée dans Networkx*

```
def remove_node(self, n):

        try:
            nbrs=self.succ[n]
            del self.node[n]
        except KeyError: # NetworkxError if n not in self
            raise NetworkxError("The node %s is not in the digraph."%(n,))
        for u in nbrs:
            del self.pred[u][n] # remove all edges n-u in digraph
        del self.succ[n]          # remove node from succ
        for u in self.pred[n]:
            del self.succ[u][n] # remove all edges n-u in digraph
        del self.pred[n]          # remove node from pred
```

Code 5: *La fonction de suppression de sommet implémentée dans Networkx*

```
def strongly_connected_component_subgraphs(G, copy=True):
    for comp in strongly_connected_components(G):
        if copy:
            yield G.subgraph(comp).copy()
        else:
            yield G.subgraph(comp)
```

Code 6: *La fonction de listage des sous-graphes des composantes fortement connexes implémentée dans Networkx*

```
def strongly_connected_components(G):
    preorder={}
    lowlink={}
    scc_found={}
    scc_queue = []
    i=0     # Preorder counter
    for source in G:
        if source not in scc_found:
            queue=[source]
            while queue:
                v=queue[-1]
                if v not in preorder:
                    i=i+1
                    preorder[v]=i
                done=1
                v_nbrs=G[v]
                for w in v_nbrs:
                    if w not in preorder:
                        queue.append(w)
                        done=0
                        break
                if done==1:
                    lowlink[v]=preorder[v]
                    for w in v_nbrs:
                        if w not in scc_found:
                            if preorder[w]>preorder[v]:
                                lowlink[v]=min([lowlink[v],lowlink[w]])
                            else:
                                lowlink[v]=min([lowlink[v],preorder[w]])
                    queue.pop()
                    if lowlink[v]==preorder[v]:
                        scc_found[v]=True
                        scc=[v]
                        while scc_queue and preorder[scc_queue[-1]]>preorder[v]:
                            k=scc_queue.pop()
                            scc_found[k]=True
                            scc.append(k)
                        yield scc
                    else:
                        scc_queue.append(v)
```

Code 7: *La fonction de listage des composantes fortement connexes implémentée dans Networkx*

```
def is_directed_acyclic_graph(G):
    if not G.is_directed():
      return False
    try: topological_sort(G, reverse=True)
      return True
    except nx.NetworkxUnfeasible:
      return False
```

Code 8: *La fonction de vérification de l'acyclicité d'un graphe*

```
def topological_sort(G, nbunch=None, reverse=False):
    if not G.is_directed():
        raise nx.NetworkxError("Topological sort not defined on undirected graphs.")
    seen = set()
    order = []
    explored = set()

    if nbunch is None:
        nbunch = G.nodes_iter()
    for v in nbunch:
        if v in explored:
            continue
        fringe = [v]
        while fringe:
            w = fringe[-1]
            if w in explored:
                fringe.pop()
                continue
            seen.add(w)
            new_nodes = []
            for n in G[w]:
                if n not in explored:
                    if n in seen:
                        raise nx.NetworkxUnfeasible("Graph contains a cycle.")
                    new_nodes.append(n)
            if new_nodes:
                fringe.extend(new_nodes)
            else:
                explored.add(w)
                order.append(w)
                fringe.pop()
    if reverse:
        return order
    else:
        return list(reversed(order))
```

Code 9: *La fonction du tri topologique implémentée dans Networkx*

```
def predecessors(self, n):

        return list(self.predecessors_iter(n))
```

Code 10: *La fonction de listage des prédécesseurs d'un sommet implémentée dans Networkx*

```
def successors(self, n):

      return list(self.successors_iter(n))
```

Code 11: *La fonction de listage des successeurs d'un sommet implémentée dans Networkx*

```
def predecessors_iter(self,n):

        try:
            return iter(self.pred[n])
        except KeyError:
            raise NetworkxError("The node %s is not in the digraph."%(n,))
```

Code 12: *La fonction d'itération sur les prédécesseurs d'un sommet implémentée dans Networkx*

```
def successors_iter(self,n):

     try:
            return iter(self.succ[n])
        except KeyError:
            raise NetworkxError("The node %s is not in the digraph."%(n,))
```

