Universidad Autónoma de Madrid
Escuela politécnica superior



# Master's Degree in Bioinformatics and Computational Biology

## Master's Degree Final Project

# ExplorePipolin: a pipeline for identification and exploration of pipolins, novel mobile genetic elements widespread among bacteria

Author: Liubov Chuprikova
Tutor: Modesto Redrejo Rodríguez
Tutor: María de Toro Hernando

JUNE 2020

# ExplorePipolin: a pipeline for identification and exploration of pipolins, novel mobile genetic elements widespread among bacteria

Author: Liubov Chuprikova
Tutor: Modesto Redrejo Rodríguez
Tutor: María de Toro Hernando

Departamento de Bioquímica, Universidad Autónoma de Madrid

Plataforma de Genómica y Bioinformática, Centro de Investigación Biomédica de La Rioja, Fundación Rioja Salud

Escuela Politécnica Superior, Universidad Autónoma de Madrid

JUNE 2020

# Abstract

**Introduction.** Pipolins constitute a new group of self-synthesizing or self-replicating mobile genetic elements (MGEs), encoding for their own replicative DNA polymerase B. These elements have been found to be mostly integrated into the genomes of bacteria from diverse phyla and also present as circular plasmids in mitochondria. Since a reduced number of pipolins has been identified and described so far, their origin and role remains unknown as well as there is little evidence of their horizontal transfer. A bioinformatics software capable of automatic identification and analysis of pipolins from bacterial genomes might ensure the progress in the accumulation of knowledge about these mobile genetic elements. Therefore, the main goal of the current project was to design and implement a pilot version of a pipeline for the identification and analysis of pipolins from *Escherichia coli* genomes. The pipeline should be flexible enough to easily extend it to other bacteria in the future. As a sub-goal, it was decided to perform a detailed analysis of pipolins of *E. coli* strains and isolates, available from the NCBI database and from the Spanish *E. coli* Reference Laboratory (LREC) collection.

**Results.** We have identified and characterised pipolin elements from 92 *E. coli* genomes, 25 of which were selected from the LREC collection and 67 – retrieved from the NCBI database. Pipolins from the *E. coli* genomes have been shown to present in a wide range of strains from different phylogroups, serotypes and clonotypes and to be highly diverse in their genetic structure and composition. Despite their great variability, the pipolin elements are flanked by conserved *att*-like terminal direct repeats and integrated into the same tRNA gene. Cophylogeny analysis showed a lack of congruence between phylogenies of some groups of the pipolins and their host strains, which is in agreement with the hypothesis of pipolins horizontal transfer.

Based on the predefined hallmark features of *E. coli* pipolins, we have designed and implemented a pilot version of the ExplorePipolin pipeline, which is intended to identify, scaffold, extract and annotate pipolin elements from bacterial genome sequences. The source code of the ExplorePipolin pipeline is available on GitHub: `https://github.com/liubovch/ExplorePipolin`. It is a command-line software that can be installed and run on any Unix-like system. Furthermore, we have created a Docker image and a Conda recipe to ease the pipeline installation and running on different systems.

**Conclusions.** The ExplorePipolin pipeline can be useful for microbiologists that are interested in pipolins. It can be also an example of a comprehensive pipeline that not only looks for pipolin markers within a genome sequence but reconstruct the whole element structure when it is possible, making further analysis more precise and straightforward. The current pipeline version is mainly tested on *E. coli* genomes, therefore, we will focus on extending it to other bacterial species in the future.

## Key words

Mobile genetic elements, pipolins, primer-independent DNA polymerase B, *Escherichia coli*, pipeline, Python.

# Acknowledgments

I am deeply grateful to my advisors Modesto Redrejo Rodríguez and María de Toro Hernando for their guidance, support and inspiration during all stages of my work on the project. I was a fruitful experience for me to work under their supervision.

I would also like to thank my husband for his great patience and the advice given from his professional point of view.

# Contents

# Figure Index



# Table Index

# 1 Introduction

## 1.1 Motivation of the Project

Pipolins constitute a new group of self-synthesizing or self-replicating mobile genetic elements (MGEs), along with eukaryotic Polintons and archaeal Casposons, encoding for their own replicative DNA polymerase from family B [1]. Similar to other self-replicating MGEs, most pipolins are integrated within bacterial chromosomes, although they have been also identified as circular plasmids both in bacteria and mitochondria. Pipolins encode for a distinct group of PolBs capable of *de novo* DNA synthesis and named primer-independent PolBs (piPolBs) [2]. Apart from that, one or more integrases of the tyrosine recombinase superfamily (Y-Rec) usually present in pipolins, suggesting that they might be responsible for pipolins excision and/or integration.

Despite the limited number of pipolins identified so far, they are widespread among diverse bacterial phyla and mitochondria [2]. Their distribution suggests of the ancient origin of these MGEs, and that they might have been horizontally transferred between bacteria. Horizontally transferred DNA elements are known to contribute to bacterial evolution and adaptation by providing useful functions or properties [3]. Therefore, analysis of pipolin prevalence and dynamics among different bacteria may help to understand their origin, evolution and role, as well as the details of their replication and excision/integration mechanisms.

Due to the novelty of pipolin elements, there are no tools or databases that might help researchers in their study of pipolins. Thus, creating such software is important to ensure the progress in the accumulation of knowledge about these elements, and also broaden the interest in pipolins for scientists in the field of microbial genomics or microbiology in general. The process of mining sequencing data for the presence of pipolins and their subsequent detailed analysis can be logically divided into steps like, 1) looking for pipolin-specific markers, 2) element structure reconstruction, 3) composition analysis of elements, 4) comparative analysis of elements. The easiest way to automate these steps is to organise them into a pipeline.

On the other hand, the pipeline implementation itself assumes that we have prior knowledge about the expected structure and composition of at least some pipolin elements. Therefore, we have decided to perform a preliminary analysis of pipolins from *Escherichia coli*, because: 1) there is a great interest in *E. coli* as a model and a pathogenic organism, which made it the best-known bacteria, 2) there is plenty of already sequenced and assembled genomes from

different *E. coli* strains and isolates in public databases, 3) a big collection of pathogenic *E. coli* from the Spanish *E. coli* Reference Laboratory (LREC) was previously surveyed for the presence of isolates encoding the piPolB gene, and their genome assemblies were also available for our analysis. The conclusions made from the preliminary analysis of *E. coli* pipolins would help us in the subsequent pipeline design and implementation.

## 1.2 The Main Goal and Objectives

The main goal of the current work was to create a pipeline for identification, extraction and annotation of pipolins from *E. coli* strains, that would be flexible enough to easily extend it to other bacterial species in the future. As a sub-goal, the detailed analysis of pipolins from *E. coli* had to be performed.

Based on the above, the following objectives were undertaken:

1. Identify pipolin-harboring *E. coli* genomes and perform detailed analysis of their pipolin elements:
   (a) Identify potential pipolin-harboring *E. coli* genomes as those encoding for the piPolB gene.
   (b) Analyse genomes for the presence of known terminal direct repeats (*att* sites) that would define pipolin element boundaries.
   (c) Extract pipolin elements from the genomes in such a manner as to make their further analysis pipolin-oriented.
   (d) Perform accurate and homogeneous annotation of pipolin genes.
   (e) Determine other potential hallmark features of pipolins with the help of pan-genome, functional and comparative analysis methods.
   (f) Analyse diversity of pipolin-harboring strains in terms of phylogroups, serotypes, sequence types and clonotypes to define whether the presence of pipolins can be associated with certain groups within *E. coli*.
   (g) Perform comparative phylogenetic analysis of pipolins and pipolin-harboring (host) bacteria to check the hypothesis of their horizontal transfer.
2. Based on the predefined hallmark features of *E. coli* pipolins, develop a pilot version of a pipeline for automatic identification, extraction and annotation of pipolins from bacterial genomes.

# 2

# Challenges in Identification and Analysis of MGEs

MGEs are one of the key players in bacterial genomes involved in genome reorganization and evolution, often responsible for acquisition by the organism of valuable adaptive traits like antimicrobial resistance (AMR), virulence factors, enzymes of secondary metabolism, etc. that can change organism's fitness, pathogenicity and diversity. It is also disputed that MGEs play an important role in social behaviour within microbial populations because a wide range of secreted proteins are found linked with mobile elements [4].

The collection of MGEs itself in a given genome is called mobilome and can comprise of plasmids, bacteriophages, transposon (Tn), insertion sequences (ISs), gene cassettes, integrons, integrative conjugative elements (ICEs) and genomic islands (GIs). All these elements vary highly in their genetic structure, length and mechanisms of transfer which makes difficult their simultaneous *in silico* analysis. Predictions are also challenged by modular nature (Figure 2.1) and rapid evolution of elements through gene acquisition and gene loss, and historical dividing of MGEs on different classes is becoming less clear these days. On the other hand, the increasing large-scale sequencing of microbial genomes places an urgent demand for such novel computational tools.

Nowadays, most of the software and databases for identification and analysis of MGEs are specialised on a certain type of elements, for example, for plasmids – PlasmidFinder [7], PLSDB [8]; prophages – PHASTER [9], Phigaro [10]; integrons – Integron Finder [11], INTEGRALL [12]; ISs – ISFinder [13]; ICEs – ICEberg [14]; GIs – IslandViewer 4 [15], Islander [16], etc. For prediction, the most commonly used are **sequence composition-based approaches** that are looking for specific signatures of horizontally transferred sequences to distinguish them from the rest of the genome [17]. Typical features of MGEs that are inspected by the programs include:

- local nucleotide composition bias (GC content, GC skew, k-mer frequencies, codon and amino acid usage)
- presence of MGE-specific mobility genes
- high prevalence of prophage-related genes
- presence of other hallmark genes, as AMR or virulence genes

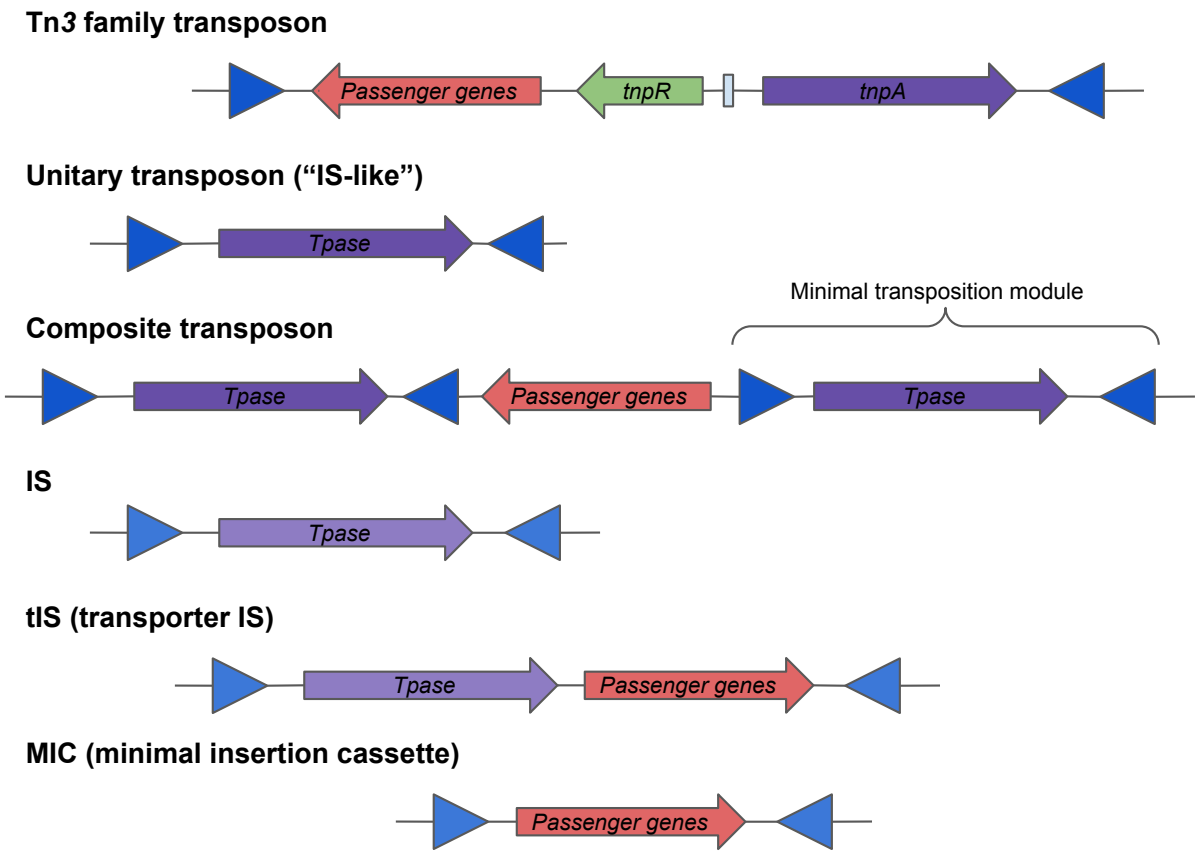



Figure 2.1: Modular structure of MGEs on the example of classic organization Tn*3* family transposon, IS and their derivatives (the figure is adapted from [5, 6]). Blue triangles, terminal inverted repeats; Tpase, transposase gene; *tnpA*, Tn*3* Tpase; *tnpR*, Tn*3* resolvase gene; light green box, *res* recombination site. Unitary transposons are solely constituted by the minimal transposition module. Unitary elements can associate to form composite transposons. tISs include passenger genes. MICs are non-autonomous elements that rely on the presence of a cognate Tpase in the cell.

- high prevalence of hypothetical proteins
- presence of direct or inverted terminal repeats

To define nucleotide composition bias, window-based methods are used: they slide the genome with a window of a chosen size and determine atypical composition with the help of scoring schemes. In order to find mobility, prophage or other marker genes, the pre-defined non-redundant databases are made to search against them using BLAST [18], HMMER [19] or INFERNAL [20] (when nucleotide secondary structure is important). This search is often combined with the window-based approaches to pinpoint clusters of genes with a high probability of being a MGE. Filtering by the presence of nearby integrase or transposase genes is used very often, as well as the presence of flanking direct or inverted repeats may serve as additional confirmation of the element's foreign origin. While most of the tools start their analysis by looking for MGE markers, some tools with totally different approaches can be encountered. For example, authors of the Islander database look first for tRNA/tmRNA genes and their fragments to identify the sequence caused this fragment displacement. Though the method allows precise prediction of mobile element boundaries, it only finds those mobile elements that use tRNA or tmRNA genes as an integration site [16].

**Machine learning (ML) methods**, although seem very promising, are not very common. A clear advantage of these methods is their ability to use all the known features of MGEs available from multiple resources and databases to train their models. Though the improved accuracy was declared for such tools years ago [21, 22, 23], they are rarely developed. On the wave of the global popularity of ML, it is tending to suspect that there are some challenges to create a flexible and highly accurate model (still not enough data, not enough computational resources, high error susceptibility).

**Comparative approaches** also could be used to predict MGEs, however, they became less popular or they are predominantly used in combination with sequence composition-based approaches [15, 24, 25]. A serious disadvantage of comparative methods is that they require a considerable set of closely related genomes for comparison, and results may vary depending on the chosen reference genomes.

