An overview of the methods used in annotation, key decisions and potential modifications.

Oakem Kyne

2024

Annotation Code Description

Contents

[2 Environment Generation 1](#_Toc173313563)

[3 Bakta annotation 2](#_Toc173313564)

[3.1 Adding new proteins 3](#_Toc173313565)

[3.1.1 For genes with lower sequence identity, you can: 4](#_Toc173313566)

[4 COG gene removal 4](#_Toc173313567)

[5 Domain based annotation of ANKs and TPRs 4](#_Toc173313568)

[6 Annotation of some specific ankyrins 5](#_Toc173313569)

[7 RAGE protein completeness checks 5](#_Toc173313570)

[8 spoT name processing 5](#_Toc173313571)

[9 RAGE classification 6](#_Toc173313572)

[9.1 Identification of potential RAGE boundaries 6](#_Toc173313573)

[9.2 Identification of RAGE derived regions 7](#_Toc173313574)

[9.2.1 Method for lists\_combined.txt development 7](#_Toc173313575)

[9.2.2 Exclusion list development 7](#_Toc173313576)

[9.3 Identification of complete RAGEs 8](#_Toc173313577)

[9.4 Modifications to RAGE criteria 8](#_Toc173313578)

[9.5 Complete RAGEs identified and complete RAGEs reclassified 8](#_Toc173313579)

[9.6 Possible modifications to RAGE criteria 8](#_Toc173313580)

[9.6.1 Possible modifications to RAGE derived regions 8](#_Toc173313581)

[9.6.2 Possible modification of complete RAGE criteria 9](#_Toc173313582)

[10 Usage 9](#_Toc173313583)

[11 Genome Summary Statistics 9](#_Toc173313584)

# Environment Generation

Enviroments required:

1. bakta environment
2. rage environment

Before running the scripts these need to be generated from the yml files and then activated when running the scripts. bakta environment for the bakta section, then the rage environment for the rest of the annotation scripts (this is called main in the 3\_annotation\_scripts directory).

There were some issues with generating the rage environment, this was resolved by enabling flexible solve on conda/mamba. The code below needs to be run before generating the environment:

`conda config --set channel\_priority flexible`

Then environments can be generated using:

`mamba env create -f name\_of\_yml.yml`

# Bakta annotation

Bakta annotation was chosen as fairly recent bacterial genome annotation package, which importantly does not require existing models for a species. It also integrates many different databases into it methodology.

This package enables the use of “expert proteins” to be provided as a .faa file and we make use of this for more accurate annotation of the O*rientia tsutsugamushi* genomes. To create this file all full-length genes annotated in the Giengkam et al (2023) paper were used, with the exception of genes with domain based names (e.g. ankyrin protiens, tetratricopeptide proteins and HATpase\_c domain containing protein). Additionally, some gene names were changed due to spelling mistakes or other naming issues (e.g. mmmA to mnmA and iepA to LepA).

This approach resulted in the correct annotation of most genes, however was insufficient for some proteins, often those with considerable sequence variability. Consequently, for a small group of proteins the required identity threshold was reduced from 90% to 80%. These are; scaA, scaC, tsa22 and tsa56, this approach was deemed reasonable for these as scaA and scaC are important host interaction proteins and likely subject to high selection pressure, and tsa22 and tsa56 are well established membrane proteins with lots of genetic diversity.

Another approach used to improve the annotation of cinA, scaA, scaB, scaD, scaE, secA, traA, traG, tsa22, tsa47 and tsa56 was to extract known sequences from NCBI and use these in the .faa file. These were extracted by searching for them and excluding partial or incomplete from the results (e.g. cinA AND txid784[Organism] NOT incomplete NOT partial), in some cases gene length was also used as a filter to remove the incomplete ones.

