pyani Documentation

Release 0.2.9

Leighton Pritchard

Jun 11, 2022

Contents:

CHAPTER 1

Build information

CHAPTER 2

`PyPI` version and Licensing

CHAPTER 3

conda version

If you're feeling impatient, please head over to the *QuickStart Guide*

CHAPTER 4

Description

`pyani` is a program and Python package that provides support for calculating average nucleotide identity (ANI) and related measures for whole genome comparisons, and for rendering relevant graphical and tabular summary output. Where available, it natively takes advantage of multicore systems, and can integrate with SGE or OGE-compatible job schedulers to manage the computationally-heavy sequence comparisons.

Installing the `pyani` Python package also installs the program `pyani`, which enables command-line based analysis of genomes. Results are stored in a private SQLite3 database local to the machine, which permits addition of genomes to a previous analysis without having to recalculate all previously-performed comparisons, facilitating incremental analysis and visualisation, and enabling incremental maintenance of results for a taxonomic group as new genome sequences become available.

If you use `pyani` in your work, we would be grateful if you could please cite us as indicated on the *Citations* page.

CHAPTER 5

Reporting problems and requesting improvements

If you encounter bugs or errors, or would like to suggest ways in which `pyani` can be improved, please raise a new issue at the `pyani` GitHub issues page.

If you'd like to fix a bug or make an improvement yourself, contributions are welcomed, and guidelines on how to do this can be found at the *Contributing to pyani* documentation page.

5.1 Citing `pyani`

We would be grateful if you could please cite the following manuscript in your work if you have found `pyani` useful:

> Pritchard *et al.* (2016) "Genomics and taxonomy in diagnostics for food security: soft-rotting enterobacterial plant pathogens" *Anal. Methods*, 2016, **8**, 12-24 DOI: 10.1039/C5AY02550H

```
@Article{C5AY02550H,
author ="Pritchard, Leighton and Glover, Rachel H. and Humphris, Sonia and␣
↪Elphinstone, John G. and Toth, Ian K.",
title  ="Genomics and taxonomy in diagnostics for food security: soft-rotting␣
↪enterobacterial plant pathogens",
journal  ="Anal. Methods",
year  ="2016",
volume  ="8",
issue  ="1",
pages  ="12-24",
publisher  ="The Royal Society of Chemistry",
doi  ="10.1039/C5AY02550H",
url  ="http://dx.doi.org/10.1039/C5AY02550H",
abstract  ="Soft rot Enterobacteriaceae (SRE) are bacterial plant pathogens that␣
↪cause blackleg{,} wilt and soft rot
diseases on a broad range of important crop and ornamental plants worldwide. These␣
↪organisms (spanning the genera Erwinia{,}
Pectobacterium{,} Dickeya{,} and Pantoea) cause significant economic and yield losses␣
↪in the field{,} and in storage. They
are transmissible through surface water{,} by trade and other movement of plant␣
↪material and soil{,} and in some cases are
```

(continues on next page)

(continued from previous page)

```
subject to international legislative and quarantine restrictions. Effective detection␣
↪and diagnosis in support of food
security legislation and epidemiology is dependent on the ability to classify␣
↪pathogenic isolates precisely. Diagnostics and
classification are made more difficult by the influence of horizontal gene transfer␣
↪on phenotype{,} and historically complex
and sometimes inaccurate nomenclatural and taxonomic assignments that persist in␣
↪strain collections and online sequence
databases. Here{,} we briefly discuss the relationship between taxonomy{,} genotype␣
↪and phenotype in the SRE{,} and their
implications for diagnostic testing and legislation. We present novel whole-genome␣
↪classifications of the SRE{,}
illustrating inconsistencies between the established taxonomies and evidence from␣
↪completely sequenced isolates. We conclude
with a perspective on the future impact of widespread whole-genome sequencing and␣
↪classification methods on detection and
identification of bacterial plant pathogens in support of legislative and policy␣
↪efforts in food security."}
```

Tip: When citing `pyani` in your text, please consider quoting the precise release version (if you are using a specific release), or the specific commit hash (if you are using the 'bleeding-edge' development verison), e.g.

`"...using PYANI (v0.3.0)..."`

or

`"...using PYANI (commit bb1cb5c)..."`

This will enable precise reproduction of this part of your work by others.

5.1.1 Publications citing `pyani`

If you are using `pyani`, you are in good company. These authors and manuscripts have employed `pyani` to help with their whole-genome classification:

(If I've missed out your paper, please do put in a pull request on `GitHub` or get in touch and I'll be happy to add it!)

2022

- Botelho *et al.* (2022) "The ESKAPE mobilome contributes to the spread of antimicrobial resistance and CRISPR-1 mediated conflict between mobile genetic elements" *bioRxiv* doi:10.1101/2022.01.03.474784
- Castro-Jaimes *et al.* (2022) "Replication initiator proteins of *Acinetobacter baumannii* plasmids: An update note" *Plasmid* doi:10.1016/j.plasmid.2021.102616
- Jiang *et al.* (2022) "Vibrio Clade 3.0: New *Vibrionaceae* Evolutionary Units Using Genome-Based Approach." *Curr Microbiol* doi:10.1007/s00284-021-02725-0
- Narsing Rao *et al.* (2022) "Genome-based reclassification of *Evansella polygoni* as a later heterotypic synonym of *Evansella clarkii* and transfer of *Bacillus shivajii* and *Bacillus tamaricis* to the genus *Evansella* as *Evansella shivajii* comb. nov. and *Evansella tamaricis* comb. nov." *Arch Microbiol* doi:10.1007/s00203-021-02720-w

2021

- Abdel-Glil *et al.* (2021) "Comparative in silico genome analysis of *Clostridium perfringens* unravels stable phylogroups with different genome characteristics and pathogenic potential" *Sci. Rep.* doi:10.1038/s41598-021-86148-8
- Abdullah *et al.* (2021) "Comparative analysis of whole genome sequences of *Leptospira* spp. from RefSeq database provide interspecific divergence and repertoire of virulence factors" *bioRxiv* doi:10.1101/2021.01.12.426470
- Al Rubaye *et al.* (2021) "Novel genomic islands and a new vanD-subtype in the first sporadic VanD-type vancomycin resistant enterococci in Norway" *PLoS One* doi:10.1371/journal.pone.0255187
- Albuquerque *et al.* (2021) "Complete Genome Sequence of Two Deep-Sea *Streptomyces* Isolates from Madeira Archipelago and Evaluation of Their Biosynthetic Potential" *marine drugs* doi:10.3390/md19110621
- Almeida *et al.* (2021) "Comparative pangenomic analyses and biotechnological potential of cocoa-related *Acetobacter senegalensis* strains" *Antonie van Leeuwenhoek* doi:10.1007/s10482-021-01684-7
- Alonso-Reyes *et al.* (2021) "Genomic insights into an andean multiresistant soil actinobacterium of biotechnological interest." *World J Microbiol Biotechnol* doi:10.1007/s11274-021-03129-9
- Asselin *et al.* (2021) "Complete genome sequence resources for the onion pathogen, *Pantoea ananatis* OC5a" *Phytopath.* doi:10.1094/phyto-09-20-0416-a
- Badhai *et al.* (2021) "Genomic plasticity and antibody response of *Bordetella bronchiseptica* strain HT200, a natural variant from a thermal spring" *FEMS Micro. Lett.* doi:10.1093/femsle/fnab035
- Baxter *et al.* (2021) "Comparative Genomics across Three Ensifer Species Using a New Complete Genome Sequence of the Medicago Symbiont *Sinorhizobium (Ensifer) meliloti* WSM1022" *microorganisms* doi:10.3390/microorganisms9122428
- Becken *et al.* (2021) "Genotypic and Phenotypic Diversity among Human Isolates of *Akkermansia muciniphila*" *mBio* doi:10.1128/mBio.00478-21
- Bei *et al.* (2021) "Shedding light on the functional role of the *Ignavibacteria* in Italian rice field soil: A metagenomic/transcriptomic analysis" *Soil Biol. Biochem.* doi:10.1016/j.soilbio.2021.108444
- Biffignandi *et al* (2021) "Genome of *Superficieibacter maynardsmithii*, a novel, antibiotic susceptible representative of *Enterobacteriaceae*" *G3* doi:10.1093/g3journal/jkab019
- Biggel *et al.* (2021) "Spread of vancomycin-resistant *Enterococcus faecium* ST133 in the aquatic environment in Switzerland" *J. Glob. Anitmicrob. Res.* doi:10.1016/j.jgar.2021.08.002
- Boeuf *et al.* (2021) "Meta-pangenomics Reveals Depth-dependent Shifts in Metabolic Potential for the Ubiquitous Marine Bacterial SAR324 Lineage" *Research Squared* doi:10.21203/rs.3.rs-225427/v1
- Carlin *et al.* (2021) "*Listeria cossartiae sp. nov.*, *Listeria immobilis sp. nov.*, *Listeria portnoyi sp. nov.* and *Listeria rustica sp. nov.*, isolated from agricultural water and natural environments" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004795
- Chen *et al.* (2021) "Integrated Phenotypic-Genotypic Analysis of *Latilactobacillus sakei* from Different Niches" *preprints* doi:10.20944/preprints202107.0457.v1
- Chew *et al.* (2021) "First isolation of *Candida oceani* from a clinical specimen" *Antonie van Leeuwenhoek* doi:10.1007/s10482-020-01512-4
- Chew *et al.* (2021) "Molecular epidemiology and phylogenomic analysis of *Mycobacterium abscessus* clinical isolates in an Asian population" *Microb. Genom.* doi:10.1099/mgen.0.000708
- Chew *et al.* (2021) "Genomic characterization of *Klebsiella quasipneumoniae* from clinical specimens - a retrospective study, Singapore" *Antimicrob. Chemother.* doi:10.1128/AAC.00412-21

