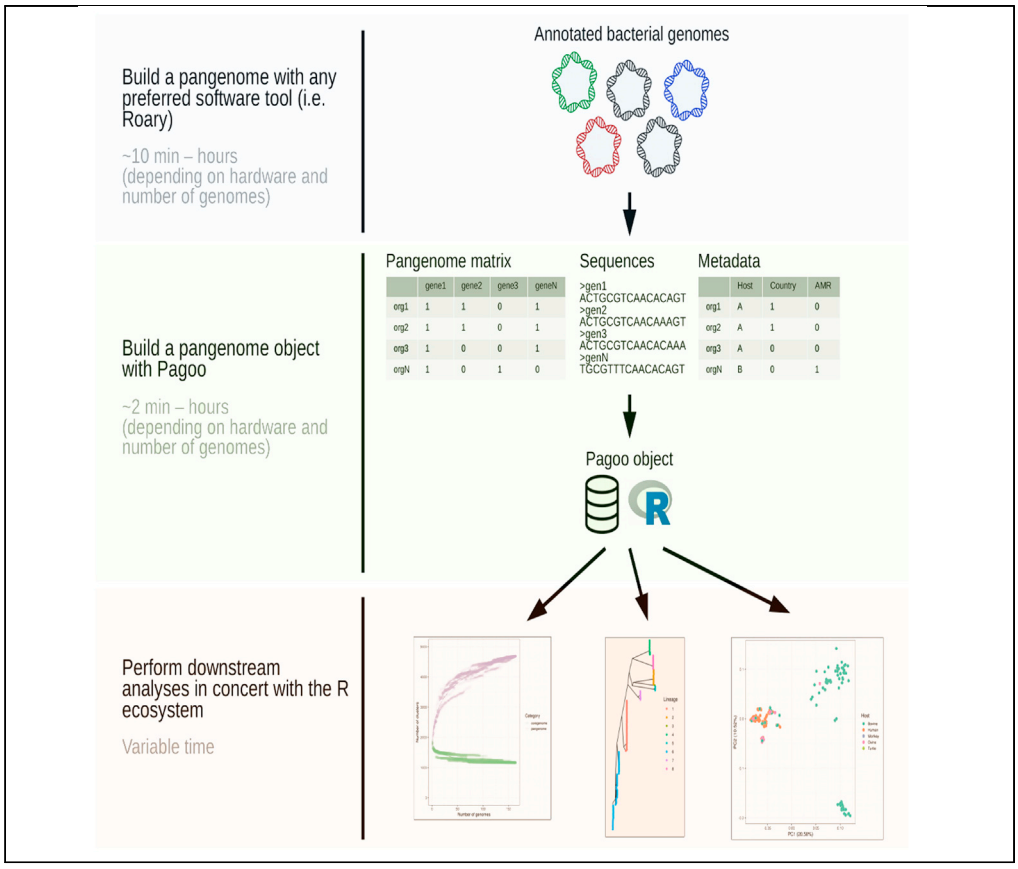
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for post-processing of bacterial pangenome data using Pagoo pipeline



Multiple downstream analyses are necessary to interpret the output of bacterial pangenome reconstruction software. This requires integrating diverse kinds of genetic and phenotypic data, which to date are left to each user's criterion. To fill this gap, we created Pagoo, a pangenome post-processing tool that leverages a standardized but flexible and extensible framework for data integration, analysis, and storage. Here, we provide the protocol for running Pagoo and performing from simple to more complex comparative analyses on bacterial pangenome data.

Ignacio Ferrés, Gregorio Iraola

iferres@pasteur.edu.uy (I.F.)
giraola@pasteur.edu.uy (G.I.)

Highlights

Bacterial pangenome analysis needs the integration of genetic and phenotypic data

This protocol shows how to achieve this using the Pagoo software package

Pagoo works in concert with other microbial genomics packages in the R ecosystem

Ferrés & Iraola, STAR Protocols 2, 100802 December 17, 2021
https://doi.org/10.1016/j.xpro.2021.100802

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for post-processing of bacterial pangenome data using Pagoo pipeline

Ignacio Ferrés[1,2,5,*] and Gregorio Iraola[1,2,3,4,6,*]

[1]Microbial Genomics Laboratory, Institut Pasteur Montevideo, Montevideo, Montevideo 11400, Uruguay

[2]Center for Innovation in Epidemiological Surveillance, Institut Pasteur Montevideo, Montevideo, Montevideo 11400, Uruguay

[3]Wellcome Sanger Institute, Hinxton, Cambridgeshire CB10 SA1, UK

[4]Center for Integrative Biology, Universidad Mayor, Providencia, Santiago de Chile, Chile

[5]Technical contact

[6]Lead contact

*Correspondence: iferres@pasteur.edu.uy (I.F.), giraola@pasteur.edu.uy (G.I.)
https://doi.org/10.1016/j.xpro.2021.100802

SUMMARY

Multiple downstream analyses are necessary to interpret the output of bacterial pangenome reconstruction software. This requires integrating diverse kinds of genetic and phenotypic data, which to date are left to each user's criterion. To fill this gap, we created Pagoo, a pangenome post-processing tool that leverages a standardized but flexible and extensible framework for data integration, analysis, and storage. Here, we provide the protocol for running Pagoo and performing from simple to more complex comparative analyses on bacterial pangenome data.

For complete details on the use and execution of this protocol, please refer to Ferrés and Iraola (2021).

BEFORE YOU BEGIN

Pagoo is designed to take the output generated by pangenome reconstruction software like Roary (Page et al., 2015), Panaroo (Tonkin-Hill et al., 2020), PEPPAN (Zhou et al., 2020) or panX (Ding et al., 2018). This protocol explains how to use built-in functions from Pagoo to automatically load and analyze output files produced by these pangenome reconstruction tools as they are widely used by the community. The output of any of these software tools can be loaded by Pagoo as a set of tables and genetic sequences. Hence, for starting using Pagoo, pangenome reconstruction needs to be performed on a set of genomes using any preferred tool.

Pangenome reconstruction

Timing: 30 min

Here, we provide a full-reproducible example on how to build a pangenome based on 168 previously published *Campylobacter fetus* genomes (Iraola et al., 2017). These genomes have been previously annotated using Prokka (Seemann, 2014) using default parameters (follow the steps described here to perform genome annotation). In this case, we will use Roary (Page et al., 2015) with default parameters to reconstruct the pangenome. It is assumed that Roary is installed and available in the user's path.

