Laboratoire de Bactériologie
Service Bactériologie, Mycologie et Parasitologie
Pôle Biologie Médicale et Anatomie Pathologique
CHU Gabriel-Montpied de Clermont-Ferrand

Université d'Auvergne
IUT Clermont-Ferrand, Campus Universitaire d'Aurillac
Département Génie Biologique

Rapport de Stage

Assemblage d'un pipeline d'analyse bioinformatique pour l'analyse de bactéries pathogènes par séquençage à haut débit

CHOLET Pierre

DUT Génie Biologique
Option Bioinformatique
Deuxième année

Maître de Stage : Pr. Bonnet R.
Tutrice de Stage : Mme. Polonais V.

Stage du 31/03/2015 au 05/06/2015
Année 2014 – 2015

Sommaire

Remerciements

Résumé

Abstract

Introduction 1

I. Présentation du lieu de stage

I.1 Centre Hospitalier Universitaire Gabriel-Montpied 2

I.2 Centre National de Référence 2

II. Partie bibliographique et présentation du projet

II.1 Antibiotiques et Mécanismes de Résistance 3

II.2 Gènes de virulence 4

II.3 Méthodes de Typage 6

II.4 Origines de réplication des plasmides et protéines Mob 7

II.5 Présentation du projet 7

III. Matériel et méthodes

III.1 Les souches bactériennes 9

III.2 Outils bioinformatiques 10

III.3 Méthodes expérimentales 12

IV. Démarche et travaux réalisés

IV.1 Pipeline d'analyses de données NGS 13

IV.2 Résultats 15

IV.3 Discussion 16

Conclusion 19

Bibliographie

Table des acronymes

Glossaire et Lexique

Table des figures et Table des tableaux

Table des annexes et annexes

Remerciements

Je tiens à remercier M. Alain MEUNIER, directeur général du Centre Hospitalier Universitaire de Clermont-Ferrand ainsi que Mme. Agnès SAVALE, directrice du site Gabriel Montpied du CHU pour leur accueil au sein du centre.

De plus je tiens à remercier M. ESCHALIER, directeur du pôle de biologie médicale et d'anatomie pathologique pour son accueil au sein du pôle.

Je remercie respectueusement le professeur Richard BONNET, directeur du laboratoire de microbiologie, professeur des universités et praticien hospitalier ainsi que son équipe de m'avoir permis d'intégrer le laboratoire et pour leur accompagnement et leur aide tout au long de ma mission.

Je remercie notamment le docteur Racha BEYROUTHY, ingénieure au laboratoire de microbiologie pour son aide précieuse et de m'avoir permis d'observer et mieux comprendre les démarches et analyses biologiques.

Je tiens également à remercier respectueusement Mme. Valérie POLONAIS, enseignante-chercheuse à l'IUT d'Aurillac pour son accompagnement, son soutient et son aide précieuse tout au long de la formation.

Enfin je tiens à remercier Mme. Stéphanie BORNES, enseignante-chercheuse à l'IUT d'Aurillac ainsi que Mme. Bérénice BATUT, intervenante à l'IUT d'Aurillac et ingénieure au CIDAM pour leur aide et conseils mais également Mme. Jocelyne HURST de nous avoir préparé à l'exercice de la rédaction de rapports universitaires et de nous en avoir inculqué les normes.

Résumé

Le **service de bactériologie** du C.H.U de Clermont-Ferrand héberge le Centre National de Référence de la résistance aux antibiotiques. Il assure la **caractérisation des bactéries multirésistantes** et des **mécanismes de résistance aux antibiotiques** en s'appuyant sur l'amplification de gènes cibles par PCR et séquençage. L'avènement du **séquençage à haut débit** rends l'ensemble des génomes accessibles permettant de révolutionner ces approches. L'implémentation d'un « **pipeline** » **d'analyses bio-informatiques** permet le **typage moléculaire** des bactéries, la **détection** de leurs **gènes de résistance**, de **virulence** ainsi que leurs **plasmides**.

Cette projet entre dans le contexte d'une étude multicentrique des Entérobactéries multirésistantes diffusant au sein des hôpitaux français. Les espèces concernées étaient *Escherichia coli*, *Enterobacter cloacae* et *Klebsiella pneumoniae.* La détection des gènes de résistance et virulence a été réalisée par l'alignement sur des bases de données de gènes cibles. L'assemblage *de Novo* des séquences permet le typage moléculaire visant à établir les liens phylogénétiques entre les bactéries.

Le pipeline a permis d'**automatiser** et de **standardiser** la caractérisation de 95 souches bactériennes en quelques semaines alors que le même travail, à l'aide d'une approche par PCR, a nécessité plus de 8 mois à une équipe de 3 personnes sans permettre une exploration aussi complète.

Abstract

The **bacteriology department** of the teaching hospital of Clermont-ferrand hosts the National resistance to antibiotics Reference Center (CNR). It ensures **multiresistant bacteria** and **antibiotics resistance mechanisms characterization** using target genes PCR amplification and sequencing. The advent of **high throughput sequencing** allows for entire genomes to be accessible, hence revolutionizing these approaches. A **bioinformatic analysis pipeline** implementation enables **molecular typing** and **resistance** and **virulence genes** and **plasmids detection** of these bacteria.

This project is involved within a multi-center study of multidrug-resistant Enterobacteriaceae reported over french hospitals. The concerned species are *Escherichia Coli*, *Enterobacter Cloacae* and *Klebsiella Pneumoniae.* Resistance and virulence genes detection was performed by alignment over target genes databases. *de Novo* assembly allows molecular typing toward defining phylogenetic relationships between bacteria.

The pipeline has **automated** and **standardized** the characterization of 95 bacterial strains in a few weeks while the same job, using a PCR approach, took more than 8 months to a team of 3 people without allowing as complete exploration.

Introduction

Le laboratoire de microbiologie du Centre Hospitalier Universitaire (CHU) de Clermont-Ferrand fait partie du Centre National de Référence (CNR) de la résistance aux antibiotiques. Dans ce cadre, des souches bactériennes multirésistantes aux antibiotiques sont explorées afin d'évaluer le risque épidémique qu'elles représentent.

Le risque épidémique dépend de plusieurs facteurs, principalement la nature des gènes de résistance, la présence de gènes de virulence, de plasmides et la nature du fond génétique. La caractérisation des bactéries porte sur la détection des gènes de résistance aux antibiotiques codant les enzymes impliqués (ex. β-lactamases, carbapénèmases) et la détection des gènes de virulence codant des adhésines (Fimbriae*), des systèmes de capture du fer (Sidérophores*) ou des toxines (ex. hémolysine*). Ces gènes conférent un avantage sélectif expliquant le succès écologique d'une souche bactérienne. Par ailleurs, ils sont parfois codés par des plasmides, éléments génétiques transférables entre bactéries qui peuvent être à l'origine d'une diffusion épidémique des gènes de résistance. L'analyse du fond génétique des bactéries, aussi appelé « core genome* » ou « génome conservé* », repose sur l'analyse de mutations au sein de gènes de ménage*.

Ces explorations sont basées sur l'amplification de gènes cibles et le séquençage des produits d'amplification. L'analyse moléculaire d'une souche requiert ainsi plus de 40 réactions d'amplification et de séquençage soit un coût excédant plusieurs centaines d'euros. Les technologies de séquençage à haut débit permettent actuellement d'accéder au séquençage des génomes bactérien pour un coût particulièrement moindre en ressources et en temps. Le CNR a ainsi décidé de modifier ses procédures au profit de ces technologies. Il ne disposait cependant pas d'un protocole d'analyse des séquençages à haut débit dont ce stage fait l'objet. Le développement d'un pipeline d'analyses bio-informatiques, automatisée et standardisé, inclue le traitement des séquences et coordonne plusieurs outils informatiques et bases de données.

L'organisme d'accueil sera traité dans une première partie, puis les concepts biologiques étudiés ainsi que la problématique seront abordés dans une deuxième partie. Une troisième partie comprendra les matériels et méthodes utilisés. Enfin, une dernière partie traitera de la démarche d'assemblage du pipeline et des résultats obtenus.

I. **Organisme**

I.1. Centre Hospitalier Universitaire Gabriel Montpied

Le CHU de Clermont-Ferrand est séparé en trois sites distincts, le CHU Gabriel Montpied, le CHU d'Estaing et le CHU Hôpital Nord. L'ensemble comptabilise près de 1996 lits (771 pour le site G. Montpied) et emploie près de 8106 personnes dont un quart représente le personnel médical (données de 2011).

En sa qualité de Centre Hospitalier Régional, il possède cinq missions distinctes dont les principales sont le **soin**, réparti en quinze pôles eux-mêmes répartis en services, l'**enseignement** pour la formation initiale de personnels médicaux et paramédicaux et la **recherche** gérée par la Délégation à la Recherche Clinique et à l'Innovation.

Le service de Bactériologie est localisé sur le site Gabriel Montpied du CHU. Il est formé par deux grands secteurs : un secteur d'analyse en Biologie Médicale, qui assure la détection des bactéries responsables d'infections chez des patients hospitalisés, et un deuxième secteur au sein duquel le stage prend place, qui correspond au Centre National de Référence (CNR) de la résistance aux antibiotiques.

I.2. Centre National de Référence

Les Centres Nationaux de Référence (CNR) sont des laboratoires localisés au sein d'établissements de santé publics ou privés, d'enseignement ou de recherche. Ils sont nommés pour cinq ans par le ministère de la Santé (article L1413-4 du code de la santé publique) sur proposition de l'Institut National de Veille Sanitaire (InVS). L'InVS est un établissement public sous tutelle du ministère de la santé. Il coordonne l'action des CNR afin d'organiser la surveillance, la vigilance et l'alerte dans des domaines critiques de santé publique. Il existe 47 CNR (cf. Tableau I) classés en groupes de A à D en fonction de leur importance stratégique en santé publique. Ils sont évalués de façon annuelle et leur reconduction repose sur une procédure d'appel d'offre tous les 5 ans.

Le CNR de la résistance aux antibiotiques est un CNR de rang A, qui présente la particularité de comprendre 4 sites : les sites de Clermont-Ferrand, Caen, Paris et le site coordinateur situé à Besançon. Le site de Clermont-Ferrand du CNR de la résistance aux antibiotiques s'intéresse à la résistance chez les entérobactéries, notamment la

CNR Agents transmissibles non conventionnels (ATNC)	CNR Anaérobies et botulisme
CNR Anaérobies et botulisme	CNR Borrelia
CNR Brucella	CNR Campylobacter et Helicobacter
CNR Chagas (maladie de) en Guyane	CNR Charbon
CNR Chlamydiae	CNR Coqueluche et autres bordetelloses
CNR Corynébactéries du complexe diphteriae	CNR Cytomégalovirus
CNR Echinococcose alvéolaire	CNR Entériques (virus)
CNR Enterovirus et Parechovirus	CNR Escherichia coli, Shigella, Salmonella
CNR Fièvres hémorragiques virales	CNR Francisella tularensis
CNR Gonocoques	CNR Haemophilus influenzae
CNR Hantavirus	CNR Hépatites à transmission entérique (A et E) (virus des)
CNR Hépatites B, C et Delta (virus des)	CNR Legionella
CNR Leishmania	CNR Leptospirose
CNR Listeria	CNR Méningocoques
CNR Mycobactéries et résistance des mycobactéries aux antituberculeux	CNR Mycoses invasives et antifongiques
CNR Orthopoxvirus	CNR Paludisme
CNR Papillomavirus	CNR Peste et autres yersinioses
CNR Pneumocoques	CNR Rage
CNR Résistance aux antibiotiques	CNR Rickettsies Coxiella et Bartonella
CNR Rougeole (virus de la) et Paramyxoviridae respiratoires	CNR Rubéoleuses (infections) maternofœoetales
CNR Staphylocoques	CNR Streptocoques
CNR Syphilis	CNR Toxoplasmose
CNR Vibrions et choléra	CNR Virus de l'immunodéficience humaine (VIH)
CNR Virus influenzae (Grippe)	

Tableau I - Liste des 47 CNR, source personnelle d'après http://www.invs.sante.fr/Espace-professionnels/Centres-nationaux-de-reference/Liens-vers-les-sites-des-CNR

résistance aux céphalosporines de dernière génération par production de β-Lactamases* à Spectre Étendu (BLSE) et de céphalosporinases. Les missions qui lui sont assignées sont l'analyse de souches multirésistantes isolées en France ou à l'étranger, la surveillance épidémiologique de la résistance en lien avec l'InVS (en collaboration avec des réseaux européens, nationaux et régionaux), la diffusion de l'information sur les phénotypes de résistance à surveiller auprès des laboratoires d'analyses médicales, l'alerte auprès de l'InVS lors d'un événement épidémiologique inhabituel et enfin le conseil auprès des pouvoirs publics, agences de sécurité sanitaire et professionnels de la santé.

II. **Partie bibliographique et présentation du projet**

II.1. Antibiotiques et Mécanismes de Résistance

Les antibiotiques sont des molécules anti-infectieuses ciblant généralement des éléments critiques dans la croissance des bactéries. Les antibiotiques β-lactamines bloquent la synthèse de la paroi bactérienne alors que les aminosides, les phénicolés et les cyclines bloquent la synthèse protéique et enfin les sulfamides et le triméthoprime agissent sur la synthèse des acides nucléiques (cf. Figure 1).

Les mécanismes de résistance aux antibiotiques sont de différentes natures. La perte de l'affinité avec les antibiotiques résulte de mutations ponctuelles de gènes codant les protéines ciblées, à la différence de la diminution de la perméabilité des bactéries aux antibiotiques qui elle, résulte de mutations des gènes codant les porines*. Dans le premier cas, les antibiotiques n'auront aucun effet alors que dans le 2ème cas, l'antibiotique bien que toujours actif ne pourra simplement pas s'incorporer à la cellule ou alors en quantité particulièrement amoindrie. On note également la production d'enzymes inactivatrices qui altèrent directement la structure des antibiotiques et enfin, l'excrétion de l'antibiotique suite à l'acquisition ou la sur-expression de systèmes d'efflux (principe de pompe vers le milieu extérieur).

Les antibiotiques β-lactamines comptent quatre sous familles : les pénicillines (pénames), les céphalosporines (céphèmes), les carbapénèmes (pénèmes) et enfin les monobactames (cf. Figure 2). Le mécanisme principal de résistance chez les

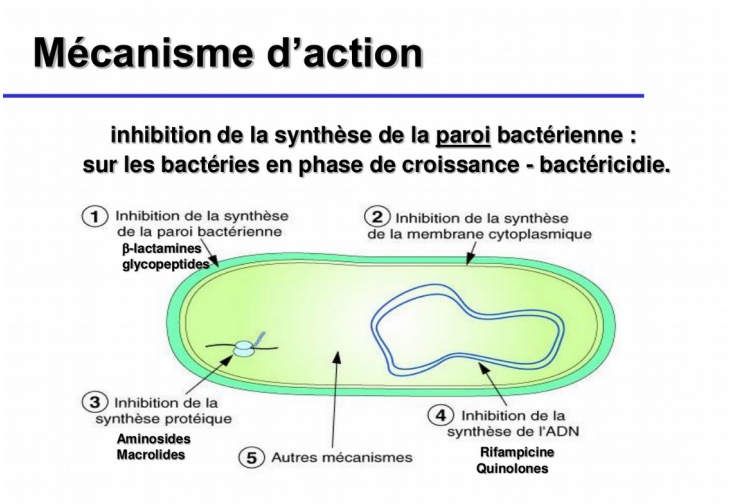
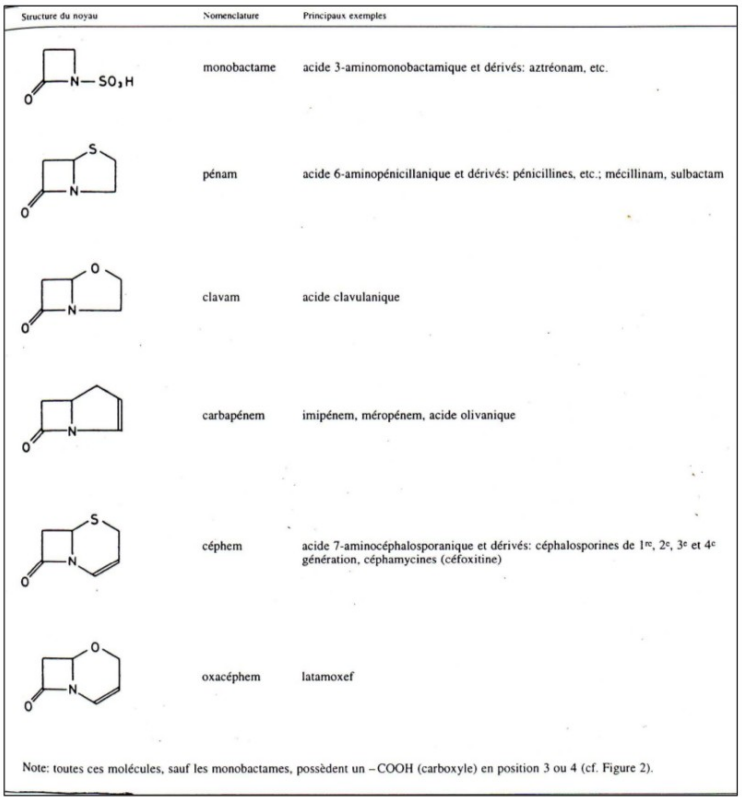


Figure 1 – Mécanismes d'action des antibiotiques
Source : Molina, Jean-Michel ; L'essentiel en pratique sur les β-lactamines



Structure du noyau	Nomenclature	Principaux exemples
	monobactame	acide 3-aminomonobactamique et dérivés: aztréonam, etc.
	pénam	acide 6-aminopénicillanique et dérivés: pénicillines, etc.; mécillinam, sulbactam
	clavam	acide clavulanique
	carbapénem	imipénem, méropénem, acide olivanique
	céphem	acide 7-aminocéphalosporanique et dérivés: céphalosporines de 1re, 2e, 3e et 4e génération, céphamycines (céfoxitine)
	oxacéphem	latamoxef

Note: toutes ces molécules, sauf les monobactames, possèdent un –COOH (carboxyle) en position 3 ou 4 (cf. Figure 2).

Figure 2 – Nomenclature des β-lactamines en fonction de la structure de leur noyau
Source : http://www.microcsb.net/IMG/pdf/doc-49.pdf

entérobactéries est la production d'enzymes inactivatrices appelées β-Lactamases. (Cavallo, Fabre, et al., 2004)

Selon la nature des substrats préférentiels, on distingue les pénicillinases, les céphalosporinases, les carbapénémases et enfin les β-Lactamases à Spectre étendu (BLSE) qui hydrolysent toutes les pénicillines et l'ensemble des céphalosporines (cf. Tableau II). Au total, plus de 500 β-Lactamases différentes ont été caractérisées. Parmi les β-Lactamases les plus courantes, on dénombre la pénicillinase TEM-1 et les BLSE de type CTX-M dont il existe 110 variants. Les principaux mécanismes de résistance aux aminosides sont également des enzymes modificatrices : des acétyltransférase, des phosphotransférase ou des nucleotidyltransférase. Les phénicolés peuvent être inactivés par des acétyltransférases ou pris en charge par des systèmes d'efflux. Enfin, la résistance aux sulfamides est généralement liée à l'acquisition d'un gène codant une cible (dihydrofolate réductase) qui présente une affinité moindre ou qui est hyper-produite.

II.2. Gènes de virulence

Les gènes de virulence sont souvent compris au sein d'îlots de pathogénicité (PAIs) intégrés au sein de plasmides ou le plus souvent du chromosome bactérien. Ils sont généralement associés à des éléments mobiles, qui favorisent leur transfert horizontal (HGT). (Croxen et Finlay, 2009) Le transfert de gènes horizontal est une méthode d'acquisition de matériel génétique sans rapport de descendance.