This code currently activates the bakta environment automatically, but this is setup for my system and will need to be changed. This is the line of code that needs modification:

`source /well/moru-batty/users/vhs789/miniforge/etc/profile.d/conda.sh`

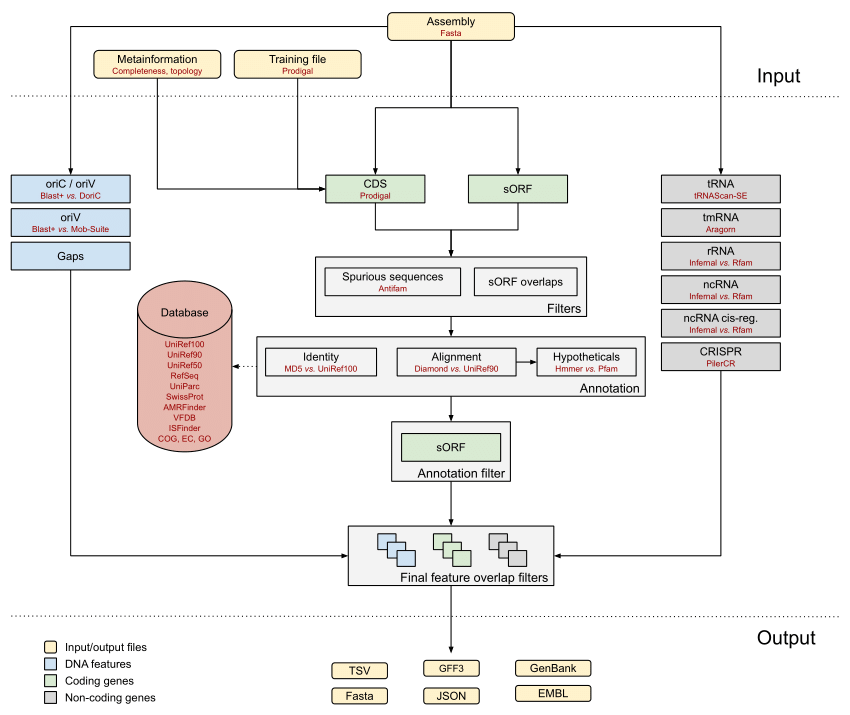


Figure 1 Bakta annotation scheme from Schwengers et al (2021), showing the different databases involved and the processing steps. However, it doesn’t show the user provided protein .faa input.

## Adding new proteins

Modification of the .faa input file for bakta is the simplest way to add/change an annotation. To do this you need:

1. Full length sequences, the more the better especially for highly variable proteins. I would suggest using at least 8.
2. These sequences should be clustered to see the sequence identity between the genes. To be used in bakta normally this should be >90% and of similar lengths (80% similarity).
3. Ensure that there are no conflicting/contradictory gene names in the .faa file by performing (cd-hit) clustering. Closely related sequences should be checked as they may be other cases of the gene, and so the gene name should be changed in the file.

### For genes with lower sequence identity, you can:

1. Modify the percentage identity in the .faa file to 80 (by including this in the fasta header; 80~~~80~~~80~~~).
2. Increase the number of sequences by identifying them in NCBI and including these in the .faa file. It is important these are complete copies.

However, this should be done with caution. Modification of the percentage identity should be performed very carefully and there should be good evidence that the gene is highly variable. It is also important to check that by doing this, genes are not mis-annotated.

Some genes cannot be handled like this if there is very low sequence identity (e.g. many of the ankyrin annotations in Giengkam et al (2023)). These should probably be annotated manually/separately.

# COG gene removal

Bakta annotation occasionally mis-annotates some protein names but not the protein product. By examining the evidence for these mis-annotations it was determined they came from the COG database used in bakta.

To resolve this issue, a list of genes mis-named in the gene name (but not product) due to COG was created (all other databases were checked when generating this list). Then the COG identifier for these was extracted and a list was made in the `cog\_list.txt` file.

The script `1\_cog\_removal.py` is then used to remove gene names where the COG identifier is present. This is possible, as bakta only writes a COG identifier if it is used for the gene name or the product (e.g. it is not superseded by another database).

In the future if other genes are identified as problematic and as having COG issues, the COG identifier can be added to the `cog\_list.txt` file. Proceed with caution, these should be checked carefully first!

# Domain based annotation of ANKs and TPRs

These domains are important in many host-pathogen interaction and so worth identifying. *Orientia* has a huge variety of these ank and tpr repeats in its genome. Consequently, we identify these repeats using a hmmer search and then annotate this information in the gbff files.

