Supplementary Bioinformatics Methods

This document contains an explanation of the bioinformatics methods required for incompatibility group/replicon typing and plasmid characterization. It is expanded from our paper in *Genome*. This document will begin with an overview of the process and will be followed by a detailed description of the methods.

Overview

The process begins with one fasta file and multiple GenBank files. The fasta file is the local download of the PlasmidFinder database referenced in our paper. The GenBank files come from the Entrez search strategy also described in the paper. The ultimate output is a CSV file and text-based report file for each input GenBank file. The CSV file contains basic information (e.g., plasmid length), the incompatibility group(s) the plasmid best aligns to, accession numbers of identical plasmids, some gene/function annotation based on key term searches of the GenBank file's CDS regions, and some other metadata extracted from the GenBank files. The text-based report is a file containing various information and statistics about each group of plasmids from the various input GenBank files. We also generated a tree to help visualize the identical plasmids.

Our process occurs stepwise, with most steps requiring the output from the previous steps. As our project developed, additional steps were inserted or modified. While most steps do depend on the output of the previous step(s), the order is in many instances arbitrary. The code is published online in this GitHub repository (https://github.com/ridgelab/plasmidCharacterization). Each output CSV file requires the following input processed from the “raw” input data (in no particular order): (a) a list of identical plasmids for each accession, (b) extracted metadata from the GenBank files, (c) gene/function annotations extracted from the GenBank files, and (d) a list of incompatibility groups. Each output statistics report file is created based on each CSV file just described.

Identical Plasmids. First, a blast database was created with makeblastdb. Each plasmid sequence (which would have to be extracted from the GenBank files) is aligned with blastn to each other plasmid sequence in a pairwise fashion. Hits were kept only if the percent identity was >=98%. Plasmids were considered identical if the hits covered >=98% of both the query and the subject sequence. We created a tree using a simple distance metric to help visualize the identical plasmids. The distance metric is the sum of the query and subject covered bases divided by the sum of the length of the query and subject sequences (see step 25 for details). The Newick formatted tree was made from the distance matrix using the makeNewick.py script from CAM (Miller et al. 2019) and is available on GitHub at https://github.com/ridgelab/cam. makeblastdb and blastn are part of the BLAST+ Suite (Altschul et al. 1990; Camacho et al. 2009).

GenBank Metadata. The sequencing technology used to sequence each plasmid was identified with GNU AWK. The remaining metadata was also obtained from the GenBank files using GNU AWK. The remaining data points are as follows: country of origin for the plasmid, isolation source for the plasmid, plasmid collection data, and source organism.

GenBank Annotations. This is by far the most complicated part of the process. First, search regions were extracted from the GenBank files. The search regions were the function, gene, note, and product sections of the CDS features. We then identified matches in these regions to key terms (these key terms were obtained as described in our paper). The search occurred under the following strategy:

The search terms are each part of one or more categories. Each can belong to multiple categories, but only if the categories are subsets of each other. Five principal categories exist, two of which have subcategories. The category structure is as follows:

- Antimicrobial Resistance
    - Beta-lactamase
        - Beta-lactamase Special
- Toxin/Antitoxin System
- DNA Maintenance/Modification
    - DNA Maintenance/Modification Special
- Mobile Genetic Elements
- Hypothetical Genes

The strategy could be described as top-to-bottom, in-to-out; i.e., Antimicrobial Resistance is more important that Toxin/Antitoxin System and Beta-lactamase Special is more important than Beta-lactamase and Antimicrobial Resistance. The reason these are shown nested instead of simply above their parents is because a match for a Beta-lactamase Special search term will increment the count for not only itself, but also its parents. If no matches are found, the CDS region being searched is classified as "Other". Some CDS regions will never be searched for these terms if they first match a term in a special "Ignored" category. Provided a CDS region is not to be ignored, it will be searched with Beta-lactamase Special terms, then Beta-lactamase terms, then Antimicrobial Resistance Terms, then Toxin/Antitoxin System terms, and so-forth, until a match is found (thus halting the search on this CDS region) or no more search terms remain, in which case it is assigned to the "Other" category. All CDS regions are converted to lowercase before being searched as described. These terms are listed, with their associated Python regular expressions, in the `doc` directory of the online repository.

Incompatibility Groups. The incompatibility fasta sequences were downloaded from the PlasmidFinder database as previously described. This was turned into a database using makeblastdb. Each plasmid sequence was then aligned to the database using blastn and hits were retained only if the percent identity was >=80%. Hits were further dropped if the subject (the sequences in the database) coverage was <60%. The “best” hits were then used to determine which incompatibility group(s) applied to each plasmid. “Best” is defined as the result(s) with the highest percent identity and those that have percent identities within only 1 percent of the highest one. makeblastdb and blastn are part of the BLAST+ Suite (Altschul et al. 1990; Camacho et al. 2009).

Detailed Methods

This section is a more detailed explanation of the bioinformatics methods required for incompatibility group/replicon typing and plasmid characterization. Please note that most of these steps will be simple data formatting. Also note that it would have been easier in some cases to combine multiple steps into one. The choice to separate each piece of the process was for clarity and to enable another to modify this process for their own purposes. Additionally, some steps might have made better sense in different orders. This process evolved as the project changed; we recognize alternate orders are plausible. For our work, all steps could be run interactively; i.e., not requiring a high-performance computing (HPC) architecture. Our work was completed on a machine running Red Hat Enterprise Linux.

Summary

This process begins with one fasta file and multiple GenBank files. The formats for these files are described in steps 1 and 3, respectively. The fasta file contains the incompatibility group sequences. In our work, this was a download of the PlasmidFinder v1.3 *Enterobacteriaceae* database (Carattoli et al. 2014). The GenBank files contain one or more GenBank records in them, where each record could itself be considered a GenBank file for a single accession number. Thus, these GenBank files are concatenations of multiple GenBank records. Effectively, this is how we grouped accessions of interest. The same accession may appear in multiple groupings. Note, if you attempt to re-use our process with your own data and have GenBank files as a single file per accession, combining them into groups will feel unnecessary. We began this way because that is what we had to start with.

The results of the entire process are CSV files with information about each plasmid in a group and a text file with summary statistics about each group. The file contains basic information (e.g., plasmid length), the incompatibility group(s) the plasmid best aligns to, and some gene/function annotation based on key term searches of the GenBank file's CDS regions. To accomplish this, each (input) group GenBank file is split into a single GenBank file per accession and the sequences are extracted as fasta files. The sequence lengths are recorded and these sequences are individually aligned (using the NCBI BLAST+ Suite (Altschul et al. 1990; Camacho et al. 2009)) to the incompatibility group sequences. After filtering out the "best" alignments, the incompatibility group is determined and saved for later assimilation into the final outputs. The CDS regions are extracted from the GenBank files and searched for key terms using regular expressions. Each key term belongs to one or more categories. Matches in each category are counted and summarized in the final output. For more details on this searching strategy, please see step #14. The key terms are listed with their Python regular expression in the supplement of our paper. Additional information, e.g., sequencing platforms, country, etc., is also available in the final outputs.