The aforementioned or similar tools are generally powerful and highly accurate in their predictions [17]. However, when talking about the discovery of novel classes of MGEs, these tools are obviously not in help as they filter out all extraordinary or peculiar elements found, in order to decrease false positives rate. That is also the main reason why pipolins cannot be defined by these tools. Comparative and **evolutionary methods** that are designed to detect horizontal gene transfer (HGT) events, disregarding of sequence composition, might be suitable [26, 27]. However, in order to predict novel classes of MGEs, they should accurately filter out not only known types of MGEs, but recognize and filter out common intra-chromosome rearrangements as duplications, deletions and inversions. So, the first discovery of novel types and classes of MGEs occurring from time to time [2, 28, 6] is believed to happen mostly as a result of researchers curiosity and serendipity – with the help of computers, but not by self-sufficient computer programs.

# 3

# Analysis of Pipolins from *E. coli* Strains

## 3.1 Identification of Pipolin-Harboring *E. coli* Strains

A survey of pipolin distribution among 2238 strains from the LREC collection had been performed previously, using a 587 nt fragment of the piPolB coding sequence from *E. coli* 3-373-03_S1_C2 strain as a marker. 25 pipolin-harboring isolates were detected, indicating that pipolins are not particularly abundant (1.1%) among pathogenic *E. coli*. Their genomes were sequenced and assembled to the level of contigs, as well as plasmids were pulled out using the methodology of PLAsmid Constellation NETwork (PLACNETw) [29, 30].

To increase the number of samples, we performed a TBLASTN search against the NCBI nucleotide database, restricted to the *E. coli* taxon (taxid:562), using piPolB amino acid sequence from *E. coli* 3-373-03_S1_C2 strain as a query. This search yielded 76 hits, corresponding to piPolB-encoding ORFs or their fragments (identity above 85%) from 67 strains (Oct 16, 2019). The corresponding genome sequences were downloaded in the form of contigs or a complete chromosome in FASTA format.

In total, 92 *E. coli* genomes (25 from LREC collection and 67 from NCBI) were employed in the subsequent analysis.

## 3.2 Prediction of Terminal Direct Repeats

The pipolin from *E. coli* 3-373-03_S1_C2 strain have been characterised earlier [2], and it has been shown to contain *att*-like terminal direct repeats, one of which is overlapping with a tRNA gene. tRNA genes frequently serve as an integration site for prokaryotic genetic elements, while terminal direct repeats or *att* sites, formed at the time of integration event, further define genetic element boundaries. Based on that, in the preliminary analysis, we used the known *att* site (about 113-118 nt long) to check whether it is present in other piPolB-containing genomes. For that, the nucleotide BLAST was performed against each of the genomes.

We were able to find at least two *att* sites in all genomes, except for the strain LREC243, which had only one *att*. In cases, when *att*s were located on the same contig or on a complete

chromosome, the piPolB gene was always sitting within the repeats, in agreement with the expected structure of pipolin elements (Figure 3.1).

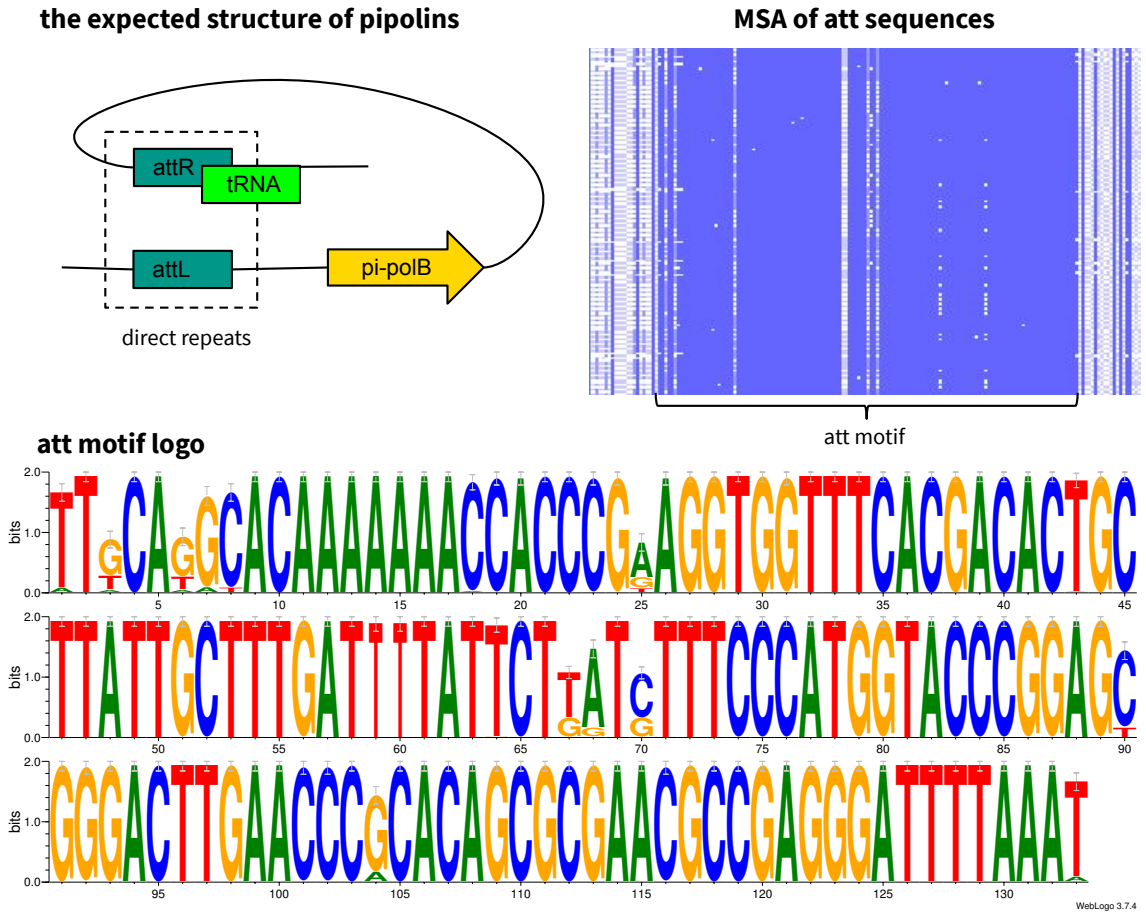



Figure 3.1: Analysis of *att* motifs from *E. coli* pipolins. All found *att* sequences were extended by around 20 nt from both sides, extracted from the genomes and aligned. Sequence logo was created for the derived conserved *att* motif (133 nt). Error bars indicate an approximate Bayesian 95% confidence interval.

The conservancy of the found *att* sites have been further investigated by extracting *att* sequences and aligning them, using MAFFT [31]:

```
$ mafft --auto att_sequences.fa > mafft_aln.fa
```

Not conserved regions from both sides of *att* motif were clipped using Jalview alignment editor [32], and a sequence logo was created using WebLogo 3 [33]. The found motif has appeared to be even longer (≈ 130-135 nt) than the original *att* sequence used for the BLAST. This is happening because the repetitions itself are not perfect, and, as a consequence, automatic repeat detection methods are not capable to find them in a precise way. Nevertheless, the presence of the same conservative *att* sites within the analysed *E. coli* strains may point to a universal integration mechanism of pipolins in *E. coli* genomes.

## 3.3 Scaffolding and Extraction of Pipolin Elements

After identification of piPolBs and terminal *att* repeats, we were able to extract pipolin elements out of the genomes. In the majority of cases, a whole pipolin flanked by *att*s was located on the chromosome or on a single contig. Those pipolins were extracted by cutting the sequence around *att*s, so that the tRNA gene used for element's integration was also included. For consistency, we referred to the *att*, overlapping with a tRNA gene, as attR and expected it always to be the rightmost *att*.

In some cases, *att* repeats and piPolB were located on different contigs, posing a challenge for us to analyse the structure of the elements. Therefore, we scaffolded the disrupted pipolin elements into a continuous sequence using a Python script, which was later included in our pipeline as a separate task (Subsection 4.2.2). Unfortunately, there were two ambiguous cases (for the strains LREC242 and LREC244) for which the scaffolding algorithm did not work. Those cases were resolved manually by visual comparison with other pipolins (Section 3.6).

## 3.4 Annotation of Pipolins

Scaffolded and extracted pipolin sequences were re-annotated by Prokka pipeline [34]. This pipeline allows using different databases for protein annotation, among those we have chosen Bacteria-specific UniProt (updated 16.10.2019), HAMAP (updated 16.10.2019) and Pfam-A (updated 08.2018). After the first try, 50% of pipolin open reading frames (ORFs) left unannotated and were classified as "hypothetical proteins".

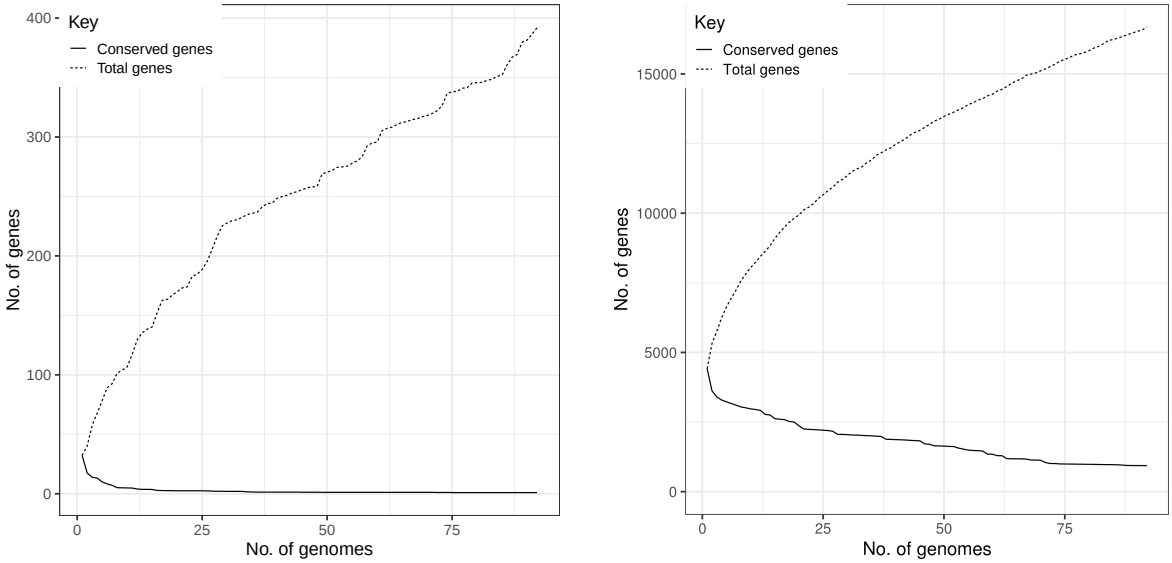
We attempted to improve the annotation using HHpred [35] for the most common pipolin ORFs, determined after the pan-genome analysis (Section 3.5). The found HHpred hits were considered as homologous to the gene of interest if 1) the estimated probability was $> 90\%$, 2) the E-value was $< 0.01$, 3) the secondary structure similarity was along the whole protein length, 4) there was a relationship among top hits, 5) only Bacteria, Archea, and Viruses were allowed as the sources of found hits. As a result, functions have been assigned to 6 more proteins (Table 3.1):

1) Uracil-DNA glycosylase (group_1)
2) Type I restriction modification system methyltransferase hsdM (hisF)
3) Metallohydrolase (group_16)
4) Type I site-specific deoxyribonuclease hsdR (group_5)
5) Excisionase (group_52)
6) Type I restriction modification enzyme hsdS (group_27)

A list of these proteins was provided to Prokka as a trusted set of pre-annotated proteins. After the second re-annotation, only $\approx 25\%$ of proteins left unclassified.

## 3.5 Pan-genome and Functional Analysis

Pan-genome analysis of pipolins gene content was carried out using Roary [36], resulting in a total of 392 genes. Remarkably, the core- and soft-core genomes are made up of a single gene cluster, the piPolB, and a XerC-like tyrosine-recombinase, respectively. In line with this, the shell genome contains only 40 genes, whereas 350 genes (89%) are cloud-genes, present in less than 15% of pipolins. Despite the great variety of different genes, some groups of pipolins share a similar gene composition and, as about 60% of genes are provided by about one-third of the pipolins (Figure 3.2a).




(a) (b)

Figure 3.2: Accumulation of conserved vs. total genes per genome in (a) pipolins pan-genome and (b) the whole pan-genome.

Shell-core genes present in more than 15% of pipolins have been analyzed by eggNog [37] and Blast KOALA [38] from KEGG orthology database. A detailed functional analysis of shell core genes is shown in Table 3.1. As mentioned above, besides piPolB, pipolins often include one or more XerC and IntS (bacteriophage-type) tyrosine recombinases. When two complete recombinase genes are present, one of them is always located next to an excisionase-like protein. A type-4 Uracil DNA glycosylase is also frequent. Other proteins with DNA binding domains like mobilization proteins as well as components of restriction-modification systems are also common.

In summary, a pipolin basic unit is composed of direct terminal repeats encompassing a piPolB gene and a variety of genes, with many of them related to the metabolism of nucleic acids.