Escherichia coli est une espèce bactérienne versatile, qui comprend des souches commensales intestinales non pathogènes chez l'hôte sain et des souches pathogènes. Cette diversité s'objective au niveau de la taille du génome, qui varie d'une souche à une autre entre 4 et 6 millions de nucléotides. Environ 2000 gènes seulement sont strictement conservés dans tous les isolats de *E. coli* et forment le « core genome* ». Ce « génome conservé* » permet de décrire plusieurs phylogroupes* bactériens au sein de l'espèce *E. coli.* Les phylogroupes* B2 et D comprennent l'essentiel des souches pathogènes alors que les souches commensales appartiennent plutôt aux phylogroupes* A ou B1. (Cuevas Ramos, 2010) Les gènes non conservés de *E. coli* varient en nombre (entre 2000 à 4000 gènes par souche), en nature et sont issus d'une collection d’environ

Tableau 1 Classification des β-lactamines selon Bush et al. [6], Ambler [5] et proposition de Giske et al. [7].
β-lactam classification according to Bush et al. [6], Ambler et al. [5] and Giske et al. [7].

Type	Bush	Ambler	Giske	Inhibiteur	Exemple
Céphalosporinase de type AmpC	1	C	$ESBL_{M-C}$	Cloxacilline	AmpC des entérobactéries du groupe 3 et leurs dérivés plasmidiques
Pénicillinase des bactéries Gram positifs	2a	A	–	Clavulanate	Pénicillinase de *Staphylococcus aureus*
Pénicillinase à spectre étroit	2b	A	–	Clavulanate	TEM-1, SHV-1
Bêta-lactamases à spectre élargi	2be	A	$ESBL_A$	Clavulanate	TEM-3, SHV-2, CTX-M, PER, VEB, GES (170Gly)
Pénicillinases résistantes aux inhibiteurs	2br	A	–	–	TEM-30
Complex mutant TEM	2ber	A	$ESBL_A$	–	TEM-50
Carbénicillinases	2c	A	–	Clavulanate	PSE-1
Oxacillinases	2d	D	–	–	OXA-1
Céfuroximases	2e	A	–	Clavulanate	Céphalosporinase de *Proteus vulgaris*
Carbapénémases	2f	A	$ESBL_{CARBA-A}$	Clavulanate	KPC, GES (170Ser et 170Asn)
Métallo-bêta-lactamases	3	B	$ESBL_{CARBA-B}$	EDTA	VIM, IMP
Autres bêta-lactamases non classées	4	–	–	–	Pénicillinase de *Burkholderia cepacia*

Tableau II – Classification des sous familles de β-Lactamases

Source : Revue Antibiotique Vol 12 issue 01

http://issuu.com/clubscientifique/docs/revue_antibiotique_vol_12_issue_01___www.clubscien/7

23000 gènes codant pour des fonctions optionnelles impliquées dans le métabolisme, la résistance aux antibiotiques ou encore la virulence. Ces derniers facteurs de virulence codés par les souches pathogènes sont nécessaires pour vaincre les défenses de l'hôte, adhérer à ses cellules, les envahir et/ou déclencher une inflammation.

Parmi ces facteurs de virulence, les adhésines sont des facteurs d'adhésion permettant la colonisation qui interagissent avec un récepteur sur la cellule de l'hôte. Elles ont le plus souvent la forme de fimbriae* mais on distingue d'autres mécanismes d'adhésions comme les adhésines afimbriales ou encore la formation de « piédestal » (restructuration de l'actine des microvillosités intestinales). (Kaper, Nataro, et Mobley, 2004) (cf. Figure 3)

Le fer étant un élément fondamental à la croissance des bactéries, elles sécrètent des sidérophores* capables de complexer les ions ferriques dans l'environnement. Une fois en associés au fer, ces sidérophores* sont captés par la bactérie à l'aide d'un récepteur spécifique à la surface de la membrane.

La capsule est un autre attribut de virulence bactérien. Elle est composée de polysaccharides* permettant aux bactéries de se dissimuler du système immunitaire, notamment envers les phagocytes. Elle confère ainsi une meilleure survie au sein de l'hôte lors de l'invasion.

Les bactéries peuvent produire différents systèmes de sécrétion permettant d'adresser des effecteurs* bactériens aux cellules de l'hôte afin de subvertir ses systèmes de défenses. Les principales sont le facteur cytotoxique nécrosant 1 (cnf1) induisant une nécrose en ciblant des protéines régulatrices, l'hémolysine A qui forme des pores et entraîne une fuite des ions au niveau de la cellule de l'hôte, les toxines sat/vat qui sont des autotransporteurs* et enfin la colibactine qui induit une cassure de l'ADN.

Suivant la nature des facteurs de virulence codés par la bactéries, les souches pathogènes seront des *E. coli* pathogènes intestinales (ECPI) responsables de diarrhées ou des *E. coli* pathogènes extra-intestinales (ECPEx) responsables d'infections urinaires ou de septicémies.

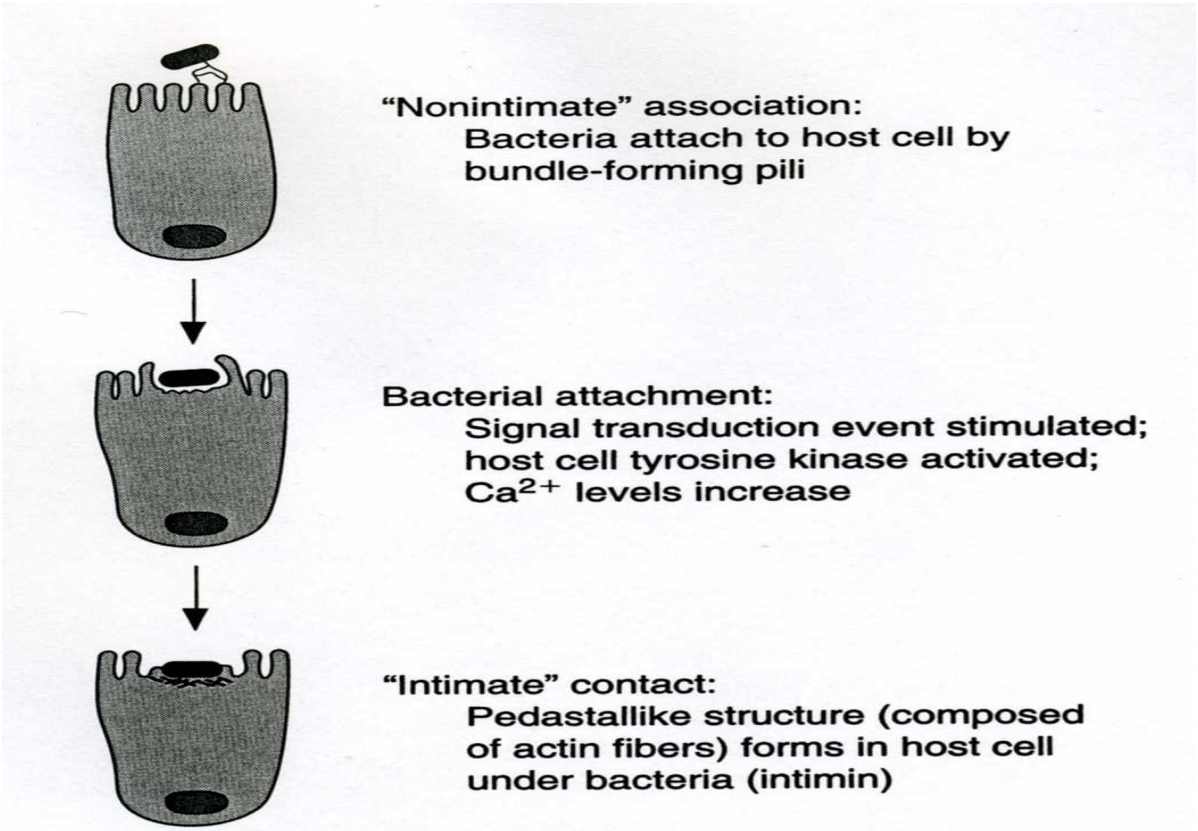


Figure 16-4 Steps in the binding of EPEC strains to host cells and effect of binding on host cell functions.

Figure 3 – Étapes de liaison des EPEC au niveau des microvillosités intestinales de l'hôte

Source : Scheftel, J-M ; Mémoire : Les facteurs de virulence des bactéries à tropisme intestinal
http://www.jn0.free.fr/M1 Physiopath/Microbiologie/les Facteurs de virulence des bactéries à tropisme intestinal.ppt

II.3. Méthodes de Typage

Le typage moléculaire des bactéries consiste à rechercher des marqueurs moléculaires spécifiques de la souche bactérienne ou d'un phylum au sein de l'espèce bactérienne analysée. La plupart des méthodes reposent sur l'analyse de la séquence nucléique de gènes conservés. La plus utilisée est le Multi-Locus Sequence Typing (MLST). Cette technique consiste à établir le profil allélique (ou Sequence Type (ST)) de 7 à 15 gènes de ménage. Les gènes sélectionnés sont différents d'une espèce à l'autre. Chez *E. coli,* les gènes utilisés sont *dinB* (ADN polymerase), *icdA* (Isocitrate dehydrogenase), *pabB* (p-aminobenzoate synthase), *polB* (Polymerase PolII), *putP* (Proline permease), *trpA, trpB* (Tryptophane synthase sous-unités A et B) et *uidA* (Beta-glucuronidase). Chaque allèle est identifié à l'aide d'un numéro dans les bases de données internationales. La combinaison des numéros d'allèles correspondant à un numéro de ST. La MLST permet d'établir des liens de parenté entre les souches proches et de relier les souches aux phylogroupes* de *E. coli* (A, B1, B2, C, F...). Elle ne permet cependant pas d'établir des liens de parenté distants et n'est pas assez discriminante pour suivre la diffusion épidémique d'une souche. (Brisse, Jaureguy, et al., 2008)

De nouvelles méthodes de typage sont à l'étude, ainsi on citera la cgMLST (core genome MLST) qui est un typage MLST qui repose sur plusieurs centaines de gènes du core genome. De ce fait, il n'est pas possible d'affecter un profil pour chaque combinaison allélique faute de base de données existante ou bien par la seule quantité de ST que cela générerait. Cette analyse, plus discriminante que la MLST originale, est alors menée selon une étude phylogénique sur la base des mutations observées. On citera également la rMLST (Ribosomal MLST) qui repose sur les allèles de 53 gènes *rps* codant les sous-unités protéiques ribosomales des bactéries. (cf. Figure 4)

Chez *K. Pneumoniae*, la présence d'une capsule permet une autre méthode, le typage capsulaire basé sur les allèles du gène wzi. Ce gène code une protéine qui permet l'attachement de la capsule à la surface de la cellule. Le type capsulaire obtenu peut être associé avec les ST déterminés à l'issue d'une MLST (cf. Tableau III) et traduire une virulence importante et/ou d'une multi-résistance aux antibiotiques. Il peut donner lieu à un arbre phylogénique pour mieux cerner les proximités et analogies entre souches. (Brisse, Passet, Haugaard, et al., 2013)

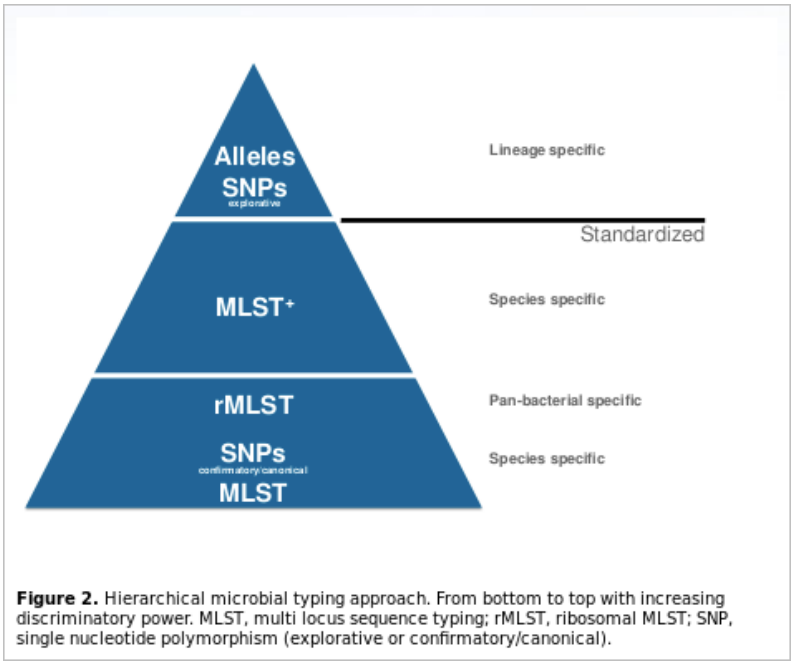


Figure 2. Hierarchical microbial typing approach. From bottom to top with increasing discriminatory power. MLST, multi locus sequence typing; rMLST, ribosomal MLST; SNP, single nucleotide polymorphism (explorative or confirmatory/canonical).

Figure 4 – Hiérarchie des différentes méthodes de typage en fonction du pouvoir discriminant
Source : http://www.ridom.de/seqsphere/mlstplus/

Table S1. Correspondence of *wzi* alleles with K-types

wzi allele	Associated K types	Reference strain	Previously characterized strains	Clinical isolates	Total	Clonal group(s) #
wzi-1	K1		4	9	13	ST23
wzi-2	K2		6	8	14	ST14, ST78, ST86, ST380
wzi-3	K3	1			1	ST3
wzi-4	K2	1			1	ST66
wzi-5	K5	1	4		5	ST60, ST61, ST91 (K. p. subsp. ozaenae)
wzi-6	K6	1			1	ST91 (K. p. subsp. ozaenae)
wzi-7	K7	1	1	2	4	ST6, ST154
wzi-8	K8	1	2	1	4	ST9, ST29
wzi-9	K9	1	2		3	ST22, ST560
wzi-10	K10	1			1	ST297
wzi-11	K11	1			1	
wzi-12	K12	1			1	
	K29			1	1	
wzi-13	K13	1			1	
wzi-14	K14		1	3	4	ST37
wzi-15	K35			1	1	
wzi-16	K16	1	1	1	3	
wzi-17	K14		1		1	
wzi-18	K18	1			1	ST562
wzi-19	K19	1			1	ST563
wzi-20	K20		3		3	
wzi-21	K21	1			1	ST290
wzi-22	K22.37	1		1	2	ST155
wzi-23	K23	1			1	ST416
	K14	1		3	4	ST1227
wzi-24	K24	1	8	2	11	ST15, ST59
	NT			1	1	
wzi-25	K25	1		1	2	ST47
wzi-26	K27			1	1	
wzi-27	K27	1		1	2	ST298
wzi-28	K28	1			1	ST564
wzi-29	NT			3	3	
wzi-30	K30	1	2		3	ST43, ST135
wzi-31	K31	1	1		2	ST11, ST34
wzi-32	K31		1		1	
wzi-34	K34	1			1	ST500
	K57				0	
wzi-35	K35		1		1	
wzi-36	K36	1			1	ST565

Tableau III – Extrait d'un tableau de correspondance des allèles du gène Wzi, types K et ST
Source : « wzi Gene Sequencing, a Rapid Method for Determination of Capsular Type for Klebsiella Strains »
http://jcm.asm.org/content/suppl/2013/11/13/JCM.01924-13.DCSupplemental/zjm999092989so2.pdf

II.4. Origines de réplication des plasmides et protéines Mob

Les plasmides sont des molécules d'ADN distinctes du chromosome bactérien présents en de nombreux exemplaires. Ils ne sont pas nécessaires mais apportent des avantages à la bactérie et participent à la propagation de gènes de résistance, de gènes métaboliques ou codant des facteurs de virulence. Ils possèdent un système de réplication autonome, appelé réplicon. Les réplicons comprennent l'origine de réplication et les protéines d'initiation de la réplication « rep ». (Carattoli, 2013) Deux plasmides ayant des origines de réplication identiques ne peuvent pas cohabiter dans une même bactérie et appartiennent donc au même groupe d'incompatibilité. Il est possible de classer et typer les plasmides en fonction de leur système de réplication dans différents groupes d'incompatibilité qui sont au nombre de 27. (Carattoli, 2009)

L'originalité des plasmides est de pouvoir être transféré de façon horizontale entre bactéries. Le mécanisme de transfert horizontal le plus efficace est la conjugaison. Les plasmides conjugables possèdent le matériel génétique nécessaire à leur transfert d'une bactérie à l'autre. Les plasmides ont en commun la relaxase, protéine « mob » multi-domaine qui reconnaît l'ADN, sépare les deux brins d'ADN du plasmide dont l'un sera marqué pour le transfert. Cependant seuls les plasmides conjugables possèdent le système de sécrétion de type IV « T4SS » aussi appelé pili sexuel et la protéine de couplage « T4CP » dont le rôle est de guider le plasmide marqué par la relaxase à travers le pili. Le T4SS permet d'établir un pont plasmidique entre la bactérie donneuse et la bactérie receveuse. La majorité des plasmides ne sont pas conjugables mais mobilisables et possèdent alors la relaxase mais pas la capacité de synthétiser le système de sécrétion. Ils doivent par conséquent faire appel à un plasmide conjugable pour être transférés.

Grâce à la conjugaison et la mobilisation, les plasmides sont responsables de la diffusion de gènes et peuvent avoir un impact écologique et épidémiologique majeur en étant notamment responsables de la diffusion des BLSE dans le monde bactérien.

II.5. Présentation du projet

En ayant fait gagner plus de 10 ans d'espérance de vie à l'espèce humaine, les molécules antibiotiques forment une classe thérapeutique sans équivalent de la

pharmacopée humaine et animale. Leur découverte constitue sans nul doute la plus grande découverte médicale du vingtième siècle. Elle a permis de révolutionner le pronostic d'infections, actuellement considérées comme bénignes, et qui était fréquemment létales à l'ère pré-antibiotique, comme une simple surinfection de plaie cutanée ou un panaris. Préserver l'efficacité de ces molécules constitue donc un enjeu majeur de santé publique.

On considère qu'il faut dix à quinze ans pour développer un nouvel antibiotique, alors que les bactéries résistantes aux antibiotiques émergent le plus souvent deux à cinq ans après les premières utilisations. Aussi, il est indispensable de développer de nouvelles molécules mais aussi de rationaliser leur utilisation pour augmenter le délai d'apparition des bactéries résistantes et limiter leur expansion. La rationalisation de l'utilisation des antibiotiques repose sur une surveillance de l'émergence et de la diffusion des mécanismes de résistance. En France, cette mission incombe au CNR de la résistance aux antibiotiques, l'objectif est de mettre à disposition des autorités sanitaires françaises et européennes des données permettant de définir des directives pour l'utilisation des antibiotiques.

Jusqu'alors la surveillance de la résistance aux antibiotiques reposait sur des approches phénotypiques et des approches moléculaires classiques (amplification génique et séquençage selon la méthode de Sanger). La multiplication des mécanismes de résistances, la création de bases de données et l’avènement de nouvelles technologies de séquençage a incité le CNR de la résistance aux antibiotiques à modifier cette approche au profit d'un séquençage du génome bactérien par séquençage à haut débit. Si le séquençage à haut débit est facile à implémenter, il manque toutefois des outils adaptés permettant une analyse automatisée et standardisée complète des données de séquençage.

Dans ce contexte, le projet présenté dans ce rapport visait à assembler un pipeline d'analyse bioinformatique pour l'étude de la résistance et de la virulence de bactéries pathogènes par séquençage à haut débit. Le projet a été mené sur un jeu de données de bactéries multirésistantes isolées dans 18 hôpitaux français représentatifs. L'objectif était de déterminer les ST et la phylogénie des souches, de détecter les gènes de résistances, de virulence ainsi que les plasmides. Pour ce faire, le projet s'articule

autour de cinq axes principaux qui constituent le pipeline d'analyse à savoir :

- la prise en charge et le prétraitement des reads* (élimination des adaptateurs et régions de mauvaise qualité),
- la détection de gènes (résistance ou virulence) et de plasmides,
- le typage de séquences issus de gènes de ménage (détermination des ST),
- l'assemblage *de Novo* des génomes,
- et l'analyse phylogénique du core genome (cf. Figure 5).

Ce travail nécessite des tâches annexes telles la construction et la vérification des bases de données, la standardisation et l'analyse de la reproductibilité des résultats et notamment le contrôle des résultats afin de vérifier qu'aucun biais n'a pu être introduit, qu'il soit humain ou non, délibéré ou non.