Code structure:

1. Parsing gbff files from bakta to faa files containing locus\_tag and sequence information
2. hmmersearch to identify the presence of tprs or anks domains
3. Parsing hmmer output
4. Adding this information back to the product of the gbff files

These 4 scripts are linked by domain\_annotation.sh which requires all files from the domain\_annotation\_package directory (excluding test directory) in the positions they are currently found in, and correct packages installed (see environment.yml). The command line usage is then: `bash domain\_annotaiton.sh <gbff files>` though this is integrated into the main.sh script.

A threshold of -E 1E-10 was used, as (1) it is the threshold used in bakta and (2) it represents a high stringency threshold avoiding false positives.

The scripts over write any previous annotations for the hmmer hits, but skips proteins annotated as traG and bamD (which sometimes contain tpr domains). In tests other annotations over written were mis-annotations, including pilW (*Orientia* doesn’t have a flagella).

The number of ankyrin repeats are included in the product description as a single hmmer model can be used for these proteins. For TPRs we used TPR\_1 model for simplicity, as in tests the model used make little or no difference in the number of TPRs identified (we also tested using all models, this resulted in high levels of complexity and increased processing requirements with little or no improvement).

# Annotation of some specific ankyrins

Giengkam et al (2023) identified many *Orientia* anks. Clustering experiments showed some of these have high sequence identity and are conserved across the 8 genomes (investigated in the paper). These were the ones targeted for annotation.

For this a blast database was created and genes were tested against them. Ankyrin proteins covered in this are: Ank03, Ank08, Ank10, Ank11, Ank12, Ank20 and Ank24.

There is so much diversity of ankyrin proteins in *Orientia*, I think additional annotations should probably be avoided until someone clusters more genomes together (many ankyrin groups clustered very poorly and so were not addressed).

Code structure:

1. 1\_gen\_2\_faa.py to convert the genbank file to a .faa file
2. 2\_blast\_processing.py performs a blast search to identify hits to the ankyrin proteins
3. 3\_writing\_to\_gbff.py this adds in the ankyrin name and product to the genbank file

These scripts are tied together in ank\_completeness\_master\_script.sh which in turn is called by the main.sh script.

# RAGE protein completeness checks

To identify complete RAGEs, full-length; traA, traB, traC, traD, traE, traF, traG, traH, traI, traK1, traK2, traL, traN, traU, traV, traW and integrase must be present. Consequently, through the use of a blast script we check the overlap between the query and the subject, if this is >=95% then the sequence is considered complete and annotated as such in the gene product (e.g. complete traN) without overwriting the existing information.

This step is essential as there are many truncated copies of these genes in the genome (from degraded RAGEs).

Code structure is essentially the same as that used in specific ank identification (much of it was recycled):

1. 1\_gen\_2\_faa.py to convert the genbank file to a .faa file
2. 2\_blast\_processing.py performs a blast search to identify hits to the ankyrin proteins
3. 3\_writing\_to\_gbff.py this adds in the complete [gene name] to the product in the .gbff file.

These scripts are tied together in gene\_completeness\_master\_script.sh which in turn is called by the main.sh script.

# spoT name processing

To handle these proteins we take 2 approaches:

1. Blast searches to identify complete/full-length/bifunctional spoT
2. Renaming genes with ppGpp or spoT in the product as "truncated RelA/SpoT homolog (RSH) protein", or as "Orientia SpoT-synthetase" if the gene length is >1200 amino acids

Whilst this isn’t the most elegant solution, Jeanne wanted the truncated spoT to be named as spoT hydrolase, and “truncated RelA/SpoT homolog (RSH) protein” was the compromise we reached as many of them were not positively identified as being of spoT origin but are of this family group. The "Orientia SpoT-synthetase" case is a bit different and identification is based on the fact that these proteins are considerably longer than the other spoT proteins in these organisms. However, it should probably be checked in the future to ensure that new proteins identified with this contain synthase domains (the pattern many not always hold).