Table 3.1: Functional characterization of the most common *E. coli* pipolin genes. Searches in eggNOG and KEGG databases, as well as HHPred search of remote homologs were performed. Genes related to nucleic acid metabolism, transposase and integrase activity are highlighted.

| **Gene** | **#Pipolins** | **Annotations** | **HHpred UniProt best hit** | **eggNOG Description** | **KEGG Description** |
|---|---|---|---|---|---|
| pipolB | 92 (100%) | Primer-independent DNA polymerase PolB | DNA polymerase (P03680) | | |
| xerC_2 | 90 (98%) | Tyrosine recombinase XerC/XerD; Prophage integrase IntS; Arm DNA-binding domain | Tyrosine recombinase XerD (P0A8P8) | Belongs to the 'phage' integrase family | |
| group_1 | 87 (95%) | Uracil-DNA glycosylase | Type-4 uracil-DNA glycosylase (Q96YD0) | | |
| group_6 | 86 (93%) | hypothetical protein | | | |
| xerC_1 | 84 (85%) | Tyrosine recombinase XerC | Integrase (P03700) | Belongs to the 'phage' integrase family | |
| hisF | 78 (83%) | Type I restriction modification system methyltransferase (hsdM); Imidazole glycerol phosphate synthase subunit HisF | hsdM1 (Q5M500) | HsdM N-terminal domain | type I restriction enzyme M protein(K03427) |
| group_16 | 76 (82%) | metallohydrolase | Uncharacterized protein (Q57587) | Metal-dependent hydrolase | uncharacterized protein (K07043) |
| group_18 | 75 (82%) | hypothetical protein | | | |
| group_5 | 75 (82%) | Type I site-specific deoxyribonuclease (hsdR) | hsdR (P10486) | Type I restriction enzyme R protein N terminus (HSDR_N) | type I restriction enzyme, R subunit (K01153) |
| group_10 | 75 (82%) | Protein of unknown function (DUF2787) | | Protein of unknown function (DUF2787) | |
| group_13 | 74 (80%) | hypothetical protein | | Protein of unknown function (DUF726) | |
| group_11 | 73 (79%) | hypothetical protein | | | |
| group_24 | 73 (79%) | hypothetical protein | | | |
| group_52 | 73 (79%) | Excisionase | Putative excisionase (A6T888) | | |
| group_3 | 64 (70%) | hypothetical protein | | | |
| group_8 | 59 (64%) | hypothetical protein | | | |
| group_19 | 56 (61%) | WYL domain | Uncharacterized protein (A0A4Y3NDN0) | transcriptional regulator | |
| group_58 | 53 (58%) | hypothetical protein | | | |

Continued on next page

Table 3.1 Continued from previous page

| Gene | #Pipolins | Annotations | HHpred UniProt best hit | eggNOG Description | KEGG Description |
|---|---|---|---|---|---|
| group_28 | 41 (45%) | Uncharacterized protein family (UPF0149) | Protein translocase subunit SecA (P28366) | Uncharacterised protein family (UPF0149) | uncharacterized protein (K07039) |
| group_17 | 38 (41%) | IS1 family transposase IS1A/IS1D | | | |
| group_23 | 31 (34%) | PD-(D/E)XK nuclease superfamily | | PD-(D/E)XK nuclease superfamily | |
| group_31 | 30 (33%) | Restriction endonuclease | Restriction endonuclease (A0A0J9X157) | Restriction endonuclease | |
| group_15 | 30 (33%) | IS1 family transposase IS1X2/IS1R | | | insertion element IS1 protein InsB (K07480) |
| group_9 | 29 (32%) | Protein of unknown function (DUF4011) | | Protein of unknown function (DUF4011) | |
| group_34 | 24 (26%) | hypothetical protein | | type I restriction enzyme, R | |
| group_53 | 23 (25%) | Protein of unknown function DUF262/DUF1524 | | Protein of unknown function (DUF1524) | |
| group_14 | 22 (24%) | Uncharacterized protein family (UPF0149) | | | |
| group_20 | 22 (24%) | WYL-domain containing protein | | | |
| group_39 | 22 (24%) | Protein of unknown function DUF262 | | Protein of unknown function (DUF1524) | |
| group_40 | 22 (24%) | hypothetical protein | | | |
| group_25 | 22 (24%) | Protein of unknown function DUF262/DUF1524 | | Protein of unknown function (DUF1524) | |
| group_55 | 21 (23%) | Protein of unknown function DUF262 | | Protein of unknown function DUF262 | |
| group_32 | 19 (21%) | hypothetical protein | | | |
| group_27 | 16 (17%) | Type I restriction modification enzyme | HsdS (Q8R9Q6) | Type I restriction modification DNA specificity domain | type I restriction enzyme, S subunit (K01154) |
| group_29 | 16 (17%) | IS3 family transposase ISEam1 | | Transposase | transposase (K07483) |
| insK | 16 (16%) | IS3 family transposase ISEc14; Putative transposase InsK | | | |

Concluded

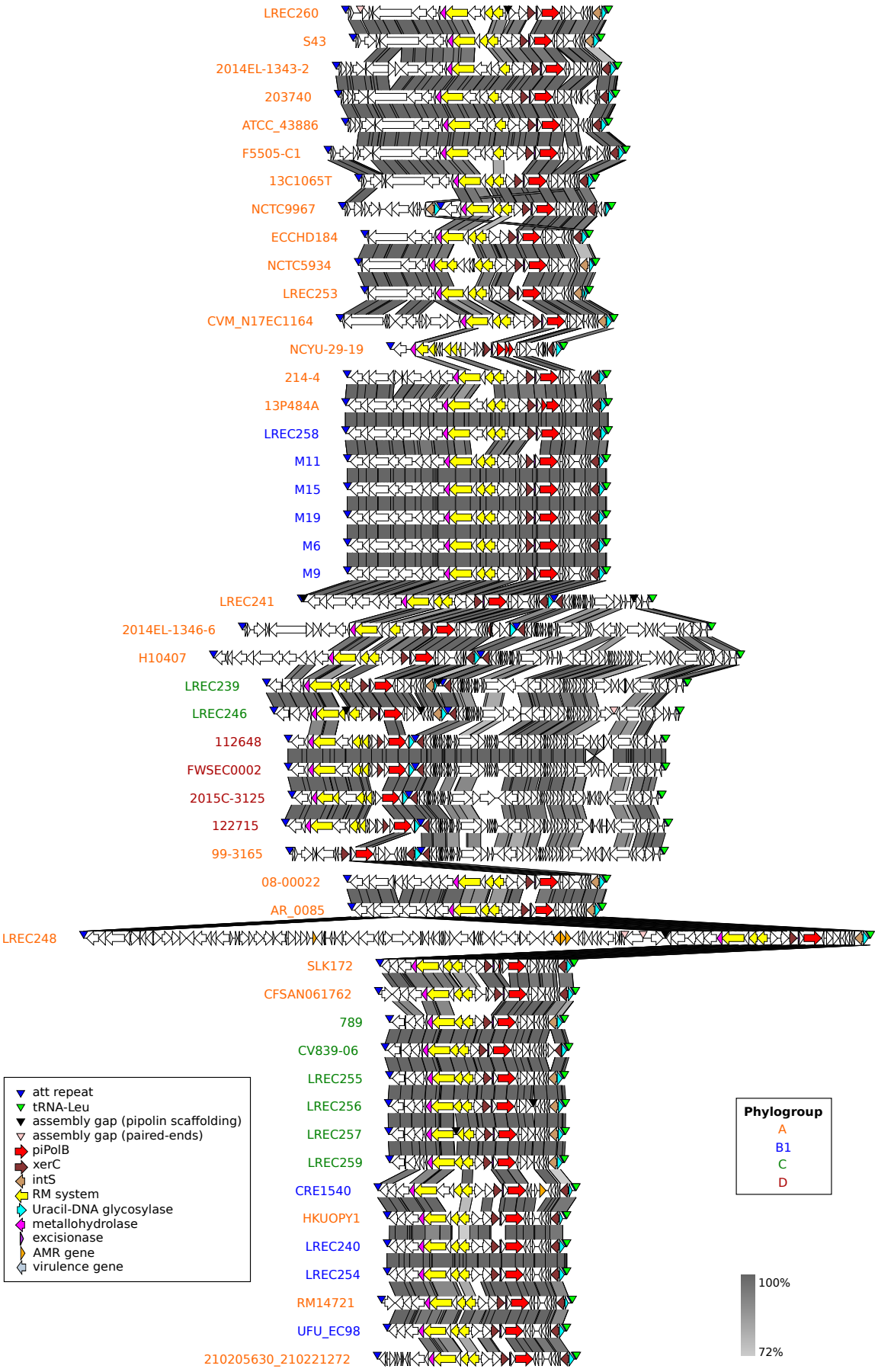
## 3.6 Comparative Analysis

Comparative representation of the genetic structure of pipolins was generated by Easyfig [39] (Figure 3.3).

All *E. coli* pipolins are integrated into the same point, at the Leu-tRNA gene, except for the pipolin from LREC252 strain that looks inconsistent with other pipolins. In some genomes, three *att* sequences were detected, as those pipolins seem to share the integration site and mechanism with some prophage, as was previously detected for the enterotoxigenic *Escherichia coli* H10407 strain [2]. Indeed, the genetic structure comparison of all pipolins confirmed that a similar Myovirus enterophage is present next to pipolins from eight strains, spanning phylogroups A (H10407, 2014EL1346-6 and 99-3165), C (LREC239 and LREC246) and D (112648, 122715, 2015C3125 and FWSEC0002) (Section 3.7). In addition, the presence of transposases and associated genes indicates that genetic islands and insertion sequences can as well contribute to the variability of pipolins, particularly in the case of the stains LREC248 and LREC252, expanding also the pipolin gene repertoire (Section 3.5).

Although a certain level of synteny and modular organization can be detected, genetic rearrangements, including inversions, duplications, and deletions, which often lead to gene exchange, are also frequent, as well as truncations and disruptions. Even truncated forms of piPolBs or XerC-like recombinases can be detected, which might lead to the impairment of replication or mobilization of pipolins.

Overall, the genetic repertoire and structure of analyzed pipolins suggest that they can exchange genetic information among *E. coli* strains.


LREC260
S43
2014EL-1343-2
203740
ATCC_43886
F5505-C1
13C1065T
NCTC9967
ECCHD184
NCTC5934
LREC253
CVM_N17EC1164
NCYU-29-19
214-4
13P484A
LREC258
M11
M15
M19
M6
M9
LREC241
2014EL-1346-6
H10407
LREC239
LREC246
112648
FWSEC0002
2015C-3125
122715
99-3165
08-00022
AR_0085
LREC248
SLK172
CFSAN061762
789
CV839-06
LREC255
LREC256
LREC257
LREC259
CRE1540
HKUOPY1
LREC240
LREC254
RM14721
UFU_EC98
210205630_210221272
att repeat
tRNA-Leu
assembly gap (pipolin scaffolding)
assembly gap (paired-ends)
piPolB
xerC
intS
RM system
Uracil-DNA glycosylase
metallohydrolase
excisionase
AMR gene
virulence gene
Phylogroup
A
B1
C
D
100%
72%

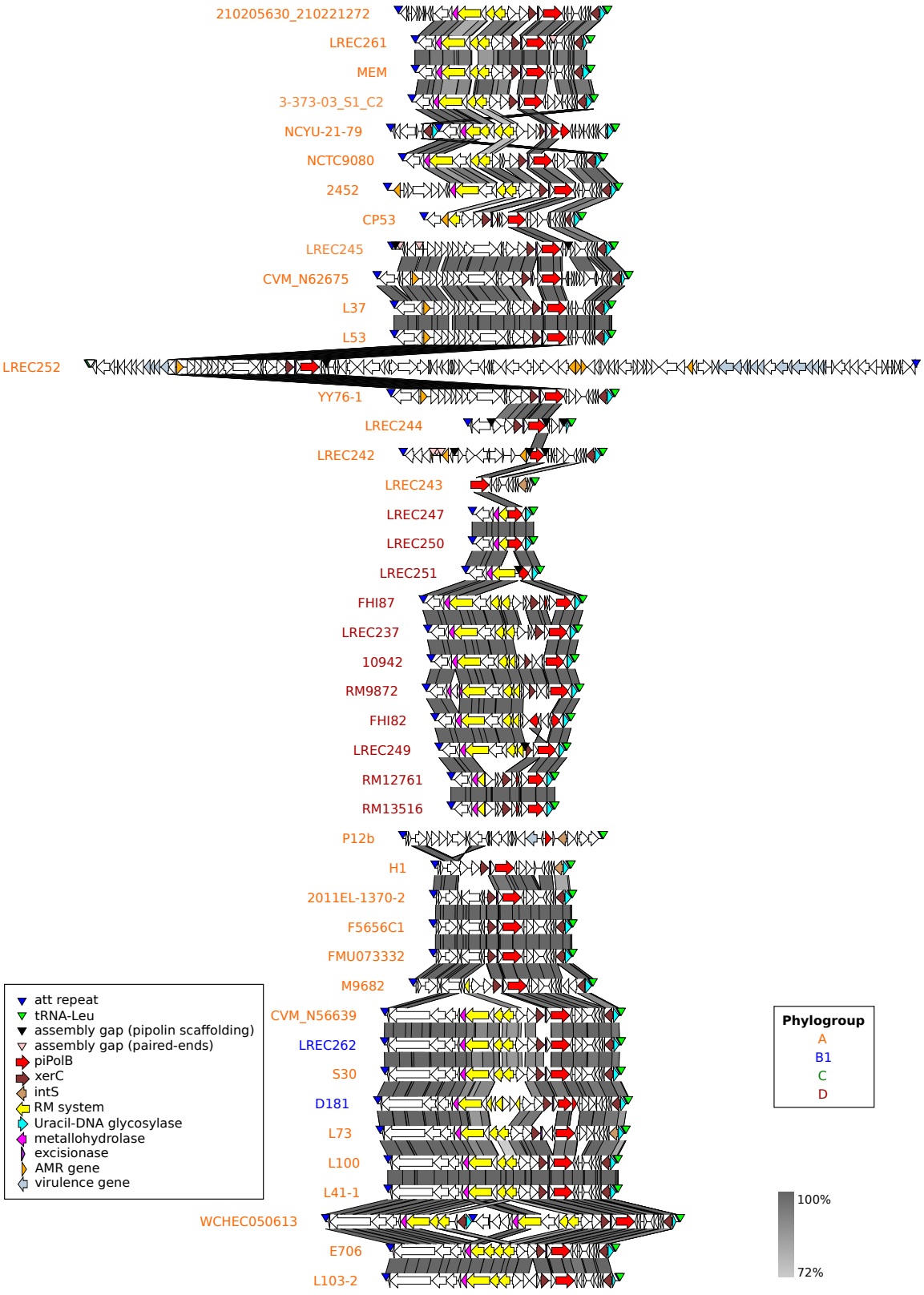



Figure 3.3: Genetic structure of *E. coli* pipolins. Protein-coding genes are represented by arrows, indicating the direction of transcription, and colored as shown on the legend. Re-annotated pipolins sorted according to the hierarchical clustering of the gene presence/absence matrix. The greyscale on the right reflects the percent of amino acid identity between pairs of sequences. Names of pipolin-carrying strains are colored based on the phylogroups.

## 3.7 Diversity of the Selected Strains

Coding DNA sequences (CDSs) predicted by Prokka were analyzed using ABRicate [40] for the presence of antibiotic resistance (ResFinder V2.1.) and virulence genes (VirulenceFinder v1.5), and identification of clonotypes (CHTyper 1.0), sequence types (MLST 2.0) and serotypes (SerotypeFinder 2.0). Phylogroups were predicted using the ClermonTyping online tool [41].

We found that pipolins were present in strains of phylogroup A (57 strains), but also in B1 (12 strains), C (8 strains) and D (15 strains), with similar distribution patterns within NCBI and LREC collections (Figure 3.4). The common presence of *E. coli* strains from phylogroup A in the dataset was somewhat expected, as this phylogroup is the most common among human isolates and thus very abundant in most collections [42, 43]. However, we were surprised by the absence of pipolins among B2 strains, despite the fact that this phylogroup is also very common in the LREC collection and, along with group D, it is responsible for most extraintestinal *E. coli* in human and animals [44]. Nevertheless, phylogroups A, B1, C and D have been proposed to belong to different ancient lineages [45], downplaying a strict vertical transmission of pipolins throughout the evolutionary diversification of *E. coli*.

Regarding multilocus sequence typing (MLST), clonotyping and serotyping, we have also observed their great variety among analysed strains (Figure 3.4). In general, strains from phylogroups A and B1 are more diverse than strains from phylogroups C and D, with clear clonality between some strains.

This diversity is even more evident when the pan-genome is analyzed, with a total of 16,675 genes, only 934 genes comprise a core-genome and more than two-thirds of the genes in the cloud-genome (11,175, 67%). As such, the number of total genes associated with the cloud gene set increased consistently with the number of genomes (Figure 3.2b).