III. **Matériel et méthodes**

III.1. Souches bactériennes

Les souches étudiées ont été préalablement sélectionnées au moyen d'un protocole spécifique dans 18 hôpitaux Français représentatifs. Les souches ont été analysées suivant un protocole spécifique pour déterminer leurs mécanismes de résistances et ont donné suite à des analyses complémentaires si nécessaire (cf. Figure 6). Ces analyses sont ciblées et requièrent des expérimentations associées à chaque gène ou famille de gènes et peuvent donc s'étaler sur plusieurs mois.

Ainsi les souches qui ont servies à mettre en place et tester le pipeline sont au nombre de 95, représentant près de 350 Go de données brutes. Parmi ces souches on distingue 30 *Enterobacter clocae*, 36 *Escherichia coli* et 29 *Klebsiella pneumoniae*. En fonction de la précision, du temps donné et du taux de recouvrement souhaité pour effectuer le séquençage (réalisé par la société Helixio), la taille des fichiers est considérablement variable, allant de 1 à 14 Go par souche. Il en résulte deux fichiers fastq par souche issus de la technologie Illumina NextSeq 500 en paired-end démultiplexé (les fragments d'ADN sont « tagués » par souche avec une séquence spécifique pour assurer leur suivi et sont tous séquencés simultanément). Les fichiers fastq comprennent un score de qualité déterminé par le séquenceur pour chaque nucléotide. En paired-end on distingue un fichier « forward » et un fichier « reverse »

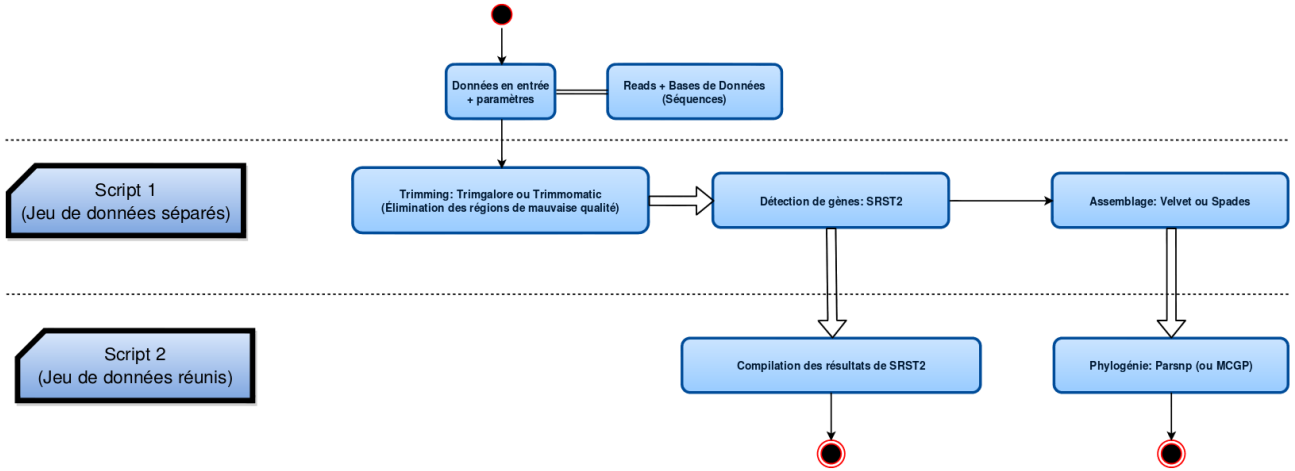


Figure 5 – Schéma définitif du pipeline

Source personnelle

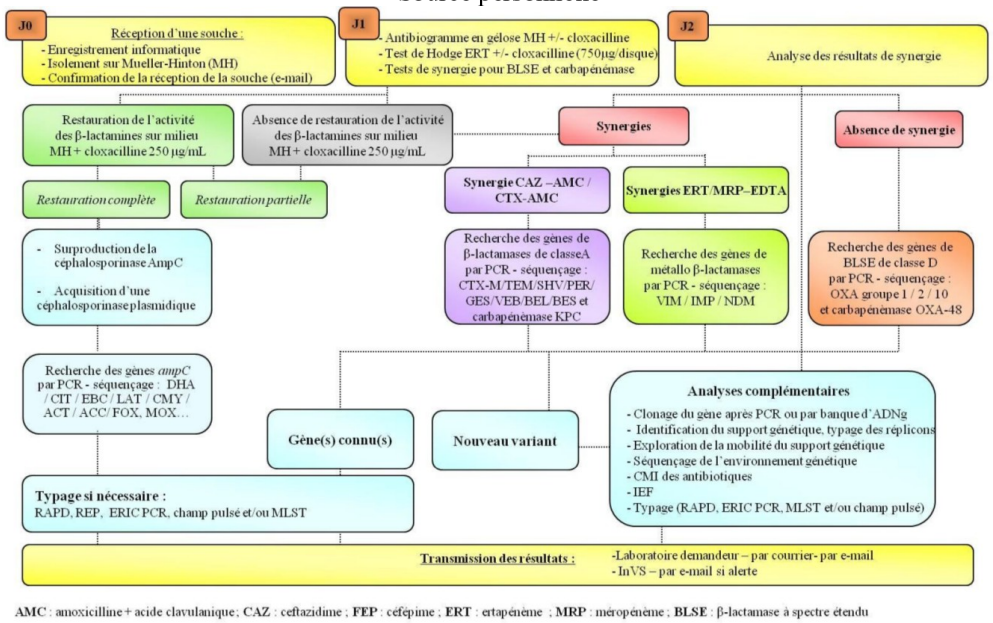


Figure 88. Démarche d'analyse des souches d'entérobactéries par le CNR

Figure 6 – Procédure des analyses des souches receptionnées

Source : Rapport d'activité 2014 des CNR – http://www.cnr-resistance-antibiotiques.fr/ressources/pages/Rapport_CNR_2014.pdf

(cf. Figure 7). Il s'est avéré après de multiples analyses qu'une souche de *Klebsiella* est en fait une *Enterobacter* et inversement et on note la présence d'une souche *Enterobacter aerogenes* menant ainsi à des résultats surprenant au premier abord.

Au total ce seront près de 200 souches qui seront à terme analysées au moyen du pipeline afin de poursuivre l'étude initiée par le CNR de la résistance. Une seconde étude portant des souches de *K. pneumoniae* productrices de carbapénèmases est par ailleurs en cours sur 170 isolats.

III.2. Outils bioinformatiques

Le langage de programmation le plus utilisé au sein du pipeline est le Python, en effet de nombreux outils incorporés au pipeline ainsi que les scripts le constituant sont écrits en Python. C'est un langage de programmation interprété orienté objet qui permet d'écrire du code fonctionnel relativement vite de par sa syntaxe et de par la variété des fonctions qu'il propose. Par soucis de compatibilité, la version utilisée est Python 2.7.6. En parallèle, BioPython qui est une bibliothèque spécifique au traitement de séquences biologiques, est utilisée ponctuellement. Elle permet par exemple de modifier à grande échelle les en-têtes d'un fichier .fasta en prenant en charge la syntaxe de l'en-tête (identifiant, nom de la séquence, description, etc.) ou encore l'alphabet utilisé (ADN, ARN, acides aminés) en vue de construire des bases de données pour certains outils. La bibliothèque graphique wxPython a également été utilisée pour la partie interface. Cette bibliothèque permet de construire des fenêtres constituant alors une interface plus intuitive et accessible qu'un programme utilisé en ligne de commande mais cette axe de développement a été abandonnée car le Pipeline est finalement destiné à une être incorporé sur une grille de calcul.

En terme de prétraitement, notamment pour le trimming, les outils utilisés sont trimmgalore (Perl) et Trimmomatic (Java). Ils permettent d'exclure les adaptateurs (séquences spécifiques) nécessaires aux méthodes de séquençage ainsi que les régions de faible qualité et éliminent automatiquement les reads* réduits à une longueur trop faible. Ils peuvent également réaliser une analyse de la qualité des séquences avec l'outil FastQC permettant de visualiser par exemple la longueur moyenne des reads*, la qualité en fonction de la longueur, la fréquence de chaque base en fonction de la longueur ou le

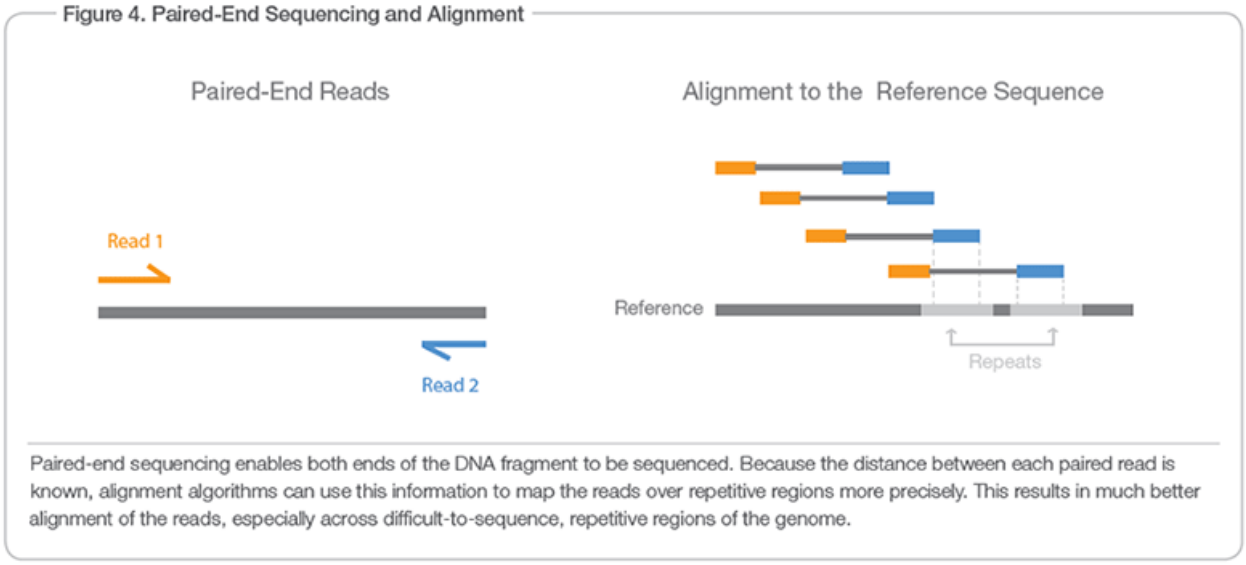


Figure 7 – Principe du séquençage paired-end (Read1 : Forward ; Read2 : Reverse)
Source : http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html

taux de bases indéterminées.

Concernant la détection de gènes, l'outil principal se nomme SRST2 (Short Read Sequence Typing), il est le fruit de la collaboration de plusieurs docteurs de différents départements de l'université de Melbourne. C'est un « outil basé sur l'alignement de reads pour la détection rapide et précise de gènes, allèles et multi-locus sequence types (MLST) à partir de données Whole Genome Sequencing (WGS)». (Holt, Inouye, Dashnow, et al, 2014) Il repose en grande partie sur le programme Bowtie2 qui permet l'alignement de reads* sur de longues séquences de référence. Les données en entrée sont les reads* en paired-end et les Bases De Données (BDD) de gènes de résistance ou de virulence et MLST. Ces BDD peuvent être construites à partir des scripts fournis avec le programme car elles nécessitent un clustering* au moyen de cd-hit (qui va regrouper les séquences par analogie au-delà d'un certain seuil) et doivent répondre à une syntaxe rigoureuse.

La BDD des gènes de résistance utilisée est une version curée manuellement de « ArgAnnot » (Antibiotic Resistance Gene-ANNOTation), une BDD produite par l'Infectiopôle Sud et Méditerranée Infection. Elle comprend 1684 gènes de résistance. La BDD de gènes de virulence comptabilise 35874 gènes et a été formatée manuellement à partir de la BDD déjà existante Virulence Factor DataBase (VFDB). Concernant le typage MLST, les séquences constituant la base de données et la correspondance entre les combinaisons d'allèles et ST ont été récupérés à partir de PubMLST.org, une BDD de schémas MLST publique. De plus, deux bases de données ont été construites pour la détection des plasmides, une permettant de différencier les 33 réplicons plasmidiques connus et une seconde permettant de détecter 638 allèles codant des protéines « mob ».

Pour l'assemblage *de Novo* (assemblage probabiliste des reads* en alignant les zones de chevauchement) des génomes, plusieurs d'outils ont été testés: Spades, Masurca, Velvet avec VelvetOptimizer, iMetAmos et SoapDeNovo. Mosaik a quant à lui été testé pour l'assemblage par alignement sur génome de référence. Seul Spades a donné des résultats satisfaisants (soit peu de long contigs* et un N50 de plusieurs dizaines de milliers de paires de bases) mais le temps nécessaire à son exécution est particulièrement élevé.

L'analyse phylogénique du core genome* séquencé a été testé avec deux outils. Le première fait appel à Maximum Common Genome Phylogeny (MCGP). Cet outil détecte les cadres de lecture ouverts* ou Open Reading Frames (ORF) avec Prodigal et les regroupe selon leur similarité avec cd-hit pour retenir les seuls gènes conservés dans toutes les souches étudiées. Après une étape de trimming, il recherche les « Single Nucleotide Polymophism » (SNP) afin de construire la phylogénie. On peut lui préférer parsnp et la suite Harvest qui en plus d'offrir la même démarche avec alignement sur génome de référence annoté, prend en charge les recombinaisons.

III.3. Méthodes expérimentales

L'extraction du matériel génétique nécessaire au séquençage à haut débit ou à l'amplification génique a été réalisée à l'aide du kit « MoBio UltraClean® Microbial DNA Isolation Kit ». Le protocole comprend les étapes de culture bactérienne (jusqu'à une densité maximale de quelques milliards de cellules par mL), la lyse cellulaire pour éliminer la structure de la cellule et libérer l'ADN, l'élimination des composants cellulaires (lipides membranaires, protéines, etc.) et enfin la précipitation de l'ADN pour obtenir une solution dont la pureté sera mesurée par spectrophotométrie au NanoDrop® (260/280 nm pour vérifier la contamination protéique et 260/230 nm pour la contamination à l'isopropanol). L'ADN a été ensuite fourni à la société Hélixio pour séquençage à l'aide de la technologie Illumina Miseq.

La PCR (Polymerase Chain Reaction) permet d'amplifier un fragment ciblé à l'aide d'amorces dessinées spécifiquement, la cible est généralement un gène d'intérêt. Il est nécessaire de constituer un « mix » contenant de la taq polymérase « Promega GoTaq® DNA Polymerase » (5 unité.μL^{-1}), du tampon 5X « Green GoTaq » (dilué pour obtenir une concentration 1X), des dinucléotides triphosphates « dNTP MasterMix », un agent intercalant qu'est le Bromure d'Ethidium (10mg.mL^{-1}) et enfin les amorces. Vient s'ajouter au kit, en dernière instance, les échantillons d'ADN à amplifier. Une PCR est constituée de nombreux cycles comprenant trois phases distincte. Une première phase de dénaturation où l'ADN double brin est séparé puis s'en suit une phase d'hybridation où l'amorce se fixe à l'un des brins et enfin une phase d'élongation où le brin complémentaire est synthétisé à partir de l'amorce. Il en résulte une grande quantité de fragments de même séquence pouvant être observés par électrophorèse au moyen d'un

agent intercalant avec des témoins positifs, négatifs (pour confirmer ou infirmer la présence d'un gène) ou éventuellement une échelle de poids moléculaire pour estimer la taille des fragments.

La purification permet d'éliminer toute trace de résidus suite à une PCR, notamment les dimères d'amorces de PCR et d'amorces amplifiées pour les PCR. Elle consiste en plusieurs étapes de filtration à base d'éthanol avec des colonnes filtrantes issues du kit « MoBio UltraClean® PCR Clean-Up Kit ». Après dosage, l'ADN peut être séquencé selon la méthode de Sanger.

IV. **Démarche et travaux réalisés**

IV.1. Pipeline d'analyses de données NGS

Le développement du pipeline a été revu plusieurs fois afin de cibler convenablement les objectifs et finalités du projet qui ont évolués tout au long de la mission. Une phase d'essai des outils utilisés a permis de sélectionner les plus performants et leurs paramètres optimaux et de s'assurer de la qualité des résultats. S'en est suivi le développement, d'abord en vue d'une interface graphique destinée à être utilisée localement, puis en ligne de commandes dans le but de pouvoir incorporer le pipeline au sein d'une grille de calcul. De plus l'acquisition de résultats s'est déroulée tout au long du projet et pas seulement à titre de test mais également pour comparer les études préalables faisant appel aux méthodes expérimentales.

Dès le début du projet, le trimming des séquences, soit le fait de s'assurer que seule les régions de bonne qualité aient été conservées, a été nécessaire à tous les traitements postérieurs. Un script a permis d'automatiser cette tâche pour l'ensemble des 95 souches.

La phase de détection de gènes a alors débuté, la base de donnée des gènes de résistance était fournie avec l'outil SRST2 mais a nécessité une correction car certaines séquences étaient mal nommées et portaient à confusion. Une marche à suivre dans le manuel du programme a permis de construire la plupart des bases de données de gènes de virulence à partir de VFDB à l'exception de *Klebsiella pneumoniae*. Les études concernant cette espèce étant moins nombreuses, la construction de cette base de

donnée s'est appuyée sur les travaux de Sylvain BRISSE de l'institut Pasteur. Une fois de plus, un script a permis de parcourir la base de données de l'institut pour récupérer les allèles des gènes de virulence de cette espèce, de les concaténer et les formater de sorte à être compatible avec SRST2.

Concernant le typage MLST, les séquences constituant la base de données et la correspondance entre les combinaisons d'allèles et ST ont été récupérés à partir de PubMLST.org, BDD de schémas MLST publique.

L'assemblage quant à lui a été un défi majeur, le choix entre l'assemblage *de Novo* et l'assemblage par alignement sur génome de référence a été discuté. Le *de Novo* est généralement utilisé quand un doute subsiste sur la nature de l'organisme séquencé, notamment en métagénomique, cependant l'alignement sur génome de référence pose la question de la légitimité du génome de référence et ne permet pas de résultats optimaux si les génomes diffèrent de trop. Dans notre cas, pour une étude épidémiologique concentrée sur les différences de caractères exprimés, le *de Novo* s'est avéré plus pertinent et s'en est suivi de nombreux essais de nombreux outils. Les résultats n'étaient pas satisfaisants, les contigs* obtenus étaient trop courts et trop nombreux. CLC genomics a permis l'assemblage *de Novo* des génomes en vue de résultats mais ne pouvait pas être intégré dans le pipeline.

Une première BDD de plasmides sommaire a permis de détecter les différents groupes d'incompatibilité puis en fonction des résultats l'analyse a été complétée avec des bases de données plus fournies spécifiques à chaque groupe d'incompatibilité.

La phylogénie de l'ensemble des souches a été réalisée pour chaque espèce séparément et nécessitait des génomes assemblés. Les résultats étaient satisfaisants dès lors que les souches mal identifiées en ont été exclues.

L'interface graphique développée avec WxPython aurait permise en plus de l'automatisation des analyses citées précédemment, d'effectuer un suivi des souches en intégrant des métadonnées en vue du stockage dans une BDD mais le projet s'est avéré trop ambitieux et l'opportunité d'une grille de calcul a définitivement exclue cette idée.

La structure finale du pipeline a donc pris la forme d'un programme en python en ligne de commande afin de pouvoir être intégré au sein d'une machine virtuelle sur

une grille de calcul permettant d'accélérer les traitements. Afin d'exploiter le potentiel d'une grille de calcul, il a été nécessaire de séparer les tâches en fonction des données en entrée. Ainsi une première partie permet d'effectuer les tâches pour une seule souche indépendamment des autres et comprend le trimming, la détection de gènes et plasmides, le typage et enfin l'assemblage et pourra. À terme, les analyses de chaque souches pourront être lancées en parallèle et non de manière sérialisée. La deuxième partie nécessite le traitement de toutes les souches pour compiler les résultats de détection de gènes et effectuer la phylogénie à partir des génomes assemblés.

IV.2. Résultats obtenus et exploitation

Bien que le résultat du projet soit le développement du pipeline (cf. Annexe 1), il a toutefois permis d'obtenir des informations essentielles sur les caractères des souches étudiées. Des statistiques relatives à l'exécution sont nécessaires à la prise en charge du pipeline sur la grille de calcul afin de paramétrer les machines virtuelles (cf. Tableau IV).