1. Download and decompress genomes from inside the R session (Box 1):

Box 1

```
tar_gz <- "cfetus_pangenome.tar.gz"

if ( !file.exists(tar_gz) ) {

      download.file(url = "https://ndownloader.figshare.com/files/26144075",

      destfile = tar_gz)

}

untar(tarfile = tar_gz, exdir = "C_fetus")
```

2. Run Roary to reconstruct the pangenome:
   a. List GFF annotation files and run Roary from inside the R session (Box 2):

Box 2

```
gffs <- list.files(path = "C_fetus",

                   pattern = "[.]gff$",

                   recursive = TRUE,

                   full.names = TRUE)

roary <- paste("roary -f ./C_fetus/roary_output",

               paste(gffs, collapse = " "))

system(roary)
```

Note: Timing of this step depends on the software selected to perform pangenome reconstruction and the number of genomes to be analyzed.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
GFF3 input files for pangenome reconstruction	Figshare	https://doi.org/10.6084/m9.figshare.13622354.v1
Software and algorithms		
Pagoo	Ferrés and Iraola, 2021	https://github.com/iferres/pagoo
Roary	Page et al., 2015	https://sanger-pathogens.github.io/Roary

MATERIALS AND EQUIPMENT

- Data (output files produced by pangenome reconstruction software - see pangenome reconstruction in before you begin). As a working example, this protocol uses the dataset listed in the key resources table.
- Pagoo works in Linux, Mac and Windows systems in which R is installed. This protocol was conducted on Linux (Fedora Generic release 26) and R v4.0.3 using a desktop computer with 64 GB RAM and an Intel Core i7-7700 (3.60GHz) processor.

STEP-BY-STEP METHOD DETAILS

Step 1: Installing Pagoo

Timing: 4 min

Full installation of Pagoo includes downloading the Pagoo package and resolving all its dependencies from CRAN. Alternatively, the last development version of Pagoo can be installed from GitHub.

1. Install Pagoo from CRAN by running the following code:

```
install.packages("pagoo")
```

2. Alternatively, install Pagoo from GitHub by running the following code:

```
if (!require("devtools"))

devtools::install_github("iferres/pagoo")
```

Step 2: Loading pangenome data from scratch

Timing: 5 min

First, we provide a toy example that describes how to create Pagoo's data structure from scratch independently of the pangenome reconstruction software that was used to generate the data. This toy dataset is included when the user installs Pagoo package.

3. **Generate a Pagoo object.** To generate a Pagoo object, the only mandatory data structure is a data.frame with information relating organisms to genes to orthologous clusters. Other optional data can be also added as additional columns or tables. Load toy example files by running the code in Box 3:

Box 3

```
library(pagoo)

tgz <- system.file("extdata", "toy_data.tar.gz", package = "pagoo")

untar(tarfile = tgz, exdir = ".")

files <- list.files(full.names = TRUE, pattern = "tsv$|fasta$")
```

   a. We will see the case_df.tsv file that is the mandatory data.frame. Run the code in Box 4 to load and inspect it:

Box 4

```
data_file <- grep("case_df.tsv", files, value = TRUE)

data <- read.table(data_file, header = TRUE,

         sep = "\t", quote = "")

head(data)

##       gene        org cluster                              annot

## 1  gene081  organismA   OG001  Thioesterase superfamily protein
```

. Continued

```
## 2  gene122  organismB   OG001          Thioesterase superfamily
## 3  gene299  organismC   OG001  Thioesterase superfamily protein
## 4  gene186  organismD   OG001  Thioesterase superfamily protein
## 5  gene076  organismE   OG001          Thioesterase superfamily
## 6  gene352  organismA   OG002  Inherit from proNOG: Thioesterase
```

b. The first column in this file identifies each gene, the second column identifies the organism to which each gene belongs and the third column shows the orthologous cluster to which each gene was assigned in the pangenome reconstruction. The fourth column shows the annotation of each gene (optional as other additional metadata columns that can be added to this table). With only this data you can start working with Pagoo as follows:

```
p <- pagoo(data = data)
```

4. **Adding metadata to organisms.** Relevant data associated to each organism (i.e., geographic origin, collection date, phenotyping, etc.) can be added to the basic pangenome structure as detailed in Box 5:

Box 5

```
orgs_file  <- grep("case_orgs_meta.tsv", files, value = TRUE)

orgs_meta <- read.table(orgs_file, header = TRUE,

                        sep = "\t", quote = "")

head(orgs_meta)

##          org sero     country
## 1  organismA    a    Westeros
## 2  organismB    b    Westeros
## 3  organismC    c    Westeros
## 4  organismD    a       Essos
## 5  organismE    b       Essos
```

⚠ CRITICAL: Beware that organism names provided in orgs_meta$org must coincide with names provided in the data$org field, in order to correctly map each variable.

Note: If partial metadata is available, for example host data is only available for a subset of organisms, fields with missing data will be automatically filled with NAs.

5. **Adding metadata to orthologous clusters.** Relevant data can be also incorporated to each orthologous cluster, for example its functional annotation as detailed in Box 6 (see here for a guide for generating functional annnotations using the eggNOG database and related tools):

Box 6

```
clust_file  <- grep("case_clusters_meta.tsv", files, value = TRUE)

clust_meta <- read.table(clust_file, header = TRUE,

                         sep = "\t", quote = "")

head(clust_meta)
```

. *Continued*

```
##    cluster    kegg    cog
## 1    OG001    <NA>      S
## 2    OG002    <NA>      S
## 3    OG003    <NA>   <NA>
## 4    OG004    <NA>      D
## 5    OG005   K01990     V
## 6    OG006    <NA>      V
```

⚠ CRITICAL: Again, the column clust_meta$cluster must contain the same identifiers as the data$cluster column to be able to map one into the other.

6. **Adding sequences.** If the user wants to add sequences, these must be provided for all organisms in the dataset. The type of data needed are multi-FASTA files where each individual sequence represents a gene whose name can be mapped to the data$gene column. Sequences need to be loaded as a list where each element is named with an organism name that maps to data$org and org_meta$org. This can be done as described in Box 7:

Box 7

```
fasta_files <- grep("[.]fasta", files, value = TRUE)

names(fasta_files) <- sub("[.]fasta", "", basename(fasta_files))

library(Biostrings)

sq <- lapply(fasta_files, readDNAStringSet)

# Names are the same as in data$org

## [1] "organismA" "organismB" "organismC" "organismD" "organismE"
```

Note: Pagoo currently supports the incorporation of DNA sequences. However, once sequences are incorporated into the Pagoo object, these can be translated and used as protein sequences for downstream analysis using Biostrings and other R packages.