In conclusion, notwithstanding the clonality of several strains, the analysis indicates that pipolins are present in a wide variety of *E. coli* strains.

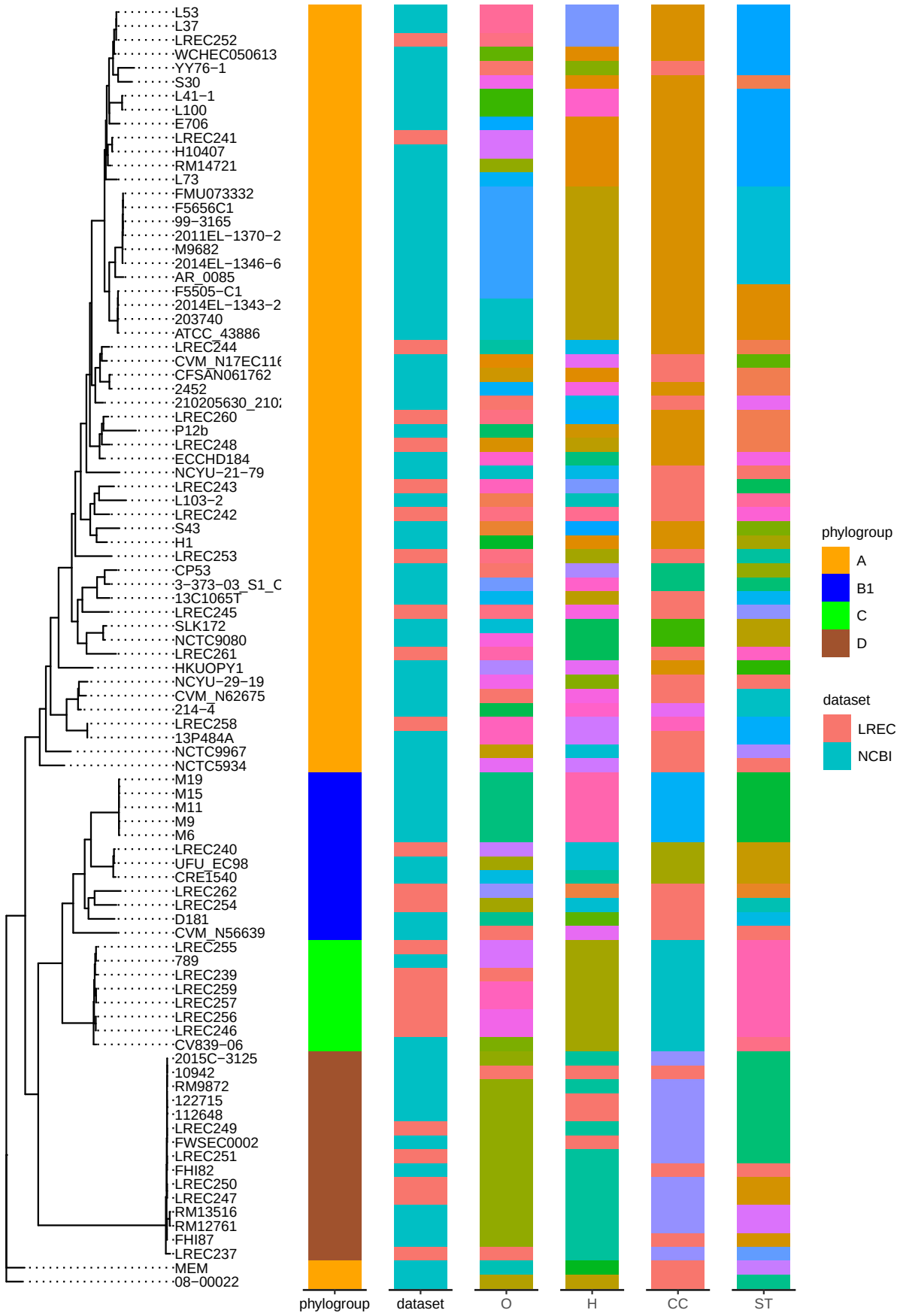



Figure 3.4: Phylogeny of pipolin-harboring *E. coli* strains along with the associated data: strain phylogroup, dataset origin, serotype (O- and H-types), clonotype (CC) and sequence type (ST).

## 3.8 Cophylogeny of Pipolins and Host Strains

The alignment of concatenated genes from the core-genome was used for the phylogeny of host strains. Phylogeny of piPolB gene was generated independently. The best-fit maximum likelihood-phylogenetic tree was built using IQ-TREE [46] upon PRANK codon aware alignment [47]. The obtained trees were then used for the comparative phylogenetic analyses with RStudio [48]. Briefly, phylogenetic trees were handled and visualized using ggtree [49] and tanglegrams for visual tree comparison were generated with Phytools [50]. We used the Dendextend package [51] to calculate the cophenetic correlation coefficient (CCC) between trees.

Since the presence of the piPolB gene is the hallmark of pipolins and it constitutes the only core gene (Section 3.5), we performed a phylogenetic analysis of the piPolB sequences from the pipolin-harboring *E. coli* strains. Although some of the annotated piPolB genes are partially truncated, particularly those from pipolins in phylogroup D strains, they have a high degree of identity, above 98.8% in the aligned regions. Phylogeny of the piPolBs underlined again the similarity among pipolins in clonal strains that belong to phylogroups C and D, but sequences from phylogroups B1 and A were mixed together (Figure 3.5).

The tanglegram in Figure 3.5 allows us to visualize the cophylogeny between piPolBs and *E. coli* strains carrying pipolins. This plot reveals a complex association pattern, with numerous crisscrossing lines that suggest incongruence between the two phylogenies. We have calculated the CCC among trees as indicative of phylogenies clustering congruence, and, in line with the figure, the CCC value is quite low, 0.21.

Overall, we can conclude that the pipolins diversity is poorly congruent with the strains phylogeny and their distribution is rather indicative of a patchy distribution amongst a wide variety of pathogenic *E. coli* strains, as expected for horizontally transferred MGEs. This pattern may reflect the wide distribution of pipolins beyond *E. coli*, dispersed among major bacterial phyla, namely Actinobacteria, Firmicutes, and Proteobacteria, as well as in mitochondria [2].

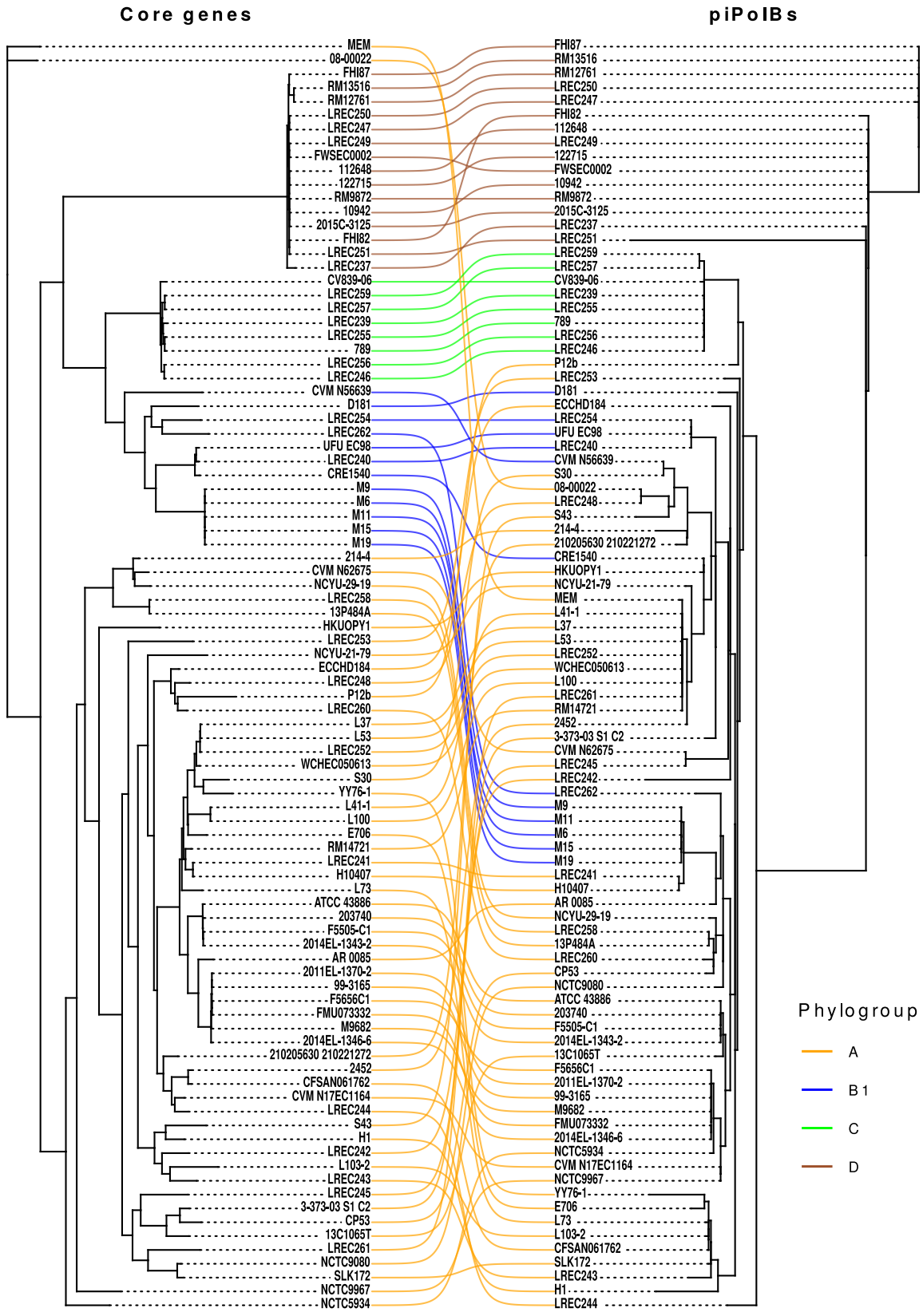



Figure 3.5: Tanglegram representation of maximum-likelihood phylogenies, constructed from the host strains core genome alignment and piPolB gene alignment. Links between pipolins and *E. coli* strains are colored based on the phylogroup.

# 4

# Pipeline Design and Implementation

## 4.1 Pipeline Design

### 4.1.1 The Main Functionality of the Pipeline

ExplorePipolin pipeline is intended to check automatically whether a given bacterial genome assembly contains pipolins, although the current version (v.0.0.a1) is designed and tested for *E. coli* pipolins. When a pipolin is detected within the analysed genome, the pipeline automatically extracts the pipolin sequence, annotates its genes, as well as searches for terminal direct repeats (*att* sites) and includes them into the annotation. The pipeline outputs results in the most common formats, like Genbank and GFF, ready for downstream analysis and/or visualization.

During the prior analysis of pipolins from *E. coli* genomes (Chapter 3), the following important steps were highlighted:

1. The presence of pipolin hallmark features need to be identified:
   (a) The presence of piPolB gene need to be predicted carefully.
   (b) The presence of *att* direct repeats can be predicted in two ways: first, it is possible to check whether repeats with similarity to the known *att* sites are present; second, *de novo* search of *att* direct repeats could be performed.
   (c) tRNAs and tmRNAs provide a valuable information for identification of *att* sites, so they need to be predicted before/during the step of *de novo* search of *att*s.
2. Pipolin features sometimes spread among several contigs, and in this case, we need to analyse whether the pipolin could be scaffolded into one continuous fragment. Scaffolding may ease pipolins structure visualization, as well as a comparison of pipolins from different strains.
3. After the detection of hallmark features, the pipolin element boundaries need to be defined and the sequence(s) corresponding to the pipolin, need to be extracted.
4. Pipolin genes need to be thoroughly and homogeneously annotated.

5. The information about found *att* repeats need to be included in the annotation separately, as gene prediction and annotation tools usually do not handle any repeats.

6. To ease subsequent visualization of pipolins using Easyfig, colouring scheme for some pipolin genes, *att* repeats and assembly gaps can be added.

It is worth noting here that we intend to use the pipeline for a broader range of bacteria in the future, therefore it is important the pipeline maintain flexibility so that its functionality could be easily modified and extended (see Subsection 4.1.4).

### 4.1.2 Functionality Provided by Existing Software

Some steps in the pipeline can be delegated to existing programs. First, prediction of the piPolB gene can be done by BLAST search, using a "reference" piPolB sequence as a query. During the prior analysis, we found that piPolB sequences from *E. coli* genomes are highly conservative, and at first glance, the nucleotide BLAST might seem a sufficient choice to predict piPolBs. But definitely, our sampling is not representative: according to the previous study, we should expect a greater variability of piPolB genes when screening genomes from other bacterial species [2]. As such, protein-protein BLAST should be preferred as more accurate and more sensitive, for the reason that protein sequences are evolutionary more conserved than nucleotide ones. Another solution would be to use HMMER, which respects protein domain structure and therefore allows more accurate identification of remote homologs. However, a multiple sequence alignment of the protein sequence family is required to build the Hidden Markov Model (HMM) profile, while the quality of the alignment and the number and diversity of the sequences it contains are crucial for the subsequent search. Thus, this method can be used in the future, when a big set of piPolB genes from different bacterial phyla is accumulated.

As for *att* terminal direct repeats, they have been shown to be highly conservative in *E. coli* pipolins (Section 3.2), so that the nucleotide BLAST can be used for their prediction. However, *att* sites from other bacterial species might totally unrelated to the *att* motif present in *E. coli* pipolins. In this case, models based on generalized or HMM profiles can be used, as well as other tools for detection of integration sites, like MGEfinder [24] or Mauve [52]. But again, at the current moment, we do not have enough information about the variability of *att* sites to apply other approaches.

As was already mentioned (Chapter 2 and Section 3.2), the knowledge of tRNAs/tmRNAs locations in the genome can help to predict some of the *att* sites. The most commonly used programs to detect tRNAs/tmRNAs are ARAGORN [53] and tRNAscan-SE [54]. Both are effective in tRNA/tmRNA search, but ARAGORN has been shown to work faster, while tRNAscan-SE – to be more sensitive. For our pipeline, we have decided first to implement the search with ARAGORN, which is more appropriate because it is already included in the dependencies list of Prokka (see below).

For the prediction and annotation of pipolin genes, we have chosen Prokka pipeline [34], which is a comprehensive tool annotating many prokaryotic genome features: protein-coding genes, tRNAs, rRNAs, non-coding RNAs and others. Prokka has other advantages compared to the frequently used tools like NCBI's PGAP [55] and RAST [56]: 1) it is a command-line tool, which facilitates its integration into other software; 2) it uses prokaryotic specific databases, which drastically reduces the amount of the required disk space; 3) it is possible to provide to Prokka your own additional/modified databases or just a set of pre-annotated proteins of good quality, which helps to improve annotation and makes gene naming consistent. Adding a custom list of proteins focused on pipolin products was used by us to refine the annotations and reduce the number of hypothetical proteins (Section 3.4).