- Christensen *et al.* High natural PHA production from acetate in *Cobetia* sp. MC34 and *Cobetia marina* DSM 4741T and in silico analyses of the genus specific PhaC2 polymerase variant" *Microb Cell Fact* doi:10.1186/s12934-021-01713-0
- Clabaut *et al.* (2021) "Variability of the response of human vaginal *Lactobacillus crispatus* to 17β-estradiol" *Sci. Reports* doi:10.1038/s41598-021-91017-5
- Danneels *et al.* (2021) "Patterns of transmission and horizontal gene transfer in the *Dioscorea sansibarensis* leaf symbiosis revealed by whole-genome sequencing" *Curr. Biol.* doi:10.1016/j.cub.2021.03.049
- Das *et al.* (2021) "Description of *Acinetobacter kanungonis* sp. nov., based on phylogenomic analysis" *Int. J. Sys. Evol. Microbiol.* doi:10.1099/ijsem.0.004833
- Costatini *et al.* (2021) "Insight into phenotypic and genotypic differences between vaginal *Lactobacillus crispatus* BC5 and *Lactobacillus gasseri* BC12 to unravel nutritional and stress factors influencing their metabolic activity" *Microb. Genomics* doi:10.1099/mgen.0.000575
- Davison *et al.* (2021) "Large-scale comparative genomics unravels great genomic diversity across the *Rickettsia* and *Ca. Megaira* genera and identifies *Torix* group as an evolutionarily distinct clade." *bioRxiv* doi:10.1101/2021.10.06.463315
- de Silva *et al.* (2021) "Revisiting the *Colletotrichum* species causing anthracnose of almond in Australia" *Aust. Plant Path.* doi:10.1007/s13313-020-00765-x
- Delgado-Blas *et al.* (2021) "Population genomics and antimicrobial resistance dynamics of *Escherichia coli* in wastewater and river environments" *Commun Biol* doi:10.1038/s42003-021-01949-x
- Devika *et al.* (2021) "In Silico Prediction of Novel Probiotic Species Limiting Pathogenic *Vibrio* Growth Using Constraint-Based Genome Scale Metabolic Modeling" *Front. Cell. Inf. Microbiol.* doi:10.3389/fcimb.2021.752477
- Díaz *et al.* (2021) "Comparative Genomic Analysis of Novel *Bifidobacterium longum* subsp. *longum* Strains Reveals Functional Divergence in the Human Gut Microbiota" *microorganisms* doi:10.3390/microorganisms9091906
- Dragoš *et al* (2021) "Phages carry interbacterial weapons encoded by biosynthetic gene clusters" *Curr. Biol.* doi:10.1016/j.cub.2021.05.046
- Ducarmon *et al.* (2021) "Microbiota-associated risk factors for asymptomatic gut colonisation with multi-drug-resistant organisms in a Dutch nursing home" *Genome Medicine* doi:0.1186/s13073-021-00869-z
- Fluit *et al.* (2021) "Characterization of clinical *Ralstonia* strains and their taxonomic position." *Antonie van Leeuwenhoek* doi:10.1007/s10482-021-01637-0
- Foucher *et al.* (2021) "Improving common bacterial blight phenotyping by using rub-inoculation and machine learning: cheaper, better, faster, stronger" *Phytopath.* doi:10.1094/PHYTO-04-21-0129-R
- Friedrich *et al.* (2021) "Complete Genome Sequence of *Stenotrophomonas indicatrix* DAIF1" *Micro Res. Ann.* doi:10.1128/MRA.01484-20
- Friedrich *et al.* (2021) "Living in a Puddle of Mud: Isolation and Characterization of Two Novel *Caulobacteraceae* Strains *Brevundimonas pondensis sp. nov.* and *Brevundimonas goettingensis sp. nov.*" *appl. microbiol.* doi:10.3390/applmicrobiol1010005
- Firedrich *et al.* (2021) "Down in the pond: Isolation and characterization of a new *Serratia marcescens* strain (LVF3) from the surface water near frog's lettuce (*Groenlandia densa*)" *PLoS One* doi:10.1371/journal.pone.0259673
- Gai *et al.* (2021) "Chromosome-scale genome sequence of *Alternaria alternata* causing Alternaria brown spot of citrus" *Mol. Plant Microbe Int.* doi:10.1094/MPMI-10-20-0278-SC