7. **Generate an object with multiple data.** Now, we can set up a Pagoo object integrating all this data, including presence/absence of each orthologous cluster in each organism, gene annotations, functional annotations, organisms metadata and sequences. To build the Pagoo object run the code in Box 8:

Box 8

```
p <- pagoo(data = data, # Required data

           org_meta = orgs_meta, # Organisms metadata

           cluster_meta = clust_meta, # Clusters metadata

           sequences = sq) # Sequences
```

8. **Adding more metadata after creating the Pagoo object.** The user can add new metadata as the outcome of downstream analysis or experiments. This can be achieved by adding new metadata columns either to each gene, cluster or organism defined in the Pagoo object. By running the following code, we illustrate how to add a new column to the $organisms field named host that describes the host where each organism was isolated from (see Box 9):

Box 9

```
host_df <- data.frame(org = p$organisms$org,
                      host = c("Cow", "Dog", "Cat", "Cow", "Sheep"))
p$add_metadata(map = "org", host_df)
p$organisms
##          org     sero    country     host
## 1  organismA        a   Westeros      Cow
## 2  organismB        b   Westeros      Dog
## 3  organismC        c   Westeros      Cat
## 4  organismD        a      Essos      Cow
## 5  organismE        b      Essos    Sheep
```

⚠ CRITICAL: To allow Pagoo to correctly map the data, values in the first column of the host_df table must be the same as in p$organisms$org, and its column header must also be named org.

Note: Preparing data from scratch and loading classes can be relatively laborious, but in real life working datasets this will be rarely needed. To avoid this, Pagoo provides helper functions that directly parse and read-in output files produced by most widely-used pangenome reconstruction software, avoiding any formatting or manipulation of data (see next step).

Step 3: Input from pangenome reconstruction software

Timing: 10 min

Pagoo allows read-in output files produced by most widely-used pangenome reconstruction software. Since its publication in 2015, Roary (Page et al., 2015) has been the standard and most cited software for pangenome reconstruction. More recently, other related software have emerged like PIRATE (Bayliss et al., 2019), Panaroo (Tonkin-Hill et al., 2020) and PEPPAN (Zhou et al., 2020) that improved different steps of pangenome reconstruction or provided new analytical approaches. Also, we have created our own pangenome reconstruction software called Pewit (unpublished but available at https://github.com/iferres/pewit), which automatically generates a Pagoo-like object to perform downstream analyses. This object contains all the methods and fields that Pagoo provides, but adding a set of methods and fields exclusive to Pewit (not covered in this protocol). Here, we provide full details on how to load output files from Roary and indicate how to load output files from other above-listed software.

9. To create a pangenome object using Pagoo from output files produced by Roary (as described in step 2), run the code described in Box 10 assuming you are placed in the output directory generated by Roary:

Box 10

```
gffs <- list.files(path = "../gffs/", pattern = "[.]gff$", full.names = TRUE)
gpa_csv <- "gene_presence_absence.csv"
library(pagoo)
p <- roary_2_pagoo(gene_presence_absence_csv = gpa_csv, gffs = gffs)
```

Note: Exactly the same approach can be used to load output files from Panaroo, by using the analogous function panaroo_2_pagoo(). Other software like PIRATE and PEPPAN provide scripts (PIRATE_to_roary.pl and PEPPAN_parse.py, respectively) that transform their output files into Roary's output format. Then, roary_2_pagoo() function can be used to generate the pangenome object from PIRATE and PEPPAN once this transformation has been applied.

Step 4: Querying pangenome data

Timing: 10 min

The user can easily explore information that is stored inside the Pagoo object using standard R notation. Indeed, this object has its own associated data and methods that can be easily queried with the '$' operator. These methods allow for the rapid subsetting, extraction and visualization of pangenome data.

10. **Summary statistics.** A pangenome can be stratified in different gene subsets according to their frequency. The core genes are defined as those present in every genome (also the term soft core is typically used for genes occurring in 95%–100% of genomes). The remaining genes are defined as the accessory genome, that can be subdivided in cloud genes or singletons (present in only one genome or only in genomes that) and shell genes which are those in the middle. Run the following code in Box 11 to see this:

Box 11

```
p$summary_stats

#    Category  Number

# 1    Total    7326

# 2    Core    1489

# 3    Shell    5010

# 4    Cloud    818
```

11. **Core level.** The core level defines the minimum number of genomes (as a percentage) in which a certain gene should be present to be considered a core gene. By default, Pagoo considers as core genes all those present in at least 95% of organisms. The core level can be modified to be more or less stringent defining the core genome. Modifying the core level will affect the pangenome object state resulting in different core, shell and cloud sets. See this running the command in Box 12:

Box 12

```
p$core_level       # [1] 95

p$core_level <- 100 # Change value

p$summary_stats    # Updated object

#    Category     Number

# 1     Total      7326

# 2      Core      1117

# 3     Shell      5391

# 4     Cloud       818
```

12. **Pangenome matrix.** The pangenome matrix is one of the most useful things when analyzing pangenomes. Typically, it represents organisms in rows and clusters of orthologous genes in columns informing about gene abundance (considering paralogues). The pangenome matrix looks like the one shown in Box 13 (printing only first 5 rows and columns):

Box 13

```
p$pan_matrix[1:5, 1:5]

#             aadK  aaeA_1  aaeA_2  aat  aat_2
# 16244_6#1      1       0       0    1      0
# 16244_6#10     0       0       0    1      0
# 16244_6#11     1       0       0    1      0
# 16244_6#12     0       0       0    1      0
# 16244_6#13     1       0       0    1      0
```

13. **Genes metadata.** Metadata associated with each individual gene can be accessed by the $genes suffix. It always contains the gene name, the organism to which it belongs, its assigned cluster and a gene identifier (gid). Optionally, it can typically include annotation data, genomic coordinates, etc. Gene metadata is splitted by cluster, so it consists of a list of dataframes (Box 14, showing only the first rows):

Box 14

```
p$genes[1]

# SplitDataFrameList of length 1

# $aadK

# DataFrame with 7 rows and 10 columns

#                    cluster          org              gene                    gid
#                   <factor>     <factor>          <factor>            <character>
# 16244_6#1_00636       aadK   16244_6#1    16244_61_00636    16244_6#1__16244_6..
# 16244_6#11_00101      aadK  16244_6#11  16244_6#11_00101   16244_6#11__16244_6..
# 16244_6#13_00100      aadK  16244_6#13  16244_6#13_00100  16244_6#13__16244_6#..
```