### 4.1.3 Functionality That Needs to Be Programmed

There are steps in the pipeline that need to be programmed "from scratch" as there are no tools that provide the required functionality and, at the same time, do not overburden the pipeline with unnecessary complexity. Some of these steps might be considered as straightforward, like extraction of pipolin sequences into separate FASTA files, introducing of *att* sites features into the annotation files (GenBank and GFF), and others, not mentioned directly, like parsing BLAST, ARAGORN, HMMER output files, introducing of assembly gap features, various sequence manipulations. These steps usually do not require serious reflection, while the presence of convenient task-specific libraries inside the programming language might help considerably.

A couple of complex steps need to be programmed from the ground up: 1) *de novo* search of *att* direct repeats, and 2) scaffolding of pipolin fragments into a single continuous sequence.

***De novo* search of *att* sites** Though *att* sites from the analysed *E. coli* strains are appeared to be conserved (Section 3.2), we have very little information about *att*s from pipolins of other bacteria. Theoretically, they may possess different *att*s, depending on the type of encoded integrase gene(s). Even in some *E. coli* pipolins, more than one XerC-like recombinases and sometimes *IntS* recombinase were detected (Section 3.4), suggesting that pipolins may use different integration sites within bacterial chromosome. Knowing the *att* repeats location helps to precisely define pipolin boundaries, though there are always no guaranties that element boundaries have remained intact. In addition, there are few examples of circular, plasmid-like pipolins in Enterobacter, Staphylococcus and Lactobacillus, which do not have *att* sites. Characterization of plasmid-like pipolins poses additional challenges that might be considered in future versions of the pipeline. Nevertheless, *de novo* search of *att* direct repeats should be implemented in order to catch not-known *att*s.

It was discussed in Chapter 2 that many tools for identification of MGEs include the step of flanking repeats analysis. For most of the tools, we do not know how exactly they perform this task since this step is usually purely documented or not documented at all. At first glance, we have a problem of finding identical non-overlapping substrings (sequences) in a long string (genome) and this problem can be easily solved. However, the overlapping region of recombining sites with strict homology might be as short as 6 bp for some of the integrases. Moreover, it was shown previously that, for some tyrosine-recombinase family proteins, strict homology between integration sites is not required [57]. So, even if we scan a short sequence of 100 kb in length for the presence of 6-bp repeats, we would likely find many of them, not to mention the situation when we would allow not strict identity.

Several tools are available for the identification of different types of repeated sequences. We were not paying attention to "knowledge-based" tools that use consensus sequence databases to search for repeats, particularly since most of them are eukaryote-specific [58]. Then, there are also "signature-based" tool that are restricted to certain types of elements, like for CRISPRs (Clustered Regularly Interspaced Palindromic Repeats) in prokaryotes [59], LTR-RTs (Long Terminal Repeat retrotransposons) in plants [60] and so on. Though, at the current moment, we should accept the fact that we do not know in detail signatures of pipolin-specific terminal repeats, apart from those found in *E. coli* genomes.

As an example of more general approaches, we might look at PHASTER web tool [9], which performs a search of *att* sites the following way. For each integrase in a cluster of prophage genes, the cluster boundaries are scanned for potential *att* sites that are identified as short nucleotide repeats (12-80 bases). At the same time, tRNA and tmRNAs are found, and repeats that do not overlap with tRNA or tmRNA gene are filtered out. We have chosen first to implement a similar approach in our pipeline, with the only difference that we will look for repeats in regions,

surrounding piPolB gene, the only hallmark of pipolins.

To find repeats, two kinds of methods are commonly used: based on the suffix tree or alignment matrix [59]. When using alignment matrices, the programs usually start by comparing a genome against itself and identifying local alignments between different regions of the genome (which can be also visualised by genomic dot plots). After that, they classify the found repeats using different approaches. With this in mind, it seems reasonable to try using the BLAST tool for the purpose of searching for local alignments (repeat candidates).

BLAST algorithm starts with so-called word matching, and the minimum word size allowed for nucleotide BLAST is 4, which would allow us to find such short repeats. One way would be to extract a region of some length that surrounds a piPolB gene and align it to itself. Though another more meaningful approach would be to extract a region of some length upstream of the piPolB and a region of the same length downstream it, and create a pairwise alignment. However, we are only interested in direct repeats, and fortunately, this can be specified by `-strand` parameter. The minimum percent identity can be regulated as well to allow non-perfect matches. The exact BLAST command for finding repeats is shown in Subsection 4.2.2.

Repeats found in this way can be further filtered by some criteria or saved into a file for future analysis.

**Scaffolding of pipolins.** While analysing the *E. coli* pipolins, we have encountered a problem when different pipolin features (piPolB and *att*s) were located on different contigs. This disconnection makes further analysis of the pipolins problematic. Taking in mind the expected pipolin structure (Figure 3.1), we have decided to scaffold pipolin fragment into a single continuous sequence by introducing the "assembly_gap" features of unknown length between contigs (DDBJ/ENA/GenBank Feature Table Definition, Version 10.9 November 2019). The term scaffolding is referred to the process when two contiguous sequences are linked together by gaps, while evidence of their adjacency comes from paired-end or mate-pair sequencing, long reads, linkage data, *etc* [61]. In our case, we know that *att*s are headed in the same direction as they are direct repeats and that one of them could overlap with a tRNA gene on the opposite strand. In addition, we might expect piPolB to have a certain direction related to the tRNA gene (Figure 3.3). The details about this step implementation can be found in Subsection 4.2.2).

### 4.1.4 Programming Language and Libraries Choice

We decided to use Python [62] to program the pipeline, as it is a simple but powerful programming language, with a comprehensive standard library, rich IDE (integrated development environment) support, and very active community. Moreover, we made an extensive use of the Biopython [63] library that comes with many useful tools for computational biology. Since GFF files parsing is not integrated into Biopython, `bcbio-gff` [64] library was used for this purpose. For creating a convenient command-line interface, Click [65] library has been chosen.

The resulting pipeline is a multi-step procedure, and, as such, it should have the following properties: 1) each step can be easily modified, as well as new steps can be easily added; 2) logging facilities to see the progress of the pipeline and to have a record of events for every pipeline run; 3) seamless support for data-parallel computation (for multiple genomes to be analysed at once); 4) automated step dependency tracking; 5) caching of step outputs to avoid unnecessary recomputing when possible.

To fulfill the above requirements we decided to use a workflow system that can simplify pipeline design and execution by providing convenient abstractions. Among a range of available libraries (Apache Airflow [66], Metaflow [67], Luigi [68], *etc.*), we have chosen Prefect [69] as it is well documented, lightweight, and has few dependencies.

## 4.2 Pipeline Implementation

### 4.2.1 Project Structure Overview (v.0.0.a1)

ExplorePipolin pipeline repository is available on GitHub: `https://github.com/liubovch/ExplorePipolin`. At the root of the repository the following file and folders can be found:

`./README.md` – user's project description (it can be found also in Appendix A)

`./LICENSE` – a full license text will be placed here;

`./setup.py` – a script describing how to build and install the package (Subsection 4.2.4);

`./docker/` – contains `Dockerfile` to build pipeline-containing Docker image (Subsection 4.2.5);

`./conda/` – contains two files to build Conda package and to create package-specific Conda environment (Subsection 4.2.6);

`./explore_pipolin/` – the package source code directory;

`./tests/` – the package unit tests directory.

The package source code directory `explore_pipolin` – the actual Python package – consists of several modules and sub-modules which divide the code by its functionality (Figure 4.1). For example, the module `explore_pipolin.flow` defines pipeline flow using Prefect library conventions (Subsection 4.2.3). The module `explore_pipolin.utilities` combines several general-purpose sub-modules: `io` sub-module contains functions to perform regular file operations, `external-tools` works with non-Python processes, and `logging` generates genome-specific log messages. The module `explore_pipolin.tasks_related` includes tasks-specific sub-modules, while the tasks themselves defined in the module `explore_pipolin.tasks` (Subsection 4.2.3). The pipeline entry-point script is defined in the module `explore_pipolin.main`.

### 4.2.2 Implementation Details

**Package classes**. For many purposes in the pipeline, we are using the Biopython library, but summing up, almost all these usages are related to reading and writing files of different formats common in bioinformatics. Even though, Biopython `SeqRecord` and `SeqFeature` classes allow to create and modify sequence annotation objects, it was not convenient to operate them for several reasons: 1) they are overwhelmed with attributes and methods, not required in our analysis, and 2) what is more important, they lack attributes and methods, not only specific for our analysis but even those that we were expecting to see. For example, to get features of a certain type from a `SeqRecord` object (features are stored in a list), we need to infer feature indices first (usually in a `for` loop), and to use them to get the features. Unfortunately, the `SeqRecord` class does not have a method to get all features of a certain type at once. Another example is the `FeatureLocation` class which stores feature's start and end positions and feature's strand. We were expecting to see in this class a method that checks if two features are overlapping but had not found any. For all those reasons, we have decided to code our own classes with the required attributes and methods.

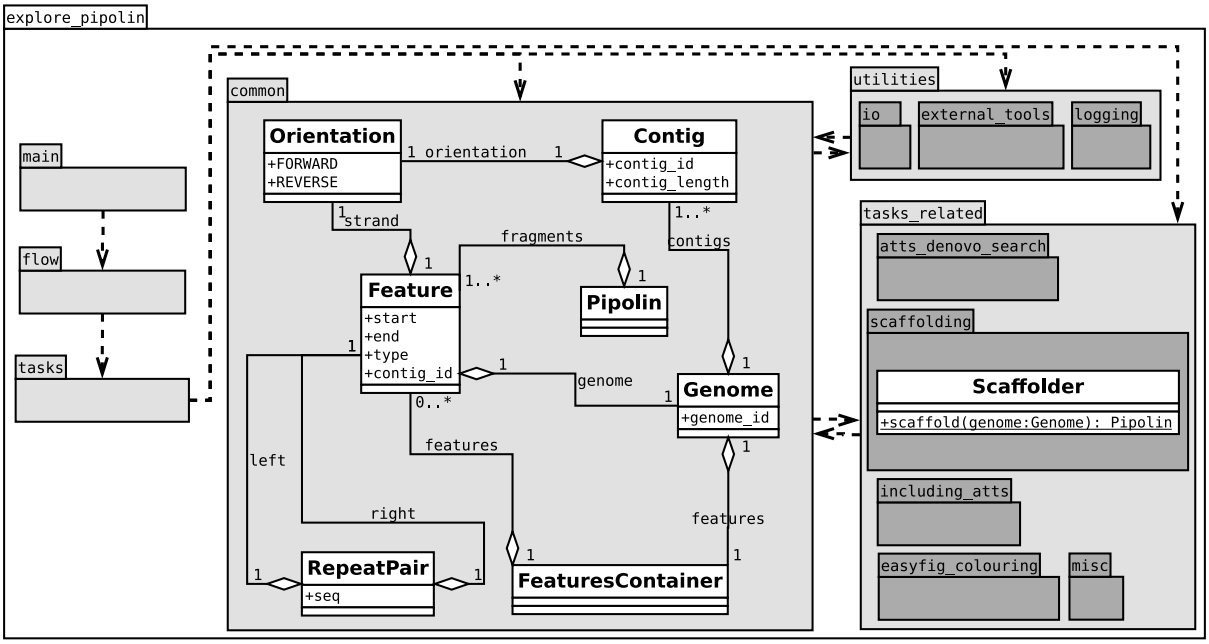



Figure 4.1: A diagram showing dependencies between package modules and classes. The package `explore_pipolin` consists of modules (light grey) and sub-modules (dark-grey). Dependencies between modules are indicated by dashed arrows, for example, the module `main` depends on the module `flow`. Dependencies between sub-modules are skipped for a clearer picture. Classes are shown as white rectangles inside modules and sub-modules. The first compartment of a rectangle shows the class name, the second – class attributes, the third – class methods (the only method is shown for the `Scaffolder` class). We have only aggregation ("has a") relationships between classes, indicated by a line with a diamond on its end. The relationships multiplicity and name are indicated as well. For example, class Contig is a part of class Genome, identified by the name `contigs`; a Genome instance might contain one or more Contig instances.

The main classes we have created to perform pipolin analysis are defined in the module `explore_pipolin.common` (Figure 4.1). Those are:

- `Orientation` – enumerates two possible orientations of a nucleotide sequence: `FORWARD` and `REVERSE` and implements methods to convert between different strand notations (for example, +1/-1 in BLAST XML or +/- in GFF3).
- `Contig` – represents a genome sequence. When we think of a bacterial genome, it is usually a single chromosome. However, when working with incomplete or not finished genomes, we would operate with contigs.
- `Feature` – an abstraction of basically any meaningful subsequence within a genome contig (or a chromosome). It also implements a method to check whether two features are overlapping.
- `FeaturesContainer` – the core class, which not only stores all features of our interest, but implements methods to query features by type, location and so on.
- `Pipolin` – is basically a container of features with type `PIPOLIN_FRAGMENT`. Compared to `FeaturesContainer`, in `Pipolin` object, `fragments` is immutable: once we have inferred the order of pipolin fragments (this is done by `Scaffolder` object), we store them in this order within `Pipolin` object.

- `RepeatPair` – stores two features of type `REPEAT` which constitute a pair of direct repeats or *att* repeat candidates.
- `Genome` – an abstraction of a genome file, provided to the pipeline as input. It aggregates almost all the aforementioned classes, directly or indirectly, to provide easy access to their attributes and methods.

**Running external tools.** To execute non-Python software within Python code, we used the `subprocess` module from the standard library. All external processes are defined in the module `explore_pipolin.utilities.external_tools`. Here we briefly discuss the processes and the additional parameters we have chosen for the pipeline (the corresponding shell commands will be shown).

To BLAST genome against the known piPolB amino acid sequence or *att* nucleotide sequence, the default `-evalue` of 10 was reduced to 0.01 to avoid low quality hits:

```
$ tblastn -query piPolB.fa -subject genome.fa -evalue 0.01 -outfmt 5

$ blastn -query att.fa -subject genome.fa -evalue 0.01 -outfmt 5
```

The command to find exact direct repeats of minimal length 6 is:

```
$ blastn -query upstream_piPolB_100000.fa -subject downstream_piPolB_100000.fa\
    -outfmt 5 -perc_identity 100 -word_size 6 -strand plus
```

, where `upstream_piPolB_100000.fa` and `downstream_piPolB_100000.fa` are subsequences that are upstream and downstream of piPolB gene. We have decided to align the subsequences of length 100 kb, as most of *E. coli* pipolins were shorter than that (Section 3.6), so it is unlikely to find *att* repeat outside of ±100 kb region around piPolB.

In all BLAST runs, the search results are stored in BLAST XML format (`-outfmt 5`), which is supported by `Bio.SearchIO.BlastIO` module from Biopython. Though BLAST XML format is harder to read by eye compared to tabular or plain text BLAST formats, it is more stable and complete and easier to parse automatically.

The command to define tRNAs/tmRNAs in a genome using ARAGORN is:

```
$ aragorn -w -o aragorn.batch genome.fa
```

By default, ARAGORN outputs not only positions and types of found tRNAs/tmRNAs, but also their two-dimensional structure. Here we used an option `-w` to have the results in a more convenient batch format. Since this format is a custom ARAGORN way to represent results, we wrote a function to parse it (can be found in `explore_pipolin.utilities.io` module).