Les paragraphes qui suivent concernent les résultats de caractérisation des souches et se concentrent exclusivement sur les résultats issus d'*Escherichia Coli*, étant l'espèce la plus étudiée à ce jour ses gènes et mécanismes sont les mieux connus.

Les résultats de la détection de gènes se présente sous la forme d'un tableau qui indique l'allèle détecté pour chaque gène s'il y a lieu. Les allèles encore inconnus, s'ils ont une identité relativement proche des allèles présents dans les bases de données sont automatiquement renseignés dans une fichier .fasta. Le tableau des résultats de la détection des gènes de résistance compte 32 gènes. Le tableau de 32 colonnes par 36 lignes qui en résulte n'étant pas pertinent en lui-même, le nombre de gènes détectés par souche rapportés à la famille d'antibiotique associée est plus représentatif des mécanismes et défenses acquises par les bactéries (cf. Annexe 2). On peut en déduire que toutes les souches étudiées possèdent au moins un gère de résistance associé aux β-Lactamines, qu'environ deux tiers des souches possèdent des mécanismes de résistance contre les Aminoglycosides et Sulfamides et enfin qu'elles sont plus vulnérables aux Tetracyclines, Phénicolés et notamment aux Macrolides pour lesquels seules 6 souches possèdent des gènes de résistance. Ce faible nombre s'explique par le

	1 souche		2 souches	26 souches
	Trimming + Détection	**Trimming + Détection + Assemblage**	**Compilation résultats + Phylogénie**	
Fastq en entrée	2 < taille < 5,5Min : 1,2 GoMax : 14,2 Go		nb_souches x 10 Mo	
			20 Mo	260 Mo
Espace nécessaire au calcul	Environ 2,6 Go	Environ 3,3 Go au strict minimum	10 Mo ± 1 Mo	97 Mo ± 1 mo
Volume des fichiers de résultats	600 ko ± 1 Mo + 1 Go si on récupère les fichiers trimmés	10 Mo ± 1 Mo + 1Go si on récupère les fichiers trimmés		
Temps d'exécution (thread = 1)	3 h 13 min	10 h 32 min	20 secondes	5 min 37 secondes

Tableau IV – Statistiques d'exécution du pipeline

Source personnelle

fait que les macrolides sont des antibiotiques qui n'affectent pas les bactéries gram – car ils sont incapables de traverser leur paroi externe.

Les facteurs de virulence détectés sont au nombre de 217 mais peuvent être regroupés en 57 fonctions (certains gènes sont complémentaires) elles mêmes réparties entre les systèmes de sécrétion, les adhésines, les invasines, les sidérophores* et les toxines (cf. Tableau V).

Le typage MLST a permis de déterminer les ST associés aux combinaisons des allèles des gènes de ménage. Cependant toutes les souches ne se sont pas vu attribuer un ST car certains gènes, trop différents ou pas suffisamment couverts par le séquençage, n'ont pas pas put être détectés (cf. Tableau VI).

L'analyse phylogénique permet la construction d'un arbre phylogénétique représentatif des différents phylogroupes* et complexes clonaux (cf. Figure 7). L'arbre phylogénétique obtenu permet de dire qu'en général les phylogroupes* A et D possèdent des BLSE de type CTX-M-1 alors que les phylogroupes* B1 et B2 possèdent des BLSE de type CTX-M-9. Pour chaque souche, on peut lire le numéro identifiant la souche, le ST déterminé avec le schéma de l'université Warwick, le complexe clonal, un deuxième ST déterminé avec le schéma de l'institut Pasteur et enfin le phylogroupe* associé.

Enfin des analyses statistiques sont effectuées par les chercheurs pour déterminer les caractères communs ainsi que les causes des différents génotypes de virulence, comme par exemple l'implication du lieu géographique, de la nature nosocomiale* ou non de l'infection, de l'âge des patients, etc. En effet une analyse en composante principale montre que les facteurs de virulence sont exprimés différentiellement selon les phylogroupes* déterminés (cf. Annexe 3).

IV.3. Discussion

Globalement, les résultats obtenus sont cohérents avec les études préalablement menées au moyen de méthodes expérimentales. Toutefois les analyses bioinformatiques ont leurs limites, si une portion du génome est mal séquencée (toutes les régions génome n'étant pas aisément séquençables), aucun gène ne pourra être détecté quelque soit les paramètres. De plus le traitement automatique des analyses peut introduire des biais non intentionnels car le volume de données à traiter est bien trop élevé. Les

Souche	Système De sécrétion	Adhésines	Invasines	Sidérophores	Toxines	Total
N°1	16	3	2	5	1	27
N°6	11	5	0	2	1	19
N°7	12	3	0	2	1	18
N°21	14	4	1	7	1	27
N°22	10	4	1	7	2	24
N°27	19	4	2	6	2	33
N°34	16	3	1	4	1	25
N°45	9	4	1	7	2	23
N°49	9	4	1	7	2	23
N°61	15	5	1	5	1	27
N°67	10	5	1	6	5	27
N°68	13	6	1	7	1	28
N°70	9	3	0	8	2	22
N°80	14	6	1	7	1	29
N°81	9	5	1	7	3	25
N°83	13	6	1	3	1	24
N°86	12	5	1	6	5	29
N°95	13	5	1	6	1	26
N°98	10	5	1	7	5	28
N°107	13	6	3	6	6	34
N°110	18	3	2	5	1	29
N°122	19	3	1	7	1	31
N°123	19	4	2	6	2	33
N°124	12	5	1	4	1	23
N°125	19	3	2	5	1	30
N°126	11	6	1	7	1	26
N°128	11	2	0	2	1	16
N°134	14	4	1	7	1	27
N°144	20	4	2	6	2	34
N°146	13	3	0	2	1	19
N°147	13	6	2	7	1	29
N°150	13	3	0	5	1	22
N°155	16	3	1	8	1	29
N°164	19	4	2	6	2	33
N°172	9	2	0	2	1	14
N°200	10	5	1	6	5	27

Tableau V – Gènes de virulence détectés par fonctions

Source personnelle

Sample	ST	dinB	icdA	pabB	polB	putP	trpA	trpB	uidA	Mismatches	Incertitude	Profondeur	maxMAF
107coli	26	4	15	10	5	2	4	1	6	0	-	73.2525	0.1515151515
110coli	8	23	9	8	12	9	11	7	13	0	-	48.402875	0.2352941176
122coli	3	3	8	5	11	8	3	5	3	0	-	61.057125	0.2
123coli	8	23	9	8	12	9	11	7	13	0	-	80.606375	0.1875
124coli	66	6	5	3	2	6	7	2	4	0	-	77.349125	0.1940298507
125coli	8	23	9	8	12	9	11	7	13	0	-	61.315	0.2
126coli	74	6	5	4	2	6	7	2	4	0	-	82.82275	0.2142857143
128coli	NF?	11	2	3	3	7	1	4	2	0	putP_7/edge4.0	31.48975	0.375
134coli	17	11	3	20	3	15	1	4	16	0	-	78.49325	0.2
144coli	8	23	9	8	12	9	11	7	13	0	-	71.71025	0.2307692308
146coli	2	8	2	7	3	7	1	4	2	0	putP_7/edge5.0	43.071625	0.4
147coli	74	6	5	4	2	6	7	2	4	0	-	65.548875	0.1818181818
150coli	NF	30	45	33	37	27	34	24	-	0	-	98.3614285714	0.1343283582
155coli	3	3	8	5	11	8	3	5	3	0	-	31.134	0.25
164coli	8	23	9	8	12	9	11	7	13	0	-	64.919125	0.1578947368
172coli	2	8	2	7	3	7	1	4	2	0	-	73.584375	0.1746031746
1ecoli	477	17	9	28	12	9	135	9	11	0	-	234.075375	0.1857142857
200coli	36	4	19	1	6	2	2	1	1	0	-	100.77825	0.1551724138
21coli	NF	30	45	33	37	27	34	24	-	0	-	63.5682857143	0.1621621622
22coli	43	9	1	15	7	4	9	6	9	0	-	78.77425	0.15
27coli	8	23	9	8	12	9	11	7	13	0	-	69.155875	0.2432432432
34coli	8	23	9	8	12	9	11	7	13	0	-	42.7045	0.2307692308
45coli	43	9	1	15	7	4	9	6	9	0	-	51.940125	0.2105263158
49coli	43	9	1	15	7	4	9	6	9	0	-	210.951625	0.1627906977
61coli	3	3	8	5	11	8	3	5	3	0	-	31.676125	0.25
67coli	9	9	20	15	7	4	9	6	9	0	-	67.10925	0.1428571429
68coli	74	6	5	4	2	6	7	2	4	0	-	53.283375	0.1944444444
6coli	471	6	6	4	2	154	7	2	4	0	-	239.201625	0.2
70coli	NF*	1	23*	2	1	1	2	3	1	icdA_23/1snp	-	87.586875	0.1739130435
7coli	2	8	2	7	3	7	1	4	2	0	-	93.84175	0.2142857143
80coli	NF*	31	46*	24	36	12	67	26	10	icdA_46/1snp	-	100.69825	0.24
81coli	43	9	1	15	7	4	9	6	9	0	-	85.176125	0.1914893617
83coli	74	6	5	4	2	6	7	2	4	0	dinB_6/edge5.0	35.117625	0.25
86coli	10	4	18	10	5	2	4	1	6	0	-	88.090125	0.1666666667
95coli	2	8	2	7	3	7	1	4	2	0	-	226.085625	0.2028985507
98coli	621	9	234	15	7	4	9	6	9	0	-	98.4255	0.1785714286

Tableau VI – ST en fonction de la combinaison des gènes de ménage détectés

Source Personnelle

premiers résultats des analyses bioinformatiques ont donc été confirmés par méthodes expérimentales. De plus les résultats incohérents ou l'absence de résultat sont systématiquement complétés par des analyses en laboratoire.

Les difficultés rencontrées étaient de natures différentes. Premièrement c'est l'assemblage qui a posé le plus de problèmes, en tant que traitement intermédiaire il était nécessaire pour réaliser la phylogénie mais les génomes assemblés n'était pas satisfaisants. Idéalement les génomes attendus auraient dû comporter quelques centaines de contigs* d'une longueur de plusieurs dizaines de milliers de paires de bases mais les résultats montraient davantage plusieurs milliers de contigs* relativement très courts avec une taille pas bien supérieure aux reads* initiaux. Seul Spades permettait un assemblage satisfaisant cependant le temps d'exécution qu'il nécessitait (plus d'une dizaine d'heures) n'a pas permis d'assembler un seul génome, notamment à cause de coupures de courant quotidiennes à l'échelle du centre hospitalier. Il a été décidé de mettre en œuvre une stratégie différente de « génomes virtuels ». Ainsi grâce au site Microscope du Génoscope, il a été possible de récupérer les gènes appartenant au core genome* de chaque espèce afin construire une BDD pour détecter tous les gènes communs de chaque espèce. Les allèles de ces gènes récupérés et concaténés, des « génomes virtuels »* ne contenant que les ORFs* du core genome* étaient à disposition. Les outils permettant l'analyse phylogénique n'en demandaient pas davantage. Lorsque s'est présenté l'opportunité de la grille de calcul, Spades est redevenu une option envisageable, raison pour laquelle il a en définitive été intégré au pipeline. Entre temps, l'assemblage avec CLC Genomics Workbench a permis de réaliser une analyse phylogénique sans biais volontaire dans la mesure où tout le génome était pris en compte, bien que les résultats étaient particulièrement similaires.

La deuxième grande difficulté a été la construction des bases de données afin de garantir leur compatibilité avec SRST2 notamment au niveau du format des en-têtes Fasta et des noms de séquences. En effet, l'en tête devait contenir 4 variables séparées par des doubles underscores « _ », dès qu'une variables était manquante ou en trop, Bowtie2 auquel SRST2 fait appel mettait un terme à son exécution en pleine analyse. De plus dès lors que les caractères « . », « / », « | », « ; » ou autres caractères interprétables dans le terminal (en ligne de commande) étaient présents dans l'en-tête,

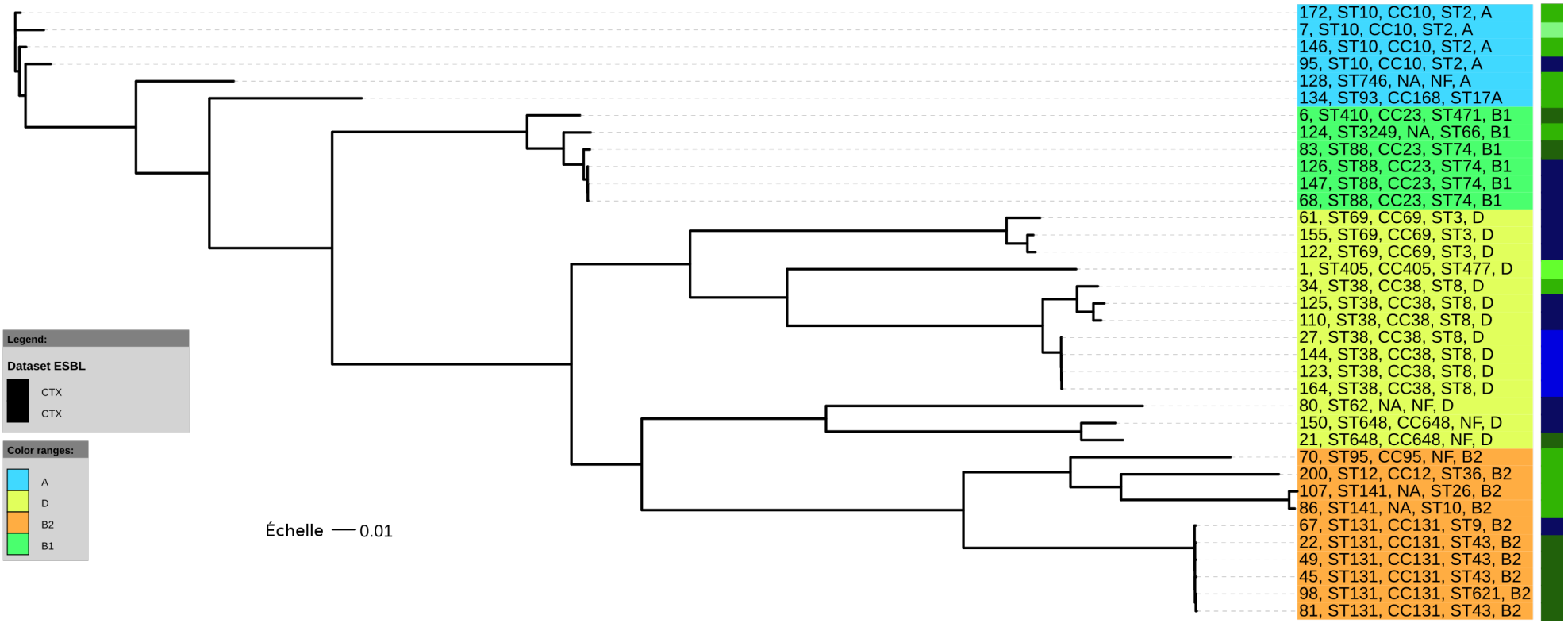


Figure 7 – Arbre phylogénétique des souches d'*Escherichia Coli* analysées

Source interne

c'est au niveau de samtools que l'exécution faisait défaut, les noms étant interprétés alors comme des commandes ou chemins de fichiers inexistants.

Enfin le dernier obstacle a été la mauvaise identification de certaines souches pour lesquelles les résultats n'étaient pas cohérents, certains gènes de ménages n'étaient pas détectés, empêchant dès lors le typage et plusieurs gènes détectés montraient une différence particulièrement élevée avec les gènes de référence des BDD avec plusieurs dizaines de SNP et de gaps (alignements de nombreux petits fragments et non de la séquence en son intégralité). Une fois l'assemblage possible et réalisé, un alignement avec l'outil BLAST a permis de se rendre compte que les souches avaient été mal identifiées en amont des analyses. Ainsi les souches 178 et K32 de *Klebsiella Pneumoniae* se sont révélées être des *Enterobacter*, la souche 92 une *Escherichia Coli* et enfin la souche E31 d'*Enterobacter Cloacae* s'est révélée être une *K. Pneumoniae*. Dans le lot des *Enterobacter* figurait cependant une *Enterobacter Aerogenes* écrasant alors l'échelle au niveau de l'arbre phylogénique.

Enfin travailler avec les bruits des travaux d'extension du bâtiment était fastidieux les premières semaines mais la gêne occasionnée était négligeable une fois l'habitude prise et des nuisances sonores au niveau du logement ont considérablement affecté ma ponctualité bien que le volume horaire a été respecté.

Conclusion

Suite à l'avènement des technologies de séquençage à haut débit et dans le cadre des études menées au CNR, le développement du pipeline d'analyses bioinformatiques doit permettre la caractérisation des souches bactériennes pathogènes.

Les analyses permettant cette caractérisation sont diverses. La détection des gènes de résistance permet une meilleure compréhension des mécanismes des enzymes intervenant sur la structure même des antibiotiques. La détection des gènes de virulence permet quant à elle de déterminer la pathogénicité et le comportement des certains phylogroupes*. La détection de plasmides permet de compléter ces analyses sans sans avoir recours à une étude complémentaire portant exclusivement sur les plasmides qui nécessiterait alors des extractions, séquençages et analyses supplémentaires. Le typage de séquence permet de discriminer les souches en fonction de ces caractères. Enfin la phylogénie permet de déterminer et quantifier les analogies et différences entre les différents phylogroupes* et complexes clonaux.

Les perspectives du développement du pipeline se sont succédé tout au long du projet. Il était en premier lieu question de développer un outil accessible au moyen d'une interface web pour coordonner les recherches au niveau d'un laboratoire mais la mise en place d'un serveur et son paramétrage n'était pas compatible avec le délai imparti, sans évoquer les démarches administratives. Il a alors été question de réaliser une interface graphique, permettant un suivi des souches tout en effectuant les analyses et en automatisant la partie technique. L'opportunité d'une grille de calcul a alors nécessité de revoir le développement pour intégrer le pipeline utilisable ligne de commande au sein de cette grille de calcul.

Au final, le pipeline a permis la caractérisation automatique et standardisée de 95 souches en quelques semaines. Il devrait être incorporer à la grille de calcul dans l'année permettant alors une étude à grande échelle des centaines de souches prévues.

Cependant, si le projet s'était étalé sur quelques semaines supplémentaires, le pipeline aurait pu être plus robuste, intégrer plus de paramètres et de modularité et le formatage des résultats aurait pu être amélioré en vue de faciliter le traitement et l'interprétation des résultats obtenus.

Bibliographie

BRISSE S., PASSET V., HAUGAARD A. B., BABOSAN A., KASSIS-CHIKHANI N., STRUVE C., DECRE D. « wzi Gene Sequencing, a Rapid Method for Determination of Capsular Type for Klebsiella Strains ». *Journal of Clinical Microbiology* [En ligne]. 1 décembre 2013. Vol. 51, n°12, p. 4073-4078. Disponible sur : < http://dx.doi.org/10.1128/JCM.01924-13 > (consulté le 2 juin 2015)

CARATTOLI A. « Plasmids and the spread of resistance ». *International Journal of Medical Microbiology* [En ligne]. août 2013. Vol. 303, n°6-7, p. 298-304. Disponible sur : < http://dx.doi.org/10.1016/j.ijmm.2013.02.001 > (consulté le 2 juin 2015)

CARATTOLI A. « Resistance Plasmid Families in Enterobacteriaceae ». *Antimicrobial Agents and Chemotherapy* [En ligne]. 1 juin 2009. Vol. 53, n°6, p. 2227-2238. Disponible sur : < http://dx.doi.org/10.1128/AAC.01707-08 > (consulté le 3 juin 2015)

CAVALLO J.-D., FABRE R., JEHL F., RAPP C., GARRABÉ E. « Bêtalactamines ». *Maladies Infectieuses*. août 2004. Vol. 1, n°3, p. 73.