14. **Clusters metadata.** Groups of orthologous genes (clusters) are also stored in Pagoo objects as a table with a cluster identifier per row, and additional columns as optional metadata (Box 15, showing only first rows):

Box 15

```
p$clusters

# DataFrame with 7326 rows and 2 columns

#   cluster              Annotation
#  <factor>              <character>
# 1  aadK       hypothetical protein
# 2  aaeA_1    Ribonuclease P prote..
# 3  aaeA_2    N-carbamoyl-D-amino..
```

. Continued

```
# 4  aat       putative ABC transpo..
# 5  aat_2      hypothetical protein
```

15. **Sequences.** Although it is an optional field (it exists only if the user provides this data as an argument when the object is created), $sequences gives access to sequence data. Sequences are stored as a DNAStringSetList object as defined in the Biostrings package. The code in Box 16 will list all sequences in clusters (only showing first rows):

Box 16

```
p$sequences
# DNAStringSetList of length 7326
# [["COQ2"]] 16244_6#1__16244_6#1_01627=ATGGCTAAATTTACTCAAATTTTAAAAGATATAAACGAA...
# [["COQ3_1"]] 16244_6#1__16244_6#1_00352=ATGAGTAACGCAAACGCATGGGACGATATGTCAAATT...
# [["COQ3_2"]] 16244_6#10__16244_6#10_01654=ATGAAAAAAACGTTTTCATTTGGAAAAAACTGGCT...
# [["COQ3_3"]] 16244_6#1__16244_6#1_00772=ATGAAAGAAAAGTTTTTTGAACTAAAAGTTTTAAGCC...
```

Then, you can list all sequences of a single cluster by running the code in Box 17 (only showing first rows):

Box 17

```
p$sequences[["aadK"]]
# DNAStringSet object of length 7:
# [1] 858 ATGAAAATGAGAACAGAGAAACA...AAAAAGAAAAATATCAAAGATAA 16244_6#1__16244_...
# [2] 858 ATGAAAATGAGAACAGAGAAACA...AAAAAGAAAAATATCAAAGATAA 16244_6#11__16244...
# [3] 858 ATGAAAATGAGAACAGAGAAACA...AAAAAGAAAAATATCAAAGATAA 16244_6#13__16244...
# [4] 858 ATGAAAATGAGAACAGAGAAACA...AAAAAGAAAAATATCAAAGATAA 16244_6#6__16244_...
```

⚠ CRITICAL: Note that sequence names are created by pasting organism names and gene names, separated by a string that by default is sep = '__' (two underscores). This is the same as the gid column in the $genes field, and is initially set when a Pagoo object is created. If you think your dataset contains names with this separator, then you should set this parameter to another string to avoid conflicts.

Note: Sequences can be written to text as multi-FASTA format files using standard methods provided by the Biostrings package.

16. **Organisms metadata.** The $organisms field contains a table with organisms and metadata as additional columns if provided (Box 18, only showing first rows):

Box 18

```
p$organisms
# DataFrame with 168 rows and 10 columns
#    org     Accession.Number  Identifier    Strain    Year    Country
#    <factor> <character>  <character>  <character>  <integer>  <character>
# 1  16244_6#1   ERS672242         FR10    2006/367h       2006       France
```

. Continued

```
# 2  16244_6#10   ERS672251        FR19    2008/755h       2008       France
# 3  16244_6#11   ERS672252        FR20    2008/898h       2008       France
# 4  16244_6#12   ERS672253        FR21     2010/41h       2010       France
# 5  16244_6#13   ERS672254        FR22    2010/524h       2010       France
```

Step 5: Data subsetting

Timing: 10 min

Data subsetting is a fundamental operation when working with pangenome, enabling structured and more in depth analyses. Pagoo provides three ways of subsetting: (i) predefined subsets, (ii) classic R's subsetting operations using square bracket operators, and (iii) removal or recovering organisms from the dataset.

17. **Predefined subsets.** As explained in step 10, elements within a pangenome can be classified in different compartments given their frequency of occurrence: core, shell and cloud. Pagoo provides operators to directly access elements in these compartments, independently if they are genes, clusters or sequences. Look at the following table for all possible combinations:

	$core_*	$shell_*	$cloud_*
$*_genes	$core_genes	$shell_genes	$cloud_genes
$*_clusters	$core_clusters	$shell_clusters	$cloud_clusters
$*_sequences	$core_sequences	$shell_sequences	$cloud_sequences

    a. As seen in the above table, the notation is quite straightforward. See example in Box 19 using $clusters to illustrate this better:

Box 19

```
dim(p$clusters)[1]

# [1] 7326

dim(p$core_clusters)[1]

# [1] 1498

dim(p$shell_clusters)[1]

# [1] 5010

dim(p$cloud_clusters)[1]

# [1] 818
```

It can be appreciated that the total number of orthologous clusters in this pangenome is 7326, but only 1498 represent clusters of core genes.

18. **Standard R subsetting notation.** Elements represented as matrices or vectors contained in the Pagoo object can be subsetted using standard R notation using square brackets. Let's see a couple of examples.
    a. Subsetting the pangenome matrix (Box 20):

Box 20

```
p$pan_matrix[1:3, 10:15]

#            accB  accB_1  accC  accC_1  accD  accD_2

# 16244_6#1     1       0     1       0     1       0

# 16244_6#10    1       0     1       0     1       0

# 16244_6#11    1       0     1       0     1       0
```

b. Subsetting core sequences (Box 21):

Box 21

```
p$core_sequences[c(1,30)]

# DNAStringSetList of length 2

# [["COQ2"]] 16244_6#1__16244_6#1_01627=ATGGCTAAATTTACTCAAATTTTAAAAGATATAAACGAA...

# [["ansA"]] 16244_6#1__16244_6#1_00415=ATGTGCTTAAAAAAGGTGTTTATACTTATGCTGATTACG...
```

19. **Dropping and recovering organisms.** The possibility of hiding certain organisms from the dataset is useful if we want to remove some genome with abnormal characteristics (i.e., potentially contaminated), if we want to focus just in a subset of genomes of interest given any metadata value, or if we included an outgroup for phylogenetic purposes but we want to discard it from downstream analyses. The following steps show how this works:.
    a. We are working with 168 organisms in the dataset, out of which 74 are from human origin (see Box 22):

Box 22

```
table(p$organisms$Host)

# Bovine Human Monkey Ovine Turtle

#     78    74      1    13      2
```

b. We will hide these 74 from the dataset (see Box 23):

Box 23

```
to_drop <- which(p$organisms$host=="Human")

p$drop(to_drop)

table(p$organisms$host)

# Bovine  Monkey  Ovine  Turtle

#     78       1     13       2
```

Note: When the user hides a set of organisms, this will have an impact on all the information stored in the object. This means that all features associated with these genomes including genes, sequences, clusters and metadata will be hidden. It is important to note that the user does not have to reassign the object to a new one, it is self-modified (in place modification) according to R6 reference semantics.

c. To recover hidden organisms run the following (see Box 24):

Box 24

```
dropped <- p$dropped p$recover(dropped)
```

Step 6: Built-in methods and visualizations

Timing: 10 min

Pagoo provides basic but fundamental statistical analyses and visualizations for straightforward exploration of pangenome features. All these methods are embedded in the Pagoo object.