This summary concludes with an outline of the steps. Each step will then be addressed in detail. The code in the detailed steps has, in many cases, been simplified. In other cases, the code is several pages long and would be difficult to copy and paste effectively. Especially with the Python code, readability suffers as lines wrap because a standard page is not wide enough to contain some code statements on a single line. Accordingly, we encourage you to visit the online repository for the code: https://github.com/ridgelab/plasmidCharacterization.

Outline of Steps

Step 1. Format Incompatibility Groups Fasta File
Step 2. Create Incompatibility Groups BLAST Database
Step 3. Split Multi-Accession GenBank Files
Step 4. Extract ORIGIN Sequence from GB to Fasta
Step 5. Extract Group Lists
Step 6. BLAST Incompatibility Groups
Step 7. Subset BLAST Results by Coverage Cutoff of 60%
Step 8. Add Incompatibility Group Family as Column to BLAST Results
Step 9. Filter Best Matches in BLAST Results
Step 10. Extract Incompatibility Families
Step 11. Extract Sequencing Technologies
Step 12. Extract Source Information
Step 13. Extract Plasmid Search Regions
Step 14. Identify Plasmid Matches
Step 15. Summarize Plasmid Matches
Step 16. Drop Plasmids
Step 17. Create Plasmid BLAST Database

Step 18. BLAST Plasmid
Step 19. Extract Identical Plasmids with BLAST Result Coverage Cutoff of 98%
Step 20. Fix Identical Plasmid Non-concordance
Step 21. Generate Plasmid CSVs
Step 22. Create Group CSVs from Plasmid CSVs
Step 23. Create Group Matches from Plasmid Matches
Step 24. Calculate Group Statistics from Group CSV
Step 25. Create Distance Matrix
Step 26. Create Distance Tree
Step 27. Add Leaf Labels to Tree
Step 28. Add Color to Leaf Labels

Step 1. Format Incompatibility Groups Fasta File

Input: Fasta file with incompatibility group sequences. Each sequence may be on one or more lines. The headers might start with “Inc”.

Output: Same fasta file as the input, but sequences occur on only one line. Headers without “Inc” now have “Inc” prepended.

Code:

Bash Command

```
awk -f formatIncGroupFasta.awk \
         original_incomp-grp.fasta \
         > incomp-grp.fasta
```

AWK Script (formatIncGroupFasta.awk)

```
#! /bin/awk -f

{
  if ( $0 ~ /^>.+$/ ) {

         if ( NR != 1 ) {
                 printf "\n";
         }

         if ( $0 ~ /^>Inc.+$/ ) {
                 print $0;
         }
         else {
                 printf "%s%s\n", ">Inc", substr($0, 2);
         }
  }
  else {
         printf "%s", $0;
  }
}

END {
  printf "\n";
}
```

Step 2. Create Incompatibility Groups BLAST database

Input: Fasta file with incompatibility group sequences. Each sequence is on only one line. The headers start with “>Inc”.

Output: BLAST database of the incompatibility group sequences.

Code:

Bash Command

```
makeblastdb \
         -dbtype nucl \
         -in  incompatibility.fasta \
         -input_type fasta \
         -title incompatibility \
         -parse_seqids \
         -hash_index \
         -out incompatibility \
         -max_file_sz 2GB \
         -logfile makeBlastDB.log
```

BLAST Software

NCBI (United States National Center for Biotechnology Information) BLAST+ Suite version 2.4.0 (Altschul et al. 1990; Camacho et al. 2009).

Step 3. Split Multi-Accession GenBank Files

Input: 1+ GenBank files, each with 1+ records. Each record is itself a GenBank file for a single Accession. Thus, the multi-accession GenBank files are simply concatenations of multiple single-accession GenBank files. Assume that these GenBank files are in a directory called `original_gb`.

Output: One GenBank file for each accession. If the same accession exists in more than one multi-accession file, assume they are the same and overwrite it. Assume that the output GenBank files will be in a directory called `plasmid_gb`.

Code:

Bash Command

```
cd plasmid_gb

while read ifn
do
         awk -f splitMultiGB.awk "${ifn}"

done < <(ls -1 original_gb/*.gb)
```

AWK Script (splitMultiGB.awk)

```
#! /bin/awk -f

BEGIN {
  FS="[ ]+";
  accession="";
  ofn="";
}

{
  if ($0 == "//" || $0 == "")
  {
         accession = "";
         ofn = "";
  }
  else if ($1 == "LOCUS")
  {
         accession = $2;
         ofn = accession ".gb";
         print $0 > ofn;
  }
  else
  {
         print $0 >> ofn;
  }
}

END {
  print "done splitting " FILENAME " by accession";
}
```

Step 4. Extract ORIGIN Sequence from GB to Fasta

Input: One GenBank file with a single accession in it. Assume it is in the directory `plasmid_gb` and it is named after the pattern `${ACCESSION}.gb`.

Output: One Fasta file with the sequence from the ORIGIN section of the GenBank file. The Fasta file has sequences that are each on only one line. It will be in the directory `plasmid_fasta`.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" ".gb"`

         awk -f extractOriginSeqFromGBtoFasta.awk \
                 "plasmid_gb/${ACCESSION}.gb" \
                 > "plasmid_fasta/${ACCESSION}.fasta"

done < <(ls -1 plasmid_gb/*.gb)
```

AWK Script (extractOriginSeqFromGBtoFasta.awk)

```
#! /bin/awk -f

BEGIN {
  FS = "[ ]+";
  origin_found = 0; # false
}

{
  if (origin_found)
  {
         sub(/ *[0-9]+ /, "", $0);
         gsub(/ +/, "", $0);
         printf toupper($0);
  }
  else if ($1 == "ORIGIN")
  {
         origin_found = 1; # true

         print ">" gensub(/^(.+)\.gb$/, "\\1", "-1", gensub(/^.*\//, "", "-1",
FILENAME));
  }
}

END {
  printf "\n";
  print "done extracting ORIGIN seq from " FILENAME " to fasta" > "/dev/stderr";
}
```

Step 5. Extract Group Lists

Input: One GenBank file with multiple accessions in it. Assume it is in the directory `original_gb` and it is named after the pattern `${GROUP}.gb`.

Output: Multiple text files, each with the extension ".list". Each file is a line separated list of accession numbers that make up the group. The files will be in a directory called `groups` with the name `${GROUP}.list`.

Code:

Bash Command