Code 13: *La fonction d'itération sur les successeurs d'un sommet implémentée dans Networkx*

```
def edges_iter(self, nbunch=None, data=False):

  if nbunch is None:
    nodes_nbrs=self.adj.items()
  else:
    nodes_nbrs=((n,self.adj[n]) for n in self.nbunch_iter(nbunch))
  if data:
    for n,nbrs in nodes_nbrs:
      for nbr,data in nbrs.items():
        yield (n,nbr,data)
  else:
    for n,nbrs in nodes_nbrs:
      for nbr in nbrs:
        yield (n,nbr)
    # alias out_edges to edges
```

Code 14: *La fonction d'itération sur les sommets d'un graphe*

```
def copy(self):

        return deepcopy(self)
```

Code 15: *La fonction de copie d'un graphe implémentée dans Networkx*

```
def shortest_path_length(G, source=None, target=None, weight=None):

    if source is None:
        if target is None:
            if weight is None:
                paths=nx.all_pairs_shortest_path_length(G)
            else:
                paths=nx.all_pairs_dijkstra_path_length(G, weight=weight)
        else:
            with nx.utils.reversed(G):
                if weight is None:
                    paths=nx.single_source_shortest_path_length(G, target)
                else:
                    paths=nx.single_source_dijkstra_path_length(G, target,weight=weight)
    else:
        if target is None:
            if weight is None:
                paths=nx.single_source_shortest_path_length(G,source)
            else:
                paths=nx.single_source_dijkstra_path_length(G,source,weight=weight)
        else:
            if weight is None:
                p=nx.bidirectional_shortest_path(G,source,target)
                paths=len(p)-1
            else:
                paths=nx.dijkstra_path_length(G,source,target,weight)
    return paths
```

Code 16: *La fonction "shortest_path_length" implémentée dans Networkx*

```
def single_source_dijkstra_path_length(G, source, cutoff=None, weight='weight'):

    push = heappush
    pop = heappop
    dist = {}  # dictionary of final distances
    seen = {source: 0}
    c = count()
    fringe = []  # use heapq with (distance,label) tuples
    push(fringe, (0, next(c), source))
    while fringe:
        (d, _, v) = pop(fringe)
        if v in dist:
            continue  # already searched this node.
        dist[v] = d
        if G.is_multigraph():
            edata = []
            for w, keydata in G[v].items():
                minweight = min((dd.get(weight, 1)
                for k, dd in keydata.items()))
      edata.append((w, {weight: minweight}))
        else:
            edata = iter(G[v].items())

        for w, edgedata in edata:
            vw_dist = dist[v] + edgedata.get(weight, 1)
            if cutoff is not None:
                if vw_dist > cutoff:
                    continue
            if w in dist:
                if vw_dist < dist[w]:
                    raise ValueError('Contradictory paths found:','negative weights?')
            elif w not in seen or vw_dist < seen[w]:
                seen[w] = vw_dist
                push(fringe, (vw_dist, next(c), w))
    return dist
```

Code 17: *La fonction "single_source_dijkstra_path_length" implémentée dans Networkx*

```
def degree_iter(self, nbunch=None, weight=None):

  if nbunch is None:
    nodes_nbrs=zip(iter(self.succ.items()),iter(self.pred.items()))
  else:
    nodes_nbrs=zip(((n,self.succ[n]) for n in self.nbunch_iter(nbunch)), ((n,self.pred[n]) for n
     in self.nbunch_iter(nbunch)))

  if weight is None:
    for (n,succ),(n2,pred) in nodes_nbrs:
      yield (n,len(succ)+len(pred))
  else:
   # edge weighted graph - degree is sum of edge weights
    for (n,succ),(n2,pred) in nodes_nbrs:
      yield (n,sum((succ[nbr].get(weight,1) for nbr in succ))+ sum((pred[nbr].get(weight,1) for
     nbr in pred)))
```

Code 18: *La fonction d'itération sur les degrés des nœuds d'un graphe implémentée dans Networkx*

```
def descendants(G, source):
    if not G.has_node(source):
      raise nx.NetworkxError("The node %s is not in the graph." % source)
    des = set(nx.shortest_path_length(G, source=source).keys()) - set([source])
    return des
```

Code 19: *La fonction de listage des descendants d'un sommet implémentée dans Networkx*

```
def ancestors(G, source):

    if not G.has_node(source):
        raise nx.NetworkxError("The node %s is not in the graph." % source)
    anc = set(nx.shortest_path_length(G, target=source).keys()) - set([source])
    return ancestors
```

Code 20: *La fonction de listage des ancêtres d'un sommet implémentée dans Networkx*

9.2 Complexité des structures de données en python

Le tableau 7 détaille la complexité des différentes opérations sur les structures de données de Python. De façon générale, n est le nombre d'éléments dans la structure en question, k désigne soit la valeur du paramètre de l'opération soit le nombre d'éléments dans le paramètre de l'opération.