20. **Principal Components Analysis (PCA).** PCA is a fundamental statistical tool that can be applied to pangenome data to see how organisms are grouped based on the diversity of their accessory genes. PCA can be calculated directly from the Pagoo object as follows (Figure 1):
    a. Generate a standard PCA object for downstream analysis:

```
pca <- p$pan_pca()
```

    b. Directly visualizing the first 2 PCs using a biplot. This uses the previous method to perform the PCA but allows you to generate a customizable ggplot2 object on the fly (see Box 25):

Box 25

```
p$gg_pca(color = "Host", size = 4) +
    theme_bw(base_size = 15) +
    scale_color_brewer(palette = "Set2")
```

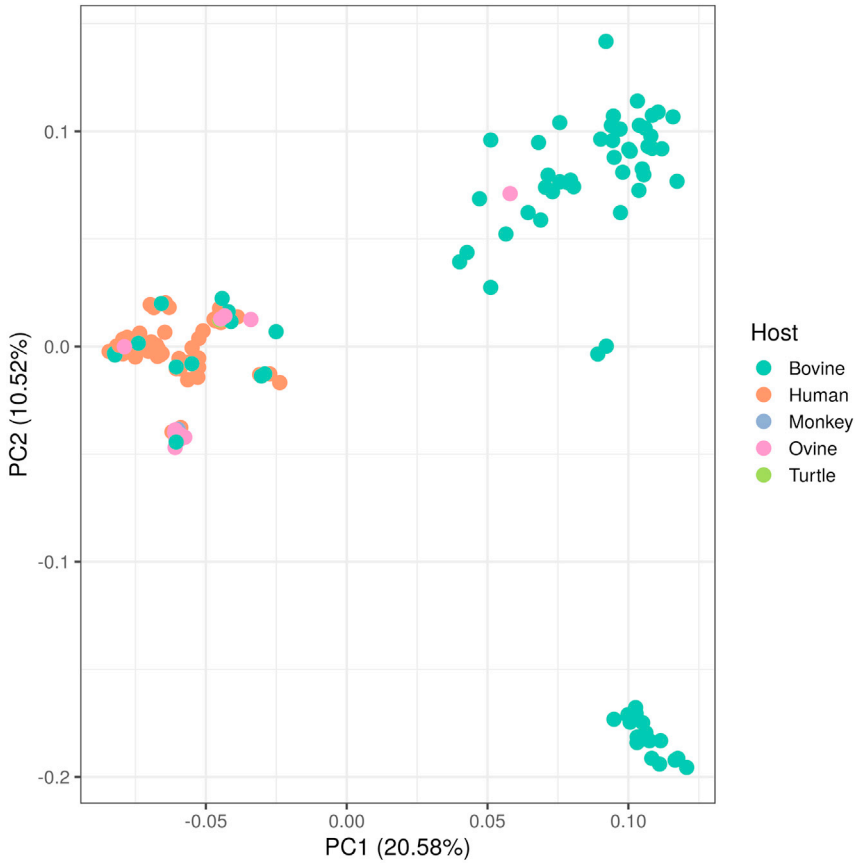


Figure 1. Principal components analysis

A PCA is generated directly from the gene presence/absence matrix and in this case organisms are colored by host of origin.

21. **Rarefaction curves.** Pangenome curves (Figure 2) show the number of gene clusters that are subsequently discovered as more genomes are added to the dataset. If the pangenome is open, more novel accessory genes will be discovered as new genomes are added and the size of the core genome will tend to decrease. Pagoo applies the Power-law distribution to fit the pangenome size and the Exponential decay function to fit the core genome size. Run this method and customize results as shown in Box 26:

Box 26

```
p$gg_curves(size = 2) +
  ggtitle("Pangenome curves") +
  geom_point(alpha = 0.1, size = 4) +
  theme_bw(base_size = 15) + ylim(0, 5000) +
  scale_color_brewer(palette = "Accent")
```

22. **Other methods.** Pagoo provides further methods for summary statistics whose application is analogous to the above described ones. These include pie charts using $gg_pie(), gene presence/absence bin maps using $gg_binmap() and gene frequency bar plots using $gg_barplot().
23. **Dynamic visualization.** The above-mentioned plots for summary statistics can be deployed through a R Shiny application, allowing responsive and dynamic exploration of pangenome data. The application can be run as follows:

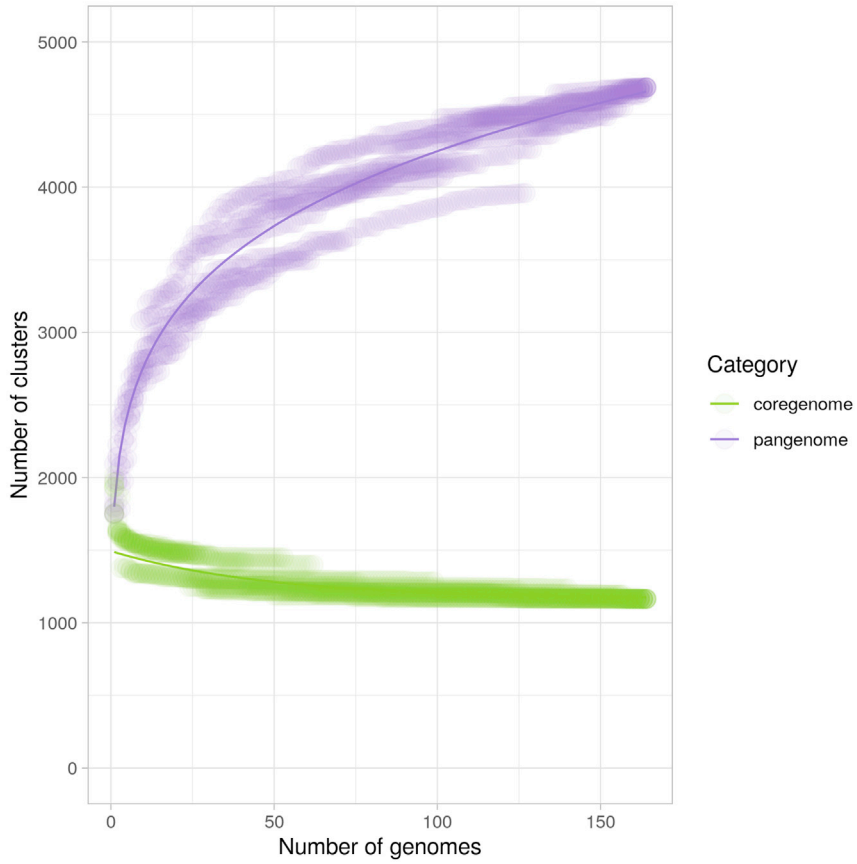


Figure 2. Pangenome curves

Pangenome curves show the accessory and core genome size and are indicative of the gene pool size in a certain dataset.

```
p$runShinyApp()
```