# RAGE classification

This contains 3 main steps:

1. Identification of potential RAGE boundaries
2. Identification of RAGE derived regions
3. Identification of potential RAGE boundaries contained within RAGE derived regions with full length genes required for the RAGEs

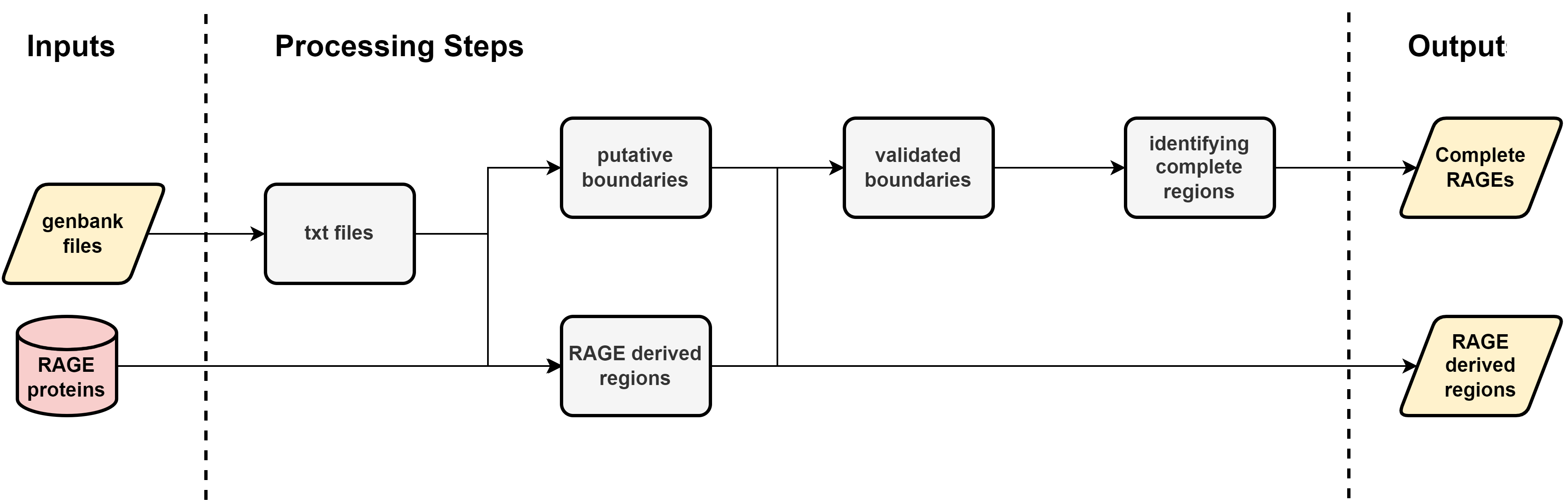


Figure 2 Flow chart of RAGE classification methodology

## Identification of potential RAGE boundaries

1. A complete RAGE must contain an integrase “facing” into the RAGE
2. It be bounded by either 2 integrases or an integrase and a dnaA

The first script identifies all of these and outputs this information as a bed file.