- Gauthier *et al.* (2021) "Genomic Perspectives on *Aeromonas salmonicida* subsp. *salmonicida* Strain 890054 as a Model System for Pathogenicity Studies and Mitigation of Fish Infections" *Front. Marine Sci.* doi:10.3389/fmars.2021.744052
- Gallardo-Benavente *et al.* (2021) "Genomics Insights into *Pseudomonas* sp. CG01: An Antarctic Cadmium-Resistant Strain Capable of Biosynthesizing CdS Nanoparticles Using Methionine as S-Source" *genes* doi:10.3390/genes12020187
- Girard *et al.* (2021) "The Ever-Expanding *Pseudomonas* Genus: Description of 43 New Species and Partition of the *Pseudomonas Putida* Group" *preprints* doi:10.20944/preprints202107.0335.v1
- Ghosh *et al.* (2021) "Reconstructing Draft Genomes Using Genome Resolved Metagenomics Reveal Arsenic Metabolizing Genes and Secondary Metabolites in Fresh Water Lake in Eastern India" *Bioinf. Biol. Insights* **'doi:10.1177/11779322211025332 https://doi.org/10.1177/11779322211025332>'_**
- Granehäll *et al.* (2021) "Metagenomic analysis of ancient dental calculus reveals unexplored diversity of oral archaeal *Methanobrevibacter*." *Microbiome* doi:https://doi.org/10.1186/s40168-021-01132-8
- Guerin *et al.* (2021) "Isolation and characterisation of ΦcrAss002, a crAss-like phage from the human gut that infects *Bacteroides xylanisolvens*" *Microbiome* doi:10.1186/s40168-021-01036-7
- Halary *et al.* (2021) "Unexpected Micro-Spatial Scale Genomic Diversity of the Bloom-Forming Cyanobacterium *Aphanizomenon gracile* and its Phycosphere" *Res. Sq.* doi:10.21203/rs.3.rs-617160/v1
- Hansen *et al.* (2021) "Metagenomic sequencing for rapid identification o f*Xylella fastidiosa* from leaf samples" *bioRxiv* doi:10.1101/2021.05.12.443947
- Hertel *et al.* (2021) "Characterization of glyphosate-resistant *Burkholderia anthina* and *Burkholderia cenocepacia* isolates from a commercial Roundup® solution" *Env. Micro. Rep.* doi:10.1111/1758-2229.13022
- Hoetzinger *et al.* (2021) "Dynamics of Baltic Sea phages driven by environmental changes" *Env. Microbiol.* doi:10.1111/1462-2920.15651
- Holzer *et al.* (2021) "Tracking the Distribution of *Brucella abortus* in Egypt Based on Core Genome SNP Analysis and In Silico MLVA-16" *microorganisms* doi:10.3390/microorganisms9091942
- von Hoyningen-Huene *et al.* (2021) "*Pontibacillus* sp. ALD_SL1 and *Psychroflexus* sp. ALD_RP9, two novel moderately halophilic bacteria isolated from sediment and water from the Aldabra Atoll, Seychelles" *PLoS ONE* doi:10.1371/journal.pone.0256639
- Huang *et al.* (2021) "Phenotypic properties and genotyping analysis of *Bacillus cereus* group isolates from dairy and potato products" *LWT* doi:10.1016/j.lwt.2021.110853
- Huang *et al.* (2021) "Genome-resolved metagenomics using environmental and clinical samples" *Brief. Bioinf.* doi:10.1093/bib/bbab030
- Huang *et al.* (2021) "Comparative Genomics and Specific Functional Characteristics Analysis of *Lactobacillus acidophilus*" *microorganisms* doi:10.3390/microorganisms9091992
- Hugouvieux-Cotte-Pattat & Van Gijsegem (2021) "Diversity within the *Dickeya zeae* complex, identification of *Dickeya zeae* and *Dickeya oryzae* members, proposal of the novel species *Dickeya parazeae* sp. nov." *Int. J. Syst. Envol. Microbiol.* doi:10.1099/ijsem.0.005059
- Huihui *et al.* (2021) "Partial biological characteristics and genomic analysis of *Vibrio cholerae* typing phage VP2" *Disease Surv.* doi:10.3784/jbjc.202105190282
- Hünnefeld *et al.* (2021) "Genome Sequence of the Bacteriophage CL31 and Interaction with the Host Strain *Corynebacterium glutamicum* ATCC 13032" *viruses* doi:10.3390/v13030495
- Ivanova *et al.* (2021) "Draft Genome Assemblies of Two *Campylobacter novaezeelandiae* and Four Unclassified Thermophilic *Campylobacter* Isolates from Canadian Agricultural Surface Water" *Microbiol. Res. Ann.* doi:10.1128/MRA.00249-21

- Jian *et al.* (2021) "Diversity and distribution of viruses inhabiting the deepest ocean on Earth" *ISME J.* doi:10.1038/s41396-021-00994-y
- Jungblut *et al.* (2021) "Genomic diversity and CRISPR-Cas systems in the cyanobacterium *Nostoc* in the High Arctic" *Env. Microbiol.* doi:10.1111/1462-2920.15481
- Karaseva *et al.* (2021) "*Fervidicoccus fontis* Strain 3639Fd, the First Crenarchaeon Capable of Growth on Lipids" *Microbiol.* doi:10.1134/S002626172104007X
- Keen *et al.* (2021) "Comparative Genomics of *Mycobacterium avium* Complex Reveals Signatures of Environment-Specific Adaptation and Community Acquisition" *mSystems* doi:10.1128/mSystems.01194-21
- Kim *et al.* (2021) "Real-Time PCR Method for the Rapid Detection and Quantification of Pathogenic *Staphylococcus* Species Based on Novel Molecular Target Genes" *foods* doi:10.3390/foods10112839
- Kim *et al.* (2021) "*Altererythrobacter lutimaris* sp. nov., a marine bacterium isolated from a tidal flat and reclassification of *Altererythrobacter deserti*, *Altererythrobacter estronivorus* and *Altererythrobacter muriae* as *Tsuneonella deserti* comb. nov., *Croceicoccus estronivorus* comb. nov. and *Alteripontixanthobacter muriae* comb. nov." *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.005134
- Koirala *et al.* (2021) "Identification of two novel pathovars of *Pantoea stewartii* subsp. *indologenes* affecting Allium sp. and millets" *Phytopathology* doi:10.1094/PHYTO-11-20-0508-R
- Kuźmińska-Bajor *et al.* (2021) "Genomic and functional characterization of five novel *Salmonella*-targeting bacteriophages." *Virol J* doi:10.1186/s12985-021-01655-4
- Lakra *et al.* (2021) "Genome based reclassification of *Deinococcus swuensis* as a heterotypic synonym of *Deinococcus radiopugnans*" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004879
- Lee *et al.* (2021) "*Bifidobacterium bifidum* strains synergize with immune checkpoint inhibitors to reduce tumour burden in mice" *Nat. Microbiol.* doi:10.1038/s41564-020-00831-6
- Lee *et al.* (2021) "Identification and Characterization of a Novel Genomic Island Harboring Cadmium and Arsenic Resistance Genes in *Listeria welshimeri*" *biomolecules* doi:10.3390/biom11040560
- Lee *et al.* (2021) "Lemierre's syndrome associated with hypervirulent *Klebsiella pneumoniae*: A case report and genomic characterization of the isolate" *IDCases* doi:10.1016/j.idcr.2021.e01173
- Lee *et al.* (2021) "Methane-derived carbon flows into host–virus networks at different trophic levels in soil" *Proc. Natl. Acad. Sci. USA* doi:10.1073/pnas.2105124118
- Lee *et al.* (2021) "Re-classification of *Streptomyces venezuelae* strains and mining secondary metabolite biosynthetic gene clusters" *iScience* doi:10.1016/j.isci.2021.103410
- Li *et al* (2021) "Novel *Paenibacillus sp.* Strains From the Perennial Ryegrass Seed Microbiome Reveal Bioprotectant and Biofertiliser Activity - Differentiating Similar Strains via Genomics and Transcriptomics" *Research Sq.* doi:10.21203/rs.3.rs-445288/v1
- Li *et al.* (2021) "Transcriptomics differentiate two novel bioactive strains of *Paenibacillus* sp. isolated from the perennial ryegrass seed microbiomeTon" *Sci. Rep.* doi:10.1038/s41598-021-94820-2
- Li *et al.* (2021) "Comparative Genomics Analyses Reveal the Differences between *B. longum* subsp. *infantis* and *B. longum* subsp. *longum* in Carbohydrate Utilisation, CRISPR-Cas Systems and Bacteriocin Operons" *microorganisms* doi:10.3390/microorganisms9081713
- Liao *et al.* (2021) "Nationwide genomic atlas of soil-dwelling *Listeria* reveals effects of selection and population ecology on pangenome evolution" *Nat. Microbiol.* doi:10.1038/s41564-021-00935-7
- Liu *et al.* (2021) "*Corynebacterium anserum* sp. nov., isolated from the faeces of greater white-fronted geese (*Anser albifrons*) at Poyang Lake, PR China" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004637
- Liu *et al.* (2021) "Isolation of the Novel Phage PHB09 and Its Potential Use against the Plant Pathogen *Pseudomonas syringae* pv. *actinidiae*" *Viruses* doi:10.3390/v13112275