CAUTION: The method described in step 23 (R-Shiny application) is currently not intended for very large datasets, as it may render slow. We recommend it to work with dozens to hundreds of genomes. For bigger pangenomes we recommend the use of the R command line.

Note: An online example for a set of 69 genomes can be found here.

Step 7: Downstream analyses using recipes

Timing: 20 min

Here we show how Pagoo can interact with other R or external tools to generate more complex analyses. We introduce the concept of Pagoo recipes, that are concise pieces of code to complete different tasks. By using these recipes (or creating new ones) the user can take full profit of functionalities provided by Pagoo to perform a variety of analyses including phylogenetics, pangenome-wide association studies, sequence comparisons, ecological measures, preparation of publication-quality figures, among others. In this protocol, we provide a couple of examples of how to create these recipes, which can be found on GitHub.

24. **Publication quality figures.** The following recipe (Box 27) allows to produce a publication quality figure showing pangenome main features using the previously

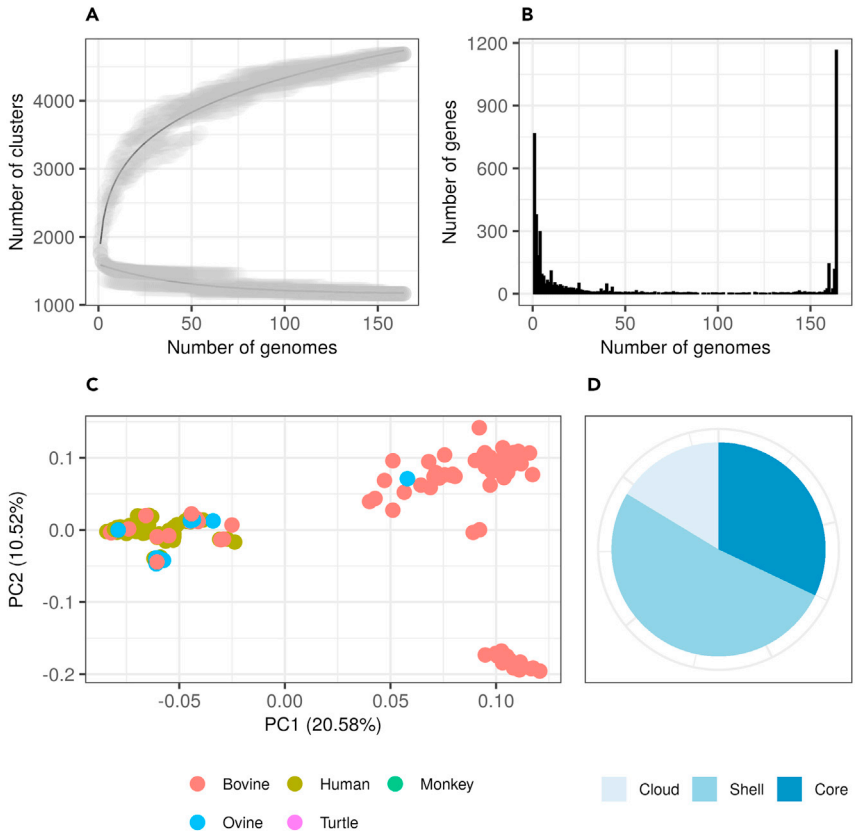


Figure 3. Visualization of pangenome features

Pagoo can be integrated with other R packages to produce publication-quality figures in a simple way. In this case, the figure shows an assembly of different analyses that summarize general features of this example pangenome: (A) pangenome curves, (B) gene frequency plots, (C) Accessory genes PCA and (D) pie chart with gene subsets.

mentioned methods (result shown in Figure 3), directly from the Pagoo object in the R session:

Box 27

```
# 1. Pangenome curves
panel1 <- p$gg_curves() +
    scale_color_manual(values = c("black", "black")) +
    geom_point(alpha = .05, size = 4, color = "grey") +
    theme_bw(base_size = 15) +
    labs(subtitle = "A") +
    theme(legend.position = "none",
      axis.title = element_text(size = 12),
      axis.text = element_text(size = 12))
# 2. Gene frequency bar plots
panel2 <- p$gg_barplot() +
    theme_bw(base_size = 15) +
    labs(subtitle = "B") +
    theme(axis.title = element_text(size = 12),
      axis.text = element_text(size = 12)) +
    geom_bar(stat = "identity", color = "black", fill = "black")
# 3. PCA of accessory genes colored by host
panel3 <- p$gg_pca(color = "Host", size = 4) +
    theme_bw(base_size = 15) +
    labs(subtitle = "C") +
    guides(color = guide_legend(nrow = 2, byrow = T)) +
    theme(legend.position = "bottom",
      legend.title = element_blank(),
      legend.text = element_text(size = 10),
      axis.title = element_text(size = 12),
      axis.text = element_text(size = 12))
# 4. Pie chart of core and accessory genes
panel4 <- p$gg_pie() + theme_bw(base_size = 15) +
    scale_fill_brewer(palette = "Blues") +
    scale_x_discrete(breaks = c(0, 25, 50, 75)) + labs(subtitle = "D") +
    theme(legend.position = "bottom", legend.title = element_blank(),
      legend.text = element_text(size = 10),
      legend.margin = margin(0, 0, 13, 0), legend.box.margin = margin(0, 0, 5, 0),
      axis.title = element_blank(), axis.ticks = element_blank(),
      axis.text.x = element_blank())
```

. Continued

```
# 5. Use patchwork to arrange plots using math operators

library(patchwork)

figure <- (panel1 + panel2) / (panel3 + panel4)
```

Note: Pagoo includes a responsive visualization dashboard through a R-Shiny application that allows the user to explore pangenome characteristics and perform standard comparative analyses like those shown in Figure 3. For a pangenome object named p, deploy the Shiny application running p$runShinyApp().