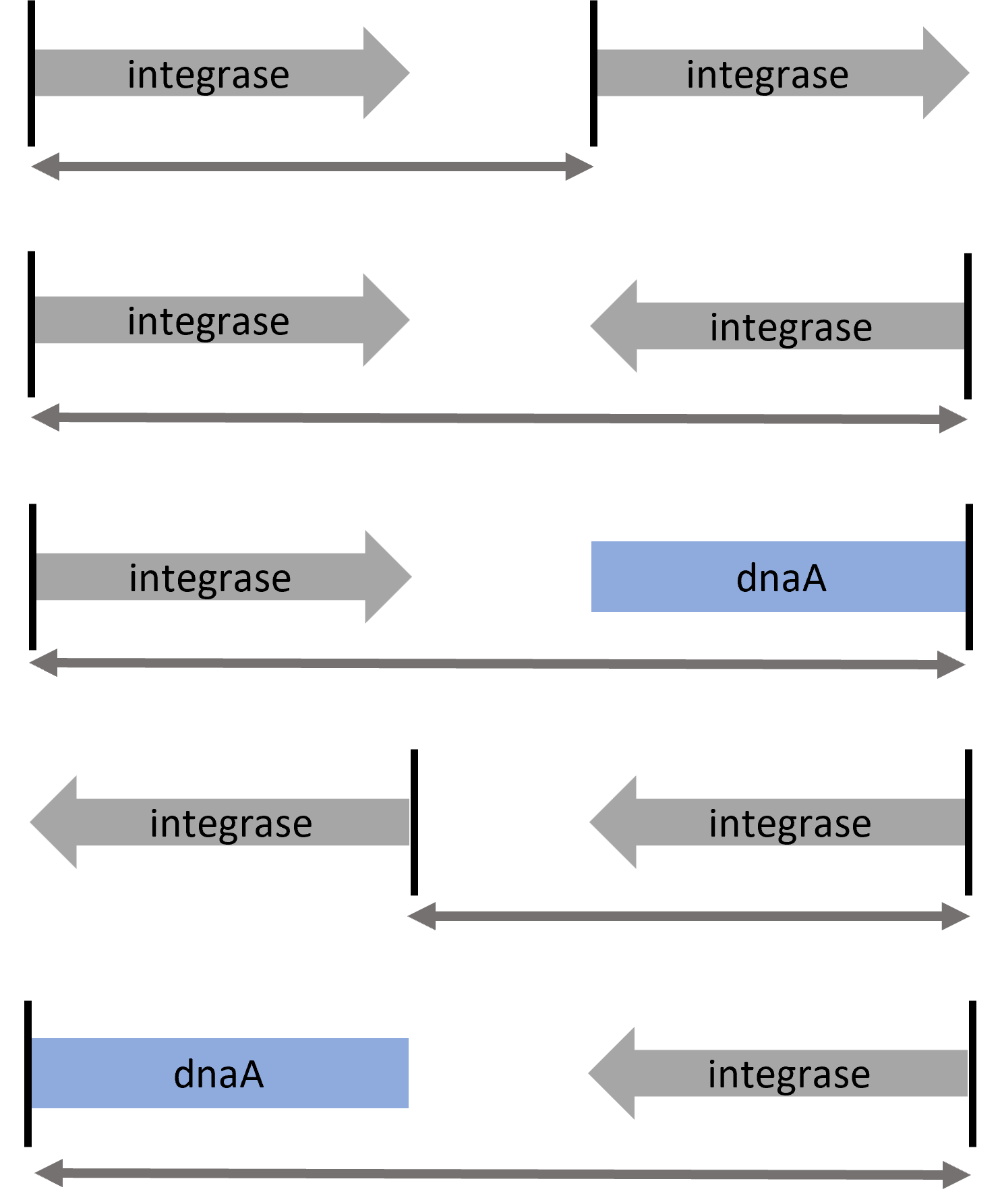


Figure 3 All possible RAGE boundaries with single headed arrows showing the direction of the integrase (facing into the RAGE, or away from the RAGE). Double headed arrows show the region identified as a possible RAGE boundary region.

## Identification of RAGE derived regions

RAGE derived regions are areas containing RAGE proteins

1. The region must contain genes found in lists\_combined.txt
2. The region must contain no genes found in exclusion\_list.txt
3. A region is permitted 1 gene which doesn’t match data in lists\_combined.txt
4. A region starts at the first gene found in lists\_combined.txt and continues until the second unmatched gene is found (unmatched genes are not written to the area) or an item from exclusion\_list.txt is identified
5. It must contain 2 or more genes (excluding skipped ones) this resolves issues where genes can be shared between either group

### Method for lists\_combined.txt development

1. Creation of a list of known RAGE proteins based on the manuscript
2. Addition of genes based on gaps in the resultant bed files (in a couple of test genomes)
3. Then checking the functioning of the list in other genomes

### Exclusion list development

Created through observation of genes with nested names but aren’t found in RAGE regions.

## Identification of complete RAGEs

1. Identification of regions contained in both the RAGE boundaries and RAGE derived regions (using bedtools)
2. Checking for the presence of each of these genes; 'complete traa', 'complete trab', 'complete trac', 'complete trad', 'complete trae', 'complete traf', 'complete trag', 'complete trah', 'complete trai', 'complete trak', 'complete tral', 'complete tran', 'complete trau', 'complete trav', 'complete traw', 'complete integrase' and 'complete trbc' . 2 copies of 'complete trad' are required.
3. 1 cargo protein and 1 transposase from each list are required

## Modifications to RAGE criteria

1. Requirement of 2 traD proteins (rather than differentiating between the 2 types of traD)
2. Requirement of trbC

Neither of these changes impact how any of the RAGEs would have been classified, but as both these proteins are thought to be important in the complete RAGEs functioning, it makes it more reflective of the actual biology.

## Complete RAGEs identified and complete RAGEs reclassified

Our new definitions, and additional investigation of the genome resulted to changes in the complete RAGEs identified.