CROXEN M. A., FINLAY B. B. « Molecular mechanisms of Escherichia coli pathogenicity ». *Nature Reviews Microbiology* [En ligne]. 25 décembre 2009. Disponible sur : < http://dx.doi.org/10.1038/nrmicro2265 > (consulté le 2 juin 2015)

CUEVAS RAMOS G. *Effets génotoxiques des souches de Escherichia coli produisant la Colibactine* [En ligne]. Toulouse : Université Toulouse III - Paul Sabatier, 2010. Disponible sur : < http://thesesups.ups-tlse.fr/940/2/Cuevas-Ramos_Gabriel.pdf > (consulté le 2 juin 2015)

INOUYE M., DASHNOW H., RAVEN L.-A., SCHULTZ M. B., POPE B. J., TOMITA T., ZOBEL J., HOLT K. E. « SRST2: Rapid genomic surveillance for public health and hospital microbiology labs ». *bioRxiv*. 2014. p. 006627.

JAUREGUY F., LANDRAUD L., PASSET V., DIANCOURT L., FRAPY E., GUIGON G., CARBONNELLE E., LORTHOLARY O., CLERMONT O., DENAMUR E., PICARD B., NASSIF X., BRISSE S. « Phylogenetic and genomic diversity of human bacteremic Escherichia coli strains ». *BMC Genomics* [En ligne]. 2008. Vol. 9, n°1, p. 560. Disponible sur : < http://dx.doi.org/10.1186/1471-2164-9-560 > (consulté le 2 juin 2015)

KAPER J. B., NATARO J. P., MOBLEY H. L. T. « Pathogenic Escherichia coli ». *Nature Reviews Microbiology* [En ligne]. février 2004. Vol. 2, n°2, p. 123-140. Disponible sur : < http://dx.doi.org/10.1038/nrmicro818 > (consulté le 2 juin 2015)

Sitographie

« Velvet: a sequence assembler for very short reads ». Disponible sur : < https://www.ebi.ac.uk/~zerbino/velvet/ > (consulté en avril 2015)

« Trimmomatic: A flexible read trimming tool for Illumina NGS data ». Disponible sur : < http://www.usadellab.org/cms/?page=trimmomatic > (consulté le 4 juin 2015c)

« UMD Genome group – Masurca Assembler ». Disponible sur : < http://www.genome.umd.edu/masurca.html > (consulté en avril 2015)

« SPAdes Genome Assembler ». Disponible sur : < http://bioinf.spbau.ru/spades > (consulté en mai 2015)

« Rapport CNR 2014 » [En ligne]. Disponible sur : < http://www.cnr-resistance-antibiotiques.fr/ressources/pages/Rapport_CNR_2014.pdf > (consulté le 7 mai 2015)

« Harvest 1.0 documentation – Parsnp quickstart ». Disponible sur : < http://harvest.readthedocs.org/en/latest/content/parsnp/quickstart.html > (consulté en mai 2015)

« Python IDE & Django IDE for Web developers : JetBrains PyCharm ». Disponible sur : < https://www.jetbrains.com/pycharm/ > (consulté en avril 2015)

« Python 2.7.10 documentation ». Disponible sur : < https://docs.python.org/2/ > (consulté en juin 2015)

« MLST Databases at University of Warwick ». Disponible sur : < http://mlst.warwick.ac.uk/mlst/dbs/Ecoli/documents/primersColi_html > (consulté le 18 mai 2015)

« Centres nationaux de référence ». Disponible sur : < http://www.invs.sante.fr/Espace-professionnels/Centres-nationaux-de-reference/Missions > (consulté le 7 mai 2015)

« Maximum Common Genome Alignment (MCGA) ». Disponible sur : < http://sourceforge.net/projects/mcga/ > (consulté en juin 2015)

« Liste et coordonnées des CNR ». Disponible sur : < http://www.invs.sante.fr/Espace-professionnels/Centres-nationaux-de-reference/Liste-et-coordonnees-des-CNR > (consulté le 7 mai 2015)

« Klebsiella Sequence Typing Home Page ». Disponible sur : < http://bigsdb.web.pasteur.fr/klebsiella/klebsiella.html > (consulté en avril 2015)

« katholt/srst2 · GitHub ». Disponible sur : < https://github.com/katholt/srst2 > (consulté

en avril 2015)
« Extraction et purification de l'ADN ». Disponible sur : < http://www.labome.fr/method/DNA-Extraction-and-Purification.html > (consulté le 18 mai 2015)
« Code de la santé publique - Article L1413-4 | Legifrance ». Disponible sur : < http://www.legifrance.gouv.fr/affichCodeArticle.do?cidTexte=LEGITEXT000006072665&idArticle=LEGIARTI000006686959 > (consulté le 7 mai 2015)
« CNR résistance aux antibiotiques - Missions ». Disponible sur : < http://www.cnr-resistance-antibiotiques.fr/missions.html > (consulté le 7 mai 2015)
« Biopython ». Disponible sur : < http://biopython.org/wiki/Main_Page > (consulté en mai 2015)
« Babraham Bioinformatics - Trim Galore! ». Disponible sur : < http://www.bioinformatics.babraham.ac.uk/projects/trim_galore/ > (consulté en avril 2015)
« Accueil CHU Clermont ». Disponible sur : < http://www.chu-clermontferrand.fr/Internet/Default.aspx > (consulté le 6 mai 2015)
« Wikipédia, l'encyclopédie libre ».Disponible sur : < https://fr.wikipedia.org/ > (consulté en juin 2015)

Table des acronymes

Acronyme	Signification
CHU	Centre Hospitalier Universitaire
CNR	Centre National de Référence
CIDAM	Conception Ingénierie et Développement de l'Aliment et du Médicament
InVS	Institut de Veille Sanitaire
BLSE	β-Lactamase à Spectre Étendu
SHV	SulfHydril Variable
CTX-M	CefoTaXimase-Munich
ARNr	Acide RiboNucléique Ribosomal
ADN	Acide DésoxyriboNucléique
HGT	Horizontal Gene Transfer
PAIs	PAthogenicity Islands
MLST	Multi-Locus Sequence Typing
ST	Sequence Type
ECPI	*E. coli* pathogènes intestinales
ECPEx	E. coli pathogènes extra-intestinales
MLEE	Multi Locus Enzyme Electrophoresis
PCR	Polymerase Chain Reaction
NGS	New Generation Sequencing
WGS	Whole Genome Sequencing
SRST2	Short Read Sequence Typing 2
BDD	Base De Données
VFDB	Virulence Factor DataBase
MCGP	Maximum Common Genome Phylogeny
ORF	Open Reading Frame
SNP	Single Nucleotide Polymophism

Glossaire

Autotransporteur : Protéines sécrétées par les bactéries qui assurent le transport de ses propres toxines à travers la membrane de la cellule de l'hôte.

Cadres de lecture ouvert : Région d'une séquence située entre un codon initiateur de la traduction et un codon stop. C'est la région codante d'un gène (en excluant les introns chez les eucaryotes).

Effecteur : Substance activant ou inhibant l'activité, c'est un inhibiteur ou un activateur.

Fimbriae : Excroissance fibreuse de petite taille au niveau de la membrane externe de certaines espèces de bactéries qui permettent d'adhérer à un substrat.

Gènes de ménage : Gène qui s'exprime dans tous les types cellulaires et dont les produits assurent les fonctions indispensables à la survie des cellules. Ils ne subissent donc pas de régulation.

Génome conservé : Régions du génome contenant les gènes conservés chez toutes les bactéries d'une même espèce.

Hémolysine : Substance (le plus souvent une protéine) susceptible de causer une destruction des globules rouges.

Sidérophore : Protéines chélatrices de fer synthétisés et sécrétés par les micro-organismes pour leur permettre de puiser le fer essentiel à leur développement.

Polysaccharide : Polymères constitués de plusieurs oses (sucres) liés entre eux par des liaisons O-osidiques.

Porine : Protéine membranaire formant des canaux permettant la diffusion de petites molécules hydrophiles à travers la membrane des cellules.

Phylogroupe : Groupe phylogénétique d'une même espèce mais suffisamment différent pour exprimer un génotype alternatif.

Nosocomial (infection) : Infection de nature hospitalière, s'oppose aux infections communautaires d nature environnementale.

Lexique

Clustering : Regroupement de différents éléments par analogie afin de constituer des clusters (groupes).

Contigs : Séquence d'acide nucléique relativement longue résultant de l'alignement de reads*.

Core genome : Génome conservé.

Read : Courte séquence d'acide nucléique issue du séquençage à partir de petits fragments d'une plus grande séquence (génome, gène, etc.).

Table des figures

Numéro	En vis-à-vis de la Page	Titre	Source
1	3	Mécanismes d'action des antibiotiques	Molina, Jean-Michel ; L'essentiel en pratique sur les β-lactamines
2	3	Nomenclature des β-lactamines en fonction de la structure de leur noyau	http://www.microcsb.net/IMG/pdf/doc-49.pdf
3	5	Étapes de liaison des EPEC au niveau des microvillosités intestinales de l'hôte	Scheftel, J-M ; Mémoire : Les facteurs de virulence des bactéries à tropisme intestinal
4	6	Hiérarchie des différentes méthodes de typage en fonction du pouvoir discriminant	Source : http://www.ridom.de/seqsphere/mlstplus
5	9	Schéma définitif du pipeline	Source personnelle
6	9	Procédure des analyses des souches réceptionnées	Rapport d'activité 2014 des CNR
7	10	Principe du séquençage paired-end	http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html
8	17	Arbre phylogénétique des souches d'*Escherichia Coli* analysées	Source interne

Table des tableaux

Numéro	En vis-à-vis de la Page	Titre	Source
I	2	Liste des 47 CNR	Personnelle d'après l'InVS
II	4	Classification des sous familles de β-Lactamases	Revue Antibiotique Vol 12 issue 01
III	6	Extrait d'un tableau de correspondance des allèles du gène Wzi, types K et ST	wzi Gene Sequencing, a Rapid Method for Determination of Capsular Type for Klebsiella Strains ; Brisse, et al. 2013
IV	15	Statistiques d'exécution du pipeline	Source personnelle
V	16	Gènes de virulence détectés par fonctions	Source Personnelle
VI	16	ST en fonction de la combinaison des gènes de ménage détectés	Source Personnelle

Table des Annexes

Annexe	Titre
1	Code du pipeline
2	Gènes de résistance détectés et classification
3	Exemple d'analyses statistiques complémentaires