- Lood *et al.* (2021) "Genomics of an endemic cystic fibrosis *Burkholderia multivorans* strain reveals low within-patient evolution but high between-patient diversity" *PLoS Pathog.* doi:0.1371/journal.ppat.1009418
- López-Pérez *et al.* (2021) "Ecological diversification reveals routes of pathogen emergence in endemic *Vibrio vulnificus* populations" *Proc. Natl. Acad. Sci. USA* doi:10.1073/pnas.2103470118
- López-Pérez *et al.* (2021) "Genomic Characterization of Imipenem- and Imipenem- Relebactam-Resistant Clinical Isolates of *Pseudomonas aeruginosa*" *mSphere* doi:10.1128/mSphere.00836-21
- Lu *et al.* (2021) "Asgard archaea in the haima cold seep: Spatial distribution and genomic insights" *Deep Sea Res. I* doi:10.1016/j.dsr.2021.103489
- Lu *et al.* (2021) "Comparative Genomic Analysis of *Bifidobacterium bifidum* Strains Isolated from Different Niches" *genes* doi:10.3390/genes12101504
- Luo *et al.* (2021) "Isolation and characterization of new phage vB_CtuP_A24 and application to control *Cronobacter* spp. in infant milk formula and lettuce" *Food Res. Int.* doi:10.1016/j.foodres.2021.110109
- Ma *et al.* (2021) "Identification of *Pectobacterium versatile* causing blackleg of potato in New York State" *Plant Disease* doi:10.1094/PDIS-09-20-2089-RE
- Majer *et al.* (2021) "Whole genome sequencing of *Streptomyces actuosus* ISP-5337, *Streptomyces sioyaensis* B-5408, and *Actinospica acidiphila* B-2296 reveals secondary metabolomes with antibiotic potential" *Biotech. Rep.* doi:10.1016/j.btre.2021.e00596
- Matarrita-Carranza *et al.* (2021) "*Streptomyces* sp. M54: an actinobacteria associated with a neotropical social wasp with high potential for antibiotic production." *Antonie van Leeuwenhoek* doi:10.1007/s10482-021-01520-y
- Matsumoto *et al.* (2021) "Complete Genome Sequence of *Acinetobacter pittii* OCU_Ac17, Isolated from Human Venous Blood" *Microbiol. Res. Ann.* doi:10.1128/MRA.00696-21
- Mao *et al.* (2021) "Comparative Genomic Analysis of *Lactiplantibacillus plantarum* Isolated from Different Niches" *genes* doi:10.3390/genes12020241
- Marks *et al.* "*Staphylococcus aureus* injection drug use-associated bloodstream infections are propagated by community outbreaks of diverse lineages" *Commun Med* doi:10.1038/s43856-021-00053-9
- McKay *et al.* (2021) "Sulfur cycling and host-virus interactions in *Aquificales*-dominated biofilms from Yellowstone's hottest ecosystems." *ISME J* doi:10.1038/s41396-021-01132-4
- Misztak *et al.* (2021) "Comparative Genomics and Physiological Investigation of a New *Arthrospira/Limnospira* Strain O9.13F Isolated from an Alkaline, Winter Freezing, Siberian Lake" *cells* doi:10.3390/cells10123411
- Moon *et al* (2021) "Mobile Colistin Resistance Gene mcr-1 Detected on an IncI2 Plasmid in *Salmonella Typhimurium* Sequence Type 19 from a Healthy Pig in South Korea" *microorganisms* doi:10.3390/microorganisms9020398
- Moya-Beltrán *et al.* (2021) "Genomic evolution of the class *Acidithiobacillia*: deep-branching *Proteobacteria* living in extreme acidic conditions" *ISME J.* doi:0.1038/s41396-021-00995-x
- Mullins *et al.* (2021) "Discovery of the Pseudomonas Polyyne Protegencin by a Phylogeny-Guided Study of Polyyne Biosynthetic Gene Cluster Diversity" *mBio* doi:10.1128/mBio.00715-21
- Nascimento *et al.* (2021) "Genomic Analysis of the 1-Aminocyclopropane-1-Carboxylate Deaminase-Producing *Pseudomonas thivervalensis* SC5 Reveals Its Multifaceted Roles in Soil and in Beneficial Interactions With Plants" *Front. Microbiol.* doi:10.3389/fmicb.2021.752288
- Nemec *et al.* (2021) "Delineation of a novel environmental phylogroup of the genus *Acinetobacter* encompassing *Acinetobacter terrae sp. nov.*, *Acinetobacter terrestris sp. nov.* and three other tentative species" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2021.126217
- Nikolaisen *et al.* (2021) "First finding of *Streptococcus phocae* infections in mink (*Neovison vison*)" *Res. Vet. Sci.* doi:10.1016/j.rvsc.2021.07.015

- Nooij *et al.* (2021) "Faecal microbiota transplantation influences procarcinogenic *Escherichia coli* in recipient recurrent *Clostridioides difficile* patients" *Gastroenterology* doi:10.1053/j.gastro.2021.06.009
- Ogg *et al.* (2021) "Pangenome analyses of LuxS-coding genes and enzymatic repertoires in cocoa-related lactic acid bacteria" *Genomics* doi:10.1016/j.ygeno.2021.04.010
- Oliveira *et al.* (2021) "High Genomic Identity between Clinical and Environmental Strains of *Herbaspirillum frisingense* Suggests Pre-Adaptation to Different Hosts and Intrinsic Resistance to Multiple Drugs" *antibiotics* doi:10.3390/antibiotics10111409
- Öhrman *et al.* (2021) "Reorganized Genomic Taxonomy of *Francisellaceae* Enables Design of Robust Environmental PCR Assays for Detection of *Francisella tularensis*" *Microorganisms* doi:10.3390/microorganisms9010146
- Öhrman *et al.* (2021) "Complete Genome Sequence of *Francisella* sp. Strain LA11-2445 (FDC406), a Novel *Francisella* Species Isolated from a Human Skin Lesion" *Micro. Res. Ann.* doi:10.1128/MRA.01233-20
- Pais *et al.* (2021) "Genomic sequencing of different sequevars of *Ralstonia solanacearum* belonging to the Moko ecotype" *Genet. Mol. Bol.* doi:10.1590/1678-4685-gmb-2020-0172
- Panwar *et al.* (2021) "Remarkably coherent population structure for a dominant Antarctic *Chlorobium* species" *Microbiome* doi:10.1186/s40168-021-01173-z
- Pardo-Esté *et al.* (2021) "Genetic Characterization of *Salmonella infantis* with Multiple Drug Resistance Profiles Isolated from a Poultry-Farm in Chile" *microorganisms* doi:10.3390/microorganisms9112370
- Pédron *et al.* (2021) "Early Emergence of *Dickeya solani* Revealed by Analysis of *Dickeya* Diversity of Potato Blackleg and Soft Rot Causing Pathogens in Switzerland" *microorganisms* doi:10.3390/microorganisms9061187
- Pedron *et al.* (2021) "*Mesorhizobium comanense* sp. nov., isolated from groundwater" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.005131
- Pérez-Carrascal *et al.* (2021) "Single-colony sequencing reveals microbe-by-microbiome phylosymbiosis between the cyanobacterium *Microcystis* and its associated bacteria." *Microbiome* doi:10.1186/s40168-021-01140-8
- Petriglieri *et al.* (2021) "Candidatus *Dechloromonas phosphoritropha* and Ca. *D. phosphorivorans*, novel polyphosphate accumulating organisms abundant in wastewater treatment systems" *ISME J.* doi:10.1038/s41396-021-01029-2
- Pidcock *et al.* (2021) "Phylogenetic systematics of *Butyrivibrio* and *Pseudobutyrivibrio* genomes illustrate vast taxonomic diversity, open genomes and an abundance of carbohydrate-active enzyme family isoforms" *Microbial Genomics* doi:10.1099/mgen.0.000638
- Puri *et al.* "Phylogenomic Framework for Taxonomic Delineation of *Paracoccus* spp. and Exploration of Core-Pan Genome" *Ind. J. Microbiol.* doi:10.1007/s12088-021-00929-3
- Román-Reyna *et al.* (2021) "Metagenomic Sequencing for Identification of *Xylella fastidiosa* from Leaf Samples" *mSystems* doi:10.1128/mSystems.00591-21
- Reichler *et al.* (2021) "Identification, subtyping, and tracking of dairy spoilage-associated *Pseudomonas* by sequencing the *ileS* gene" *J. Dairy Sci.* doi:10.3168/jds.2020-19283
- Ryngajłło *et al.* (2021) "Complete genome sequence of lovastatin producer *Aspergillus terreus* ATCC 20542 and evaluation of genomic diversity among *A. terreus* strains" *Appl. Microbiol. Biotechnol.* doi:10.1007/s00253-021-11133-0
- Saati-Santamaría *et al.* (2021) "Phylogenomic Analyses of the Genus _Pseudomonas_ Lead to the Rearrangement of Several Species and the Definition of New Genera" *Biology* doi:10.3390/biology10080782
- Sakiyama *et al.* (2021) "Complete Genome Sequence of a Clinical Isolate of *Acinetobacter baumannii* Harboring 11 Plasmids" *Microbiol. Res. Ann.* doi:10.1128/MRA.00695-21