Tab. 7: *Complexité des différentes structures de données en python. (source : https ://wiki.python.org/moin/TimeComplexity).*

TYPE D'OPÉRATION	LE CAS MOYEN	LE PIRE CAS
Liste		
Copier	O(n)	O(n)
Ajouter	O(1)	O(1)
Insérer	O(n)	O(n)
Extraire élément	O(1)	O(1)
Mettre à jour un élément	O(1)	O(1)
Supprimer un élément	O(n)	O(n)
Itération	O(n)	O(n)
Extraire une portion	O(k)	O(k)
Supprimer une portion	O(n)	O(n)
Ajouter une portion	O(k+n)	O(k+n)
Extend	O(k)	O(k)
Trier	O(n log n)	O(n log n)
Multiplier	O(nk)	O(nk)
x dans s	O(n)	
min(s), max(s)	O(n)	
Calculer taille	O(1)	O(1)
Ensemble		
x dans s	O(1)	O(n)
Union s|t	O(len(s)+len(t))	
Intersection s&t	O(min(len(s), len(t))	O(len(s) * len(t))
Différence s-t	O(len(s))	
Dictionnaire		
Copier	O(n)	O(n)
Extraire élément	O(1)	O(n)
Mettre à jour élément	O(1)	O(n)
Supprimer élément	O(1)	O(n)
Itération	O(n)	O(n)

9.3 Séquences et base de données utilisées dans les différentes expérimentations

9.3.1 Création d'une base de donnée locale avec makeblastdb de blast+

Durant ce stage on a travillé avec BLAST local (balstn) en utilisant la version "blast 2.2.25". La première étape est de s'assurer que "blastn" est installé sur la machine sur laquelle on travaille (commande : blastn -version), sinon il faut l'installer comme suit :

```
sudo apt-get install ncbi-blast+
```

Code 21: *Installation blast+ sous linux*

Une fois blastn est installé sur notre machine, il est facile de créer des bases de données locales avec la commande "makeblastdb" décrite ci-dessous (*cf.* code 22 et illustré dans l'encadré 23 :

```
makeblastdb [-h] [-help] [-in input_file] [-dbtype molecule_type]
    [-title database_title] [-parse_seqids] [-input_type type] [-hash_index]
    [-mask_data mask_data_files] [-gi_mask]
    [-gi_mask_name gi_based_mask_names] [-out database_name]
    [-max_file_sz number_of_bytes] [-taxid TaxID] [-taxid_map TaxIDMapFile]
    [-logfile File_Name] [-version]
```

Code 22: *Usage de la commande de création de la base de donnée locale : makeblastdb*

```
makeblastdb $-in /home/amal/sequences-a-aligner.fasta -dbtype nucl -out seq10db -title "seq10db"
    $
```

Code 23: *Exemple de création de base de donnée locale avec makeblastdb*

9.3.2 Lactococcus lactis : portal-lysine

Lors de l'extraction des séquences, il était essentiel de prendre en considération la circularité des génomes. Pour cela, le script d'extraction de séquence de même que le fichier .csv utilisé lors de la "translation" des positions du graphe, prenait 4 positions sur le génome qu'on note :

- ❒ Position A correspond à la position de début de la séquence à extraire.
- ❒ Position B correspond à la position de fin du génome si la position de fin de la séquence à extraire est inférieur celle de début de la séquence à extraire sinon elle vaut 0.
- ❒ Position C : si la position B est différente de 0, alors C vaut 1 sinon C correspond à la position de fin de la séquence à extraire.
- ❒ Position D : si la C=1 alors D correspond à la position de fin de la séquence à extraire sinon D=0

Cette définition est valable pour l'ensemble des jeux de données utilisés.

En ce qui concerne les essaies réalisées sur *L.lactis*, nous avons travaillé avec des séquences ancres des gènes de portal et de la lysine (*cf.* tableau 8).