Annexe 1 – Code du pipeline

Script 1 Trimming, Détection, Assemblage

```
#!/usr/bin/env python
#  coding=utf-8

"""
TTRaVDAP (Trimming, Typing, Resistance and Virulence genes and plasmids Detection, Assembly, Phylogeny)
Pipeline de Trimming, Typage MLST, Détection de gènes de Résistance et Virulence, de Plasmides et Assemblage
Ecrit par Pierre CHOLET sous la supervision de Richard BONNET, CHU Gabriel Montpied, Laboratoire de Microbiologie,
2015

Logiciels utilisés:
- Trim Galore:  F. Krueger - Babraham Institute Bioinformatics group
- CutAdapt:     M. Martin - Department of Computer Science, TU Dortmund, Germany (DOI:
http://dx.doi.org/10.14806/ej.17.1.200)
- Trimmomatic:  A. M. Bolger; M. Lohse; B. Usadel - Trimmomatic: A flexible trimmer for Illumina Sequence Data.
Bioinformatics, btu170. (2014)
- SRST2:        M. Inouye; K. Holt et al. - SRST2: Rapid genomic surveillance for public health and hospital
microbiology labs (http://genomemedicine.com/content/6/11/90)
- Spades:       S. Nurk, A. Bankevich et al. - Assembling Genomes and Mini-metagenomes from Highly Chimeric Reads
(2013) (http://link.springer.com/chapter/10.1007%2F978-3-642-37195-0_13)
"""

# Imports

import argparse
import glob
import os
import re
import sys
import time

##############
# Attributs #
##############

# Chemins (à modifier pour la grille de calcul ?)
trim_galore_loc = '/usr/local/trim_galore_0.3.7/trim_galore'                                    # Chemin vers trim_galore
cutadapt_loc = '/usr/local/cutadapt-1.7.1/bin'                                                  # Chemin vers cutadapt
trimmomatic_loc = '/usr/local/Trimmomatic-0.32/trimmomatic'                                     # Chemin vers trimmomatic
trimmomatic_adapters_dir = '/usr/local/Trimmomatic-0.32/adapters'                               # Chemin vers le dossier des
adaptateurs (séquence qui permettent de démarrer le séquençage) pour trimmomatic
srst2_loc = '/usr/local/bin/srst2'                                                              # Chemin vers srst2
spades_loc = '/usr/local/spades-3.5.0/bin/spades.py'                                            # Chemin vers Spades

# Trimming
trim_dir = 'Trimmed'                                                                            # Dossier des séquences
trimmées en sortie
trimg_fastqc_dir = 'Fastqc'                                                                     # Dossier des analyses de
qualité de FastQC

trim_qualite_score_defaut = [28]  # 0.15% d'erreurs au niveau de la qualité                    # Score de qualité minimum par
défaut pour le trimming
trim_trimmer_defaut = ['trimmomatic']                                                           # Outil de trimming par défaut
trim_phred_defaut = [33]                                                                        # Encodage des caractères de
qualité par défaut (phred33 = Illumina 1.9+; phred64 = Illumina 1.5)
trim_min_len_defaut = [100]                                                                     # Longueur minimale des reads
après trimming par défaut
trim_bypass_defaut = ''                                                                         # Bypass de la commande (si
l'utilisateur souhaite utiliser sa propre commande pour le trimming)

# Trimming: trim_galore
trimg_stringence_defaut = [5]                                                                   # Stringence par défaut pour
trim_galore (taux de recouvrement de l'adapteur, 1 -> même si une seule base de l'adapter est détectée, elle est
trimmée)
trimg_5prime_clip = 20                                                                          # Nombre de bases par défaut à
trimmer à en 5' si le flag --trm_clip est présent
trimg_3prime_clip = 10                                                                          # Nombre de bases par défaut à
trimmer à en 3' si le flag --trm_clip est présent

# SRST2
mlst_dir = 'MLST'                                                                               # Dossier des résultats de
MLST (arborescence: MLST/[SOUCHE]/[fichiers de résultats])
resist_dir = 'Resistance'                                                                       # Dossier des résultats de
détection de gènes de résistance (même arborescence)
virulence_dir = 'Virulence'                                                                     # Dossier des résultats de
détection de gènes de virulence (idem)
plasmides_dir = 'Plasmides'                                                                     # Dossier des résultats de
détection de gènes de plasmides (arborescence: Plasmides/[BDD]/[Souche]/[fichiers])

suffixe_forward = "_forward_paired"                                                             # Suffixe des fichiers
"forward" trimmés (|------->   |)
suffixe_reverse = "_reverse_paired"                                                             # Suffixe des fichiers
"reverse" trimmés (|   <-------|)

srst_profondeur_defaut = [10]                                                                   # Pronfondeur minimale par
défaut pour srst2 au delà de laquelle un allèle est détecté de manière "sûre" (incertaine sinon)
srst_profondeur_ext_defaut = [5]                                                                # Pronfondeur minimale au
```

```
niveau des extrémités par défaut pour srst2 au delà de laquelle un allèle est détecté de manière "sûre" (incertaine
sinon)
srst_couverture_defaut = [90]                                                    # Taux de couverture minimale
pour détecter un allèle (90% d'analogie min par défaut, sinon allèle rejeté)
srst_divergence_defaut = [10]                                                    # Taux de divergence max au
delà duquel un allèle est trop différent et est donc rejeté

# SPAdes
assemblage_dir = 'Assembly'                                                      # Dossier de sortie des
génomes assemblés par Spades
# /!\ Valeur de la mémoire à modifier pour aller avec la grille de calcul
spades_memory_defaut = 250                                                       # Mémoire par défaut pour
laquelle SPAdes se termine si elle est dépassée

##############
# Méthodes  #
##############


def parse_arguments():
    """Méthode qui définit les paramètres du pipeline"""
    # Paramètres
    parametres = argparse.ArgumentParser(sys.argv[0],
                                         description="""Pipeline d'analyse de génomes bactériens, trimming, typage
MLST, détection de gènes de résistance, virulence, assemblage.""",
                                         prefix_chars='-',
                                         add_help=True,
                                         epilog='Écrit par BONNET R., CHOLET P. Laboratoire de Microbiologie, CHU G.
Montpied Clermont-Ferrand, 2015')
    # Trimming
    parametres.add_argument('--inf',            action='store', nargs=1,    type=str,   required=True,
dest='input_forward',
                            help='Fichier "forward" du séquençage Paired-End')

    parametres.add_argument('--inr',            action='store', nargs=1,    type=str,   required=True,
dest='input_reverse',
                            help='Fichier "reverse" du séquençage Paired-End')

    parametres.add_argument('-s', '--trm_score', action='store', nargs=1,   type=int,   required=False, dest='score',
                            help='Seuil de qualité minimale (défaut=' + str(trim_qualite_score_defaut) + ')',
                            default=trim_qualite_score_defaut)

    parametres.add_argument('-t', '--trimmer',  action='store', nargs=1,    type=str,   required=False,
dest='trimmer',
                            help='Outil de trimming (défaut=trimmomatic)',
                            default=trim_trimmer_defaut,
                            choices=['galore', 'trimmomatic', 'skip'])

    parametres.add_argument('-n', '--threads',  action='store', nargs=1,    type=int,   required=False,
dest='threads',
                            help='Nombre de threads (processeurs) (défaut=1)',
                            default=[1])

    parametres.add_argument('--trm_stringence', action='store', nargs=1,    type=int,   required=False,
dest='stringence',
                            help='Stringence pour trim_galore',
                            default=trimg_stringence_defaut)

    parametres.add_argument('--trm_clip',       action='store_true',    dest='clip',
                            help='Trimmer 20 pb en 5\' et 10 pb en 3\'')

    parametres.add_argument('-l', '--trm_min_len', action='store', nargs=1, type=int,   required=False,
dest='min_len',
                            help='Taille minimale pour retenir un read trimmé (défaut=' + str(trim_min_len_defaut) +
')',
                            default=trim_min_len_defaut)

    parametres.add_argument('--trm_bypass',     action='store', nargs='+',  type=str,   required=False,
dest='trm_bypass',
                            help='Commande à entrer pour le trimming, entre quotes (ex: "-q 28 --fastq") (Attention au
trimmer choisi)')

    parametres.add_argument('-p', '--phred',    action='store', nargs=1,    type=int,   required=False, dest='phred',
                            help='Encodage du score phred du fastq (défaut=33)',
                            default=['33'],
                            choices=[33, 64])

    parametres.add_argument('--trm_adapter',    action='store', nargs=1,    type=str,   required=False,
dest='adaptateur',
                            help='Adaptateur spécifique du séquençage pour trim_galore')

    # Détection/typage
    parametres.add_argument('--suffixe_forward', action='store', nargs=1,  type=str,   required=False,
dest='suffixe_f',
                            help='Suffixe du fichier "Forward" de reads déjà trimmé')

    parametres.add_argument('--suffixe_reverse', action='store', nargs=1,  type=str,   required=False,
dest='suffixe_r',
                            help='Suffixe du fichier "Reverse" de reads déjà trimmé')

    parametres.add_argument('--srst_depth',     action='store', nargs=1,    type=int,   required=False,
dest='srst_profondeur',
                            help='Profondeur min
 pour marquer l\'allèle comme ambigu',
                            default=srst_profondeur_defaut)

    parametres.add_argument('--srst_edge_depth', action='store', nargs=1,   type=int,   required=False,
```

```
dest='srst_profondeur_ext',
                            help='Profondeur min des extrémités pour marquer l\'allèle comme ambigu',
                            default=srst_profondeur_ext_defaut)

    parametres.add_argument('--srst_coverage',  action='store', nargs=1,    type=int,   required=False,
dest='srst_couverture',
                            help='Taux de couverture (en %%) minimum pour rapporter un allèle (défaut: 90)',
                            default=srst_couverture_defaut)

    parametres.add_argument('--srst_divergence', action='store', nargs=1,   type=int,   required=False,
dest='srst_divergence',
                            help='Taux de divergence (en %%) maximum pour exclure un allèle (défaut: 10)',
                            default=srst_divergence_defaut)

    parametres.add_argument('--mlst_db',        action='store', nargs=1,    type=str,   required=False,
dest='mlst_db',
                            help='Chemin vers la base de donnée MLST',
                            default=[])

    parametres.add_argument('--mlst_definition', action='store', nargs=1,   type=str,   required=False,
dest='mlst_definitions',
                            help='Chemin vers le fichier de définitions correspondant à la base de donnée MLST')

    parametres.add_argument('--mlst_delimiter', action='store', nargs=1,    type=str,   required=False,
dest='mlst_delimiteur',
                            help='Délimiteur gène/allèle du fichier de définitions')

    parametres.add_argument('--resist_db',      action='store', nargs='+',  type=str,   required=False,
dest='detection_resistance_db',
                            help='Chemin(s) vers la(les) base(s) de données de gènes de résistance (Fasta)')

    parametres.add_argument('--vir_db',         action='store', nargs='+',  type=str,   required=False,
dest='detection_virulence_db',
                            help='Chemin(s) vers la(les) base(s) de données de gènes de virulence (Fasta)')

    parametres.add_argument('--plsm_mlst',      action='store', nargs=1,    type=str,   required=False,
dest='plasmides_mlst',
                            help='Chemin(s) vers la(les) base(s) de données de MLST pour les plasmides (Fasta)')

    parametres.add_argument('--plsm_def',       action='store', nargs=1,    type=str,   required=False,
dest='plasmides_definitions',
                            help='Chemin(s) vers le(les) fichiers de profils MLST de plasmides (Fasta)')

    parametres.add_argument('--plsm_del',       action='store', nargs=1,    type=str,   required=False,
dest='plasmides_delimiteur',
                            help='Délimiteur gène/allèle des fichiers de défnitions de MLST des plasmides')

    parametres.add_argument('--plsm_db',        action='store', nargs='+',  type=str,   required=False,
dest='plasmides_db',
                            help='Chemin(s) vers la(les) base(s) de données de gènes de plasmides (Fasta)')

    # Assemblage
    parametres.add_argument('--spades_kmer',     action='store', nargs=1,   type=str,   required=False,
dest='spades_kmers',
                            help='Liste de kmers à utiliser séparés par une virgule et < 128 (ex: 21,53,77)')

    parametres.add_argument('--spades_cutoff',   action='store', nargs=1,   type=str,   required=False,
dest='spades_cutoff',
                            help='Cutoff seuil de couverture pour Spades (flottant ou "auto")')

    parametres.add_argument('--skip_assembly',   action='store_true',   dest='skip_assembly',
                            help='Désactive l\'assemblage avec SPAdes')

    # FLAGS
    parametres.add_argument('--log',            action='store_true',    dest='log',
                            help='Active les fichiers de logs des différents outils')

    parametres.add_argument('--no_verbose',  action='store_false',    dest='verbose',
                            help='Désactive le mode verbose (informations supplémentaires)')

    parametres.add_argument('--keep_files',     action='store_false',   dest='rm_files',
                            help='Empêche la suppression des fichiers annexes et rapports')

    return parametres.parse_args()


# Trimmomatic
def trmtc_trimming(reads_f, reads_r, name, loc, score, threads, phred, length, clip5p, clip3p, bypass, verbose):  #
trim_qualite_score
    """Méthode de construction de la commande à exécuter pour trimmer avec Trimmomatic"""
    trimming_command = 'java -jar ' + loc + \
                       ' PE -threads ' + str(threads) + \
                       ' -' + str(phred) + \
                       ' ' + reads_f + ' ' + reads_r + ' ' + \
                       name + suffixe_forward + '.fq.gz ' + name + '_forward_unpaired.fq.gz ' + \
                       name + suffixe_reverse + '.fq.gz ' + name + '_reverse_unpaired.fq.gz' + \
                       ' ILLUMINACLIP:'+trimmomatic_adapters_dir+'/TruSeq3-PE.fa:2:30:10:8:TRUE'

    if clip5p and clip3p:
        trimming_command += ' HEADCROP:' + str(clip5p) + \
                            ' CROP:' + str(clip3p)

    trimming_command += ' LEADING:' + str(score) + \
                        ' TRAILING:' + str(score) + \
                        ' MINLEN:' + str(length)
    if bypass:
        if verbose:
```

```
            print "/!\ Attention: Bypass de la commande de trimmomatic."
        trimming_command = bypass

    return trimming_command


# Trim_galore
def trmg_trimming(reads_f, reads_r, loc, score, fq_path, phred, stringency, clip5p, clip3p, length, adapter, bypass,
verbose):
    """Méthode de construction de la commande à exécuter pour trimmer avec Trim_galore"""
    trimming_command = loc + ' -q ' + str(score) + \
                             ' --' + phred + \
                             ' --paired ' \
                             ' --fastqc ' \
                             ' --fastqc_args "--outdir ' + fq_path + '"' \
                             ' --stringency ' + str(stringency)

    if clip5p and clip3p:
        trimming_command += ' --clip_R1 ' + str(clip5p) + \
                            ' --clip_R2 ' + str(clip5p) + \
                            ' --three_prime_clip_R1 ' + str(clip3p) + \
                            ' --three_prime_clip_R2 ' + str(clip3p)

    if adapter:
        trimming_command += ' --adapter ' + adapter

    trimming_command += ' --gzip' \
                        ' --length ' + str(length) + \
                        ' ' + reads_f + ' ' + reads_r

    if bypass:
        if verbose:
            print "/!\ Attention: Bypass de la commande de trim_galore."
        trimming_command = bypass

    return trimming_command


# Détection de gènes/typage avec Srst2
def srst2_mlst(name, seq_dir, loc, profondeur, profondeur_ext, couverture, divergence, db, definitions, delimiteur,
log):
    """Méthode de construction de la commande à exécuter pour la détection du profil MLST"""
    mlst_command = loc + " --input_pe " + seq_dir + "/" + name + suffixe_forward + ".fq.gz " + seq_dir + "/" + name +
suffixe_reverse + ".fq.gz" + \
                         " --min_depth " + str(profondeur) + \
                         " --min_edge_depth " + str(profondeur_ext) + \
                         " --min_coverage " + str(couverture)

    if divergence:
        mlst_command += " --max_divergence " + str(divergence)

    mlst_command += " --forward " + suffixe_forward + \
                    " --reverse " + suffixe_reverse + \
                    " --mlst_db " + db

    if definitions:
        mlst_command += " --mlst_definitions " + definitions
    if delimiteur:
        mlst_command += " --mlst_delimiter " + delimiteur

    mlst_command += " --output mlst" + \
                    " --report_all_consensus"

    if log:
        mlst_command += " --log"

    return mlst_command


def srst2_detection(name, seq_dir, loc, profondeur, profondeur_ext, couverture, divergence, db, cat, log):
    """Méthode de construction de la commande à exécuter pour la détection de gènes (résistance comme virulence géré
par l'argument 'cat')"""
    detect_command = loc + " --input_pe " + seq_dir + "/" + name + suffixe_forward + ".fq.gz " + seq_dir + "/" + name
+ suffixe_reverse + ".fq.gz" + \
                           " --min_depth " + str(profondeur) + \
                           " --min_edge_depth " + str(profondeur_ext) + \
                           " --min_coverage " + str(couverture)

    if divergence:
        detect_command += " --max_divergence " + str(divergence)

    detect_command += " --forward " + suffixe_forward + \
                      " --reverse " + suffixe_reverse + \
                      " --gene_db " + db

    detect_command += " --output " + cat + \
                      " --report_all_consensus"

    if log:
        detect_command += " --log"

    return detect_command


# Assemblage
def spades_assembly(loc, name, seq_dir, threads, kmers, cutoff, phred):
    """Méthode de construction de la commande à exécuter pour l'assemblage des génomes avec SPAdes"""
```

```
    spades_command = loc + " -1 " + seq_dir + "/" + name + suffixe_forward + ".fq.gz" + \
                           " -2 " + seq_dir + "/" + name + suffixe_reverse + ".fq.gz" + \
                           " --careful" + \
                           " -t " + str(threads) + \
                           " -m " + str(spades_memory_defaut) + \
                           " -k " + str(kmers)

    if cutoff:
        spades_command += " --cov-cutoff " + str(cutoff)

    if phred:             # Enlever le paramètre pour une détection automatique de l'encodage du score phred
(paramètres présent par soucis de cohérence)
        spades_command += " --phred-offset " + str(phred)

    spades_command += " -o ."

    return spades_command


# Utilitaires
def
 check_outdir(categorie, name):
    """Méthode qui permet de vérifier l'existence du dossier de résultat sous la forme [Trimmed|MLST|Genes|etc.]/[Nom
de la souche]."""
    if not os.path.exists(categorie):                                                    # Si le dossier de
catégorie (Trimming, MLST, Resist, Viru, etc.) n'existe pas, on le crée
        os.mkdir(categorie)
    compteur = 1
    if os.path.exists(categorie + "/" + name):                                           # Si un dossier portant
le nom de la souche existe déjà dans le dossier de catégorie, on modifie le nom puis on le crée
        output = name + "_" + str(compteur)
        while os.path.exists(categorie + "/" + output):
            output = name + "_" + str(compteur)
            compteur += 1
        os.mkdir(categorie + "/" + output)
    else:
        os.mkdir(categorie + "/" + name)
        output = file_name
    return categorie + "/" + output


def check_sam_mod():
    """Méthode appelée après srst2 qui recherche des fichiers sam ou sam.mod (présence non systématique et
comportement inconnu) à supprimer si le flag --keep_files n'est pas présent."""
    # Cette méthode un peu trop lourde pour ce qu'elle fait sert surtout à ne pas avoir d'erreur du genre "rm: cannot
remove ‘*.sam’: No such file or directory" et garder le pipeline propre
    files_in_folder = os.listdir(".")
    sam_is_there = False
    mod_is_there = False
    for files_in_folder_name in files_in_folder:
        if re.search("\.sam", files_in_folder_name):
            sam_is_there = True
        if re.search("\.mod", files_in_folder_name):
            mod_is_there = True
    if sam_is_there:
        if mod_is_there:
            return "rm *.sam; rm *.mod"
        else:
            return "rm *.sam"
    else:
        if mod_is_there:
            return "rm *.mod"
        else:
            return False

#############
# Main      #
#############

if __name__ == '__main__':

    heure_dep = time.time()                        # Permet de calculer les temp d'exécution

############################################################
#                    ARGUMENTS                             #
############################################################

    arguments = parse_arguments()                  # Récupération des arguments

    # Chemin des fichiers (vérifie que les chemins sont valides et détermine le chemin absolu si un chemin relatif a
été entré)
    forward_file = ''                              # Déclaration pour éviter des avertissements, les attributs sont
changés si les chemins sont valide, sinon sortie du pipeline
    reverse_file = ''
    if os.path.exists(arguments.input_forward[0]) and os.path.exists(arguments.input_reverse[0]):
        if re.match("/", arguments.input_forward[0]):
            forward_file = arguments.input_forward[0]
        else:
            forward_file = os.path.abspath(arguments.input_forward[0])
        if re.match("/", arguments.input_reverse[0]):
            reverse_file = arguments.input_reverse[0]
        else:
            reverse_file = os.path.abspath(arguments.input_reverse[0])
    else:
        exit("/!\ Chemin(s) vers les reads fastq invalide(s), sortie du pipeline.")

    # Calcul du nom de fichier
```

```
    reverse_file_name = os.path.split(reverse_file)[1]                          # path == /head/head2/head3/tail ->
récupère tail
    file_name = os.path.split(forward_file)[1]                                  # os.path.split()[0] == heads |
os.path.split() [1] == tail
    for decrementation in range(len(file_name), 0, -1):                         # Décrémentation de la longueur de
"file_name"
        if not reverse_file_name.startswith(file_name):                         # Compare les 2 noms de fichiers pour
trouver une similitude
            file_name = file_name[0:decrementation]                             # Réduit la chaîne de caractère de 1 à
chaque itération
    if len(file_name) < 2:                                                      # Si "file_name" est trop court, pas
de similitude
        file_name = "default_file_name"                                         # Donc "file_name" par défaut
    else:                                                                       # Sinon on nettoie un peu le nom
(souvent du type [SOUCHE]_R{1|2})
        file_name = file_name.rstrip("R")
        file_name = file_name.rstrip("_")

    # Nombre de threads
    n_threads = arguments.threads[0]

# Arguments du trimming
    skip_trimming = True if arguments.trimmer[0] == "skip" else False           # Opérateur ternaire, skip_trimming
VRAI si argument == "skip" sinon FAUX
    trimtc = True if arguments.trimmer[0] == "trimmomatic" else False           # 2ème Opérateur ternaire, trimtc VRAI
si argument == "trimmomatic" sinon FAUX

    trim_qualite_score = arguments.score[0]

    trim_phred = "phred" + str(arguments.phred[0])

    trimg_stringence = arguments.stringence[0]

    if arguments.clip:
        trim_clip5p = trimg_5prime_clip
        trim_clip3p = trimg_3prime_clip
    else:
        trim_clip5p = False
        trim_clip3p = False

    trim_min_len = arguments.min_len[0]

    if arguments.adaptateur:
        trim_adaptateur = arguments.adaptateur[0]
    else:
        trim_adaptateur = False

    # Bypass de la commande de trimming
    if arguments.trm_bypass:
        if skip_trimming:
            print "/!\ Attention, bypass de la commande de trimming inutile, trimming sauté '-t skip'"
        if trimtc:
            trim_bypass_command = "java -jar " + trimmomatic_loc + " " + arguments.trm_bypass[0]            # Chemin
vers trimmomatic + commande bypassée
        else:
            trim_bypass_command = trim_galore_loc + " " + arguments.trm_bypass[0]                           # Pareil
pour trim_galore
    else:
        trim_bypass_command = False

# Arguments srst2
    if arguments.suffixe_f:
        suffixe_forward = arguments.suffixe_f[0]

    if arguments.suffixe_r:
        suffixe_reverse = arguments.suffixe_r[0]

    profondeur_min = arguments.srst_profondeur[0]

    profondeur_ext_min = arguments.srst_profondeur_ext[0]

    taux_couverture = arguments.srst_couverture[0]

    taux_divergence = arguments.srst_divergence[0]

    # Mlst ('None' affecté aux paramètres qui sont modifiés si les arguments sont présents afin que paramètres soient
déclarés)
    mlst_database = None
    mlst_definitions = None
    mlst_delimiteur = None
    if arguments.mlst_db:
        if os.path.exists(arguments.mlst_db[0]):
            mlst_database = os.path.abspath(arguments.mlst_db[0])
        else:
            print "/!\ Attention, Bdd MLST '" + arguments.mlst_db[0] + "' introuvable, veuillez corriger le chemin."
        if os.path.exists(arguments.mlst_definitions[0]):
            mlst_definitions = os.path.abspath(arguments.mlst_definitions[0])
        else:
            print "/!\ Attention, définitions de la Bdd MLST '" + arguments.mlst_db[0] + "' introuvable."
        if arguments.mlst_delimiteur:
            mlst_delimiteur = arguments.mlst_delimiteur[0]
        else:
            print "/!\ Attention, délimiteur de la Bdd MLST non défini, délimiteur par défaut de srst2: '-'"

    # Gènes
        # Liste des bdd de résistance
    resistance_database = []                                            # Définition d'une liste vide de base de
```

```
données de résistance (évite qu'elle retourne un objet de type 'None' si non spécifié en argument)
    if arguments.detection_resistance_db:                               # Si des chemins sont passés en arguments
        for db_element in arguments.detection_resistance_db:            # On les parcourt un par un
            if re.search("\*", db_element):                             # On cherche la présence d'une wildcard ("*")
                for db_path in glob.glob(db_element):                   # On en déduit les chemins possibles et on
ajoute le chemin absolu (sinon le chemin spécifié avec une wildcard n'est pas reconnu)
                    resistance_database.append(os.path.abspath(db_path))
            else:                                                       # Sinon on ajoute juste le chemin absolu à la
liste
                if os.path.exists(db_element):
                    resistance_database.append(os.path.abspath(db_element))
                else:
                    print "/!\ Attention, Bdd de résistance '" + db_element + "' introuvable, veuillez corriger le
chemin."

        # Liste des bdd de virulence (pareil)
    virulence_database = []
    if arguments.detection_virulence_db:
        for db_element in arguments.detection_virulence_db:
            if re.search("\*", db_element):
                for db_path in glob.glob(db_element):
                    virulence_database.append(os.path.abspath(db_path))
            else:
                if os.path.exists(db_element):
                    virulence_database.append(os.path.abspath(db_element))
                else:
                    print "/!\ Attention, Bdd de résistance '" + db_element + "' introuvable, veuillez corriger le
chemin."
    # Plasmides
        # Plasmides_MLST
    plasmides_mlst = False
    plasmides_definitions = False
    plasmides_delimiteur = False
    if arguments.plasmides_mlst:
        if os.path.exists(arguments.plasmides_mlst[0]):
            plasmides_mlst = os.path.abspath(arguments.plasmides_mlst[0])
        else:
            print "/!\ Attention, Bdd MLST  de plasmides '" + arguments.plasmides_mlst[0] + "' introuvable, veuillez
corriger le chemin."
        if os.path.exists(arguments.plasmides_definitions[0]):
            plasmides_definitions = os.path.abspath(arguments.plasmides_definitions[0])
        else:
            print "/!\ Attention, définitions de la Bdd MLST de plasmides '" + arguments.plasmides_definitions[0] + "'
introuvable."
        if arguments.plasmides_delimiteur:
            plasmides_delimiteur
 = arguments.plasmides_delimiteur[0]
        else:
            print "/!\ Attention, délimiteur de la Bdd MLST de plasmides non défini, délimiteur par défaut de srst2:
'-'"

        # Plasmides Gènes
    plasmides_database = []
    if arguments.plasmides_db:
        for db_element in arguments.plasmides_db:
            if re.search("\*", db_element):
                for db_path in glob.glob(db_element):
                    plasmides_database.append(os.path.abspath(db_path))
            else:
                if os.path.exists(db_element):
                    plasmides_database.append(os.path.abspath(db_element))
                else:
                    print "/!\ Attention, bdd de plasmides '" + db_element + "' introuvable, veuillez corriger le
chemin."

# Arguments Spades
    if arguments.spades_kmers:
        spades_kmers = arguments.spades_kmers[0]
    else:
        spades_kmers = "21,33,55,77"

    if arguments.spades_cutoff:
        spades_cutoff = arguments.spades_cutoff[0]
    else:
        spades_cutoff = False

    skip_assembly = arguments.skip_assembly

# Flags
    log_flag = arguments.log
    if log_flag:
        if not os.path.exists('Log'):                                           # Vérifie si le dossier de Log est
présent et le crée sinon
            os.mkdir('Log')

    verbose_flag = arguments.verbose

    rm_files = arguments.rm_files

############################################################
#                       EXECUTION                          #
############################################################

# TRIMMING
    if not skip_trimming:
        # Détails sur le score seuil
        trim_qualite_score_to_pourcentage = (10**(-trim_qualite_score/10))*100      # Calcule le pourcentage d'erreur
```

```
à partir du score seuil

        if verbose_flag:
            print "\n-> Nom de la souche soumise:     " + file_name
            print "-> Score de qualité de trimming: " + str(trim_qualite_score) + " (soit un pourcentage d'erreur de "
+ str(trim_qualite_score_to_pourcentage) + "%)"

        # Construction de la commande
        if trimtc:
            trm_command = trmtc_trimming(forward_file, reverse_file, file_name, trimmomatic_loc, trim_qualite_score,
n_threads, trim_phred, trim_min_len, trim_clip5p, trim_clip3p, trim_bypass_command, verbose_flag)
        else:
            trm_command = trmg_trimming(forward_file, reverse_file, trim_galore_loc, trim_qualite_score,
trimg_fastqc_dir, trim_phred, trimg_stringence, trim_clip5p, trim_clip3p, trim_min_len, trim_adaptateur,
trim_bypass_command, verbose_flag)

        if verbose_flag:
            print "-> Aperçu de la commande de trimming:\n\n" + trm_command + "\n"

        # Dossier de sortie
        trimming_output_dir = os.path.abspath(check_outdir(trim_dir, file_name))
        os.chdir(trimming_output_dir)

        # Exécution de la commande dans le dossier de sortie
        trim_has_failed = os.system(trm_command)

        # Nettoyage des fichiers
        if trimtc:
            if rm_files:
                os.system("rm *unpaired*")
        else:
            os.rename(file_name+'_R1_val_1.fq.gz', file_name + suffixe_forward + '.fq.gz')
            os.rename(file_name+'_R2_val_2.fq.gz', file_name + suffixe_reverse + '.fq.gz')
            if rm_files:
                os.system('rm *.txt')

        os.chdir("../..")                                       # On sort du fichiers de résultats en remontant
l'arborescence (../<-[Trimmed]/<-[Souche]/Fichiers Trimmés)

        # Rapport du trimming
        if trim_has_failed:
            if verbose_flag:
                print "-> Le Trimming a échoué."
            os.system("rm " + trimming_output_dir + "/*")
            os.rmdir(trimming_output_dir)
            sys.exit("/!\ Trimming échoué, sortie du pipeline.")
        else:
            if verbose_flag:
                print "-> Trimming effectué avec succès. Dossier de sortie: \"" + trimming_output_dir + "\""

    else:
        if verbose_flag:
            print "\n-> Nom de la souche soumise: " + file_name
            print "-> Trimming annulé ('-t skip')"
        trimming_output_dir = forward_file.rsplit("/", 1)[0]    # On remplace le dossier de sortie de trimming par le
dossier qui contient les séquences déjà trimmées soumises en paramètres

# DETECTION
    # MLST
    if mlst_database:                                           # Si on spécifie une chemin vers une bdd de MLST, on
effectue la mlst, sinon pas besoin et on passe à la détection de gènes
        # Construction de la commande
        srst_command = srst2_mlst(file_name, trimming_output_dir, srst2_loc, profondeur_min, profondeur_ext_min,
taux_couverture, taux_divergence, mlst_database, mlst_definitions, mlst_delimiteur, log_flag)

        # Dossier de sortie (comme pour le trimming)
        mlst_output_dir = check_outdir(mlst_dir, file_name)
        os.chdir(mlst_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande MLST:\n\n" + srst_command + "\n"

        mlst_has_failed = os.system(srst_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename("mlst.log", "../../Log/" + file_name + "_mlst.log")
            if rm_files:
                os.system("rm *.pileup; rm *.bam")
                sam_mod_to_remove = check_sam_mod()
                if sam_mod_to_remove:
                    os.system(sam_mod_to_remove)
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers de MLST impossible, l'analyse ne s'est probablement pas déroulé
normalement."
        finally:
            os.chdir("../..")

        # Rapport de la MLST
        if mlst_has_failed:
            if verbose_flag:
                print "-> Le typage MLST a échoué."
            os.system("rm " + mlst_output_dir + "/*")
            os.rmdir(mlst_output_dir)
        else:
            if verbose_flag:
```

```
                print "-> Typage MLST effectué. Résultats: " + mlst_output_dir

    else:
        if verbose_flag:
            print "-> Pas de typage MLST effectué."

    # Résistance
    if len(resistance_database) > 0:                            # Si au moins une bdd en paramètre
        gene_cat = "resistance"                                 # Catégorie résistance
        # Construction de la commande
        srst_command = srst2_detection(file_name, trimming_output_dir, srst2_loc, profondeur_min, profondeur_ext_min,
taux_couverture, taux_divergence, " ".join(resistance_database), gene_cat, log_flag)

        # Dossier de sortie (comme pour le trimming)
        detect_output_dir = check_outdir(resist_dir, file_name)
        os.chdir(detect_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande de détection des gènes de résistance:\n\n" + srst_command + "\n"

        # Exécution
        detect_has_failed = os.system(srst_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename(gene_cat + ".log", "../../Log/" + file_name + "_" + gene_cat + ".log")
            if rm_files:
                os.system("rm *.pileup; rm *.bam")
                sam_mod_to_remove = check_sam_mod()
                if sam_mod_to_remove:
                    os.system(sam_mod_to_remove)
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers de détection de gènes de résistance impossible, l'analyse ne s'est
probablement pas déroulé normalement."
        finally:
            os.chdir("../..")

        # Rapport de la détection de gènes de résistance
        if detect_has_failed:
            if verbose_flag:
                print "-> La détection de gènes de résistance a échouée."
            os.system("rm " + detect_output_dir + "/*")
            os.rmdir(detect_output_dir)
        else:
            if verbose_flag:
                print "-> Détection de gènes de résistance effectuée. Résultats: " + detect_output_dir

    else:
        if verbose_flag:
            print "-> Pas de détection de gènes de résistance (pas de bdd)."

    # Virulence
    if len(virulence_database) > 0:
        gene_cat = "virulence"                                  # Catégorie virulence
        # Construction de la commande
        srst_command = srst2_detection(file_name, trimming_output_dir, srst2_loc, profondeur_min, profondeur_ext_min,
taux_couverture, taux_divergence, " ".join(virulence_database), gene_cat, log_flag)

        # Dossier de sortie (comme pour le trimming)
        detect_output_dir = check_outdir(virulence_dir, file_name)
        os.chdir(detect_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande de détection des gènes de virulence:\n\n" + srst_command + "\n"

        # Exécution
        detect_has_failed = os.system(srst_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename(gene_cat + ".log", "../../Log/" + file_name + "_" + gene_cat + ".log")
            if rm_files:
                os.system("rm *.pileup; rm *.bam")
                sam_mod_to_remove = check_sam_mod()
                if sam_mod_to_remove:
                    os.system(sam_mod_to_remove)
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers de détection de gènes de virulence impossible, l'analyse ne s'est
probablement pas déroulé normalement."
        finally:
            os.chdir("../..")

        # Rapport de la détection de gènes de virulence
        if detect_has_failed:
            if verbose_flag:
                print "-> La détection de gènes de virulence a échouée."
            os.system("rm " + detect_output_dir + "/*")
            os.rmdir(detect_output_dir)
        else:
            if verbose_flag:
                print "-> Détection de gènes de virulence effectuée. Résultats: " + detect_output_dir

    else:
        if verbose_flag:
            print "-> Pas de détection de gènes de
```

```
 virulence (pas de bdd)."

    # MLST Plasmides
    if plasmides_mlst:                                          # Si on spécifie une chemin vers une bdd de MLST, on
effectue la mlst, sinon pas besoin et on passe à la détection de gènes
        # Construction de la commande
        srst_command = srst2_mlst(file_name, trimming_output_dir, srst2_loc, profondeur_min, profondeur_ext_min,
taux_couverture, taux_divergence, plasmides_mlst, plasmides_definitions, plasmides_delimiteur, log_flag)

        # Dossier de sortie (comme pour le trimming)
        if not os.path.exists(plasmides_dir):
            os.mkdir(plasmides_dir)
        mlst_output_dir = check_outdir(plasmides_dir+"/MLST", file_name)
        os.chdir(mlst_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande MLST pour les plasmides :\n\n" + srst_command + "\n"

        mlst_has_failed = os.system(srst_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename("mlst.log", "../../../Log/" + file_name + "_mlst_plasmides.log")
            if rm_files:
                os.system("rm *.pileup; rm *.bam")
                sam_mod_to_remove = check_sam_mod()
                if sam_mod_to_remove:
                    os.system(sam_mod_to_remove)
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers de MLST de plasmides impossible, l'analyse ne s'est probablement pas
déroulé normalement."
        finally:
            os.chdir("../../..")

        # Rapport de la MLST
        if mlst_has_failed:
            if verbose_flag:
                print "-> Le typage MLST de plasmides a échoué."
            os.system("rm " + mlst_output_dir + "/*")
            os.rmdir(mlst_output_dir)
        else:
            if verbose_flag:
                print "-> Typage MLST de plasmides effectué. Résultats: " + mlst_output_dir

    else:
        if verbose_flag:
            print "-> Pas de typage MLST de plasmides effectué."

    # Gènes Plasmides
    if len(plasmides_database) > 0:
        gene_cat = "plasmides"                                  # Catégorie plasmides
        # Construction de la commande
        srst_command = srst2_detection(file_name, trimming_output_dir, srst2_loc, profondeur_min, profondeur_ext_min,
taux_couverture, taux_divergence, " ".join(plasmides_database), gene_cat, log_flag)

        # Dossier de sortie
        if not os.path.exists(plasmides_dir):
            os.mkdir(plasmides_dir)
        detect_output_dir = check_outdir(plasmides_dir+"/Genes", file_name)
        os.chdir(detect_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande de détection des plasmides :\n\n" + srst_command + "\n"
```

```
        # Exécution
        detect_has_failed = os.system(srst_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename(gene_cat + ".log", "../../../Log/" + file_name + "_" + gene_cat + ".log")
            if rm_files:
                os.system("rm *.pileup; rm *.bam")
                sam_mod_to_remove = check_sam_mod()
                if sam_mod_to_remove:
                    os.system(sam_mod_to_remove)
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers de détection de plasmides impossible, l'analyse ne s'est probablement
pas déroulé normalement."
        finally:
            os.chdir("../../..")

        # Rapport de la détection de plasmides
        if detect_has_failed:
            if verbose_flag:
                print "-> La détection de plasmides a échouée."
            os.system("rm " + detect_output_dir + "/*")
            os.rmdir(detect_output_dir)
        else:
            if verbose_flag:
                print "-> Détection de plasmides effectuée. Résultats: " + detect_output_dir

    else:
        if verbose_flag:
            print "-> Pas de détection de plasmides (pas de bdd)."
# ASSEMBLAGE
    if not skip_assembly:
        # Construction de la commande pour l'assemblage avec Spades
        assembly_command = spades_assembly(spades_loc, file_name, trimming_output_dir, n_threads, spades_kmers,
spades_cutoff, trim_phred[5:])

        assembly_output_dir = check_outdir(assemblage_dir, file_name)
        os.chdir(assembly_output_dir)

        if verbose_flag:
            print "-> Aperçu de la commande d'assemblage.\n\n" + assembly_command + "\n"

        # Exécution
        assembly_has_failed = os.system(assembly_command)

        # Nettoyage des fichiers
        try:
            if log_flag:
                os.rename('spades.log', '../../Log/'+file_name+"_spades.log")
            if rm_files:
                os.system("rm -r corrected/")
                os.system("rm -r K*/")
                os.system("rm -r misc/")
                os.system("rm -r mismatch_corrector/")
                os.system("rm -r tmp/")
                for dir_content in os.listdir("."):
                    if not re.search("contigs\.fasta", dir_content) and not re.search("scaffolds\.fasta",
dir_content):
                        os.remove(dir_content)
                os.rename("contigs.fasta", file_name+"_contigs.fasta")
                os.rename("scaffolds.fasta", file_name+"_scaffolds.fasta")
        except (IOError, OSError):
            print "/!\ Nettoyage des fichiers d'assemblage impossible, l'analyse ne s'est probablement pas déroulé
normalement."
```

```
        finally:
            os.chdir("../..")

        # Rapport de détection de l'assemblage
        if assembly_has_failed:
            if verbose_flag:
                print "-> L'assemblage a échoué."
            os.system("rm -r " + assembly_output_dir + "/*")
            os.rmdir(assembly_output_dir)
        else:
            if verbose_flag:
                print "-> Assemblage effectué avec succès. Génome dans " + assembly_output_dir

    else:

        if verbose_flag:
            print "-> Assemblage non effectué. --skip_assembly"

# Fin de l'exécution du pipeline (stats, traitement)
    # Calcul de la durée d'exécution
    duree_exec = time.time()-heure_dep
    if verbose_flag:
        print "-> Pipeline exécuté en {:.0f} minute(s) et {:.2f} seconde(s).".format(duree_exec / 60, duree_exec % 60)
```