- Schlez *et al.* (2021) "*Corynebacterium rouxii*, a recently described member of the *C. diphtheriae* group isolated from three dogs with ulcerative skin lesions" *Ant. van Leeuw.* doi:10.1007/s10482-021-01605-8
- Santos *et al.* (2021) "*Phaffia brasiliana* sp. nov., a yeast species isolated from soil in a Cerrado–Atlantic Rain Forest ecotone site in Brazil" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.005080
- Saw *et al.* (2021) "Complete Genome Sequencing of a Novel *Gloeobacter* Species from a Waterfall Cave in Mexico" *Genome Biol. Evol.* doi:10.1093/gbe/evab264
- Schörner *et al.* (2021) "Genomic analysis of *Neisseria elongata* isolate from a patient with infective endocarditis" *FEBS Open Bio* doi:10.1002/2211-5463.13201
- von Schwartzenberg *et al.* (2021) "Caloric restriction disrupts the microbiota and colonization resistance" *Nature* doi:10.1038/s41586-021-03663-4
- Sedaghatjoo *et al.* (2021) "Development of a loop-mediated isothermal amplification assay for the detection of *Tilletia controversa* based on genome comparison" *Sci. Reports.* doi:10.1038/s41598-021-91098-2
- Seibert *et al.* (2021) "*Chlamydia buteonis* in birds of prey presented to California wildlife rehabilitation facilities" *PLoS One* doi:10.1371/journal.pone.0258500
- Singh *et al.* (2021) "Enrichment and description of novel bacteria performing syntrophic propionate oxidation at high ammonia level" *Env. Micro.* doi:10.1111/1462-2920.15388
- Singh *et al.* (2021) "Genome-based reclassification of *Amycolatopsis eurytherma* as a later heterotypic synonym of *Amycolatopsis thermoflava*" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004642
- Son *et al.* (2021) "*Serratia rhizosphaerae sp. nov.*, a novel plant resistance inducer against soft rot disease in tobacco" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004788
- Sorokin *et al.* (2021) "*Natronoglycomyces albus gen. nov., sp. nov*, a haloalkaliphilic actinobacterium from a soda solonchak soil" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.004804
- Strube (2021) "RibDif: can individual species be differentiated by 16S sequencing?" *Bioinf. Adv.* doi:10.1093/bioadv/vbab020
- Suarez *et al.* (2021) "Whole-Genome sequencing and comparative genomics of *Mycobacterium spp.* from farmed Atlantic and coho salmon in Chile" *Antonie van Leeuw.* doi:10.1007/s10482-021-01592-w
- Tegopoulos *et al.* (2021) "Genomic and Phylogenetic Analysis of *Lactiplantibacillus plantarum* L125, and Evaluation of Its Anti-Proliferative and Cytotoxic Activity in Cancer Cells" *biomedicines* doi:10.3390/biomedicines9111718
- Tian *et al.* (2021) "LINflow: a computational pipeline that combines an alignment-free with an alignment-based method to accelerate generation of similarity matrices for prokaryotic genomes" *PeerJ* doi:10.7717/peerj.10906
- Tian *et al.* (2021) "Antifungal mechanism of *Bacillus amyloliquefaciens* strain GKT04 against *Fusarium* wilt revealed using genomic and transcriptomic analyses" *Microbiol. Open* doi:10.1002/mbo3.1192
- Turco *et al.* (2021) "Draft Genome Sequence of a New *Fusarium* Isolate Belonging to *Fusarium tricinctum* Species Complex Collected From Hazelnut in Central Italy" *Front. Plant Sci.* doi:10.3389/fpls.2021.788584
- Undabarrena *et al.* (2021) "*Rhodococcus* comparative genomics reveals a phylogenomic-dependent nonribosomal peptide synthetase distribution: insights into biosynthetic gene cluster connection to an orphan metabolite" *Microb. Genom.* doi:10.1099/mgen.0.000621
- van der Lelie *et al.* (2021) "Rationally designed bacterial consortia to treat chronic immune-mediated colitis and restore intestinal homeostasis" *Nat. Comms.* doi:10.1038/s41467-021-23460-x
- Vargas-Peralta *et al.* (2021) "Identification of *Pectobacterium* species isolated from the soft rot of tetecho (*Neobuxbaumia tetetzo*), a columnar cactus, and associated metagenomics" *bioRxiv* doi:10.1101/2021.02.01.429127