Tab. 8: *Détail des 27 séquences utilisées dans les expérimentations relatives à L.lactis*

Phage	Position A	Position B	Position C	Position D
Phage_L1	2924	0	17648	0
Phage_L2	2913	0	17648	0
Phage_L3	3065	0	18262	0
Phage_L4	3065	0	18262	0
Phage_L5	2913	0	17648	0
Phage_6	2913	0	17342	0
Phage_L7	2913	0	17342	0
Phage_L8	3065	0	18262	0
Phage_L9	3065	0	18263	0
Phage_L10	3065	0	18262	0
Phage_L11	2913	0	17342	0
Phage_L12	2913	0	17306	0
Phage_L13	2913	0	17306	0
Phage_L14	2913	0	17246	0
Phage_L15	2913	0	17648	0
Phage_L16	2913	0	17306	0
Phage_L17	3087	0	18284	0
Phage_L18	3087	0	18285	0
Phage_L19	2913	0	17306	0
Phage_L20	3087	0	18284	0
Phage_L21	3087	0	18284	0
Phage_L22	2913	0	17342	0
Phage_L23	2923	0	17316	0
Phage_L24	2913	0	17411	0
Phage_L25	3087	0	18284	0
Phage_L26	3087	0	18284	0
Phage_L27	2913	0	17342	0

Tab. 9: *Récapitulatif des 31 phages de Staphylococcusaureus utilisés dans le premier jeux de données. (lien NCBI).*

Code	Phage	Accession	lg génome	RinaA- M1	Amidase-M2	lg M1-M2
F0	phiETA2	*NC* : 008798	43265	15612	41937	26325
F2	phiETA3	*NC* : 008799	43282	16404	41954	25550
F23	alpha80	*NC* : 009526	43864	15798	42155	26357
F27	phiNM1	*NC* : 008583	43128	16009	42383	26374
F29	phage11	*NC* : 004615	43604	15917	42365	26448
F30	phage29	*NC* : 007061	42802	42111..42802	1..25140	25831
F33	Rosa	*NC* : 007058	43155	42550..43155	1..25970	26575
F39	phage53	*NC* : 007049	43883	43430..43883	1..25903	26356
F40	phage85	*NC* : 007050	44283	43826..44283	1..25368	25825
F41	phiNM2	DQ530360	43145	16005	42472	26467
F3	phiETA	*NC* : 003288	43081	16275	41753	25478
F5	phage55	*NC* : 007060∗	41902	41292..41902	1..25052	25662
F6	phage71	*NC* : 007059∗	43114	42432..43114	1..24666	25348
F24	phage80	DQ908929*	42140	14384	40286	25902
F28	phage88	*NC* : 007063∗	43231	42629..43231	1..25092	25694
F35	phiNM4	DQ530362*	43189	16579	42438	25859
F42	phage92	*NC* : 007064∗	42431	41826..42431	1..25023	25628
F7	phi12	*NC* : 004616	44970	19560	44173	24613
F11	phiSLT	*NC* : 002661	42942	15752	40377	24625
F19	42e	*NC* : 007052	45861	44881..45861	1..23279	24259
F22	IPLA35	*NC* : 011612	45344	18963	43578	24615
F31	tp3102	*NC* : 009762	45710	20362	44964	24602
F12	phiPVL108	*NC* : 008689∗	44857	16838	39948	23110
F13	phiPVL	*NC* : 002321∗	41401	39932..41401	1..21653	23122
F14	tp3103	*NC* : 009763∗	41966	14835	37536	22701
F15	phi13	*NC* : 004617∗	42722	15557	38276	22719
F25	phiPV83	*NC* : 002486∗	45636	15298	39930	24632
F34	phage77	*NC* : 005356	41708	40729..41708	1..21026	22005
F17	phiN315	*NC* : 004740	44082	16781	39762	22981
F43	P954	*NC* : 013195	40761	17738	39370	21632
F26	phage69	*NC* : 007048	42732	42226..42732	1..25939	26445

9.3.3 STAPHYLOCCOCUS AUREUS : PORTAL-TAPE MEASURE

De la même manière que pour *L.lactis*, nous avons travaillé avec des séquences comprises ente deux séquences ancres. Pour S.aureus, nous avons fait le choix de travailler deux jeux de données à savoir :

- ❒ le premier correspond aux séquences comprises entre le gène de l'amidase et le gène de RinA (*cf.* tableaux 10 et 9).
- ❒ le deuxième correspond aux séquences comprises entre le gène Portal et le gène de Tape measure. À titre informatif, ce dernier gène doit son nom au fait que sa longueur est proportionnelle à la longueur de la queue du phage [5] (*cf.* tableau 11).