```
while read ifn
do
         awk -f extractGroupLists.awk \
                 "${ifn}"

done < <(ls -1 original_gb/*.gb)
```

AWK Script (extractGroupLists.awk)

```
#! /bin/awk -f

BEGIN {
  FS="[ ]+";
  accession="";
  ofn="";
}

{
  if (NR == 1)
  {
         ofn = gensub(/^(.+)\.gb$/, "\\1", "-1", gensub(/^.*\//, "", "-1",
FILENAME)) ".list";
  }

  if ($1 == "LOCUS")
  {
         accession = $2;
         print accession >> ofn;
  }
}

END {
  print "done extracting accessions from " FILENAME;
}
```

Step 6. BLAST Incompatibility Groups

Input: Fasta files. Each contains the sequence from a single accession. Assume they are in the directory `plasmid_fasta` and they are named after the pattern `${ACCESSION}.fasta`.

Input: The incompatibility groups BLAST database created in step #1. It is named `incompatibility`.

Output: One tab-separated value file for each input file. Each file is a modified version of the BLAST output format 6. The format is specified as seen using the -outfmt option with blastn. The columns are as follows: qseqid, sseqid, pident, length, evalue, qframe, qlen, qstart, qend, sframe, slen, sstart, send, qseq, and sseq. The files will be in a directory called `blast_results` and named after the pattern `${ACCESSION}_fmt6c.tsv`. Note thaat a match was not included in the output if the percent identity was <80%.

Code:

Bash Command

```
THREADS=8

while read ifn
do
         ACCESSION=`basename "${ifn}" ".fasta"`

         blastn \
                 -query "${ifn}" \
                 -strand both \
                 -task blastn \
                 -db icompatibility \
                 -out blast_results/${ACCESSION}_fmt6c.tsv \
                 -outfmt "6 qseqid sseqid pident length evalue qframe qlen qstart
qend sframe slen sstart send qseq sseq" \
                 -num_threads ${THREADS} \
                 -perc_identity 80

done < <(ls -1 plasmid_fasta/*.fasta)
```

BLAST Software

NCBI (United States National Center for Biotechnology Information) BLAST+ Suite version 2.4.0 (Altschul et al. 1990; Camacho et al. 2009).

Step 7. Subset BLAST Results by Coverage Cutoff of 60%

Input: Tab-separated value files. Each contains the results from blasting the sequence of a single accession against the incompatibility groups BLAST database. Assume they are in the directory `blast_results` and they are named after the pattern `${ACCESSION}_fmt6c.tsv`.

Output: One tab-separated value file for each input file. Each file is a copy of its respective input file except some results may be omitted if the coverage was less than 60%. The files will be in a directory called `blast_results` and named after the pattern `${ACCESSION}_fmt6c_cov60.tsv`. Note that a new column was inserted as column number 14 (1-based indexing). The columns will now be as follows: qseqid, sseqid, pident, length, evalue, qframe, qlen, qstart, qend, sframe, slen, sstart, send, scov, qseq, and sseq.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_fmt6c.tsv"`

         awk -f subCovCutoff60.awk \
                 "${ifn}" \
                 > "blast_results/${ACCESSON}_fmt6c_cov60.tsv"

done < <(ls -1 blast_results/*_fmt6c.tsv)
```

AWK Script (subCovCutoff60.awk)

```
#! /bin/awk -f

BEGIN {
  FS="\t";
  OFS="\t";
  ORS="\n";
  count=0;
}

{
  # 4 = length, 11 = slen, scov = length / slen
  scov = $4 / $11;
  if (scov >= 0.6)
  {
          count += 1

          # keep 1-13, add new column, keep 14-15 (will become 15-16)
          for (i = 1; i <= 13; i++)
          {
                  printf "%s", $i OFS;
          }

          printf "%f", scov OFS;

          for (i = 14; i <= NF; i++)
          {
                  printf "%s", $i (i == NF ? ORS : OFS);
          }
  }
}

END {
  print FILENAME ": " count > "/dev/stderr";
}
```

Step 8. Add Incompatibility Group as Column to BLAST Results

Input: Tab-separated value files. Each contains the results from blasting the sequence of a single accession against the incompatibility groups BLAST database. It has an added column with the subject coverage and has only records with coverage >60%. Assume they are in the directory `blast_results` and they are named after the pattern `${ACCESSION}_fmt6c_cov60.tsv`.

Output: One tab-separated value file for each input file. Each file is a copy of its respective input file except that an additional column is added. This column has the family or root of the incompatibility group from column #2 (sseqid). The files will be in a directory called `blast_results` and named after the pattern `${ACCESSION}_fmt6c_cov60_fam.tsv`. Note that a new column was inserted as column number 3 (1-based indexing). The columns will now be as follows: qseqid, sseqid, fam, pident, length, evalue, qframe, qlen, qstart, qend, sframe, slen, sstart, send, scov, qseq, and sseq.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_fmt6c_cov60.tsv"`

         awk -f addFamCol.awk \
                "${ifn}" \
                > "blast_results/${ACCESSON}_fmt6c_cov60_fam.tsv"

done < <(ls -1 blast_results/*_fmt6c_cov60.tsv)
```

AWK Script (addFamCol.awk)

```
#! /bin/awk -f

BEGIN {
  FS="\t";
  OFS="\t";
  ORS="\n";
}

{
  # 2 = subject_id, keep 1-2, add new column, keep 3-16 (will become 4-17)
  for (i = 1; i <= 2; i++)
  {
         printf "%s", $i OFS;
  }

  printf "%s", gensub(/^([^(_]+).*$/, "\\1", "-1", $2) OFS;

  for (i = 3; i <= NF; i++)
  {
         printf "%s", $i (i == NF ? ORS : OFS);
  }
}
```

Step 9. Filter Best Matches in BLAST Results

Input: Tab-separated value files. Each contains the results from blasting the sequence of a single accession against the incompatibility groups BLAST database. It has two added columns with the subject coverage (and has only records with coverage >60%) and family. Assume they are in the directory `blast_results` and are named after the pattern `${ACCESSION}_fmt6c_cov60_fam.tsv`.

Output: One tab-separated value file for each input file. Each file is a copy of its respective input file except that some results are omitted. The "best" results are retained. "Best" is defined as the result(s) with the highest percent identity and those that have percent identities within only 1 percent of the highest one. The files will be in a directory called `blast_results` and named after the pattern `${ACCESSION}_fmt6c_cov60_fam_best.tsv`. As in the input file, the columns will be as follows: qseqid, sseqid, fam, pident, length, evalue, qframe, qlen, qstart, qend, sframe, slen, sstart, send, scov, qseq, and sseq.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_fmt6c_cov60_fam.tsv"`

         python3 filterBestResults.py \
                 "${ifn}" \
                 > "blast_results/${ACCESSON}_fmt6c_cov60_fam_best.tsv"

done < <(ls -1 blast_results/*_fmt6c_cov60_fam.tsv)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (filterBestResults.py)

This script has at least one line that is too long to represent in this document without sacrificing readability. Please view it in the freely-accessible online repository.

Step 10. Extract Incompatibility Families

Input: Tab-separated value files. Each contains the results from blasting the sequence of a single accession against the incompatibility groups BLAST database. It has two added columns with the subject coverage (and has only records with coverage >60%) and family. Only the “best” results remain. Assume they are in the directory `blast_results` and are named after the pattern `${ACCESSION}_fmt6c_cov60_fam_best.tsv`.

Output: One file for each input file. Each file is a line-delimited list of incompatibility group roots/families. The files will be in a directory called `blast_results` and named after the pattern `${ACCESSION}_families.list`.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_fmt6c_cov60_fam_best.tsv"`

         cut -f 3 "${ifn}" \
                 | sort \
                 | uniq \
                 > blast_results/"${ACCESSON}_families.list"

done < <(ls -1 blast_results/*_fmt6c_cov60_fam_best.tsv)
```