- Verma *et al.* (2021) "Genome analyses of 174 strains of *Mycobacterium tuberculosis* provide insight into the evolution of drug resistance and reveal potential drug targets" *Microb. Genom.* doi:10.1099/mgen.0.000542
- Viera *et al.* (2021) "A highly specific *Serratia*-infecting T7-like phage inhibits biofilm formation in two different genera of the Enterobacteriaceae family" *Res. Microbiol.* doi:10.1016/j.resmic.2021.103869
- Vincent *et al.* (2021) "AsaGEI2d: a new variant of a genomic island identified in a group of *Aeromonas salmonicida* subsp. *salmonicida* isolated from France, which bears the pAsa7 plasmid" *FEMS Micro. Lett.* doi:10.1093/femsle/fnab021
- Volpiano *et al.* (2021) "Genomic Metrics Applied to *Rhizobiales (Hyphomicrobiales)*: Species Reclassification, Identification of Unauthentic Genomes and False Type Strains" *Front. Microbiol.* doi:10.3389/fmicb.2021.614957
- Wang *et al.* (2021). "Spontaneous Bacterial Peritonitis Caused by *Bordetella hinzii*." *Emerging Infectious Diseases* doi:10.3201/eid2711.211428
- Wang *et al.* (2021) "Dynamic impact of virome on colitis and colorectal cancer: Immunity, inflammation, prevention and treatment" *Sem. Cancer Biol.* doi:10.1016/j.semcancer.2021.10.004
- Watson *et al.* (2021) "Adaptive ecological processes and metabolic independence drive microbial colonization and resilience in the human gut" *bioRxiv* doi:10.1101/2021.03.02.433653
- Wu *et al.* (2021) "Metagenomic insights into nitrogen and phosphorus cycling at the soil aggregate scale driven by organic material amendments" *Sci. Tot. Env.* doi:10.1016/j.scitotenv.2021.147329
- Wu *et al.* (2021) "Moisture modulates soil reservoirs of active DNA and RNA viruses." *Commun Biol* doi:10.1038/s42003-021-02514-2
- Wu *et al.* (2021) "An Effective Preprocessing Method for High-Quality Pan-Genome Analysis of *Bacillus subtilis* and *Escherichia coli*" *Essential Genes and Genomes* doi:10.1007/978-1-0716-1720-5_21
- Wu *et al.* (2021) "DNA Viral Diversity, Abundance, and Functional Potential Vary across Grassland Soils with a Range of Historical Moisture Regimes" *mBio* doi:doi.org/10.1128/mBio.02595-21
- Xiao *et al.* (2021) "Carbapenem-resistant *Acinetobacter Baumannii* Ventilator-Associated Pneumonia in Critically Ill Patients: Potential Inference with Respiratory Tract Microbiota Dysbiosis" *Res. Sq.* doi:10.21203/rs.3.rs-736916/v1
- Young *et al.* (2021) "Defining the *Rhizobium leguminosarum* Species Complex" *genes* doi:10.3390/genes12010111
- Zeng *et al.* (2021) "Novel phage vB_CtuP_B1 for controlling *Cronobacter malonaticus* and *Cronobacter turicensis* in ready-to-eat lettuce and powered infant formula" *Food Res. Int.* doi:10.1016/j.foodres.2021.110255
- Zhou *et al.* (2021) "Comparative genomic analysis of *Mycoplasma anatis* strains" *Genes and Genomics* doi:0.1007/s13258-021-01129-5
- Zhou *et al.* (2021) "Development of a high resolution melting method based on a novel molecular target for discrimination between *Bacillus cereus* and *Bacillus thuringiensis*" *Food Res. Int.* doi:10.1016/j.foodres.2021.110845
- Zvi-Kedem *et al.* (2021) "The worm affair: Genetic diversity in two species of symbionts that co-occur in tubeworms from the Mediterranean Sea" *Env. Microbiol.* doi:10.1111/1758-2229.12994

2020

- Akob *et al.* (2020) "Mixotrophic Iron-Oxidizing *Thiomonas* Isolates from an Acid Mine Drainage-Affected Creek" *App. Env. Microbiol.* doi:10.1128/AEM.01424-20
- Albert *et al.* (2020) "Comparative Pangenomics of the Mammalian Gut Commensal *Bifidobacterium longum*" *Microorganisms* doi:10.3390/microorganisms8010007

- Bech *et al.* (2020) "Marine Sediments Hold an Untapped Potential for Novel Taxonomic and Bioactive Bacterial Diversity" *mSystems* doi:/10.1128/mSystems.00782-20
- Biggel *et al.* (2020) "Horizontally acquired papGII-containing pathogenicity islands underlie the emergence of invasive uropathogenic *Escherichia coli* lineages" *Nat, Comm.* doi:10.1038/s41467-020-19714-9
- Bisanz *et al.* (2020) "A Genomic Toolkit for the Mechanistic Dissection of Intractable Human Gut Bacteria" *Cell Host & Microbe* doi:10.1016/j.chom.2020.04.006
- Bridel *et al.* (2020) "Genetic diversity and population structure of *Tenacibaculum maritimum*, a serious bacterial pathogen of marine fish: from genome comparisons to high throughput MALDI-TOF typing" *Vet. Res.* doi:10.1186/s13567-020-00782-0
- Brock *et al.* (2020) "Endosymbiotic adaptations in three new bacterial species associated with *Dictyostelium discoideum*: *Paraburkholderia agricolaris* sp. nov., *Paraburkholderia hayleyella* sp. nov., and *Paraburkholderia bonniea* sp. nov" *PeerJ* doi:10.7717/peerj.9151
- Busch *et al.* (2020) "Using affinity propagation clustering for identifying bacterial clades and subclades with whole-genome sequences of *Francisella tularensis*" *PLoS Neg. Trop. Dis.* doi:10.1371/journal.pntd.0008018
- Cai *et al.* (2020) "Comparative genomics of *Klebsiella michiganensis* BD177 and related members of *Klebsiella* sp. reveal the symbiotic relationship with *Bactrocera dorsalis*" *BMC Genetics* doi:10.1186/s12863-020-00945-0
- Cassaniti *et al.* (2020) "Authors' response: COVID-19: how accurate are seroprevalence studies?" *Eurosurveillance* doi:10.2807/1560-7917.ES.2020.25.30.2001437
- Chibani *et al.* (2020) "Genomic variation among closely related *Vibrio alginolyticus* strains is located on mobile genetic elements" *BMC Genomics* doi:10.1186/s12864-020-6735-5
- Christman *et al.* (2020) "Novel clostridial lineages recovered from metagenomes of a hot oil reservoir" *Sci. Rep.* doi:10.1038/s41598-020-64904-6
- Christman *et al.* (2020) "Methanogens Within a High Salinity Oil Reservoir From the Gulf of Mexico" *Front. Microbiol.* doi:10.3389/fmicb.2020.570714
- Cunningham-Oakes *et al.* (2020) "Genome Sequence of *Pluralibacter gergoviae* ECO77, a Multireplicon Isolate of Industrial Origin" *Microbiol. Res. Ann.* doi:10.1128/MRA.01561-19
- Dahihausen *et al.* (2020) "Isolation and sequence-based characterization of a koala symbiont: *Lonepinella koalarum*" *PeerJ* doi:10.7717/peerj.10177
- Dam *et al.* (2020) "Targeted Cell Sorting Combined With Single Cell Genomics Captures Low Abundant Microbial Dark Matter With Higher Sensitivity Than Metagenomics" *Front. Microbiol.* doi:10.3389/fmicb.2020.01377
- Damnjanovich *et al.* (2020) "Bacteriophage genotyping using BOXA repetitive-PCR" *BMC Microbiol.* doi:10.1186/s12866-020-01770-2
- Dangel *et al.* (2020) "*Corynebacterium silvaticum* sp. nov., a unique group of NTTB corynebacteria in wild boar and roe deer" *Int. J. Syst. Evol. Microb.* doi:10.1099/ijsem.0.004195
- de Andrade Alves *et al.* "First isolation and whole-genome sequencing of a *Shewanella algae* strain from a swine farm in Brazil" *BMC Microbiol.* doi:10.1186/s12866-020-02040-x
- Déraspe *et al.* (2020) "Genome Sequence of a *Klebsiella pneumoniae* NDM-1 Producer Isolated in Quebec City" *Microbiol. Resour. Announc.* doi:10.1128/MRA.00829-19
- Donner *et al.* (2020) "Septic shock caused by *Capnocytophaga canis* after a cat scratch" *Eur. J. Clin. Microbiol. Inf. Dis.* doi:10.1007/s10096-020-03922-8
- Duar *et al.* (2020) "Comparative Genome Analysis of *Bifidobacterium longum* subsp. *infantis* Strains Reveals Variation in Human Milk Oligosaccharide Utilization Genes among Commercial Probiotics" *Nutrients* doi:10.3390/nu12113247