Tab. 10: *Détail des 31 séquences utilisées dans les expérimentations relatives à S.aureus : premier jeu de données*

Phage	Position A	Position B	Position C	Position D
phiETA2	15612	0	41937	0
phiETA3	16404	0	41954	0
80alpha	15798	0	42155	0
phiNM1	16009	0	42383	0
phage11	15917	0	42365	0
phage29	42111	42802	1	25140
ROSA	42550	43155	1	25970
phage53	43430	43883	1	25903
phage85	43826	44283	1	25368
phage85	16005	0	42472	0
phiETA	16275	0	41753	0
phage55	41292	41902	1	25052
phage71	42432	43114	1	24666
phage80	14384	0	40286	0
phage88	42629	43231	1	25092
phage71	16579	0	42438	0
phage92	41826	42431	1	25023
phi12	19560	0	44173	0
phiSLT	15752	0	40377	0
42e	44881	45861	1	23279
IPLA35	18963	0	43578	0
tp3102	20362	0	44964	0
phiPVL108	16838	0	39948	0
PVL	39932	41401	1	21653
tp3103	14835	0	37536	0
phi13	15557	0	38276	0
phiPV83	15298	0	39930	0
phage77	40729	41708	1	21026
phiN315	16781	0	39762	0
P954	17738	0	39370	0
phage69	42226	42732	1	25939

Tab. 11: *Détail des 31 séquences utilisées dans les expérimentations relatives à S.aureus : deuxième jeu de données*

Phage	Position A	Position B	Position C	Position D
phiETA2	17937	0	31238	0
phiETA3	18864	0	31293	0
phiETA	18704	0	31091	0
phage55	1819	0	14206	0
phage71	926	0	14072	0
phi12	22773	0	41841	0
phiSLT	18972	0	38044	0
phiPVL108	20730	0	29314	0
PVL	2451	0	11032	0
tp3103	18727	0	27310	0
phi13	19468	0	28051	0
phiN315	19878	0	28203	0
42e	2230	0	21298	0
IPLA35	22181	0	41246	0
80alpha	18122	0	31421	0
phage80	16815	0	29430	0
phiPV83	19200	0	29708	0
phage69	1815	0	15117	0
phiNM1	18333	0	31655	0
phage88	1000	0	14241	0
phage11	18241	0	31543	0
phage29	1734	0	14349	0
tp3102	23564	0	42631	0
ROSA	1000	0	15280	0
phage77	2120	0	10444	0
phiNM4	19038	0	31725	0
phage53	1871	0	15170	0
phage85	1955	0	15277	0
phiNM2	18329	0	31649	0
phage92	997	0	14238	0
P954	20834	0	29133	0

9.4 Description des scripts développés lors du stage

Dans cette section, je détaille quelques scripts développés lors de ce stage.

9.4.1 Script d'extraction des séquences comprises entre les séquences ancres

Le script *database_seq.py* permet d'extraire les séquences à aligner et qui sont comprises entre les séquences ancres. Le script prend 5 paramètres à savoir :
1^{er} cas : si la position de début est avant la position de fin sur le génome, alors le script prend en entrée le nom du fichier .fasta contenant la séquence du génome en question suivi des position de début et de fin de la séquence à extraire suivi de " 0 0 ". Ex : (./*database_seq.py* phiPV83.fasta 15298 39930 0 0").
1^{er} cas : si la position de début est après la position de fin sur le génome, alors le script prend en paramètre le nom du fichier suivi de la position de début de la séquence à extraire suivi de "1" suivi de la taille du génome dans le fichier en entrée suivi de la position de fin de la séquence à extraire ."./*database_seq.py* phage77.fasta 40729 1 41708 21026").
Le script génère un fichier .fasta formaté (60 nucléotides par ligne) dont le nom est composé du nom du fichier donné en entrée précédé par le terme "database". Ex : databasephiPV83 ou encore databasephage77.fasta.

9.4.2 SCRIPT DE GÉNÉRATION DES SÉQUENCES DE SIMULATIONS

Ce code a été utilisé notamment pour générer les différentes séquences mimant la présence des principaux réarrangements génomiques. Le script prend en paramètre les noms des phages (4) suivi de leur séquences à générées sous forme de lettres où chaque lettre sera replacée par une séquence d'une longueur de 400 bp. Le script génère 4 fichiers dont le nom est composé du nom du phage et du code de sa séquence. Exemple de commande selon le type de réarrangement :

- Insertion :./testinsertion.py -A "JRBM" -B "JRBM" -C "JBM" -D "JBM".
- Duplication :./testinsertion.py -A "JRRBM" -B "JRBM" -C "JRBM" -D "JRBM".
- Problème de colinéarité : ./testinsertion.py -A "JRBM" -B "JBNM" -C "JNRM" -D "JNRM".