Step 11. Extract Sequencing Technologies

Input: GenBank files for each plasmid. We assume they are in the directory `plasmid_gb` and they are named after the pattern `${ACCESSION}.gb`.

Output: One tab-separated value file. The file has one column for the accession number, one column containing the sequencing technology string taken from the GenBank file, and several columns containing counts for the various sequencing technologies and groups of technologies. The file is assumed to be called `seqTechs.tsv` in the `plasmid_seqTech` directory. The columns are as follows: accession, sequencing_technologies, num_total, num_short, num_long, num_illumina, num_454, num_abi, num_sanger, num_torrent, num_pacbio, and num_nanopore.

Code:

Bash Command

```
printf "%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n" \
  'accession' \
  'sequencing_technologies' \
  "num_total" \
  "num_short" \
  "num_long" \
  "num_illumina" \
  "num_454" \
  "num_abi" \
  "num_sanger" \
  "num_torrent" \
  "num_pacbio" \
  "num_nanopore" \
  > "plasmid_seqTech/seqTech.tsv"

while read ifn
do
         ACCESSION=`basename "${ifn}" ".gb"`

         printf '%s\t' "${ACCESSION}" >> "plasmid_seqTech/seqTech.tsv"

         awk -f sequesterSeqTech.awk \
                 "${ifn}" \
                 >> "plasmid_seqTech/seqTech.tsv"

done < <(ls -1 plasmid_gb/*.gb)
```

AWK Script (sequesterSeqTech.awk)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 12. Extract Source Information

Input: GenBank files for each plasmid. We assume they are in the directory `plasmid_gb` and they are named after the pattern `${ACCESSION}.gb`.

Output: One tab-separated value file. The file has one column for the accession number and one column for each of these subsections of the GenBank file source section: organism, isolation source, country, and collection_date. The file is assumed to be called `sourceInfo.tsv` in the `plasmid_sourceInfo` directory. The columns are as follows: accession, organism, isolation_source, country, and collection_date.

Code:

Bash Command

```
printf "%s\t%s\t%s\t%s\t%s\n" \
  'accession' \
  'organism' \
  'isolation_source' \
  'country' \
  'collection_date' \
  > "plasmid_sourceInfo/sourceInfo.tsv"

while read ifn
do
         ACCESSION=`basename "${ifn}" ".gb"`

         printf '%s\t' "${ACCESSION}" >> "plasmid_sourceInfo/sourceInfo.tsv"

         awk -f snagSourceInfo.awk \
                 "${ifn}" \
                 >> " plasmid_sourceInfo/sourceInfo.tsv "

done < <(ls -1 plasmid_gb/*.gb)
```

AWK Script (snagSourceInfo.awk)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 13. Extract Plasmid Search Regions

Input: This Python program requires 3 inputs. 1- The accession number of the plasmid it will extract the search regions from. 2- The directory where the output will be placed. 3- The directory where the GenBank file is located for that plasmid. We assume the GenBank file is named after the pattern `${ACCESSION}.gb`.

Output: One text file containing the lines from input GenBank file that will be searched using the key terms. We assume the output file will be named after the following pattern: `${ACCESSION}_searchRegions.txt`. For convenience, it will also generate a copy of the input GenBank file with shell color codes, marking the CDS and source regions in blue, the portions of the CDS and source regions that will be included in green, and the portion of the CDS and source regions that will not be searched in red. The FEATURE line will be blue. This file will have the same name as the `.txt` file, but will have the extension `.gb` instead of `.txt`. Note that intended search space is to consider each CDS region as a separate entity. However, only the following subsections of each CDS region are to be considered: `/function`, `/gene`, `/note`, and `/product`.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" ".gb"`

         python3 extractPlasmidSearchRegions.py \
                 "${ACCESSION}" \
                 plasmid_searchRegions \
                 plasmid_gb

done < <(ls -1 plasmid_gb/*.gb)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (extractPlasmidSearchRegions.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 14. Identify Plasmid Matches

Input: This Python program requires 3 inputs. 1- The accession number of the plasmid in which it will identify matches. 2- The directory where the input search regions file is located. 3- The directory where the output matches will be placed. We assume the input search regions file is named after the pattern `${ACCESSION}_searchRegions.txt`.

Output: One tab-separated value file containing matches. We assume the output file will be named after the following pattern: `${ACCESSION}_matches.tsv`. The columns of the file are as follows:

1. Ignored (True/False)
2. Categories (c1[,c2,…,cN])
3. Search Term
4. CDS Region

Column 1 is a simple flag denoting if the term was to be ignored. This could also be determined based on the second column, but it was convenient to have a simple flag as its own column. Column 2 contains the category (categories) that the search term belonged to. Column 3 contains the regular expression used. Column 4 contains the CDS region that was searched (all tabs and newlines were converted to `\t` (backslash and a t, not a tab) and `\n` (backslash and an n, not a newline) to not interfere with the tab-separated value file format and keep each record on a single line).

Search Strategy: The search terms are each part of one or more categories. It can belong to multiple categories only if the categories are subsets of each other. Five principal categories exist, two of which have subcategories. The category structure is as follows:

- Antimicrobial Resistance
    - Beta-lactamase
        - Beta-lactamase Special
- Toxin/Antitoxin System
- DNA Maintenance/Modification
    - DNA Maintenance/Modification Special
- Mobile Genetic Elements
- Hypothetical Genes

The strategy could be described as top-to-bottom, in-to-out; i.e., Antimicrobial Resistance is more important that Toxin/Antitoxin System and Beta-lactamase Special is more important than Beta-lactamase and Antimicrobial Resistance. The reason these are shown nested instead of simply above their parents is because a match for a Beta-lactamase Special search term will increment the count for not only itself, but also its parents. If no matches are found, the CDS region being searched is classified as "Other". Some CDS regions will never be searched for these terms if they first match a term in a special "Ignored" category. Provided a CDS region is not to be ignored, it will be searched with Beta-lactamase Special terms, then Beta-lactamase terms, then Antimicrobial Resistance Terms, then Toxin/Antitoxin System terms, and so-forth, until a match is found (thus halting the search on this CDS region) or no more search terms remain (it is assigned to the "Other" category). All CDS regions are converted to lowercase before being searched as described. See our paper for a table of search terms.