Gilliam is still recognised as having the 3 complete RAGEs previously identified. However, now Kato has lost its (as it only contains one traD, the traDti, this invalidates it under either definition) and a new one has been identified in UT76 (previously classified as complete with truncated genes, but it contains the complete set of full length genes (in my analysis or the previous ones)).

In initial testing we also identified RAGE kato\_02 as being complete. This was based on sequence similarity of the integrase to Karp\_02484 integrase. However, this was subsequently reclassified as a truncated intergrase (through discussions with Saire and Jeanne) as it is missing the arm-type binding motif. Thus, we changed the classification of this integrase to truncated (this is in line with the original manuscript).

## Possible modifications to RAGE criteria

### Possible modifications to RAGE derived regions

1. Increase the number of permitted skips, simply change the value in the 2\_rage\_derived\_regions.py script. The section where this would be modified is commented
2. Inclusion list, genes can be added or removed from this txt file affecting genes recognised as RAGE. This is the simplest way to make changes.
3. Exclusion list, genes which should not be appearing in the RAGE derived regions can be added to this. If there are issues with certain proteins being included, this is another very simple way to make modifications.

### Possible modification of complete RAGE criteria

It is possible to change the requirements by adding or removing genes (names, products or parts of either) to the tra\_genes list. However, you will likely have to create a method to identify complete versions of any genes you add.

Transposase or cargo list can also be modified, however for these lists only one hit from each is required. So, it is unlikely to make much of a difference.

# Usage

The code can be run through a single script (main.sh) on any number of genbank files (.gbff files).

Requirements:

1. The rage conda environment must be active (there is a yaml file containing the information for this (rage.yml))
2. Genbank files must not contain “-“ in their names
3. The script should be run from the directory containing only the genbank files which you wish to annotate
4. The directory should be in the same directory as the other scripts i.e. the 3\_annotation\_scripts (this is a result of path issues resulting from dropbox and the method used to resolve them)

Usage: `bash Path/to/script/main.sh \*.gbff`

# Genome Summary Statistics

After all the post-bakta processing, it is useful to generate some genome summary information. This includes; genome size, number of contigs, number of tRNAs, number of coding sequences, coding density, hypotheticals, total length of rage regions, total length of interRAGE regions, percentage of the genome in RAGE regions, the number of RAGE regions, the number of complete RAGEs, the number of spoT-synthase, bifunctional spoT and truncated spoT/RelA, and the number of both ankyrins and tetratricopeptide containing proteins.

As this information comes from 4 different sources for each genome, it was decided not to turn this into a single package/chunk of code. But rather to provide the code for each individual bit of information.

|  |  |
| --- | --- |
| Statistic | Method (gbff files are the ones post processing) |
| genome\_length\_(bases) | seqkit stats genome.fasta (seqkit needs to be installed) |
| number\_of\_contigs | seqkit stats genome.fasta |
| number\_of\_tRNAs | bakta genome.txt output |
| number\_of\_coding\_sequences | bakta genome.txt output |
| coding\_density | bakta genome.txt output |
| hypotheticals | bakta genome.txt output (this may be lower with ankyrin and tpr annotation... I’ve decided this probably doesn’t matter as these proteins are still largely hypothetical) |
| size\_of\_RAGE\_regions | for file in \*.bed; do awk '{sum += $3 - $2} END {print FILENAME, sum; sum=0}' "$file"; done (files in rage\_derived) |
| size\_of\_interRAGE\_regions | =genome\_size-size\_of\_RAGE\_regions |
| percentage\_genome\_RAGE | =size\_of\_RAGE\_regions/genome\_size\*100 |
| number\_of\_RAGE\_regions | wc -l processing\_outputs/rage\_derived/\* |
| number\_of\_complete\_RAGEs | wc -l processing\_outputs/complete\_rage/\* |
| number\_spoT-synthase | grep -c "Orientia SpoT-synthetase" \*.gbff |
| number\_bifunctional\_spoT | grep -c "full length bifunctional spoT" \*.gbff |
| number\_of\_truncated\_RelA/SpoT\_homolog\_proteins | grep -c "truncated RelA/SpoT homolog (RSH) protein" \*.gbff |
| number\_of\_ankyrins | grep -c -i "ankyrin" \*.gbff |
| number\_of\_tetratricopeptides | grep -c -iE "tpr |tetratricopeptide" \*.gbff |