The command to annotate pipolin sequences using Prokka is:

```
$ prokka --outdir outdir --prefix genome --locustag genome --rawproduct\
    --cdsrnaolap --rfam --proteins proteins.fa --force genome.fa
```

, where `--prefix` is output files prefix, which is set to the genome file name (to distinguish between different genomes), `--locustag` is a Locus tag prefix used inside annotation files, `--force` – allows annotations from different genomes to be saved in the same directory, and `--proteins` – allows to provide a list of pre-annotated genes, such as we used in Section 3.4. Other parameters are not mandatory for the pipeline and do not affect results significantly.

***De novo* search of *att* repeats**. At the current moment, we implemented this step only for complete genomes, although, it might be extended to incomplete genomes without changing the algorithm. In brief, we start this step by searching for direct repeats around piPolB gene. Then, we filter the found repeats to discard those that are overlapping with the known *att* site. In the

end, we filter the remaining repeats to leave those that are overlapping with a tRNA or tmRNA gene.

When we analysed piPolB-containing *E. coli* genomes in this way, we had not found any other *att* repeats apart from those that were overlapping with the known ones. This result is somewhat expected as it is unusual for a MGE to have two different integration sites. In most cases, this found *de novo att* repeats were only around 20-25 nt long, representing short regions of the perfect identity. Nevertheless, the approach might be helpful in the identification of other kinds of *att* repeats that might present in genomes of other bacterial species.

**Scaffolding of pipolins**. This step is done within a `Scaffolder` class object (Figure 4.1) in case when pipolin hallmark features (piPolB gene and *att* sites are localized on more than one contig. The `Scaffolder` object takes a `Genome` instance as its main attribute and invokes the `scaffold()` method, which in turn returns an instance of `Pipolin` class (ordered sequence of pipolin fragments).

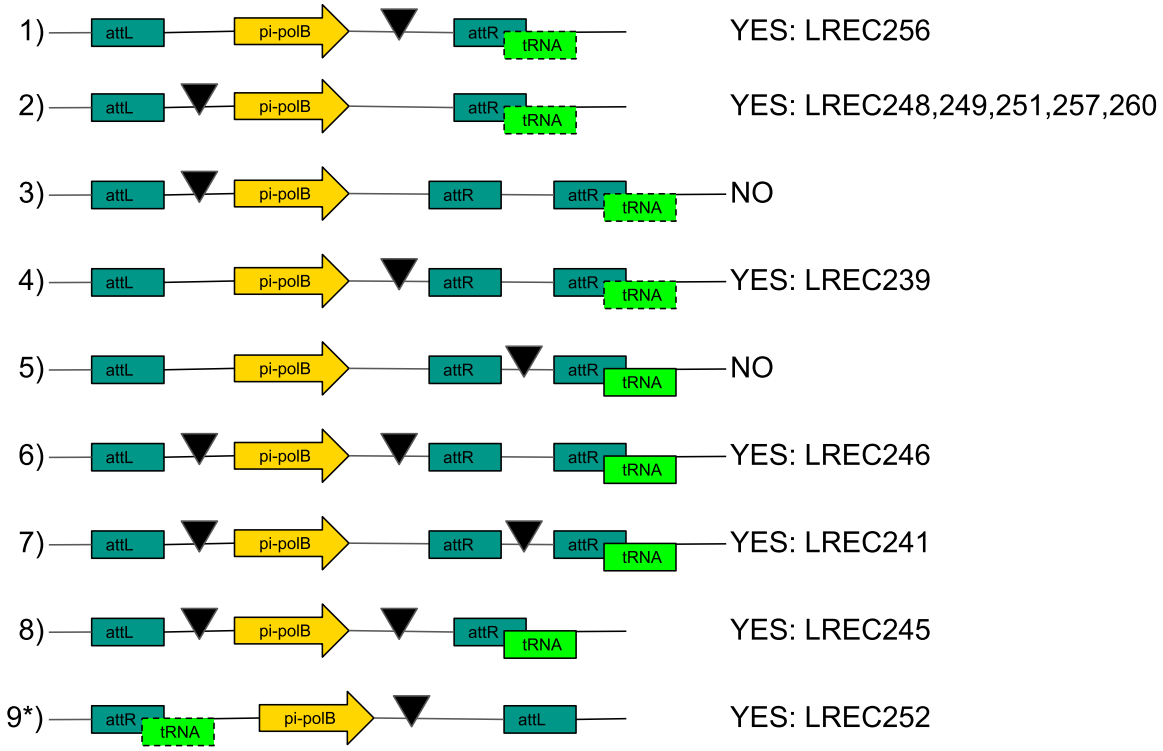



Figure 4.2: Combinations of pipolin fragments that can be scaffolded. Assembly gap is shown as a black triangle. tRNA feature with dashed border line indicates that its presence or absence is not required to infer pipolin fragments order. YES/NO on the right states whether the following combination is present among analysed pipolins and, when YES, the corresponding *E. coli* strains are also listed. The combination 9 (which actually comes from case 1) is a non-consistent case because the tRNA is appeared to be on the left side. The detailed structure of the following pipolins is shown on Figure 2.1.

There are eight unambiguous combinations of pipolin fragments that can be scaffolded by our algorithm (Figure 4.2). Those are coming from the following assumptions: 1) *att* repeats are headed in the same direction (on the scheme they are located on the plus strand as well as piPolB gene); 2) piPolB is sitting within *att* sites; 3) if one of *att* sites overlaps with tRNA/tmRNA gene on the opposite strand, this *att* site should be on the right side; 4) if piPolB gene is sitting alone on the contig, the contig direction is set so that piPolB is on the plus strand. It is worth to note that these assumptions do not guarantee the correct pipolin scaffolding, but for sure they will facilitate the subsequent analysis of putative pipolins. The scaffolding algorithm pseudocode is

shown as well (Algorithm 1), where CreateFragment function cuts a contig and rotates the obtained fragment, so that piPolB and *att*s are on the plus strand, and OrderFragments is concerned that piPolB sits within *att* sites.

**Algorithm 1:** Scaffolding of pipolin fragments into a single pipolin

```
Input: Genome class instance
Output: Pipolin class instance
begin
    PipolbAttContigs ← GetPipolbAttContigs()
    PipolbOnlyContigs ← GetPipolbOnlyContigs()
    AttOnlyContigs ← GetAttOnlyContigs()
    if Length(PipolbAttContigs) = 1 then
        if Length(AttOnlyContigs) = 1 then
            fragmentRight ← CreateFragment(PipolbAttContigs [0])
            fragmentLeft ← CreateFragment(AttOnlyContigs [0])
            Pipolin ← OrderFragments(fragmentRight, fragmentLeft)
            return Pipolin
        else
            error Cannot assemble!
        end
    else if Length(PipolbAttContigs) = 0 then
        if Length(PipolbOnlyContigs) = 1 then
            if Length(AttOnlyContigs) = 2 then
                fragmentRight ← CreateFragment(AttOnlyContigs [w/ tRNA])
                fragmentLeft ← CreateFragment(AttOnlyContigs [w/o tRNA])
                fragmentMiddle ← CreateFragment(PipolbOnlyContigs [0])
                Pipolin ← fragmentLeft, fragmentMiddle, fragmentRight
                return Pipolin
            else
                error Cannot assemble!
            end
        else
            error Cannot assemble!
        end
    else
        error Cannot assemble!
    end
end
```

After scaffolding of disrupted *E. coli* pipolins with this algorithm, we got only one case (for strain LREC252) looking totally inconsistent with the expected structure (Figure 4.2). Then, we also had two ambiguous cases (for strains LREC242 and LREC244), with two contigs containing piPolB gene (one of which was actually a short fragment less than 100 amino acids). We have resolved their order manually by visual comparison with other pipolins using Easyfig software (Figure 3.3).

**Testing the code.** To automatically test different components of the pipeline, we are using the `unittest` testing framework available from the standard Python library. Each package module tests are organized in a separate script, for example, `test_common.py` includes tests for the module `explore_pipolin.common`. In the current version of the pipeline, we do not have tests for all the modules, though, we are planning to extend the test coverage in the future.

### 4.2.3 Workflow Overview

The Prefect workflow management library that we used refers to each step in a pipeline as a **Task**. Tasks are basically Python functions that can receive inputs and produce outputs. However, apart from that, a task also produces metadata about its state which are then passed to downstream tasks. In the pipeline, we created tasks by decorating Python functions with the provided `@task` decorator:

```
# inside ./explore_pipolin/tasks.py
from prefect import task

@task
def create_genome_from_file(genome_file) -> Genome:
    ... # function body

@task()
def find_pipolbs(genome: Genome, out_dir):
    ... # function body
```

A task also can take arguments allowing its customization, for example, indicating whether it should run or not depending on the upstream task state, or whether it should cache its inputs/outputs, and so on. In most of the cases, we had no need to change task default parameters. Individual tasks have been combined into a **Flow** which is basically a script that illustrates the dependencies between tasks. Once a flow has been defined, it can be executed by calling `run()` method on it. To pass arguments (inputs) to a flow, special tasks called **Parameters** had to be defined:

```
# inside ./explore_pipolin/flow.py
from prefect import Flow, Parameter, unmapped
from explore_pipolin import tasks

def get_flow():
    with Flow('MAIN') as flow:
        genome_file = Parameter('genome_file')
        out_dir = Parameter('out_dir')

        genome = tasks.create_genome_from_file.map(genome_file)

        genome = tasks.find_pipolbs.map(genome=genome, out_dir=unmapped(
                                                out_dir))

        ... # other downstream tasks

    return flow

# inside ./explore_pipolin/main.py
get_flow().run(genome_file= ... , out_dir= ... )    # where genome_file
# can be an iterable object including several files
```

Data-parallel computation was achieved through the use of **Mapping** concept: when an iterable input is provided, a mapped task is copied for each element in input. This concept allowed us to specify several genome sequence files as input and to have independent parallel flows running for each genome sequence file. In the example above, the task `create_genome_from_file` was mapped through the `genome_file` argument, while the next task `run_blast_against_pipolb` was mapped through the results of the previous task. At the same time, we did not need to iterate over `ref_pipolb` and `out_dir` arguments, that is why they were passed to `unmapped` function from Prefect.

When running a flow, we can monitor the progress of pipeline execution by leveraging logging

mechanisms that are built in Prefect. There is also a possibility to extend Prefect's default logs by accessing the `logger` from the execution context:

```
    # inside ./explore_pipolin/tasks.py
    from prefect import task
    from prefect import context

    @task
    def are_pipolbs_present(genome: Genome):
        logger = context.get('logger')

        if len(genome.pipolbs) == 0:
            logger.warning('No piPolBs were found!')    # it will be logged!
            return False

        return True
```

By the default, Prefect logs are streamed to standard output and, for our pipeline, analysing two genome files, the standard output will look similar to this:

```
$ explore_pipolin --out-dir outdir genome_wo_pipolin.fa genome_w_pipolin.fa
[timestamp] INFO - prefect.FlowRunner | Beginning Flow run for 'MAIN'
[timestamp] INFO - prefect.FlowRunner | Starting flow run.

...    # logs of upstream tasks

[timestamp] INFO - prefect.TaskRunner | Task 'are_pipolbs_present[0]': Starting
    task run...
[timestamp] WARNING - prefect.are_pipolbs_present[0] | No piPolBs were found!
[timestamp] INFO - prefect.TaskRunner | Task 'are_pipolbs_present[0]': finished
    task run for task with final state: 'Success'
[timestamp] INFO - prefect.TaskRunner | Task 'are_pipolbs_present[1]': Starting
    task run...
[timestamp] INFO - prefect.TaskRunner | Task 'are_pipolbs_present[1]': finished
    task run for task with final state: 'Success'

...    # logs of downstream tasks

[timestamp] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks
   succeeded
```

From the log above we could see the warning message that was thrown for the first genome, not containing piPolB gene. Although Prefect default logging is quite useful, when running for several genomes, it starts to be too long. Not to mention that it is problematic to discriminate log messages thrown for a certain genome file from this output. For that reason, we have decided to implement genome-specific logging using the Python standard library. Details of its implementation can be found in the module `explore_pipolin.utilities.logging`. Below is an example of a log file for one of the strains:

```
$ cat outdir/LREC241.log
[timestamp] INFO: prefect.find_pipolbs[1] ( LREC241 ) starting...
[timestamp] INFO: prefect.find_pipolbs[1] ( LREC241 ) done
[timestamp] INFO: prefect.are_pipolbs_present[1] ( LREC241 ) starting...
[timestamp] INFO: prefect.are_pipolbs_present[1] ( LREC241 ) done
[timestamp] INFO: prefect.find_atts[0] ( LREC241 ) starting...
[timestamp] INFO: prefect.find_atts[0] ( LREC241 ) done
[timestamp] INFO: prefect.find_trnas[0] ( LREC241 ) starting...
[timestamp] INFO: prefect.find_trnas[0] ( LREC241 ) done
[timestamp] INFO: prefect.find_atts_denovo[0] ( LREC241 ) starting...
[timestamp] WARNING: prefect.find_atts_denovo[0] ( LREC241 ) This step is only
   for complete genomes. Skip...
[timestamp] INFO: prefect.are_atts_present[0] ( LREC241 ) starting...
```

```
[timestamp] INFO: prefect.are_atts_present[0] ( LREC241 ) done
[timestamp] INFO: prefect.analyse_pipolin_orientation[0] ( LREC241 ) starting
    ...
[timestamp] INFO: prefect.analyse_pipolin_orientation[0] ( LREC241 ) done
[timestamp] INFO: prefect.scaffold_pipolins[0] ( LREC241 ) starting...
[timestamp] WARNING: prefect.scaffold_pipolins[0] ( LREC241 ) >>> Scaffolding
    is required!
[timestamp] INFO: prefect.scaffold_pipolins[0] ( LREC241 ) done
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) starting...
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) @pipolin
    fragment length 365 from NODE_38
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) @pipolin
    fragment length 49199 from NODE_42
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) @pipolin
    fragment length 2708 from NODE_18
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) @@@pipolin
    record total length 52472
[timestamp] INFO: prefect.extract_pipolin_regions[0] ( LREC241 ) done
[timestamp] INFO: prefect.annotate_pipolins[0] ( LREC241 ) starting...
[timestamp] INFO: prefect.annotate_pipolins[0] ( LREC241 ) done
[timestamp] INFO: prefect.include_atts[0] ( LREC241 ) starting...
[timestamp] INFO: prefect.include_atts[0] ( LREC241 ) done
```

The aforementioned are the main Prefect concepts we have been using in our pipeline. A static representation of the pipeline's flow graph can be seen in Figure 4.3. Caching of pipeline outputs and parallel tasks execution are not implemented at the current moment. It requires around 30 minutes to analyse 92 *E. coli* genomes on a standard laptop with 4 GB of memory and 4 CPUs (when all CPUs are leveraged at the annotation step by Prokka).

### 4.2.4 Installation From Source

Since there are only Python libraries among the package build dependencies (Click, Biopython, bcbio-gff and Prefect), it can be easily built and installed using `pip` Python package installer (https://github.com/liubovch/ExplorePipolin#install-from-source). For this purpose, `setup.py` script is used, listing build dependencies, entry points, package data files and other package metadata.