25. **Core genome phylogeny and population structure.** The following recipe (Box 28) shows how to build a phylogenetic tree directly from the Pagoo object by using concatenated alignments of each individual core gene. This is performed by interacting with diverse R packages like Biostrings and DECIPHER (Wright, 2015) for sequence handling and alignment, phangorn (Schliep, 2011) for phylogenetic reconstruction, and ggtree (Yu et al., 2017) for tree visualization.

Box 28

```
# Load required packages

library(magrittr)

library(DECIPHER)

library(Biostrings)

library(phangorn)

library(ggtree)

library(rhierbaps)

# Drop Cft and set core level to 100

cft <- p$organisms[which(p$organisms$Subspecies=="Cft"),"org"]

p$drop(cft)

p$core_level <- 100

# Align individual core genes

ali <- p$core_seqs_4_phylo() %>%

  lapply(DECIPHER::AlignTranslation)

# Identify neutral core clusters using Tajima's D

tajD <- ali %>%

    lapply(ape::as.DNAbin) %>%

    lapply(pegas::tajima.test) %>%

    sapply("[[", "D")

neutral <- which(tajD <= 2 & tajD >= -2)

# Concatenate neutral core gene clusters

concat_neu <- ali[neutral] %>%

     do.call(Biostrings::xscat, .) %>%

     setNames(p$organisms$org) %>%
```

. Continued

```
    as("matrix") %>%
    tolower()
# Find population structure with RhierBAPS
rhb <- hierBAPS(snp.matrix = concat_neu, n.pops = 10,
      max.depth = 1, n.extra.rounds = 5)
# Add lineage information to organisms metadata
res <- rhb$partition.df
lin <- data.frame(org = as.character(res[, 1]),
        lineage = as.factor(res[, 2]))
p$add_metadata(map = "org", data = lin)
# Compute phylogeny
tre <- concat_neu %>%
  phangorn::phyDat(type = "DNA") %>%
  phangorn::dist.ml() %>%
  phangorn::NJ()
# Draw phylogeny with lineage and host information
gg1 <- ggtree(tre, ladderize = T, layout = "slanted") %<+%
  as.data.frame(p$organisms) +
  geom_tippoint(aes(color = as.factor(lineage))) +
  labs(subtitle = "A") +
  scale_color_discrete("Lineage")
gg2 <- ggtree(tre, ladderize = T, layout = "slanted") %<+%
  as.data.frame(p$organisms) +
  geom_tippoint(aes(colour = as.factor(Host))) +
  labs(subtitle = "B") +
  scale_colour_discrete("Host")
fig2 <- gg1 + gg2
```

Results presented in Figure 4 exemplify how a relatively complex task that needs of many steps like extracting core genes, keeping those that show signal of neutral evolution, aligning and concatenating them, determining population structure to finally perform a phylogenetic reconstruction, can be achieved directly from the Pagoo pangenome object with different sequential code recipes using different microbial genomics packages within the R environment.

Step 8: Saving and loading pangenome data

Timing: 5 min

Once the pangenome object is created, Pagoo provides two methods for saving and reloading it to a new R session:

26. Saving as plain text files (Box 29):

Box 29

```
outdir <- paste(".", "my_pangenome", sep = "/")

p$write_pangenome(dir = outdir)

list.files(outdir, full.names = TRUE)

## [1] "./my_pangenome/clusters.tsv"

## [2] "./my_pangenome/data.tsv"

## [3] "./my_pangenome/organisms.tsv"
```

This will create a directory with 3 text files. The advantage of this approach is that you can analyze it outside R, the disadvantage is that full reproducibility can be compromised since reading text could be less stable (classes or number precision can be lost).

27. Saving as R data format (Box 30):

Box 30

```
rds <- paste("./my_pangenome", "pangenome.RDS", sep = "/")

p$save_pangenomeRDS(file = rds)

p2 <- load_pangenomeRDS(rds)
```

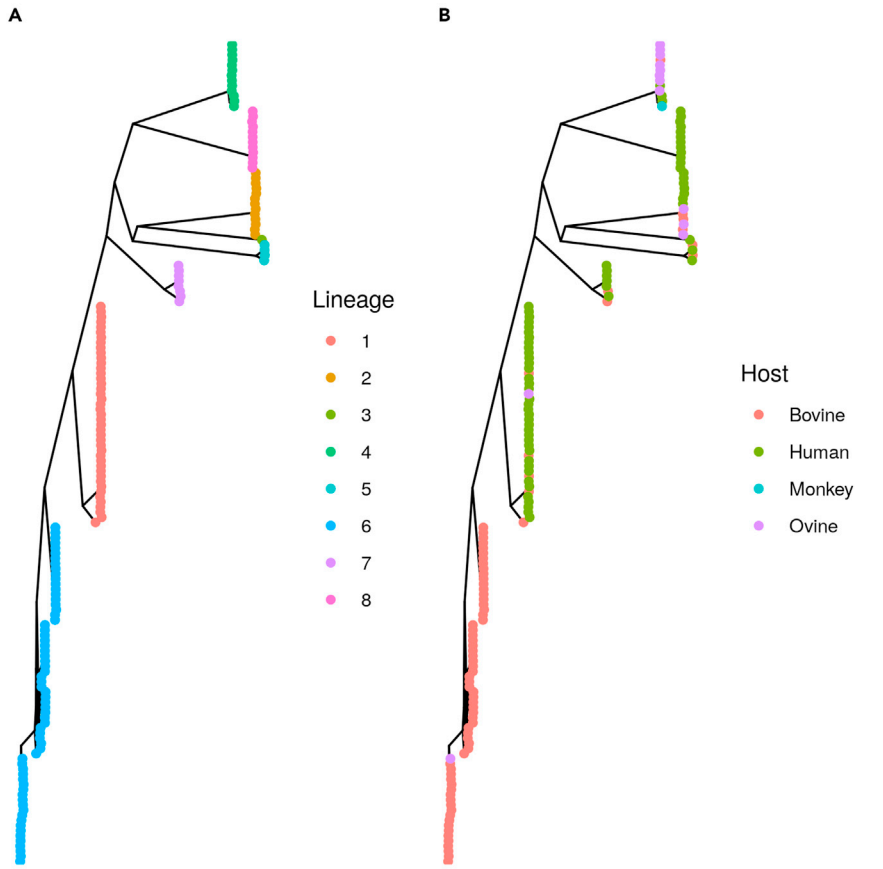


Figure 4. Core genome phylogenies

This shows the output of the above-described recipe aiming to generate a core genome phylogeny directly from the pangenome object. Panel (A) shows the three colored by lineage defined in the same recipe through a population structure analysis and panel (B) shows the tree colored by host.