- Elcheninov *et al.* (2020) "*Thermogemmata fonticola* gen. nov., sp. nov., the first thermophilic planctomycete of the order *Gemmatales* from a Kamchatka hot spring" *Syst. App. Micro.* doi:10.1016/j.syapm.2020.126157
- Feng *et al.* (2020) "Phylogenetic and genomic analysis reveals high genomic openness and genetic diversity of *Clostridium perfringens*" *Microb. Gen.* doi:10.1099/mgen.0.000441
- Ferrerira *et al.* "Genome-based reclassification of *Azospirillum brasilense* Sp245 as the type strain of *Azospirillum baldaniorum* sp. nov" *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.004517
- Fishbein *et al.* (2020) "Randomized Controlled Trial of Oral Vancomycin Treatment in *Clostridioides difficile*-Colonized Patients" *mSphere* doi:10.1128/mSphere.01296-20
- France *et al.* (2020) "Complete Genome Sequences of Six *Lactobacillus iners* Strains Isolated from the Human Vagina" *Microbiol. Res. Ann.* doi:10.1128/MRA.00234-20
- Francoeur *et al.* (2020) "Bacteria Contribute to Plant Secondary Compound Degradation in a Generalist Herbivore System" *mBio* doi:doi.org/10.1128/mBio.02146-20
- Freitas *et al.* (2020) "Yeast communities associated with cacti in Brazil and the description of *Kluyveromyces starmeri* sp. nov. based on phylogenomic analyses" *Yeast* doi:10.1002/yea.3528
- Friedrich *et al.* (2020) "First Complete Genome Sequences of *Janthinobacterium lividum* EIF1 and EIF2 and their Comparative Genome Analysis" *Genome Biol. Evol.* doi:10.1093/gbe/evaa148
- Furrer *et al.* (2020) "Phage vB_BveM-Goe7 represents a new genus in the subfamily *Bastillevirinae*" *Arch. Virol.* doi:10.1007/s00705-020-04546-1
- Gabor *et al.* (2020) "A New Species of Genus *Pseudomonas*" United States Patent Application 20200216503 20200216503
- Gai *et al.* (2020) "The Genome Sequence of the Citrus Melanose Pathogen *Diaporthe citri* and Two Citrus related *Diaporthe* species" *Phytopathology* doi:10.1094/PHYTO-08-20-0376-SC
- Gardon *et al.* (2020) "A drift-barrier model drives the genomic landscape of a structured bacterial population" *Molecular Ecol.* doi:10.1111/mec.15628
- Girard *et al.* (2020) "Reliable Identification of Environmental *Pseudomonas* Isolates Using the *rpoD* Gene" *Microorganisms* doi:10.3390/microorganisms8081166
- González-Dominici *et al.* (2020) "Genome Analysis and Genomic Comparison of the Novel Species *Arthrobacter ipsi* Reveal Its Potential Protective Role in Its Bark Beetle Host" *Microbial Ecol.* doi:10.1007/s00248-020-01593-8
- González-Gómez *et al.* (2020) "Phylogenomic Analysis Supports Two Possible Origins for Latin American Strains of *Vibrio parahaemolyticus* Associated with Acute Hepatopancreatic Necrosis Disease (AHPND)" *Curr. Microbiol.* doi:10.1007/s00284-020-02214-w
- Gramaje *et al.* (2020) "Comparative Genomic Analysis of *Dactylonectria torresensis* Strains from Grapevine, Soil and Weed Highlights Potential Mechanisms in Pathogenicity and Endophytic Lifestyle" *J. Fungi* doi:10.3390/jof6040255
- Graña-Miraglia *et al.* (2020) "Spirochetes isolated from arthropods constitute a novel genus *Entomospira* genus novum within the order Spirochaetales" *Sci. Rep.* doi:10.1038/s41598-020-74033-9
- Hempel *et al.* (2020) "Complete Genome Sequence of *Bacillus velezensis* Strain S4, Isolated from Biochar-Treated Soil" *Microbiol. Res. Ann.* doi:10.1128/MRA.00352-20
- Hinger *et al.* (2020) "Phylogenomic Analyses of Members of the Widespread Marine Heterotrophic Genus *Pseudovibrio* Suggest Distinct Evolutionary Trajectories and a Novel Genus, *Polycladidibacter* gen. nov." *Appl. Env. Microbiol.* doi:10.1128/AEM.02395-19
- Hollensteiner *et al.* (2020) "Genome Sequence of *Komagataeibacter saccharivorans* Strain JH1, Isolated from Fruit Flies" *Microbiol. Res. Announc.* doi:10.1128/MRA.00098-20

- Hulin *et al.* (2020) "Cherry picking by pseudomonads: after a century of research on canker, genomics provides insights into the evolution of pathogenicity towards stone fruits" *Plant Pathology* doi:10.1111/ppa.13189
- Ibarra Caballero *et al.* (2020) "Genome comparison and transcriptome analysis of the invasive brown root rot pathogen, *Phellinus noxius*, from different geographic regions reveals potential enzymes associated with degradation of different wood substrates" *Fungal Biology* doi:10.1016/j.funbio.2019.12.007
- Inderbitzin *et al.* (2020) "Species identification in plant-associated prokaryotes and fungi using DNA" *Phytobiomes J.* doi:10.1094/PBIOMES-12-19-0067-RVW
- Jin *et al* (2020) "Complete genome sequence of fish-pathogenic *Aeromonas hydrophila* HX-3 and a comparative analysis: insights into virulence factors and quorum sensing" *Sci. Rep.* doi:10.1038/s41598-020-72484-8
- Joglekar *et al.* (2020) "Polyphasic analysis reveals correlation between phenotypic and genotypic analysis in soybean bradyrhizobia (*Bradyrhizobium* spp.)" *Syst. Appl. Microb.* doi:10.1016/j.syapm.2020.126073
- Joutsen *et al.* (2020) "Two copies of the *ail* gene found in *Yersinia enterocolitica* and *Yersinia kristensenii*" *Vet. Micro.* doi:10.1016/j.vetmic.2020.108798
- Jung *et al.* (2020) "Genome Analysis of *Enterococcus mundtii* Pe103, a Human Gut-Originated Pectinolytic Bacterium" *Curr. Microbiol.* doi:10.1007/s00284-020-01932-5
- Kim *et al.* (2020) "Genome analysis of *Lactobacillus plantarum* subsp. *plantarum* KCCP11226 reveals a well-conserved C30 carotenoid biosynthetic pathway" *3 Biotech.* doi:10.1007/s13205-020-2149-y
- Kim *et al.* (2020) "Comparative Genomics Determines Strain-Dependent Secondary Metabolite Production in *Streptomyces venezuelae* Strains" *Biomolecules* doi:10.3390/biom10060864
- Kornienko *et al.* (2020) "Contribution of *Podoviridae* and *Myoviridae* bacteriophages to the effectiveness of anti-staphylococcal therapeutic cocktails" *Sci. Rep.* doi:10.1038/s41598-020-75637-x
- Kroll *et al.* (2020) "Microbiota supplementation with *Bifidobacterium* and *Lactobacillus* modifies the preterm infant gut microbiota and metabolome: an observational study" *Cell Reports Medicine* doi:10.1016/j.xcrm.2020.100077
- Kuleshov *et al.* (2020) "Whole genome sequencing of *Borrelia miyamotoi* isolate Izh-4: reference for a complex bacterial genome" *BMC Genomics* doi:10.1186/s12864-019-6388-4
- Kumar *et al.* (2020) "Phylogenetic Relationship Among Brackishwater *Vibrio* Species" *Evol. Bioinf.* doi:10.1177/1176934320903288
- La China *et al.* (2020) "Genome sequencing and phylogenetic analysis of K1G4: a new *Komagataeibacter* strain producing bacterial cellulose from different carbon sources" *Biotech. Lett.* doi:10.1007/s10529-020-02811-6
- Lacault *et al.* (2020) "Zucchini vein clearing disease is caused by several lineages within *Pseudomonas syringae* species complex." *Phytopathology* doi:10.1094/PHYTO-07-19-0266-R
- Leyer *et al.* (2020) "*Avrilella dinanensis* gen. nov., sp. nov., a novel bacterium of the family *Flavobacteriaceae* isolated from human blood" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2020.126124
- Li *et al.* (2020) "Metabolic diversification of anaerobic methanotrophic archaea in a deep-sea cold seep" *Marine Life Sci. Tech.* doi:10.1007/s42995-020-00057-9
- Li *et al.* (2020) "Comparative Genomics Reveals Broad Genetic Diversity, Extensive Recombination and Nascent Ecological Adaptation in *Micrococcus luteus*" *Research Square* doi:10.21203/rs.3.rs-62334/v1
- Li *et al.* (2020) "Isolation and Characterization of *Bacillus cereus* Phage vB_BceP-DLc1 Reveals the Largest Member of the Phi29-Like Phages" *Microorganisms* doi:10.3390/microorganisms8111750
- Liu *et al.* (2020) "Whole genome sequence and comparative genome analyses of multi-resistant *Staphylococcus warneri* GD01 isolated from a diseased pig in China" *PLoS One* doi:10.1371/journal.pone.0233363
- Liu *et al.* (2020) "Pathogenicity of a *Vibrio owensii* strain isolated from *Fenneropenaeus chinensis* carrying pirAB genes and causing AHPND" *Aquaculture* doi:10.1016/j.aquaculture.2020.735747