Code:

Bash Command

```
while read ifn
do
        ACCESSION=`basename "${ifn}" "_searchRegions.txt"`

        python3 identifyPlasmidMatches.py \
                "${ACCESSION}" \
                plasmid_searchRegions \
                plasmid_matches

done < <(ls -1 plasmid_searchRegions/*_searchRegions.txt)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (identifyPlasmidMatches.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 15. Summarize Plasmid Matches

Input: This Python program requires 2 inputs. 1- The accession number of the plasmid in which it will summarize matches. 2- The directory where the input matches are to be found and the output summarized matches will be placed. We assume the input matches file is named after the pattern `${ACCESSION}_matches.tsv` in a directory called `plasmid_matches`.

Output: One tab-separated value file containing summarized matches. It will have two lines only. The first is a header line; the second the data. We assume the output file will be named after the following pattern: `${ACCESSION}_matches-summary.tsv` in a directory called `plasmid_matches`. The columns of the file are as follows:

1. Accession #
2. Antimicrobial Resistance CDS
3. Antimicrobial Resistance CDS %
4. Beta-lactamase CDS
5. Beta-lactamase CDS %
6. Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy #
7. Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy # % of Beta-lactamase
8. Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy # % of Total
9. Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Absent (Yes/No)
10. Plasmid Transfer CDS
11. Plasmid Transfer CDS %
12. Toxin/Antitoxin System CDS
13. Toxin/Antitoxin System CDS %
14. Toxin/Antitoxin System Present (Yes/No)
15. DNA Maintenance/Modification CDS
16. DNA Maintenance/Modification CDS %
17. DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy #
18. DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy # % of DNA Maintenance/Modification
19. DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy # % of Total
20. DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Present (Yes/No)
21. Mobile Genetic Elements CDS
22. Mobile Genetic Elements CDS %
23. Hypothetical Genes CDS
24. Hypothetical Genes CDS %
25. Other CDS
26. Other CDS %
27. Total CDS

This data, with the exception of the first column, will be copied into the plasmid csv file created later. Column number 6 will also be used to drop plasmids.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_sorted_matches.tsv"`

         python3 summarizePlasmidMatchInfo.py \
                 "${ACCESSION}" \
                 plasmid_matches

done < <(ls -1 plasmid_matches/*_sorted_matches.tsv)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (summarizePlasmidMatchInfo.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 16. Drop Plasmids

Input: This script acts on all the plasmids directly (i.e., not calling on a subroutine in Python or AWK for each of the plasmids). It requires no user input directly as it ascertains the plasmid accession numbers from file names. It also relies on the directory structure to find the files named after the pattern `${ACCESSION}_matches-summary.tsv` in a directory called `plasmid_matches`.

Output: Two new directories in the `groups` directory: `keep` and `discard`. Inside the `discard` directory will be a file called `discard.list`. It will contain the accession numbers (one per line) that are to be excluded from the rest of the analysis. The same is true in the `keep` directory, except the accession numbers are the ones that will be retained for the rest of the analysis and the file will be called `keep.list`. Also in the `keep` directory is a new group list file for each of the groups found in `groups` directory. These lists are the same as the originals except that the discarded accessions have been removed.

Code:

Bash Command

```
while read ifn
do
        GROUP=`basename "${ifn}" ".list"`

        while read ACCESSION
        do
                COUNT=`tail -n 1 \
                        "plasmid_matches/${ACCESSION}_matches-summary.tsv" \
                        | cut -d '\t' -f 6 \
                        | tr -d '"'`

                if [ $COUNT -ge 1 ] && [ $COUNT -le 6 ]
                then
                        printf "${ACCESSION}\n" >> "groups/keep/${GROUP}.list"
                        printf "${ACCESSION}\n" >> "groups/keep/keep.list"
                else
                        printf "${ACCESSION}\n" >> "groups/discard/discard.list"
                fi

        done < "${ifn}"

done < <(ls -1 groups/*.list)
```

Step 17. Create Plasmid BLAST Database

Input: Fasta files for each plasmid. We assume they are in the directory `plasmid_fasta` and they are named after the pattern `${ACCESSION}.fasta`.

Output: One BLAST database. We are creating this so we can do pairwise BLAST between the plasmid fastas. The objective is to identify plasmids that are "identical". Identical will, for our purposes, be defined as >=98% percent identity and >=98% query *and* subject coverage.

Code:

Bash Command

```
cat plasmid_fasta/*.fasta > plasmid_blast_results/plasmids.fasta

cd plasmid_blast_results

makeblastdb \
         -dbtype nucl \
         -in plasmids.fasta \
         -input_type fasta \
         -title plasmids \
         -parse_seqids \
         -hash_index \
         -out plasmids \
         -max_file_sz 2GB \
         -logfile makeBlastDB.log
```

BLAST Software

NCBI (United States National Center for Biotechnology Information) BLAST+ Suite version 2.4.0 (Altschul et al. 1990; Camacho et al. 2009).

Step 18. BLAST Plasmid

Input: Fasta files. Each contains the sequence from a single accession. Assume they are in the directory `plasmid_fasta` and they are named after the pattern `${ACCESSION}.fasta`.

Input: The plasmids BLAST database created in step #12. It is named `plasmids`.

Output: One tab-separated value file for each input file. Each file is a modified version of the BLAST output format 6. The format is specified as seen using the -outfmt option with blastn. The columns are as follows: qseqid, sseqid, pident, length, evalue, qframe, qlen, qstart, qend, sframe, slen, sstart, send, qseq, and sseq. The files will be in a directory called `plasmid_blast_results` and named after the pattern `${ACCESSION}_fmt6c.tsv`. Note that a match was not included in the output if the percent identity was <98%.

Code:

Bash Command