Though, apart from build dependencies, the package needs additional run dependencies: command-line BLAST (BLAST+), ARAGORN and Prokka, – which might be difficult to install depending on the operating system you are using. We can note, however, that the pipeline will only work on Unix-like systems because Prokka run dependency has such a restriction.

To make the pipeline installation and running reproducible, Docker image and installation via Conda are provided (see below).

### 4.2.5 Deployment Using Docker

Docker is a tool that allows creating a so-called Docker image of the application. While creating an image, Docker is guided by instructions or commands written in a `Dockerfile`. Once an image is created, it is ready to be used by running applications installed in it.

A `Dockerfile` that is used to build an image with the ExplorePipolin pipeline can be found in the GitHub repository: https://github.com/liubovch/ExplorePipolin/blob/master/docker/Dockerfile. It is a simple `Dockerfile`, containing the following instructions:

1. The parent image is defined at the beginning of the file using command `FROM`. The image

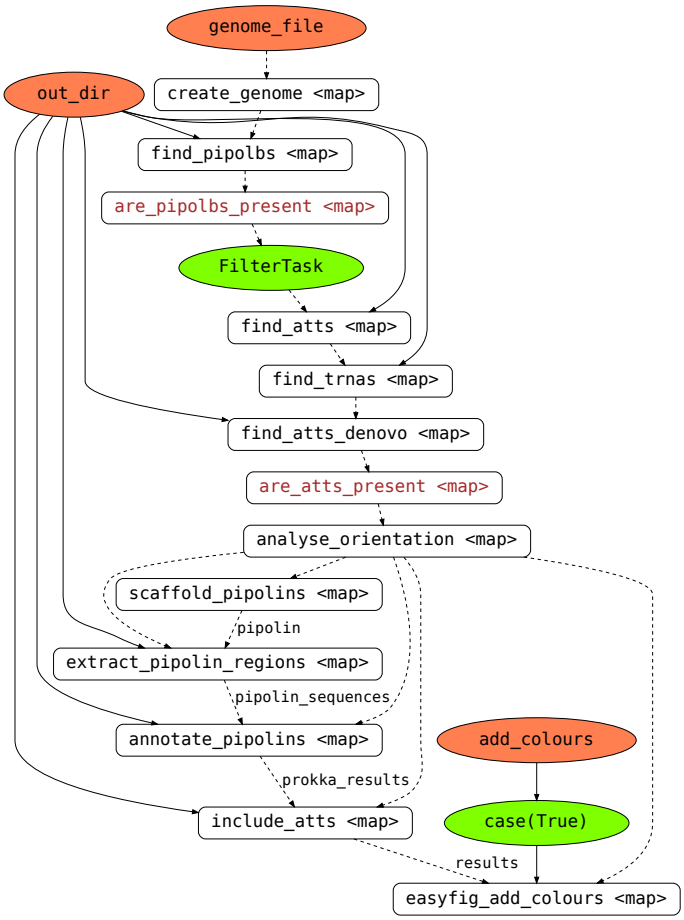



Figure 4.3: Static representation of ExplorePipolin's flow graph. The nodes with a coral background are Parameter tasks and labeled with the input name. The nodes with white background correspond to functional tasks and labeled with task names (brown font colour denotes internal check tasks). The nodes with a green background are special structures allowing filtering tasks by a condition (FilterTask) or create a conditional block in a flow (case). Edges represent task dependencies.

is based on the Debian 10 ("Buster") parent image available from Docker Hub (https://hub.docker.com/).

2. Several `RUN` instructions go after to run shell commands, the purpose of which is to install the pipeline dependencies.

    - We tried to consolidate several commands under a single `RUN` as it is recommended in the Docker Documentation (https://docs.docker.com/develop/dev-best-practices/) in order to reduce the final image size.

3. Prokka dependency is installed from the source as it is not available in Debian 10. The `ENV` instruction is used to add Prokka in the `PATH` environmental variable.

4. ExplorePipolin source code is downloaded from the repository and installed within an additional `RUN` instruction.

5. Finally, using `ENTRYPOINT` instruction, the container is configured to run ExplorePipolin's executable (`explore_pipolin`)

The `Dockerfile` needs to be placed in an empty directory. Since we had decided to upload the image to GitHub Packages (https://github.com/features/packages), the image was built with the following command:

```
$ sudo docker build -t\
    docker.pkg.github.com/OWNER/REPOSITORY/IMAGE_NAME:VERSION\
    path_to_directory_with_Dockerfile
```

, and uploaded to the GitHub Packages:

```
$ docker push docker.pkg.github.com/OWNER/REPOSITORY/IMAGE_NAME:VERSION
```

The final image occupies around 3 GB of disk space.

### 4.2.6 Deployment Using Conda

Another convenient way to use software that will guarantee the reproducibility of its installation is to use Conda environments. To build a Conda package, a build recipe need to be specified in a `meta.yaml` file. The corresponding `meta.yaml` to build the ExplorePipolin pipeline can be found in the GitHub repository (https://github.com/liubovch/ExplorePipolin/blob/master/conda/meta.yaml). It contains the following sections:

- `package` – contains the package name and version (the only mandatory fields);
- `source` – specifies that the source code of the package is coming from the latest release at GitHub;
- `build` – indicates that the pipeline is installed using `pip` Python package installer;
- `requirements` – lists build and run requirements;
- `test` – tells how the package can be tested;
- `about` – contains the information about the package: homepage URL pointing to the GitHub repository, licence name and licence file location.

The package can be built and installed into a separate environment with the following commands:

```
$ conda-build path_to_directory_with_metayaml -c bioconda -c conda-forge\
    --no-anaconda-upload
$ conda create -n env_name explore-pipolin -c local -c bioconda -c conda-forge
```

Now we can share the project environment across operating systems by exporting the environment specifications into a separate so-called "environment.yml" file:

```
$ conda env export -n env_name -f explore_pipolin-0.0.1.yml
```

, and providing it for a user at the GitHub repository along with the built package (see https://github.com/liubovch/ExplorePipolin#install-using-conda).

# 5

# Conclusions and Future Directions

In the current work, we have identified and characterised pipolin elements from 92 *E. coli* genomes, 25 of which were identified from the Spanish *E. coli* Reference Laboratory (LREC) collection and 67 – retrieved from the NCBI database. Pipolins from the *E. coli* genomes have been shown to present in a wide range of strains from different phylogroups, serotypes and clonotypes and to be highly diverse in their genetic structure and composition. Despite their great variability, the pipolin elements are flanked by conserved *att*-like terminal direct repeats and integrated into the same tRNA gene. Cophylogeny analysis showed a lack of congruence between phylogenies of some groups of the pipolins and their host strains, which is in agreement with the hypothesis of pipolins horizontal transfer.

Based on the predefined hallmark features of *E. coli* pipolins, we have designed and implemented a pilot version of the ExplorePipolin pipeline, which is intended to identify, scaffold (when it is possible), extract and annotate pipolin elements from bacterial genome sequences. The source code of the ExplorePipolin pipeline is available on GitHub: https://github.com/liubovch/ExplorePipolin. It is command-line software that can be installed and run on any Unix-like system. We have created a Docker image and a Conda recipe to ease the pipeline installation and running on different systems.

The current pipeline version (v.0.0.a1) is mainly tested on *E. coli* genomes, therefore we will focus on extending it to other bacterial species in the future work. We are expecting to introduce the following upgrades:

1. Improve piPolB detection step. First, it sounds reasonable to use an HMM profile of piPolB gene from diverse groups of bacteria. Second, we need to be sure that the given HMM profile matches only bacterial piPolB gene and not a closely-related rPolB gene from phages, otherwise additional restrictions should be added. Third, it is probably required to distinguish between whole and truncated CDS of the gene because a truncated gene could point to a defective pipolin.

2. Improve *att* detection step. This step is the most intricate in the pipeline, because *att* sites can vary in their sequence composition and length, and MGEs are known to use different bacterial genes as their integration site, not always tRNA/tmRNA genes. Also, the presence of *att*s will confirm that particular pipolins are indeed "integrative" and help to establish pipolin boundaries. Thus, possible upgrades would be:

- adding a new dependency tool capable to detect integration sites, like MGEfinder or Mauve, or implementing a similar comparative approach;
- creating a database of *att* sites, which can be done using the same comparative approaches;
- an option to provide alternative *att* sequences;

3. Add an option for reporting possible circular pipolins. As for the analysed *E. coli* genomes, we have not encountered piPolBs belonging to plasmid sequences. Though, we are expecting to have circular pipolins in other bacteria. A special feature of such pipolins is the lack of terminal direct repeats. It should be possible to recognise circular pipolins when they are present within a bunch of contigs and to extract properly.

4. Overcome the current version limitations: 1) extend *de novo* search of *att*s to incomplete genomes, 2) make it possible to analyse several pipolin elements per genome: although, it is highly unlikely to have more than one pipolin per genome, it is possible to have circular pipolin along with the one integrated into the chromosome.

5. Increase the test coverage by writing more unit tests.

6. Caching of pipeline intermediate results and parallel task execution can be implemented to make pipeline calculations faster.

Another considerable improvement would be to create a website with a database of known pipolins. With the help of the pipeline, other bacterial species can be searched and the database can be easily updated. The website might in turn incorporate the pipeline as a separate online tool, as well as other tools allowing comparative analysis of chosen pipolin elements. Therefore, the proposed web-based resource would facilitate browsing of already discovered bacterial pipolins and identification and analysis of new pipolins.

Finally, ExplorePipolin pipeline can be useful for microbiologist that are interested in pipolins. It can be also an example of a comprehensive pipeline that not only looks for pipolin markers within a genome sequence but reconstruct the whole pipolin element structure when it is possible, making further analysis more precise and straightforward. Furthermore, the pipeline can be updated to work with other elements, similar to pipolins, particularly with archaeal Casposons.

# Acronyms

**AMR** antimicrobial resistance.

**CCC** cophenetic correlation coefficient.

**CDS** coding DNA sequence.

**GI** genomic island.

**HGT** horizontal gene transfer.

**HMM** Hidden Markov Model.

**ICE** integrative conjugative element.

**IS** insertion sequence.

**LREC** Spanish *E. coli* Reference Laboratory.

**MGE** mobile genetic element.

**ML** machine learning.

**ORF** open reading frame.

**piPolB** primer-independent PolB.

**Tn** transposon.

# Bibliography


[1] Mart Krupovic and Eugene V Koonin. Self-synthesizing transposons: unexpected key players in the evolution of viruses and defense systems. *Current opinion in microbiology*, 31:25–33, 2016.

[2] Modesto Redrejo-Rodríguez, Carlos D Ordóñez, Mónica Berjón-Otero, Juan Moreno-González, Cristian Aparicio-Maldonado, Patrick Forterre, Margarita Salas, and Mart Krupovic. Primer-independent DNA synthesis by a family B DNA polymerase from self-replicating mobile genetic elements. *Cell reports*, 21(6):1574–1587, 2017.

[3] Christian JH von Wintersdorff, John Penders, Julius M van Niekerk, Nathan D Mills, Snehali Majumder, Lieke B van Alphen, Paul HM Savelkoul, and Petra FG Wolffs. Dissemination of antimicrobial resistance in microbial ecosystems through horizontal gene transfer. *Frontiers in microbiology*, 7:173, 2016.

[4] Teresa Nogueira, Daniel J Rankin, Marie Touchon, François Taddei, Sam P Brown, and Eduardo PC Rocha. Horizontal gene transfer of the secretome drives the evolution of bacterial cooperation and virulence. *Current Biology*, 19(20):1683–1691, 2009.

[5] Emilien Nicolas, Michael Lambin, Damien Dandoy, Christine Galloy, Nathan Nguyen, Cédric A Oger, and Bernard Hallet. The *Tn*3-family of replicative transposons. *Mobile DNA III*, pages 693–726, 2015.

[6] Patricia Siguier, Lionel Gagnevin, and Michael Chandler. The new IS1595 family, its relation to IS1 and the frontier between insertion sequences and transposons. *Research in microbiology*, 160(3):232–241, 2009.

[7] Alessandra Carattoli, Ea Zankari, Aurora Garcìa-Fernandez, Mette Volby Larsen, Ole Lund, Laura Villa, Frank Møller Aarestrup, and Henrik Hasman. PlasmidFinder and pMLST: *in silico* detection and typing of plasmids. *Antimicrobial Agents and Chemotherapy*, pages AAC–02412, 2014.

[8] Valentina Galata, Tobias Fehlmann, Christina Backes, and Andreas Keller. PLSDB: a resource of complete bacterial plasmids. *Nucleic acids research*, 47(D1):D195–D202, 2019.

[9] David Arndt, Jason R Grant, Ana Marcu, Tanvir Sajed, Allison Pon, Yongjie Liang, and David S Wishart. PHASTER: a better, faster version of the PHAST phage search tool. *Nucleic acids research*, 44(W1):W16–W21, 2016.

[10] Elizaveta V Starikova, Polina O Tikhonova, Nikita A Prianichnikov, Chris M Rands, Evgeny M Zdobnov, Elena N Ilina, and Vadim M Govorun. Phigaro: high throughput prophage sequence annotation. *Bioinformatics*, 2019.

[11] Jean Cury, Thomas Jové, Marie Touchon, Bertrand Néron, and Eduardo PC Rocha. Identification and analysis of integrons and cassette arrays in bacterial genomes. *Nucleic acids research*, 44(10):4539–4550, 2016.

[12] Alexandra Moura, Mário Soares, Carolina Pereira, Nuno Leitão, Isabel Henriques, and António Correia. INTEGRALL: a database and search engine for integrons, integrases and gene cassettes. *Bioinformatics*, 25(8):1096–1098, 2009.

[13] Patricia Siguier, Jocelyne Pérochon, L Lestrade, Jacques Mahillon, and Michael Chandler. ISfinder: the reference centre for bacterial insertion sequences. *Nucleic acids research*, 34(suppl_1):D32–D36, 2006.

[14] Meng Liu, Xiaobin Li, Yingzhou Xie, Dexi Bi, Jingyong Sun, Jun Li, Cui Tai, Zixin Deng, and Hong-Yu Ou. ICEberg 2.0: an updated database of bacterial integrative and conjugative elements. *Nucleic acids research*, 47(D1):D660–D665, 2019.

[15] Claire Bertelli, Matthew R Laird, Kelly P Williams, Simon Fraser University Research Computing Group, Britney Y Lau, Gemma Hoad, Geoffrey L Winsor, and Fiona SL Brinkman. IslandViewer 4: expanded prediction of genomic islands for larger-scale datasets. *Nucleic acids research*, 45(W1):W30–W35, 2017.

[16] Corey M Hudson, Britney Y Lau, and Kelly P Williams. Islander: a database of precisely mapped genomic islands in tRNA and tmRNA genes. *Nucleic acids research*, 43(D1):D48–D53, 2015.

[17] Claire Bertelli, Keith E Tilley, and Fiona SL Brinkman. Microbial genomic island discovery, visualization and analysis. *Briefings in bioinformatics*, 20(5):1685–1698, 2019.