9.4.3 SCRIPT DE LA DEUXIÈME APPROCHE :CODE_V0.PY

Pour lancer le code, nous avons besoin de préciser 6 paramètres dont 3 optionnels (ceux précédés par –) à savoir :

❶ **Paramètres obligatoires (précédés par 1 tiret -) :**

- ❒ Nom de la base de données contenant les séquences à blaster (précédé par -D).
- ❒ Nom du fichier utiliser pour la création de la base de donnée (précédé par -F).
- ❒ Nom du fichier qui sera utiliser pour la translation des positions dans le graphe des régions (précédé par -T). Ce fichier permettra de localiser les positions dans le graphe des régions généré à l'étape 5 sur le génome. Cette étape est importante dans le sens où elle permet de faciliter l'interprétation et la lecture du graphe en se basant sur les annotations.

❷ **Paramètres optionnels (précédés par 2 tirets –) :**

- ❒ Paramètres du BLAST (*penality* et *record* précédés respectivement par –P et par –R). Si ces derniers ne sont pas précisés alors ils prennent les valeurs définis par défaut comme c'est spécifiées dans le code (à savoir penality= 1 et reward = -5).
- ❒ Seuil de fusion (précédé par –S,). S'il n'est pas précisé alors il prend la valeur indiquée par défaut dans le code à savoir 5 bp.

Exemple de lancement du script code_v0.py :

```
code_v0.py –F Fichier_name –T fichier_translation –D database_name ––R Reward_value ––P
    Penalty_value ––S Seuil_fusion_value
```

Code 24: *Lancement du script code_v0.py*

9.5 PIPLINE

La figure 40 ci-dessous résume les différentes étapes de notre approche en commançant par l'extraction des séquences à la génération du graphe des régions et en passant par la création de la base de données locale.

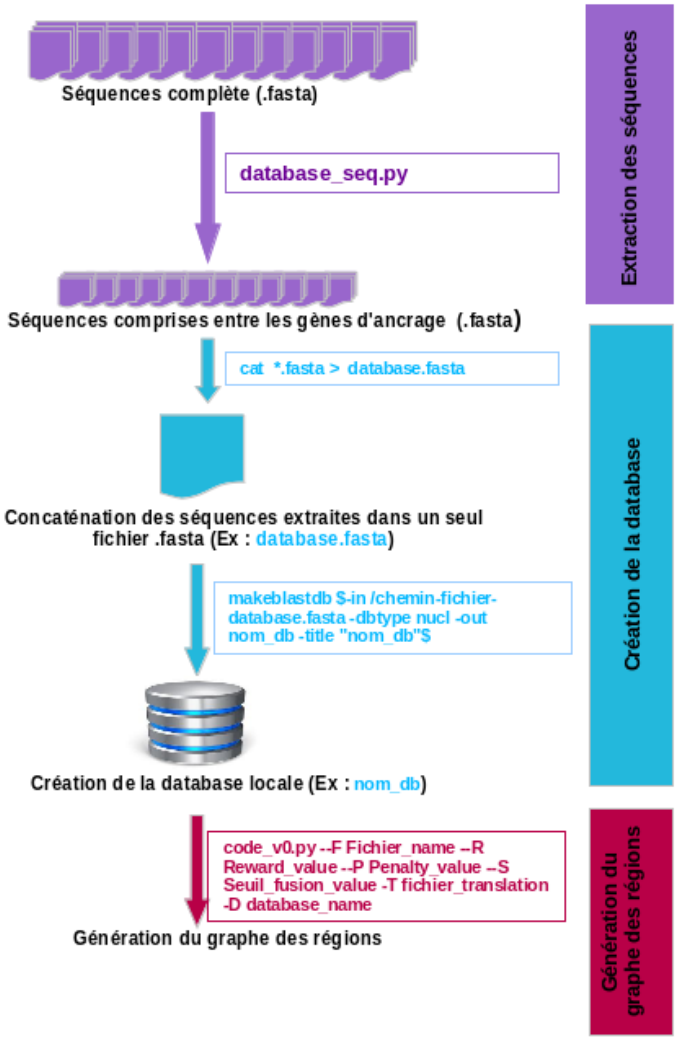


Fig 40: *Pipline.*

9.6 DESCRIPTION DE L'ARCHIVE

La figure 41 illustre l'organisation de l'archive rendue (*Archive_stage_ZINE_EL_AABIDINE_M2*) à la fin de mon stage. L'archive contient 12 dossiers à savoir :

❒ **Simulation_des_cas_d'insertion** : contient les données et les résultats liés aux simulations d'insertion. Il contient 4 dossiers à savoir :

➠ **cas1** : contient les résultats de l'analyse de la présence d'insertion simple (chez 1 phage sur 4 à savoir : phageA_RBNJ, phageB_RBVNJ, phageC_RBNJ e phageD_RBNJ).