```
THREADS=8

while read ifn
do
         ACCESSION=`basename "${ifn}" ".fasta"`

         blastn \
                 -query "${ifn}" \
                 -strand both \
                 -task blastn \
                 -db plasmids \
                 -out plasmid_blast_results/${ACCESSION}_fmt6c.tsv \
                 -outfmt "6 qseqid sseqid pident length evalue qframe qlen qstart 
qend sframe slen sstart send qseq sseq" \
                 -num_threads ${THREADS} \
                 -perc_identity 98

done < <(ls -1 plasmid_fasta/*.fasta)
```

BLAST Software

NCBI (United States National Center for Biotechnology Information) BLAST+ Suite version 2.4.0 (Altschul et al. 1990; Camacho et al. 2009).

Step 19. Extract Identical Plasmids with BLAST Result Coverage Cutoff of 98%

Input: Tab-separated value files. Each contains the results from blasting the sequence of a single accession against the plasmids BLAST database. Assume they are in the directory `plasmids_blast_results` and they are named after the pattern `${ACCESSION}_fmt6c.tsv`. Note that these BLAST results all have >=98% sequence identity.

Output: One file for each input file. Each file is a line-delimited list of accessions associated with "identical" plasmids. The files will be in a directory called `plasmid_blast_results` and named after the pattern `${ACCESSION}_identicalPlasmids.list`. The BLAST results are further filtered based on the query and subject coverage; each must be >= 98%. Coverage is determined based on number of bases covered by the other sequence. This coverage can come from one or more BLAST hits, as long as the total number of covered bases is >=98% of the number of possible bases.

Code:

Bash Command

```
while read ifn
do
         ACCESSION=`basename "${ifn}" "_fmt6c.tsv"`

         python3 queryAndSubCovCutoff98-multiHit.py \
                 "${ifn}" \
                 > "plasmid_blast_results/${ACCESSON}_identicalPlasmids.list"

done < <(ls -1 blast_results/*_fmt6c.tsv)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (queryAndSubCovCutoff98-multiHit.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 20. Fix Identical Plasmid Non-concordance

Input: This Python program requires 6 inputs. 1- the path of the coverage information files. 2- the path of the identical plasmid files. Inputs 3-6 are suffixes to file names; the assumed base of the name is the accession number. 3- the suffix of the input coverage info file. 4- the suffix of the output coverage info file. 5- the suffix of the input identical plasmids file. 6- the suffix of the output identical plasmids file.

Output: Two text files. The first will be the output coverage info file. It will be the concordant version of its respective input file. The second will be the output identical plasmids file. It will be the concordant version of its respective input file. We assume they are both in the `plasmid_blast_results` directory and have the suffixes `_covInfo_concordant.tsv` and `_identicalPlasmids_concordant.list`, respectively. Another term for concordance might be reciprocal. This step accounts for inconsistencies in BLAST outputs. One might get hits from sequence A to B with >=98% identity and >=98% query and subject coverage, yet get no hits from B to A. This non-concordance is “fixed” in this step to force reciprocity of the BLAST hits. These hits are not updated in the BLAST output file, though the outcome is affected in the two output files from this step.

Code:

Bash Command

```
python3 fixIdenticalPlasmidsNonConcordance.py \
         plasmid_blast_results \
         plasmid_blast_results \
         "_covInfo.tsv" \
         "_covInfo_concordant.tsv" \
         "_identicalPlasmids.list" \
         "_identicalPlasmids_concordant.list"
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (fixIdenticalPlasmidsNonConcordance.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 21. Generate Plasmid CSVs

Input: This Python program requires 8 inputs. 1- The accession number of the plasmid it will generate a CSV file for. 2- The directory where the output CSV file is to be placed. 3- The directory where the plasmid fasta file is located. We assume it is named after the pattern `${ACCESSION}.fasta`. 4- The directory where the input plasmid matches file is located. We assume it is named after the pattern `${ACCESSION}_matches-summary.tsv`. 5- The directory where the input incompatibility groups (derived from the BLAST results) are located. We assume it is named after the pattern `${ACCESSION}_families.list`. 6- The filename of the source info. We assume it is named `sourceInfo.tsv` in the `plasmid_sourceInfo` directory. 7- The directory of the plasmid BLAST results. We assume it is called `plasmid_blast_results`. 8- The filename of the sequence technologies information. We assume it is at `plasmid_seqTech/seqTech.tsv`.

Output: One comma-separated value file. It will be placed in the directory specified in the input position 2. We assume the output file will be named after the following pattern: `${ACCESSION}.csv`. The columns of the file are as follows:

> "Accession #","Identical Plasmids","Source: Organism","Source: Isolation Source","Source: Country","Source: Collection Date","Sequencing Technologies","Sequencing Technologies Count","Short Read Count","Long Read Count","Illumina Count","Roche 454 Count","ABI Solid Count","Sanger Count","Ion Torrent Count","PacBio Count","ONT Count","Plasmid Length","Antimicrobial Resistance CDS","Antimicrobial Resistance CDS %","Beta-lactamase CDS","Beta-lactamase CDS %","Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy #","Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy # % of Beta-lactamase","Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Copy # % of Total","Beta-lactamase Special (Carbapenem*,IMP,KPC,NDM,VIM) Absent (Yes/No)","Plasmid Transfer CDS","Plasmid Transfer CDS %","Toxin/Antitoxin System CDS","Toxin/Antitoxin System CDS %","Toxin/Antitoxin System Present (Yes/No)","DNA Maintenance/Modification CDS","DNA Maintenance/Modification CDS %","DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy #","DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy # % of DNA Maintenance/Modification","DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Copy # % of Total","DNA Maintenance/Modification Special (mucA,mucB,polymerase,umuC,umuD) Present (Yes/No)","Mobile Genetic Elements CDS","Mobile Genetic Elements CDS %","Hypothetical Genes CDS","Hypothetical Genes CDS %","Other CDS","Other CDS %","Total CDS","Incompatibility Groups"

Code:

Bash Command

```
while read ifn
do
        ACCESSION=`basename "${ifn}" ".fasta"`

        python3 generatePlasmidCSV.py \
                "${ACCESSION}" \
                plasmid_csv \
                plasmid_fasta \
                plasmid_matches \
                blast_results \
                plasmid_sourceInfo/sourceInfo.tsv \
                plasmid_blast_results \
                plasmid_seqTech/seqTech.tsv

done < <(ls -1 plasmid_fasta/*.fasta)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (generatePlasmidCSV.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 22. Create Group CSVs from Plasmid CSVs

Input: The inputs required are the group list files that contain the plasmids in each group (see step #4) and the individual plasmid CSVs (see step #12). The group list files are assumed to be in the directory `groups` and named after the pattern `${GROUP}.list`. The plasmid CSVs are assumed to be in the `plasmid_csv` directory and named after the pattern `${ACCESSION}.csv`.

Output: One comma-separated value file containing the same header line as all the plasmid CSVs and a concatenation of the non-header lines from the plasmid CSVs. We assume the output file will be in the directory `group_csv` and will be named after the following pattern: `${GROUP}.csv`.