Script 2 Compilation et Formatage des résultats, Phylogénie

```
#!/usr/bin/env python
#  coding=utf-8

"""
TTRaVDAP (Trimming, Typing, Resistance and Virulence genes and plasmids Detection, Assembly, Phylogeny)
Deuxième partie du pipeline, compilation des résultats (et formattage) et phylogénie
Ecrit par Pierre CHOLET sous la supervision de Richard BONNET, CHU Gabriel Montpied, Laboratoire de Microbiologie,
2015

Logiciels utilisés:
- SRST2:        M. Inouye; K. Holt et al. - SRST2: Rapid genomic surveillance for public health and hospital
microbiology labs (http://genomemedicine.com/content/6/11/90)
- Parsnp:       T.J. Treangen; B.D. Ondov; S. Koren; A.M. Phillippy - The Harvest suite for rapid core-genome
alignment and visualization of thousands of intraspecific microbial genomes. Genome Biology
(http://harvest.readthedocs.org/en/latest/)
"""

# Imports

import argparse
import glob
import os
import re
import sys
import time


#############
# Attributs #
#############

# Chemins (à modifier pour la grille de calcul ?)
srst2_loc = '/usr/local/bin/srst2'                                                  # Chemin vers srst2
parsnp_loc = '/usr/local/harvest-1.2/parsnp'                                        # Chemin vers Parsnp

# Dossiers de résultats (GARDER LES MêMES VALEURS QUE DANS LA PREMIèRE PARTIE DU PIPELINE)
mlst_dir = 'MLST'                                                                   # Dossier des résultats de
MLST (arborescence: MLST/[SOUCHE]/[fichiers de résultats])
resist_dir = 'Resistance'                                                           # Dossier des résultats de
détection de gènes de résistance (même arborescence)
virulence_dir = 'Virulence'                                                         # Dossier des résultats de
détection de gènes de virulence (idem)
plasmides_dir = 'Plasmides'                                                         # Dossier des résultats de
détection de gènes de plasmides (arborescence: Plasmides/[BDD]/[Souche]/[fichiers])
assemblage_dir = 'Assembly'                                                         # Dossier de sortie des
génomes assemblés par Spades
rapports_dir = 'Rapport'                                                            # Dossier de rapport des
résultats compilés de srst2
phylogenie_dir = 'Phylogenie'                                                       # Dossier des résultats de
phylogénie
log_dir = 'Log'                                                                     # Dossier des fichiers de log
trimm_dir = 'Trimmed'                                                               # Dossier des fichiers des
```

```
séquences trimmées

phylogenie_temporaire = 'TMP_phylogenie'                                                    # Dossier temporaire qui
rassemble les génomes pour la phylogénie

# Dictionnaire des gènes de résistance (Clé: 3 premières lettres du groupe de gènes en respectant la casse; Valeur:
Nom de la feuille de calcule pour grouper)
tableau_resistance = {
    "CTX": "B-lactamine",
    "OXA": "B-lactamine",
    "TEM": "B-lactamine",
    "Pen": "B-lactamine",
    "Amp": "B-lactamine",
    "AMP": "B-lactamine",
    "Str": "Aminoside",
    "Aac": "Aminoside",
    "Aph": "Aminoside",
    "Aad": "Aminoside",
    "Sul": "Sulfonamide",
    "Dfr": "Sulfonamide",
    "Tet": "Tetracycline",
    "Cml": "Chloramphenicol",
    "Cat": "Chloramphenicol",
    "Flo": "Chloramphenicol",
    "Mph": "Macrolide",
    "Sam": "Autre"
    }

#############
# Méthodes  #
#############


def parse_arguments():
    """Méthode qui définit les paramètres de la 2ème partie du pipeline"""
    # Paramètres
    parametres = argparse.ArgumentParser(sys.argv[0],
                                         description="""2ème partie du pipeline d'analyse de génomes bactériens,
compilation et formattage des résultats et phylogénie.""",
                                         prefix_chars='-',
                                         add_help=True,
                                         epilog='Écrit par BONNET R., CHOLET P. Laboratoire de Microbiologie, CHU G.
Montpied Clermont-Ferrand, 2015')

    parametres.add_argument('--gb_ref',         action='store', nargs=1,    type=str,   required=False,
dest='genbank_ref',
                            help='Génome de référence annoté au format genbank',
                            default=[])

    parametres.add_argument('--fasta_ref',      action='store', nargs=1,    type=str,   required=False,
dest='fasta_ref',
                            help='Génome de référence au format fasta (inutile si le genebank a été spécifié)')

    parametres.add_argument('--genome_type',    action='store', nargs=1,    type=str,   required=False,
dest='gen_type',
                            help='Type des fichiers du génome utilisé',
                            choices=['contigs', 'scaffolds'],
                            default=['contigs'])

    parametres.add_argument('--no_recombination', action='store_false', dest='no_recombination',
                            help='Ne pas prendre en compte les recombinaisons (omettre l\'option -x de parsnp)')

    parametres.add_argument('--skip',           action='store', nargs=1,    type=str,   required=False, dest='skip',
                            help='Étape à ne pas exécuter {compilation|phylogenie}',
                            choices=['compilation', 'phylogenie'],
                            default=[])

    parametres.add_argument('--no_verbose',  action='store_false',    dest='verbose',
                            help='Désactive le mode verbose (informations supplémentaires)')

    parametres.add_argument('--keep_files',     action='store_false',   dest='rm_files',
                            help='Empêche la suppression des fichiers annexes et rapports')

    parametres.add_argument('--clear_all',     action='store_true',   dest='clear',
                            help='Attention: cette option annule la compilation de résultats et la phylogénie mais
supprime TOUS les fichiers issus du pipeline (incluant tous les résultats, veillez à les récupérer avant)')

    return parametres.parse_args()


def compile_results(dir_to_parse, result_type):
    """Méthode qui recherche tous les fichiers de résultats en fonction du dossier passé en paramètre et les compile
grâce à Srst2 (résultats "__fullgenes__" incompatibles)"""

    result_files = []                                                                       # Liste des résultats trouvés
    result_files_fullgenes = dict()                                                         # Dictionnaire clé: analyse
(MLST, Résistance, Virulence, etc..) / valeur: liste des chemins vers les fichiers de résultats "__fullgenes__" (au
cas où formattage des résultats)

    if os.path.exists(dir_to_parse):                                                        # Si le dossier en paramètre
existe
        for strains_dirs in os.listdir(dir_to_parse):                                       # On liste les fichiers et
dossiers
            if os.path.isdir(os.path.abspath(dir_to_parse+"/"+strains_dirs)):           # Si c'est dossier (forcément
un dossier relatif à une souche)
                if verbose_flag:
                    print "-> Dossier de souche trouvé: " + strains_dirs
```

```
                for strains_files in os.listdir(dir_to_parse+"/"+strains_dirs):         # On liste l'intérieur des
dossiers des souches, un par un

                    if re.search("_results\.txt", strains_files):                      # Regex: si on trouve
"_results.txt" c'est très probablement un fichier de résultats
                        if re.search("fullgenes", strains_files):                      # On recherche les "fullgenes"
car Srst2 ne sait pas les traiter, inutile de lui donner
                            regex = re.compile("fullgenes__(\w+)")                     # Regex pour récupérer le nom
de la BDD utilisée et ainsi classer les résultats "fullgenes" par BDD
                            categorie = re.search(regex, strains_files).group(1)
                            if categorie not in result_files_fullgenes.keys():         # Si la clé n'existe pas on la
crée et on associe une liste de chemins de ficheirs de résultats, sinon on concatène la liste
                                result_files_fullgenes[categorie] =
[os.path.abspath((dir_to_parse+"/"+strains_dirs+"/"+strains_files))]
                            else:

result_files_fullgenes[categorie].append(os.path.abspath((dir_to_parse+"/"+strains_dirs+"/"+strains_files)))
                        else:                                                          # Sinon ce sont des résultats
normaux traitables par Srst2 donc on rajoute à la liste principale
                            result_files.append(os.path.abspath(dir_to_parse+"/"+strains_dirs+"/"+strains_files))

        # Construction de la commande de compilation des résultats
        compiling_command = srst2_loc + " --prev_output " + " ".join(result_files) + " --output
"+rapports_dir+"/"+result_type+"_Full_Report"

        if verbose_flag:
            print "-> Aperçu de la commande de compilation des résultats:\n" + compiling_command

        # Dossier de sortie (créé s'il n'existe pas)
        if not os.path.exists(rapports_dir):
            os.mkdir(rapports_dir)

        # Exécution de la commande
        compiling_has_failed = os.system(compiling_command)

        if compiling_has_failed:
            if verbose_flag:
                print "-> La compilation des résultats de " + result_type + " a échouée."
        else:
            if verbose_flag:
                print "-> Compilation des résultats de " + result_type + " réussie. Rapports dans le dossier '" +
rapports_dir + "'"

        # Traitement des fichiers fullgenes
        # for db in result_files_fullgenes.keys():
            # Srst2 ne sait pas traiter les les fichiers "fullgenes"

            # Passer les fichiers
 à srst2 -> Inutile car SRST2 ne les traitera pas mais au besoin s'il en vient nécessaire de les traiter
            # compiling_command = srst2_loc + " --prev_output " + " ".join(result_files_fullgenes.get(db)) + " --
output "+rapports_dir+"/"+result_type+"_"+db+"_Full_Report"
            # print compiling_command
            # os.system(compiling_command)

            # Printer les chemins des listes de chaque clé du dictionnaire -> Inutile également, à titre de
vérification mais au besoin
            # print db
            # for file_path in result_files_fullgenes[db]:
            #     print "\t"+file_path

    else:
        if verbose_flag:
            print "Dossier " + dir_to_parse + " introuvable, compilation des résultats impossible."


def format_resistance_results(resistance_result_file, tableau_associatif):
    """Méthode qui permet de formatter les résultats (des gènes de résistance); méthode à refactoriser voir à
recoder"""
    readed_file_to_format = open(resistance_result_file, "r")

    all_my_lines = []

    for line in readed_file_to_format:      # Récupération de toutes les lignes du fichier
        all_my_lines.append(line)

    readed_file_to_format.close()

    if verbose_flag:
        print "-> Nombre de souches détectés lors du formattage des résultats de résistance: " +
str(len(all_my_lines)-1)
        print "-> Nombre de gènes détectés lors du formattage des résultats de résistance: " +
str(len(all_my_lines[0].split("\t"))-1)

    my_genes_id = set()

    nombre_gene = 0
    last_elem = ''

    line_zero = all_my_lines[0].rstrip("\n").split("\t")        # Liste qui contient tous les gènes de la première
ligne (+ "Sample")
    # Décommenter les lignes qui suivent pour éliminer le premier élément "Sample" qui ne correspond pas à un gène, au
risque de passer à coté d'un gène qui se nomme "Sam"
    # sam_got_rid_of = False
    for elem in line_zero:
        # if elem == "Sam":
        #       sam_got_rid_of = True
        # elif sam_got_rid_of:
```

```
        if not last_elem == elem:
            nombre_gene += 1
        my_genes_id.add(elem[:3])
        last_elem = elem
    nombre_gene -= 1  # Enlève la case sample du nombre de gènes, ligne à commenter si les lignes ci-dessus sont
décommentées

    # Écriture du classeur openoffice au format .odt
    my_xml = open(os.path.splitext(resistance_result_file)[0] + ".ods", "w")
    my_xml.write('<?xml version="1.0" encoding="UTF-8"?>')
    my_xml.write('<office:document xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
office:mimetype="application/vnd.oasis.opendocument.spreadsheet">\n')  # Format pour open office
    my_xml.write('<office:body>')

    gene_familly_unique = set()

    for genes in my_genes_id:                                   # Pour chaque famille de gènes
        gene_familly_unique.add(tableau_associatif[genes])      # On ajoute la famille des antibiotiques associée

    for gene_familly in gene_familly_unique:                    # Pour chaque famille d'antibiotiques dont au moins 1
gène a été détecté
        my_xml.write('<office:spreadsheet> <table:table table:name="' + gene_familly + '">')    # Nouvelle feuille de
calcul

        column_to_keep = []
        last_row = ''

        for line in all_my_lines:                               # On parcourt les lignes
            line_list = line.rstrip("\n").split("\t")           # On récupère chaque champ (chaque allèle)
            if line_list[0] == "Sample":                        # Si le premier champ == "Sample" c'est la première
ligne donc on va écrire les balises pour ouvrir un Tableau
                for i in range(0, len(line_list)):
                    if i == 0:                                  # Si premier élément, début de ligne, donc ouvrir les
balises ligne (row) et écrire le nom des souches
                        my_xml.write('<table:table-row> <table:table-cell office:value-type="string"> <text:p>' +
line_list[i] + '</text:p> </table:table-cell>\n')
                        last_row = line_list[i]
                    else:                                       # Sinon écrire les gènes
                        gene_list = [pairs[0] for pairs in tableau_associatif.items() if pairs[1] == gene_familly]
# Si ils font partis des familles de gènes de la famille d'antibio
                        for gene in gene_list:
                            if re.match(gene, line_list[i]):
                                if not line_list[i] == last_row:
                                    column_to_keep.append(i)
                                    my_xml.write('<table:table-cell office:value-type="string"> <text:p>' +
line_list[i] + '</text:p> </table:table-cell>\n')

                                last_row = line_list[i]
            else:  # A partir de la deuxième ligne du tableau
                gene_tot_in_clust = 0
                genes_count = 0
                for i in range(0, len(line_list)):
                    if i == 0:  # Première colonne (souche)
                        # MyList.append
                        my_xml.write('<table:table-row> <table:table-cell office:value-type="string"> <text:p>' +
line_list[i] + '</text:p> </table:table-cell>\n')
                    else:  # Gènes
                        if i in column_to_keep:  # Si ce n'est pas un doublon
                            my_xml.write('<table:table-cell office:value-type="string"> <text:p>' + line_list[i] + '</
text:p> </table:table-cell>\n')
                            gene_tot_in_clust += 1
                            if not re.match("-", line_list[i]):  # Si ça ne match pas du vide
                                genes_count += 1
                # my_xml.write('<table:table-cell> <text:p>' + str(genes_count) + '</text:p> </table:table-cell>\
n<table:table-cell office:value-type="string"> <text:p>' + str(genes_count*100/gene_tot_in_clust) + '%</text:p>
</table:table-cell>\n')
            my_xml.write('</table:table-row>')
        my_xml.write('</table:table>'
                     '</office:spreadsheet>')

    my_xml.write('</office:body>'
                 '</office:document>')

    my_xml.close()


def format_virulence_results(virulence_result_file):
    """Méthode analogue au formattage des résultats de résistance mais sans onglets car catégorisation beaucoup plus
complexe"""

    too_many_columns_file = open(virulence_result_file, "r")

    all_my_lines = []

    for line in too_many_columns_file:
        all_my_lines.append(line)

    too_many_columns_file.close()

    if verbose_flag:
        print "-> Nombre de souches détectés lors du formattage des résultats de virulence: " + str(len(all_my_lines)-
1)
        print "-> Nombre de gènes détectés lors du formattage des résultats de virulence: " +
str(len(all_my_lines[0].split("\t"))-1)
```

```
    my_genes_id = set()

    line_zero = all_my_lines[0].rstrip("\n").split("\t")
    for elem in line_zero:
        my_genes_id.add(elem[:3])

    my_xml = open(os.path.splitext(virulence_result_file)[0] + ".ods", "w")
    my_xml.write('<?xml version="1.0" encoding="UTF-8"?>')
    my_xml.write('<office:document xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
office:mimetype="application/vnd.oasis.opendocument.spreadsheet">\n')  # Format openoffice
    my_xml.write('<office:body>')

    # my_genes_id.discard("Sam")

    for genes in my_genes_id:
        my_xml.write('<office:spreadsheet> <table:table table:name="' + genes + '">')

        column_to_keep = []
        last_row = ''

        for line in all_my_lines:
            line_list = line.split("\t")
            if line_list[0] == "Sample":
                for i in range(0, len(line_list)):
                    if i == 0:
                        my_xml.write('<table:table-row> <table:table-cell office:value-type="string"> <text:p>' +
line_list[i] + '</text:p> </table:table-cell>\n')
                        last_row = line_list[i]
                    else:
                        if re.match(genes, line_list[i]):
                            if not line_list[i] == last_row:
                                column_to_keep.append(i)
                                my_xml.write('<table:table-cell office:value-type="string"> <text:p>' + line_list[i] +
'</text:p> </table:table-cell>\n')

                            last_row = line_list[i]
            else:  # A partir de la deuxième ligne du tableau
                gene_tot_in_clust = 0
                genes_count = 0
                for i in range(0, len(line_list)):
                    if i == 0:  # Première colonne (souche)
                        # MyList.append
                        my_xml.write('<table:table-row> <table:table-cell office:value-type="string"> <text:p>' +
line_list[i] + '</text:p> </table:table-cell>\n')
                    else:  # Gènes
                        if i in column_to_keep:  # Si ce n'est pas un doublon
                            my_xml.write('<table:table-cell office:value-type="string"> <text:p>' + line_list[i] + '</
text:p> </table:table-cell>\n')
                            gene_tot_in_clust += 1
                            if not re.match("-", line_list[i]):  # Si ça ne match pas du vide
                                    genes_count += 1
                # my_xml.write('<table:table-cell> <text:p>' + str(genes_count) + '</text:p> </table:table-cell>\
n<table:table-cell office:value-type="string"> <text:p>' + str(genes_count*100/gene_tot_in_clust) + '%</text:p>
</table:table-cell>\n')
            my_xml.write('</table:table-row>')
        my_xml.write('</table:table>'
                     '</office:spreadsheet>')

    my_xml.write('</office:body>'
                 '</office:document>')


# Utilitaires
def clear_all():
    """Méthode qui supprimes tous les fichiers puis tous les dossiers issus du pipeline, incluant tous les
résultats"""
    dir_to_clean_list = (mlst_dir, resist_dir, virulence_dir, plasmides_dir, assemblage_dir, rapports_dir,
phylogenie_dir, log_dir, trimm_dir)  # Liste de tous les dossiers
    for dir_to_clean in dir_to_clean_list:
        if os.path.exists(dir_to_clean) and os.path.isdir(dir_to_clean):
            clear_has_failed = os.system("rm -r " + dir_to_clean)

            if clear_has_failed:
                if verbose_flag:
                    print "-> Le nettoyage des fichiers du dossier '" + dir_to_clean + "' a échoué."
            else:
                if verbose_flag:
                    print "-> Nettoyage des fichiers du dossier '" + dir_to_clean + "' effectué."

#############
# Main      #
#############

if __name__ == '__main__':

    heure_dep = time.time()                         # Permet de calculer le temps d'exécution

############################################################
#                       ARGUMENTS                          #
############################################################

    arguments = parse_arguments()                   # Récupération des arguments

    if arguments.genbank_ref:
        ref_genbank = arguments.genbank_ref[0]
```

```
        ref_fasta = False
    elif arguments.fasta_ref:
        ref_genbank = False
        ref_fasta = arguments.fasta_ref[0]
    else:
        ref_genbank = False
        ref_fasta = False
        print "/!\ Attention, aucun génome de référence spécifié, impossible de lancer l'analyse phylogénique."

    genomes_type = arguments.gen_type[0]

    recombination = arguments.no_recombination

    skip_compil = False
    skip_phylogeny = False
    if arguments.skip[0] == "compilation":
        skip_compil = True
    elif arguments.skip[0] == "phylogenie":
        skip_phylogeny = True

    # Flags
    verbose_flag = arguments.verbose

    rm_files = arguments.rm_files

    clear_all_files = arguments.clear


############################################################
#                      EXECUTION                           #
############################################################

    if clear_all_files:                             # Si le flag clear_all_files est présent, on ne fait nettoyer le
répertoire
        clear_all()
    else:                                           # Sinon on effectue la 2ème partie du pipeline
        if not skip_compil:
            # Compilation des Résultats
            compile_results(mlst_dir, "MLST")
            compile_results(resist_dir, "Resistance")
            compile_results(virulence_dir, "Virulence")
            compile_results(plasmides_dir+"/MLST", "Plasmides_MLST")
            compile_results(plasmides_dir+"/Genes", "Plasmides_Genes")
            # Formattage des résultats
            try:
                format_resistance_results(rapports_dir+"/Resistance_Full_Report__compiledResults.txt",
tableau_resistance)
                format_virulence_results(rapports_dir+"/Virulence_Full_Report__compiledResults.txt")
            except KeyError:
                print "Erreur de formattage des résultats."
        else:
            if verbose_flag:
                print "-> Compilation des résultats annulée"

        # Phylogénie
        if not skip_phylogeny:
            if ref_genbank or ref_fasta:
                if not os.path.exists(phylogenie_dir):
                    os.mkdir(phylogenie_dir)

                if not os.path.exists(phylogenie_temporaire):
                    os.mkdir(phylogenie_temporaire)

                if genomes_type == 'contigs':
                    genomes_list = glob.glob(os.path.join(assemblage_dir+"/*", "*coli.fasta"))
                else:
                    genomes_list = glob.glob(os.path.join(assemblage_dir+"/*", "*_scaffolds.fasta"))

                for genome in genomes_list:
                    os.system("cp " + genome + " " + phylogenie_temporaire + "/")

                phylogenie_command = parsnp_loc + " -d " + phylogenie_temporaire + "/"
                if ref_genbank:
                    phylogenie_command += " -g " + ref_genbank
                else:
                    phylogenie_command += " -r " + ref_fasta
                phylogenie_command += " -o " + phylogenie_dir
                if recombination:
                    phylogenie_command += " -x"
                if verbose_flag:
                    phylogenie_command += " -v"

                if verbose_flag:
                    print "-> Aperçu de la commande de l'analyse phylogénique:\n" + phylogenie_command

                phylogenie_has_failed = os.system(phylogenie_command)

                try:
                    if not os.path.exists(log_dir):
                        os.mkdir(log_dir)
                    os.rename("parsnpAligner.log", log_dir+"/parsnp_phylogenie.log")
                    if rm_files:
                        os.system("rm " + phylogenie_dir + "/*.ini")
                        os.system("rm " + phylogenie_dir + "/all*mumi*")
                except (IOError, OSError):
                    print "/!\ Nettoyage des fichiers de phylogénie impossible, l'analyse ne s'est probablement pas
déroulé normalement."
```