➠ **cas2** : contient les résultats de l'analyse de la présence d'insertion multiples (chez 2 phages sur 4 à savoir les phages : phageA_RBNJ, phageB_RBINJ, phageC_RBINJ et phageD_RBNJ).

➠ **cas3** : contient les résultats de l'analyse de la présence d'insertion multiples (chez 3 phages sur 4 à savoir les phages : phageA_RBNJ, phageB_RBINJ, phageC_RBINJ et phageD_RBINJ).

➠ **cas4** : contient les résultats de l'analyse de la présence d'insertion multiples (2 variants de l'insert).

❒ **Simulation_des_cas_de_duplication** : contient les données et les résultats liés aux simulations de duplication. Il contient 4 dossiers à savoir :

➠ **cas1** : contient les résultats de l'analyse de la présence de duplication en tendem simple (chez 1 phage sur 4 :).

➠ **cas2** : contient les résultats de l'analyse de la présence de duplication en tendem multiples (3 duplications en tendem chez 1 phage sur 4).

➠ **cas3** : contient les résultats de l'analyse de la présence de duplication en tendem multiples (chez 3 phages sur 4).

➠ **cas4** : contient les résultats de l'analyse de la présence de duplication multiples mais pas en tendem (chez 1 phage sur 4).

❒ **Simulation_des_cas_d'inversion** : contient les données et les résultats liés aux simulations vraie colinéarité/inversion. Il contient 3 dossiers à savoir :

➠ **cas1** : contient les résultats de l'analyse de la présence d'inversion simple mettant en jeu 2 phages sur 4 (Ex : phageA_RBNJ, phageB_RBINJ, phageC_RBNIJ, phageD_RBNJ).

➠ **cas2** : contient les résultats de l'analyse de la présence d'inversion multiples mettant en jeu les 4 phages (Ex : phageA_RBNIJ, phageB_RBINJ, phageC_RBNIJ, phageD_RBINJ).

❒ **Simulation_des_cas_de_colinéarité** : contient les données et les résultats liés aux simulations de problème de colinéarité. Il contient 1 dossier à savoir :

➠ **cas1** : contient les résulttas de l'analyse de la présence de colinéarité à 3 en analysant les phages phageA_JRBM, phageB_JBNM, phageC_JNRM et phageD_JNRM.

❒ **Analyse_de_S.aureus_Portal_Tape_measure** : contient les données et les résultats liés aux différentes analyses réalisées chez *S.aureus* (Portal-Tape measure).

❒ **Analyse_de_S.aureus_Amidase_RinA** : contient les données et les résultats liés aux différentes analyses réalisées chez *S.aureus* (Amidase-RinA).

❒ **Analyse_de_L.lactis_Portal_lysine** : contient les données et les résultats liés aux différentes analyses réalisées chez *L.lactis* (Portal-lysine).

❒ **Séquences_fasta_L.lactis** : les séquences fasta des génomes de *L.lactis* analysés (28 séquences).

❒ **Séquences_fasta_S.aureus** : les séquences fasta des génomes de *S.aureus* analysés (31 séquences).

❒ **Annotation_L.lactis** : les annotations fonctionnelles des génomes de *L.lactis* analysés (28 séquences).

❒ **Scripts** : contient les différents scripts nécessaires pour la reproduction des différents étapes et résultats de notre étude.

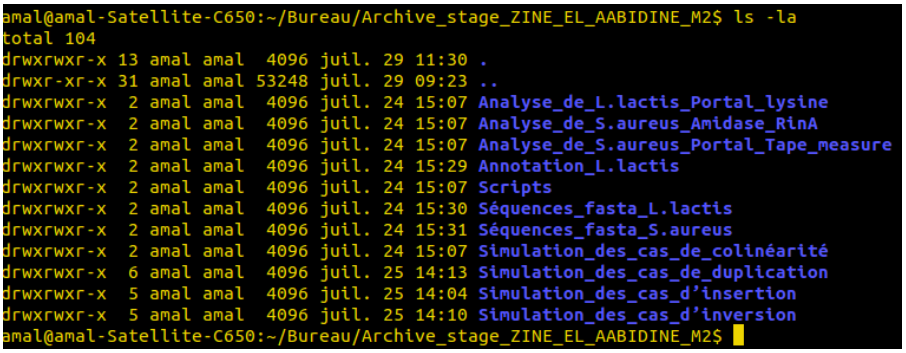


Fig 41: *Description de l'archive.*

RÉFÉRENCES

[1] Sulakvelidze A, A Sulakvelidze, Zemphira A, and J.M Glenn. Bacteriophage therapy. *Antimicrobial Agents Chemotheraby*, 45 :649–659, 2001.