[18] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[19] Robert D Finn, Jody Clements, and Sean R Eddy. HMMER web server: interactive sequence similarity searching. *Nucleic acids research*, 39(suppl_2):W29–W37, 2011.

[20] Eric P Nawrocki and Sean R Eddy. Infernal 1.1: 100-fold faster RNA homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[21] Dongsheng Che, Cory Hockenbury, Robert Marmelstein, and Khaled Rasheed. Classification of genomic islands using decision trees and their ensemble algorithms. *BMC genomics*, 11(2):S1, 2010.

[22] Dongsheng Che, Han Wang, John Fazekas, and Bernard Chen. An accurate genomic island prediction method for sequenced bacterial and archaeal genomes. *J Proteomics Bioinform*, 7(8):214, 2014.

[23] Sergio Arredondo-Alonso, Malbert RC Rogers, Johanna C Braat, Tess D Verschuuren, Janetta Top, Jukka Corander, Rob JL Willems, and Anita C Schürch. mlplasmids: a user-friendly tool to predict plasmid-and chromosome-derived sequences for single species. *Microbial genomics*, 4(11), 2018.

[24] Matthew G Durrant, Michelle M Li, Benjamin A Siranosian, Stephen B Montgomery, and Ami S Bhatt. A bioinformatic analysis of integrative mobile genetic elements highlights their role in bacterial adaptation. *Cell Host & Microbe*, 27(1):140–153, 2020.

[25] Tiago F Jesus, Bruno Ribeiro-Gonçalves, Diogo N Silva, Valeria Bortolaia, Mário Ramirez, and João A Carriço. Plasmid ATLAS: plasmid visual analytics and identification in high-throughput sequencing data. *Nucleic acids research*, 47(D1):D188–D194, 2019.

[26] Matt Ravenhall, Nives Škunca, Florent Lassalle, and Christophe Dessimoz. Inferring horizontal gene transfer. *PLoS computational biology*, 11(5), 2015.

[27] Timothy Sampson, Gregory W Broussard, Laura J Marinelli, Deborah Jacobs-Sera, Mondira Ray, Ching-Chung Ko, Daniel Russell, Roger W Hendrix, and Graham F Hatfull. Mycobacteriophages BPs, Angel and Halo: comparative genomics reveals a novel class of ultra-small mobile genetic elements. *Microbiology*, 155(Pt 9):2962, 2009.

[28] Mart Krupovic, Kira S Makarova, Patrick Forterre, David Prangishvili, and Eugene V Koonin. Casposons: a new superfamily of self-synthesizing DNA transposons at the origin of prokaryotic CRISPR-Cas immunity. *BMC biology*, 12(1):36, 2014.

[29] Saskia-Camille Flament-Simon, María De Toro, Liubov Chuprikova, Miguel Blanco, Juan Moreno-González, Margarita Salas, Jorge Blanco, and Modesto Redrejo Rodríguez. High diversity and variability of pipolins among a wide range of pathogenic *Escherichia coli* strains. *bioRxiv*, 2020.

[30] Luis Vielva, María de Toro, Val F Lanza, and Fernando de la Cruz. PLACNETw: a web-based tool for plasmid reconstruction from bacterial genomes. *Bioinformatics*, 33(23):3796–3798, 2017.

[31] Kazutaka Katoh and Daron M Standley. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780, 2013.

[32] Andrew M Waterhouse, James B Procter, David MA Martin, Michèle Clamp, and Geoffrey J Barton. Jalview version 2—a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, 2009.

[33] Gavin E Crooks, Gary Hon, John-Marc Chandonia, and Steven E Brenner. WebLogo: a sequence logo generator. *Genome research*, 14(6):1188–1190, 2004.

[34] Torsten Seemann. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, 2014.

[35] Johannes Söding, Andreas Biegert, and Andrei N Lupas. The HHpred interactive server for protein homology detection and structure prediction. *Nucleic acids research*, 33(suppl_2):W244–W248, 2005.

[36] Andrew J Page, Carla A Cummins, Martin Hunt, Vanessa K Wong, Sandra Reuter, Matthew TG Holden, Maria Fookes, Daniel Falush, Jacqueline A Keane, and Julian Parkhill. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693, 2015.

[37] Jaime Huerta-Cepas, Damian Szklarczyk, Davide Heller, Ana Hernández-Plaza, Sofia K Forslund, Helen Cook, Daniel R Mende, Ivica Letunic, Thomas Rattei, Lars J Jensen, et al. eggNOG 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic acids research*, 47(D1):D309–D314, 2019.

[38] Minoru Kanehisa, Yoko Sato, and Kanae Morishima. BlastKOALA and ghostKOALA: KEGG tools for functional characterization of genome and metagenome sequences. *Journal of molecular biology*, 428(4):726–731, 2016.

[39] Mitchell J Sullivan, Nicola K Petty, and Scott A Beatson. Easyfig: a genome comparison visualizer. *Bioinformatics*, 27(7):1009–1010, 2011.

[40] ABRicate: Mass screening of contigs for antimicrobial resistance or virulence genes. https://github.com/tseemann/abricate.

[41] Johann Beghain, Antoine Bridier-Nahmias, Hervé Le Nagard, Erick Denamur, and Olivier Clermont. ClermonTyping: an easy-to-use and accurate *in silico* method for *Escherichia* genus strain phylotyping. *Microbial genomics*, 4(7), 2018.

[42] Nancy de Castro Stoppe, Juliana S Silva, Camila Carlos, Maria IZ Sato, Antonio M Saraiva, Laura MM Ottoboni, and Tatiana T Torres. Worldwide phylogenetic group patterns of *Escherichia coli* from commensal human and wastewater treatment plant isolates. *Frontiers in microbiology*, 8:2512, 2017.

[43] Méril Massot, Anne-Sophie Daubié, Olivier Clermont, Francoise Jaureguy, Camille Couffignal, Ghizlane Dahbi, Azucena Mora, Jorge Blanco, Catherine Branger, France Mentré, et al. Phylogenetic, virulence and antibiotic resistance characteristics of commensal strain populations of *Escherichia coli* from community subjects in the Paris area in 2010 and evolution over 30 years. *Microbiology*, 162(4):642, 2016.

[44] Olivier Clermont, Julia K Christenson, Anne-Sophie Daubié, David M Gordon, and Erick Denamur. Development of an allele-specific PCR for *Escherichia coli* B2 sub-typing, a rapid and easy to perform substitute of multilocus sequence typing. *Journal of microbiological methods*, 101:24–27, 2014.

[45] José Maria Gonzalez-Alba, Fernando Baquero, Rafael Cantón, and Juan Carlos Galán. Stratified reconstruction of ancestral *Escherichia coli* diversification. *BMC genomics*, 20(1):936, 2019.

[46] Lam-Tung Nguyen, Heiko A Schmidt, Arndt Von Haeseler, and Bui Quang Minh. IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular biology and evolution*, 32(1):268–274, 2015.

[47] Ari Löytynoja. Phylogeny-aware alignment with PRANK. pages 155–170, 2014.

[48] RStudio. Integrated development for R. https://rstudio.com/.

[49] Guangchuang Yu, David K Smith, Huachen Zhu, Yi Guan, and Tommy Tsan-Yuk Lam. ggtree: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods in Ecology and Evolution*, 8(1):28–36, 2017.

[50] Liam J Revell. phytools: an R package for phylogenetic comparative biology (and other things). *Methods in ecology and evolution*, 3(2):217–223, 2012.

[51] Tal Galili. dendextend: an R package for visualizing, adjusting and comparing trees of hierarchical clustering. *Bioinformatics*, 31(22):3718–3720, 2015.

[52] Aaron E Darling, Bob Mau, and Nicole T Perna. progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. *PloS one*, 5(6), 2010.

[53] Dean Laslett and Bjorn Canback. ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic acids research*, 32(1):11–16, 2004.

[54] Patricia P Chan, Brian Y Lin, Allysia J Mak, and Todd M Lowe. tRNAscan-se 2.0: improved detection and functional classification of transfer RNA genes. *bioRxiv*, page 614032, 2019.

[55] Tatiana Tatusova, Michael DiCuccio, Azat Badretdin, Vyacheslav Chetvernin, Eric P Nawrocki, Leonid Zaslavsky, Alexandre Lomsadze, Kim D Pruitt, Mark Borodovsky, and James Ostell. NCBI prokaryotic genome annotation pipeline. *Nucleic acids research*, 44(14):6614–6624, 2016.

[56] Ramy K Aziz, Daniela Bartels, Aaron A Best, Matthew DeJongh, Terrence Disz, Robert A Edwards, Kevin Formsma, Svetlana Gerdes, Elizabeth M Glass, Michael Kubal, et al. The RAST Server: rapid annotations using subsystems technology. *BMC genomics*, 9(1):75, 2008.

[57] Lara Rajeev, Karolina Malanowska, and Jeffrey F Gardner. Challenging a paradigm: the role of DNA homology in tyrosine recombinase reactions. *Microbiol. Mol. Biol. Rev.*, 73(2):300–309, 2009.

[58] Weidong Bao, Kenji K Kojima, and Oleksiy Kohany. Repbase Update, a database of repetitive elements in eukaryotic genomes. *Mobile Dna*, 6(1):11, 2015.

[59] Charles Bland, Teresa L Ramsey, Fareedah Sabree, Micheal Lowe, Kyndall Brown, Nikos C Kyrpides, and Philip Hugenholtz. CRISPR recognition tool (CRT): a tool for automatic detection of clustered regularly interspaced palindromic repeats. *BMC bioinformatics*, 8(1):209, 2007.

[60] Shujun Ou and Ning Jiang. LTR_retriever: a highly accurate and sensitive program for identification of long terminal repeat retrotransposons. *Plant physiology*, 176(2):1410–1422, 2018.

[61] Janna L Fierst. Using linkage maps to correct and scaffold *de novo* genome assemblies: methods, challenges, and computational tools. *Frontiers in genetics*, 6:220, 2015.

[62] Python programming language. http://python.org.

[63] Biopython: Python tools for computational molecular biology. http://biopython.org.

[64] bcbio-gff: a Python library to read and write Generic Feature Format (GFF). https://github.com/chapmanb/bcbb/tree/master/gff.

[65] Click Python package. https://click.palletsprojects.com/.

[66] Apache Airflow: a platform to programmatically author, schedule and monitor workflows. https://airflow.apache.org/.

[67] Metaflow: A framework for real-life data science. https://metaflow.org/.

[68] Luigi: a Python package to build complex pipelines of batch jobs. https://github.com/spotify/luigi.

[69] Prefect: a new standard in dataflow automation. https://www.prefect.io/.

# A

# Users manual

Pipolins constitute a new group of self-synthesizing or self-replicating mobile genetic elements (MGEs). They are widespread among diverse bacterial phyla and mitochondria.

> **Redrejo-Rodríguez, M., *et al.*** Primer-independent DNA synthesis by a family B DNA polymerase from self-replicating Mobile genetic elements. *Cell reports*, 2017[1]
>
> **Flament-Simon, S.C., de Toro, M., Chuprikova, L., *et al.*** High diversity and variability of pipolins among a wide range of pathogenic *Escherichia coli* strains. *bioRxiv*, 2020[2]

**ExplorePipolin** is a search tool that identifies and analyses pipolins within bacterial genome.

## A.1 Requirements

- pip
- BLAST+[3]
- ARAGORN[4]
- Prokka[5]

## A.2 Installation

### A.2.1 Install from source

1. Install the requirements (see above).
2. `wget https://github.com/liubovch/ExplorePipolin/archive/0.0.a1.zip`
3. `unzip 0.0.a1.zip && cd ExplorePipolin-0.0.a1`

[1] <https://doi.org/10.1016/j.celrep.2017.10.039>
[2] <https://www.biorxiv.org/content/10.1101/2020.04.24.059261v1>
[3] <https://www.ncbi.nlm.nih.gov/books/NBK279690/>
[4] <https://github.com/TheSEED/aragorn>
[5] <https://github.com/tseemann/prokka>

4. `pip install .` (install in user site-package) or `sudo pip install .` (requires superuser privileges)

NOTE: before installing, it is possible to run unit tests: `pytest` or `python setup.py test` (from the source root directory).

**How to uninstall:**

```
(sudo) pip uninstall ExplorePipolin
```

### A.2.2 Install using Conda

- Before installing ExplorePipolin, make sure you'are running the latest version of Conda:

```
conda update conda
```

```
conda install wget
```

- Create a new environment that is specific for ExplorePipolin. You can choose whatever name you'd like for the environment.

```
wget https://github.com/liubovch/ExplorePipolin/releases/download/0.0.a1/\
```

```
explore-pipolin-0.0.a1-py_0.yml
```

```
conda env create -n ExplorePipolin-0.0.a1 -file explore-pipolin-0.0.a1-py_0.yml
```

- Download and install ExplorePipolin into the created environment:

```
wget https://github.com/liubovch/ExplorePipolin/releases/download/0.0.a1/\
```

```
explore-pipolin-0.0.a1-py_0.tar.bz2
```

```
conda install -n ExplorePipolin-0.0.a1 explore-pipolin-0.0.a1-py_0.tar.bz2
```

- Clean up (optional):

```
rm explore-pipolin-0.0.a1-py_0.yml explore-pipolin-0.0.a1-py_0.tar.bz2
```

- Activate the environment and check the installation:

```
conda activate ExplorePipolin-0.0.a1
```

```
explore_pipolin -h
```

## A.3 Quick usage

### A.3.1 Test run

As input, **ExplorePipolin** takes FASTA file(s) with genome sequence(s). A genome sequence can be either a single complete chromosome (preferred) or contigs (in a single multiFASTA file).

```
-> explore_pipolin -h
```

```
Usage:  explore_pipolin [OPTIONS] GENOMES...
```

```
ExplorePipolin is a search tool that identifies and analyses pipolin elements
```

```
within bacterial genome(s).
```

```
Options:
```

```
-out-dir PATH [required]
```

```
-h, -help Show this message and exit.
```

### A.3.2 Output files

The output directory will contain several folders:

`pipolbs_search` – BLAST search results for piPolB genes

`atts_search` – BLAST search results for the known *att* sites

`atts_denovo_search` – Results of *de novo* search for *att* sites

`trnas_search` – ARAGORN search results for tRNAs/tmRNAs

`pipolin_sequences` – extracted pipolin sequences in FASTA format

`prokka_results` – Prokka annotation results (check files description here[6])

`results` – GenBank and GFF annotation results with the *att*s included, log files

## A.4 Running with Docker

See here[7] to install Docker.

**NOTE:** superuser privileges are required to run the analysis and around 3GB of disk space for the image.

```
sudo docker pull docker.pkg.github.com/liubovch/explorepipolin/\
explore_pipolin:0.0.a1
sudo docker tag docker.pkg.github.com/liubovch/explorepipolin/\
explore_pipolin:0.0.a1 explore_pipolin
sudo docker run -rm explore_pipolin -h
sudo docker run -rm -v $(pwd):/output -w /output explore_pipolin -out-dir output
./input_genomes/*.fa #(example run)
```

[6] <https://github.com/tseemann/prokka/blob/master/README.md#output-files>
[7] <https://docs.docker.com/install/>