This solution preserves data structures, such as column data types (numeric, character or factor). So, this method is more stable (compatible between Pagoo versions), secure (uses the same metadata classes), and convenient (the exact state of the object can be saved and restored, keeping all modifications performed to the object during the analysis). Importantly, this option allows to store all pangenome data in a single and easily shareable file.

Note: Authors recommend using the R data format for saving and loading Pagoo objects.

EXPECTED OUTCOMES

Pangenome analysis is a key approach to explore genetic diversity occurring in bacterial populations. Despite the availability of many different software tools for pangenome reconstruction, there are much less alternatives to perform downstream pangenome data analysis in a simple and standardized way. Pagoo is a flexible and extensible tool that systematize pagenome data in a single programming environment, allowing dynamic data analysis and integration with widely used packages for comparative genomics available in the R ecosystem. This protocol aims to be a guide for the use of Pagoo, providing the user with the fundamental skills to work with pangenome data within the R environment. Beyond the user will be able to perform standard phylogenetic analyses, genotype-phenotype association analyses or functional enrichment analyses, Pagoo is an open and flexible framework that allows the development of new modules or code recipes to fulfill specific tasks. Additionally, Pagoo facilitates data storage and sharing, since all data can be saved in a single file that can be recovered later in an independent R session or written to plain text.

LIMITATIONS

Despite the R ecosystem being vast and currently including dozens of packages that allow performing diverse comparative genomic analyses, it is possible that the user finds some specific tasks that are difficult to implement in R or are better covered by other pangenome analysis tools. Pagoo has been tested with hundreds to few thousands of genomes. A limiting aspect here is the time it can take to generate the Pagoo object from bigger datasets. This is particularly relevant for dynamic visualizations that Pagoo provides through its Shiny application.

TROUBLESHOOTING

Problem 1

Copying and modifying the copy of a pangenome object also modifies the original one.

Potential solution

R6 classes are environments which use reference semantics. That means that these kinds of objects need a special method to get copied and if they are simply assigned to a new variable, this variable will point to the original one, and will not get duplicated. To avoid this, use the $clone() method to generate an independent copy of the object.

Problem 2

An error occurs when trying to load a pangenome.

Potential solution

Check that key columns ('gene', 'cluster', and 'org') contain the same elements between the different tables (gene, cluster and organisms). Pagoo checks that these match each other and raises an error if there are missing or different key values. See step 2, Boxes 2, 3, 4, 5, 6, 7, 8, and 9.

Problem 3

I dropped some organisms but I don't remember which ones and also want to recover them.

Potential solution

Use the $dropped field to look if there are some hidden organisms, and then use the $recover() method to recover them. For example, if all dropped organisms want to be recovered from a pangenome object p, then use p$recover(p$dropped). See step 5, section 19.

Problem 4

The state of the pangenome object has changed after saving it and loading in a different R session.

Potential solution

Use save_pangenomeRDS() and load_pangenomeRDS() for saving and loading, respectively. This will load the pangenome object exactly in the same state as it was saved. See step 8, Box 30.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Gregorio Iraola (giraola@pasteur.edu.uy).

Materials availability

This study did not generate new unique reagents.

Data and code availability

The published article includes all datasets/code generated or analyzed during this study.

ACKNOWLEDGMENTS

This work has been partially funded by Banco de Seguros del Estado (BSE) from Uruguay and Fondo de Convergencia Estructural del Mercosur (FOCEM) grant COF 03/11. I.F. is funded by grant ANII-POS_NAC_2018_1_151494 from Agencia Nacional de Investigación e Innovación (ANII), Uruguay. We thank Pablo Fresia, Andrés Parada, and Daniela Costa for insightful comments and suggestions during testing of Pagoo.

AUTHOR CONTRIBUTIONS

I.F. and G.I. conceived the idea. I.F. developed software and generated the analytical protocols described here. I.F. and G.I. wrote the manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Bayliss, S.C., Thorpe, H.A., Coyle, N.M., Sheppard, S.K., and Feil, E.J. (2019). PIRATE: A fast and scalable pangenomics toolbox for clustering diverged orthologues in bacteria. GigaScience *8*, giz119.

Ding, W., Baumdicker, F., and Neher, R.A. (2018). panX: pan-genome analysis and exploration. Nucleic Acids Res. *46*, e5.

Ferrés, I., and Iraola, G. (2021). An object-oriented framework for evolutionary pangenome analysis. Cell Reports Methods *1*, S2667-2375(21)00140-5.

Iraola, G., Forster, S.C., Kumar, N., Lehours, P., Bekal, S., García-Peña, F.J., Paolicchi, F., Morsella, C., Hotzel, H., Hsueh, P.-R., et al. (2017). Distinct Campylobacter fetus lineages adapted as livestock pathogens and human pathobionts in the intestinal microbiota. Nat. Commun. *8*, 1367.

Page, A.J., Cummins, C.A., Hunt, M., Wong, V.K., Reuter, S., Holden, M.T.G., Fookes, M., Falush, D., Keane, J.A., and Parkhill, J. (2015). Roary: rapid large-scale prokaryote pan genome analysis. Bioinformatics *31*, 3691–3693.

Schliep, K.P. (2011). phangorn: phylogenetic analysis in R. Bioinformatics *27*, 592–593.

Seemann, T. (2014). Prokka: rapid prokaryotic genome annotation. Bioinformatics *30*, 2068–2069.

Tonkin-Hill, G., MacAlasdair, N., Ruis, C., Weimann, A., Horesh, G., Lees, J.A., Gladstone, R.A., Lo, S., Beaudoin, C., Floto, R.A., et al. (2020). Producing polished prokaryotic pangenomes with the Panaroo pipeline. Genome Biol. *21*, 180.

Wright, E.S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics *16*, 322.

Yu, G., Smith, D.K., Zhu, H., Guan, Y., and Lam, T.T.-Y. (2017). ggtree: an r package for visualization and annotation of phylogenetic trees with their covariates and other associated data. Methods Ecol. Evol. *8*, 28–36.

Zhou, Z., Charlesworth, J., and Achtman, M. (2020). Accurate reconstruction of bacterial pan- and core genomes with PEPPAN. Genome Res. *30*, 1667–1679.