Code:

Bash Command

```
while read ifn
do
        GROUP=`basename "${ifn}" ".list"`
        ofn="group_csv/${GROUP}.csv"

        # get and write a header
        hfn=plasmid_csv/`head -q -n 1 "${ifn}"`".csv"
        head -q -n 1 "${hfn}" > "${ofn}"

        # get and write the non-headers lines
        nhfns=`cat "${ifn}" | sed -r 's,^(.+)$,plasmid_csv/\1.csv,' | tr '\n' ' '`
        tail -q -n +2 ${nhfns} >> "${ofn}"

done < <(ls -1 groups/*.list)
```

sed Note

sed must be GNU (https://www.gnu.org) sed. `-r` does not enable extended regular expression syntax with BSD (http://www.bsd.org) sed.

Step 23. Create Group Matches from Plasmid Matches

Note that this step is not technically necessary to generate the desired output (the group CSV files (step #13) and the group statistics files (step #15)). This is really for convenience in inspecting results.

Input: The inputs required are the group list files that contain the plasmids in each group (see step #4) and the individual plasmid matches (see step #11). The group list files are assumed to be in the directory `groups` and named after the pattern `${GROUP}.list`. The plasmid matches are assumed to be in the `plasmid_matches` directory and named after the pattern `${ACCESSION}_matches.tsv`.

Output: One text file containing the matches for the group. We assume the output file will be in the directory `group_matches` and will be named after the following pattern: `${GROUP}_matches.tsv`.

Code:

Bash Command

```
while read ifn
do
         GROUP=`basename "${ifn}" ".list"`
         ofn="group_matches/${GROUP}_matches.tsv"

         fns=`cat "${ifn}" | sed -r 's,^(.+)$,plasmid_matches/\1_matches.tsv,' | tr
'\n' ' '`
         head -q -n 1 ${fns} | head -n 1 > "${ofn}"
         tail -q -n +2 ${fns} >> "${ofn}"

done < <(ls -1 groups/*.list)
```

sed Note

sed must be GNU (https://www.gnu.org) sed. `-r` does not enable extended regular expression syntax with BSD (http://www.bsd.org) sed.

Step 24. Calculate Group Statistics from Group CSV

Input: This Python program requires 2 inputs. 1- The CSV file for a group. Here, we show the CSV files in the directory `group_csv`, named after the pattern `${GROUP}.csv`. 2- The output statistics file for the group. Here, we show the statistics files in the directory `group_stats`, named after the pattern `${GROUP}.stats`.

Output: One text file named as described in position 2 of the input to the Python program. That file is formatted as follows:

```
GROUP_NAME
==========
Total # of Plasmids: ##

Incompatibility Groups Structure:
  Inc.         Plasmid   Size         Size
  Group        Count     Mean         St. Dev.
  IncGrp1      #         #.###        #.###
  IncGrp2      #         ######.###   #####.###
  .
  .
  .
  IncGrpN      #         #####.###    ####.###

Plasmid Lengths Summary:
       Min: ####
       Max: ######
    Median: #####
      Mean: ######.###
  St. Dev.: ######.###

Key Words Structure:
  Key                       Plasmid   Size         Size
  Word                      Count     Mean         St. Dev.
  anti_microb_resist        ##        ######.###   ######.###
  anti_microb_resist_not    #         ######.###   ######
  beta_lact                 ##        ######.###   ######.###
  beta_lact_not             #         ######.###   ######
  plasmid_transfer          ##        ######.###   ######.###
  plasmid_transfer_not      #         #####.###    #####.###
  toxin                     ##        ######.###   #####.###
  toxin_not                 ##        #####.###    ######.###
  dna_maint                 ##        ######.###   ######.###
  dna_maint_not             #         ######.###   ######
  mob_gen_elem              ##        ######.###   ######.###
  mob_gen_elem_not          #         ######.###   ######.###
  hypo_genes                ##        ######.###   ######.###
  hypo_genes_not            #         ######.###   ######
  other                     ##        ######.###   ######.###
  other_not                 #         ######.###   ######.###

Plasmid Structure:
  This information is already reported in the CSV file: GROUP_NAME.csv

Sequencing Technologies:
  Sequencing       Num         Occurances per    Percent Total    Percent Known
  Technology       Plasmids    Plasmid           Plasmids         Plasmids
  Known            ##          NA                ##.###           ###.###
  Unknown          ##          NA                ##.###           #.###
  Illumina         ##          #.###             ##.###           ##.###
  Roche ###        #          #.###            #.###           ##.###
  ABI Solid        #          #.###            #.###           #.###
  Sanger           #          #.###            #.###           #.###
  Ion Torrent      #          #.###            #.###           #.###
  PacBio           #          #.###            ##.###          ##.###
  ONT              #          #.###            #.###           #.###
  Short            ##         #.###            ##.###          ##.###
  Long             #          #.###            ##.###          ##.###
  Multiple Short   #          #.###            #.###           #.###
  Multiple Long    #          #.###            #.#             #.#
  Short Only       ##         #.###            ##.###          ##.###
  Long Only        #          #.###            #.###           #.###
  Short & Long     #          #.###            #.###           ##.###

Identical Plasmids Summary:
                           Plasmids (GROUP_NAME): ##
                               Discrete Plasmids: ##
         Indiscrete Plasmids (inside GROUP_NAME): ##
        Indiscrete Plasmids (outside GROUP_NAME): ##
                             Indiscrete Plasmids: ##
                   Groups of Indiscrete Plasmids: ##
                          Group Member Count Min: ##
                          Group Member Count Max: ##
                       Group Member Count Median: ##
                         Group Member Count Mean: ##.###
                     Group Member Count St. Dev.: ##.###

Identical Plasmids Groups:
       Discrete (GROUP_NAME):
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########

         Indiscrete Group #1:
                 ########  ########  ########
                 ########  ########  ########
                 ########  ########  ########
                 ########

         ...
         ...
         ...

         Indiscrete Group #n:
                ########  ########
```

Code:

Bash Command

```
while read gfn
do
        GROUP=`basename "${gfn}" ".list"`

        ifn="group_csv/${GROUP}.csv"
        ofn="group_stats/${GROUP}.stats"

        python3 calcGroupCSVstats.py\
                "${ifn}" \
                "${ofn}"

done < <(ls -1 groups/*.list)
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (calcGroupCSVstats.py)

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Step 25. Create Distance Matrix

Input: This script acts on all the files directly (i.e., not calling on a subroutine in Python or AWK for each of the accession numbers). It requires no user input directly as it ascertains the plasmid accession numbers from file names. It also relies on the directory structure to find the files named after the pattern `${ACCESSION}_identicalPlasmids_concordant.list` in a directory called `plasmid_blast_results`.

Output: One file per each accession. Each file is effectively a single row in the distance matrix. Once they are all created, they are combined into an additional file, the full distance matrix. The distance matrix is a full matrix (not only the bottom or upper halves); it is a csv file. The format looks like this:

Accession	A	B	C	D
A	0	x	y	a
B	x	0	i	j
C	y	i	0	k
D	a	j	k	0

Code:

Bash Command

This script is too long to reasonably represent in this document. Please view it in the freely-accessible online repository.

Distance Metric Definition and Examples:

Definition

The distance metric is the sum of the query and subject covered bases divided by the sum of the query and subject sequences. A covered base is defined as a base covered by (i.e., included in) the alignment. Given that d is the distance between a query and subject sequence, c is the coverage (i.e., bases included in the alignment) from a given sequence, and l is the length of a given sequence, the distance metric can be expressed in equation notation:

$$d = \frac{c_{query} + c_{subject}}{l_{query} + l_{subject}}$$

This metric is calculated for each pair of query and subject sequences; in other words, it is calculated in an all-vs-all fashion between the sequence for each plasmid.

Example

Given a query sequence that is 10 bases long and a subject sequence that is 20 bases long, consider an alignment that has a length of 6 bases and looks like this:

```
  Query:  AAAAACGGGG
Subject:      A-GGGGTTTTTGGGGGCCCCC
```

The length of the alignment is 6. The number of covered bases (i.e., bases in the alignment) from the query sequence is 6. For the subject sequence, the number is 5. The distance can be found using the equation:

$$d = \frac{c_{query} + c_{subject}}{l_{query} + l_{subject}} = \frac{6+5}{10+20} = \frac{11}{30} \cong 0.367$$

This example is for a single pairwise comparison and would need to be repeated for every pair of plasmids.

Step 26. Create Distance Tree

Input: The input is the distance matrix from the previous step. We assume it is called `dist_matrix.csv` in the `tree` directory.

Output: One text file called `dist_tree.newick` in the `tree` directory. It is in the Newick tree format.

Code:

Bash Command

```
makeNewick.py \
         -i "tree/dist_matrix.csv" \
         -o "tree/dist_tree.newick"
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (makeNewick.py)

This script is not part of this package. It must be downloaded and installed separately. The only substantive requirement is Python 3.5+. makeNewick.py comes from a software package called CAM - **C**odon **A**version **M**otifs for Alignment-free Phylogenies (Miller et al. 2019). CAM is freely-available on GitHub at https://github.com/ridgelab/cam.

Step 27. Add Leaf Labels to Tree

Input: The input is the distance tree in Newick format from the previous step. We assume it is called `dist_tree.newick` in the `tree` directory. It also requires the location of source information (e.g., the country of origin of the plasmid) file and the name of the output file.

Output: This step appends additional information to the accession numbers that are the leaf labels in the tree. It creates a new tree, also in Newick format. We assume the output tree is in the `tree` directory and is called `dist_tree_labels.newick`.

Code:

Bash Command

```
python3 modifyLeafLabels.py \
         "plasmid_sourceInfo/sourceInfo.tsv" \
         "tree/dist_tree.newick" \
         "tree/dist_tree_labels.newick"
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (modifyLeafLabels.py)

This script has at least one line that is too long to represent in this document without sacrificing readability. Please view it in the freely-accessible online repository.

Step 28. Add Color to Leaf Labels

Input: The input is the labeled distance tree in Newick format from the previous step. We assume it is called `dist_tree_label.newick` in the `tree` directory. Additional input is a colors mapping file. We assume it is called `colors.tsv`. The format is assumed to be one entry per line, where each entry has one column for the group and another column for the hex color (without the # symbol). The file we used is as follows:

```
IMP   FF0000
KPC   0000FF
NDM   00B600
VIM   000000
```

As you can see, we were looking for four groups for this tree figure in our analysis: IMP, KPC, NDM, and VIM. The final inputs required are a list of associated accession numbers for each group. We assume the files are named after the pattern `${GROUP}.list` in the directory `groups/keep`.

Output: This step includes the Newick-formatted tree from the input in a new Nexus file. It relies on a taxa block to specify colors for the leaf labels. As an example, the leaf label will have the label (e.g., `\t'some label here'`) followed by the color specification (e.g., `[&!color=#6789AB]`). This Nexus file will be available for directly opening with FigTree (https://github.com/rambaut/figtree) and, presumably, by other tree viewing/editing software. We assume the output tree is in the `tree` directory and is called `dist_tree_labels_colors.nexus`.

Code:

Bash Command

```
python3 convertNewick2NexusAndAddColor.py \
        "tree/dist_tree_labels_colors.nexus" \
        "colors.tsv" \
        "tree/dist_tree_labels.newick" \
        "groups/keep/IMP.list" \
        "groups/keep/KPC.list" \
        "groups/keep/NDM.list" \
        "groups/keep/VIM.list"
```

Python Version

Python 3.6.4 (https://www.python.org).

Python Script (convertNewick2NexusAndAddColor.py)

This script has at least one line that is too long to represent in this document without sacrificing readability. Please view it in the freely-accessible online repository.

A comment on data availability

The version of the PlasmidFinder database that we downloaded is no longer available. Accordingly, we release the fasta file we downloaded for reproducibility purposes. However, we advise a fresh download for any new experiments. This file may be found in the repository at the following path: `data/original_incompatibility_groups/incompatibility.fasta`. Similarly, many GenBank files have been updated since our download on 1 March 2018. We likewise release the versions we downloaded here for reproducibility purposes. However, we recommend fresh downloads of these files for new analyses. A script (labelled as “Step 0”) is released with the online code repository for such a purpose. Please note that additional plasmids could now (and should) be included if the Entrez search strategy were to be re-done. The script would not reflect such changes as it downloads the specific GenBank groupings we used via accession numbers, completely ignoring the Entrez strategy. This is appropriate for reproducing our results, but it would probably not be ideal for a future study.

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. Basic Local Alignment Search Tool. Journal of Molecular Biology **215**: 403-410. doi:10.1016/S0022-2836(05)80360-2.

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., and Madden, T.L. 2009. BLAST+: architecture and applications. BMC Bioinformatics **10**: 421. doi:Artn 421\nDoi 10.1186/1471-2105-10-421.

Carattoli, A., Zankari, E., Garcia-Fernandez, A., Voldby Larsen, M., Lund, O., Villa, L., Moller Aarestrup, F., and Hasman, H. 2014. In silico detection and typing of plasmids using PlasmidFinder and plasmid multilocus sequence typing. Antimicrob Agents Chemother **58**(7): 3895-3903. doi:10.1128/aac.02412-14.

Miller, J.B., McKinnon, L.M., Whiting, M.F., and Ridge, P.G. 2019. CAM: an alignment-free method to recover phylogenies using codon aversion motifs. PeerJ **7**: e6984. doi:10.7717/peerj.6984.