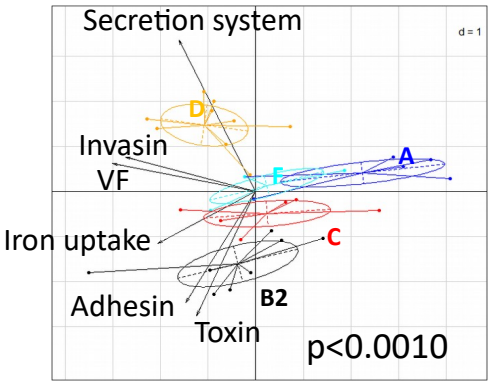
```
                finally:
                    os.system("rm -r " + phylogenie_temporaire)

                # Rapport de la phylogénie
                if phylogenie_has_failed:
                    if verbose_flag:
                        print "-> L'analyse phylogénique a échouée."

                else:
                    if verbose_flag:
                        print "-> Analyse phylogénique effectué. Résultats dans '" + phylogenie_dir + "'"
            else:
                if verbose_flag:
                    print "-> Analyse phylogénique annulée, pas de génome de référence spécifié."
        else:
            if verbose_flag:
                print "-> Phylogénie annulée."
    # Fin de l'exécution du pipeline (stats, traitement)
        # Calcul de la durée d'exécution
        duree_exec = time.time()-heure_dep

        if verbose_flag:
            print "-> Pipeline exécuté en {:.0f} minute(s) et {:.2f} seconde(s).".format(duree_exec / 60, duree_exec %
60)
```

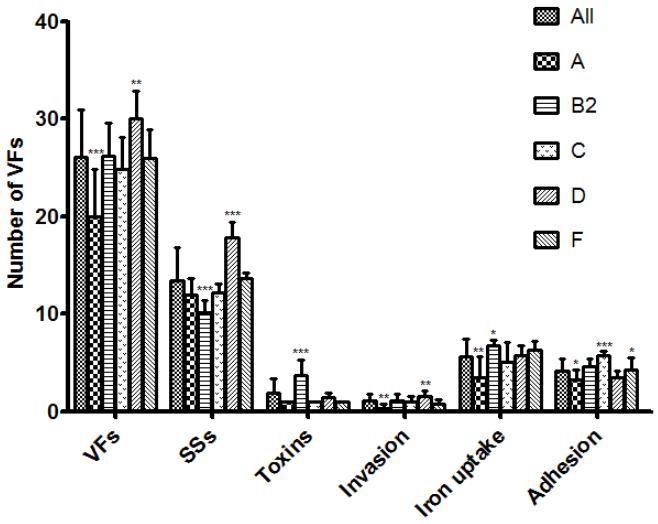
Annexe 2 – Gènes de résistance détectés et classification

Échantillon	B-Lactamines	Aminosides	Sulfamides et Triméthoprime	Cyclines	Phénicolés	Macrolides	Total	Phylogroupe	ST (Warwick)	Complexe Clonal	ST (Pasteur)	Groupe BLSE	Type de CTX-M
N°1	2	3	2	0	1	0	8	D	ST405	CC405	ST477	CTXM1	CTXM139
N°6	3	5	3	4	2	1	18	B1	ST410	CC23	ST471	CTXM1	CTXM15
N°7	2	4	4	2	1	0	13	A	ST10	CC10	ST2	CTXM1	CTXM32
N°21	3	3	2	1	2	1	12	D	ST648	CC648	NA	CTXM1	CTXM15
N°22	2	2	0	0	1	0	5	B2	ST131	CC131	ST43	CTXM1	CTXM15
N°27	2	1	2	0	0	0	5	D	ST38	CC38	ST8	CTXM9	CTXM27
N°34	2	1	3	2	0	0	8	D	ST38	CC38	ST8	CTXM1	CTXM1
N°45	2	0	0	0	0	0	2	B2	ST131	CC131	ST43	CTXM1	CTXM15
N°49	3	3	2	0	1	1	10	B2	ST131	CC131	ST43	CTXM1	CTXM15
N°61	1	0	0	0	0	0	1	D	ST69	CC69	ST3	CTXM9	CTXM14
N°67	2	4	3	2	0	1	12	B2	ST131	CC131	ST9	CTXM9	CTXM14
N°68	2	1	2	0	2	0	7	B1	ST88	CC23	ST74	CTXM9	CTXM14
N°70	2	1	2	0	0	0	5	B2	ST95	CC95	NA	CTXM1	CTXM1
N°80	1	0	0	0	0	0	1	D	ST62	ST62	NA	CTXM9	CTXM14
N°81	2	1	0	2	1	0	6	B2	ST131	CC131	ST43	CTXM1	CTXM15
N°83	3	2	2	1	2	0	10	B1	ST88	CC23	ST74	CTXM1	CTXM15
N°86	1	0	0	0	0	0	1	B2	ST141	ST141	ST10	CTXM1	CTXM1
N°95	2	3	1	2	0	0	8	A	ST10	CC10	ST2	CTXM9	CTXM14
N°98	2	2	2	2	1	1	10	B2	ST131	CC131	ST621	CTXM1	CTXM15
N°107	1	0	1	2	0	0	4	B2	ST141	ST141	ST26	CTXM1	CTXM1
N°110	2	2	1	0	0	0	5	D	ST38	CC38	ST8	CTXM9	CTXM14
N°122	2	4	2	0	0	0	8	D	ST69	CC69	ST3	CTXM9	CTXM14
N°123	1	0	0	0	0	0	1	D	ST38	CC38	ST8	CTXM9	CTXM27
N°124	1	4	2	1	0	0	8	B1	ST3249	ST3249	ST66	CTXM1	CTXM1
N°125	2	2	2	1	1	1	9	D	ST38	CC38	ST8	CTXM9	CTXM14
N°126	2	1	2	0	2	0	7	B1	ST88	CC23	ST74	CTXM9	CTXM14
N°128	1	0	1	2	0	0	4	A	ST746	CC746	NA	CTXM1	CTXM1
N°134	1	4	0	0	1	1	7	A	ST93	CC168	ST17	CTXM1	CTXM1
N°144	1	0	0	0	0	0	1	D	ST38	CC38	ST8	CTXM9	CTXM27
N°146	1	1	1	0	0	0	3	A	ST10	CC10	ST2	CTXM1	CTXM1
N°147	2	1	2	1	1	0	7	B1	ST88	CC23	ST74	CTXM9	CTXM14
N°150	1	0	0	1	0	0	2	D	ST648	CC648	NA	CTXM9	CTXM14
N°155	2	0	0	0	0	0	2	D	ST69	CC69	ST3	CTXM9	CTXM14
N°164	1	0	0	0	0	0	1	D	ST38	CC38	ST8	CTXM9	CTXM27
N°172	1	0	0	2	0	0	3	A	ST10	CC10	ST2	CTXM1	CTXM1
N°200	1	1	2	0	0	0	4	B2	ST12	CC12	ST36	CTXM1	CTXM1
Total	100,00%	66,67%	63,89%	44,44%	38,89%	19,44%	100,00 %						

Annexe 3 – Exemple d'analyses statistiques complémentaires

E. coli phylogroup





CTX-M group