[2] ST Abedon, C Thomas-Abedon, A Thomas, and H Mazure. Bacteriophage prehistory. *Bacteriophage*, 3 :174–178, 2011.

[3] Griffiths AJF, JH Miller, and et al. DT Suzuki. *An Introduction to Genetic Analysis. 7th edition.* New York : W. H. Freeman ;, 2000.

[4] Alberts B, A Johnson, and J Lewis et al. *Molecular Biology of the Cell. 4th edition.* New York : Garland Science ;, 2002.

[5] Mahdi Belcaid, Anne Bergeron, and Guylaine Poisson. The evolution of the tape measure protein : units, duplications and losses. *BMC Bioinformatics*, 12(S-9) :S10, 2011.

[6] Desplats C and HM Krisch. The diversity and evolution of the *T4-type* bacteriophages. *Research in Microbiology*, 154 :259–267, 2003.

[7] E. Castro-Nallar, H. Chen, S. Gladman, S.C. Moore, T. Seemann, I.B. Powell, A. Hillier, K.A. Crandall, and P.S. Chandr. Population genomics and phylogeography of an australian dairy factory derived lytic bacteriophage. *Genome Biology and Evolution*, 4(3) :382–393, 2012.

[8] S Chibani-Chennoufi, C Canchaya, A Bruttin, and H Brüssow. Comparative genomics of the t4-like *Escherichia coli Phage JS98* : Implications for the evolution of *T4* phages. *Journal of bacteriology*, 186 :8276–8286, 2004.

[9] Botstein D. A theory of modular evolution for bacteriophages. *Annals of the New York Academy of Sciences*, 354 :484–490, 1980.

[10] Desiere F, S Lucchini, and H Brussow. Evolution of *Streptococcus thermophilus* bacteriophage genomes by modular exchanges followed by point mutations and small deletions and insertions. *Virology*, 241 :345–356, 1998.

[11] D'Herelle F. On an invisible microbe antagonistic toward dysenteric bacilli : brief note by mr. f. d'herelle, presented by mr. roux. 1917. *Research in Microbiology*, 158 :553–554, 2007.

[12] D'Hérelle Félix. Sur un microbe invisible antagoniste des bacilles dysentérique. *Académie des sciences Paris*, 165 :373–375, 1917.

[13] Rousseau GM and S Moineau. Evolution of *Lactococcus lactis* phages within a cheese factory. *Appl Environ Microbiol*, 75 :5336–5344, 2009.

[14] Martinsohn JT, Radman M, and MA Petit. The lambda red proteins promote efficient recombination between diverged sequences : implications for bacteriophage genome mosaicism. *PLoS Genetic*, 2008.

[15] Swenson KM, P Guertin, H Deschênes, and A Bergeron. Reconstructing the modular recombination history of *Staphylococcus aureus* phages. *BMC Bioinformatics*, 14 :S15–S17, 2013.

[16] Krupovic M, D Prangishvili, RW Hendrix, and DH Bamford. Genomics of bacterial and *archaeal* viruses : dynamics within the prokaryotic virosphere. *Microbiol Mol Biol Rev*, 75 :610–635, 2011.

[17] Grindley ND, KL Whiteson, and PA Rice. Mechanisms of site-specific recombination. *Annual review of biochimy*, 75 :567–605, 2006.

[18] Esko Nuutila and Eljas Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Inf. Process. Lett.*, 49(1) :9–14, January 1994.

[19] Lucchini S, F Desiere, and H Brüssow. Comparative genomics of *Streptococcus thermophilus* phage species supports a modular evolution theory. *Journal of virology*, 73 :8647–8656, 1999.

[20] Nolivos S. *Etude du mécanisme de résolution des dimères de chromosomes chez les Streptocoques*. PhD thesis, l'Université Toulouse III - Paul Sabatier, 2010.

[21] MCM Smith and HM Thorpe. Diversity in the serine recombinases. *Molecular Microbiology*, Volume 44 :299–307, 2002.

[22] Robert Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1972.

[23] FW Twort, LRCP Lond, and MRCS. An investigation on the nature of ultra-microsomic viruses. *The Lancet*, 186 :1241–1243, 1915.