- Long *et al.* (2020) "Polyclonality, Shared Strains, and Convergent Evolution in Chronic CF *S. aureus* Airway Infection" *Am. J. Resp. Crit. Care Med.* doi:10.1164/rccm.202003-0735O
- Machado *et al.* (2020) "Molecular relationships of *Campomanesia xanthocarpa* within Myrtaceae based on the complete plastome sequence and on the plastid ycf2 gene" *Genet. Mol. Biol.* doi:10.1590/1678-4685-gmb-2018-0377
- Méndez *et al.* (2020) "Comparative Genomics of Pathogenic *Clavibacter michiganensis* subsp. *michiganensis* Strains from Chile Reveals Potential Virulence Features for Tomato Plants" *Microorganisms* doi:10.3390/microorganisms8111679
- Min *et al.* (2020) "Complete Genomic Analysis of *Enterococcus faecium* Heat-Resistant Strain Developed by Two-Step Adaptation Laboratory Evolution Method" *Front. Bioeng. Biotechnol.* doi:10.3389/fbioe.2020.00828
- Mino *et al.* (2020) "*Hydrogenimonas urashimensis* sp. nov., a hydrogen-oxidizing chemolithoautotroph isolated from a deep-sea hydrothermal vent in the Southern Mariana Trough" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2020.126170
- Miyoshi *et al.* (2020) "Early-life microbial intervention reduces colitis risk promoted by antibiotic-induced gut dysbiosis" *bioRxiv* doi:10.1101/2020.03.11.987412
- Modesto *et al.* (2020) "Bifidobacteria in two-toed sloths (*Choloepus didactylus*): phylogenetic characterization of the novel taxon *Bifidobacterium choloepi* sp. nov." *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.004506
- Moon *et al.* (2020) "First Report of an *Escherichia coli* Strain Carrying the Colistin Resistance Determinant *mcr-1* from a Dog in South Korea" *Antibiotics* doi:10.3390/antibiotics9110768
- Moore *et al.* (2020) "Draft Genome Sequence of *Lactobacillus rhamnosus* NCB 441, Isolated from Egyptian White Domiati Cheese" *Micro. Res. Ann.* doi:10.1128/MRA.01191-20
- Mu *et al.* (2020) "*Bradymonabacteria*, a novel bacterial predator with versatile survival strategies in saline environments" *Microbiome* doi:10.21203/rs.2.20535/v1
- Mu *et al.* (2020) "*Tichowtungia aerotolerans* gen. nov., sp. nov., a novel representative of the phylum *Kiritimatiellaeota* and proposal of *Tichowtungiaceae* fam. nov., *Tichowtungiales* ord. nov. and *Tichowtungiia* class. nov." *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.004370
- Müller *et al.* (2020) "*Aliarcobacter butzleri* from Water Poultry: Insights into Antimicrobial Resistance, Virulence and Heavy Metal Resistance" *Genes* doi:10.3390/genes11091104
- Mullins *et al.* (2020) "Genomic Assemblies of Members of *Burkholderia* and Related Genera as a Resource for Natural Product Discovery" *Micro. Res. Ann.* doi:10.1128/MRA.00485-20
- Mullins *et al.* (2020) "Reclassification of the biocontrol agents *Bacillus subtilis* BY-2 and Tu-100 as *Bacillus velezensis* and insights into the genomic and specialized metabolite diversity of the species" *Microbiol.* doi:10.1099/mic.0.000986
- Nilsson *et al.* (2020) "Diversity and Host Interactions Among Virulent and Temperate Baltic Sea *Flavobacterium* Phages" *Viruses* doi:10.3390/v12020158
- Norris *et al.* (2020) "*Acidithiobacillus ferrianus* sp. nov.: an ancestral extremely acidophilic and facultatively anaerobic chemolithoautotroph" *Extremophiles* doi:0.1007/s00792-020-01157-1
- Oshkin *et al.* (2020) "Pan-Genome-Based Analysis as a Framework for Demarcating Two Closely Related Methanotroph Genera Methylocystis and Methylosinus" *Microorganisms* doi:10.3390/microorganisms8050768
- Pandey *et al.* (2020) "Evidence of homologous recombination as a driver of diversity in *Brachyspira pilosicoli*" *Micro. Genom.* doi:10.1099/mgen.0.000470
- Paulsen *et al.* (2020) "Production of the antimicrobial compound tetrabromopyrrol and the *Pseudomonas* quinolone system precursor, 2-heptyl-4-quinolone, by a novel marine species *Pseudoalteromonas galatheae* sp. nov." *Sci. Rep.* doi:10.1038/s41598-020-78439-3

- Pang *et al.* (2020) "The Genomic Context for the Evolution and Transmission of Community-Associated *Staphylococcus aureus* ST59 Through the Food Chain" *Front. Microbiol.* doi:10.3389/fmicb.2020.00422
- Panwar *et al.* (2020) "Influence of the polar light cycle on seasonal dynamics of an Antarctic lake microbial community" *Microbiome* doi:10.1186/s40168-020-00889-8
- Parlikar *et al.* (2020) "Understanding genomic diversity, pan-genome, and evolution of SARS-CoV-2" *PeerJ* doi:10.7717/peerj.9576
- Pasanen *et al.* (2020) "*Pectobacterium parvum* sp. nov., having a Salmonella SPI-1-like Type III secretion system and low virulence" *Int. J. Syst. Evol. Microb.* doi:10.1099/ijsem.0.004057
- Peral-Aranega *et al.* (2020) "Bacteria Belonging to *Pseudomonas typographi* sp. nov. from the Bark Beetle *Ips typographus* Have Genomic Potential to Aid in the Host Ecology" *Insect* doi::10.3390/insects11090593
- Pierry *et al.* (2020) "High-quality draft genome sequence resources of eight *Xylella fastidiosa* strains isolated from citrus, coffee, plum and hibiscus in South America" *Phytopathology* doi:10.1094/PHYTO-05-20-0162-A
- Pierry *et al.* (2020) "Genetic Diversity of *Xylella fastidiosa* Plasmids Assessed by Comparative Genomics" *Trop. Plant Path.* doi:doi.org/10.1007/s40858-020-00359-4
- Portier *et al.* (2020) "Updated taxonomy of *Pectobacterium* genus in the CIRM-CFBP bacterial collection: when newly described species reveal "old" endemic population" *Preprints* doi:10.20944/preprints202008.0608.v1
- Rackaityte (2020) "Viable bacterial colonization is highly limited in the human intestine in utero" *Nature Medicine* doi:10.1038/s41591-020-0761-3
- Roach *et al.* (2020) "Whole genome sequencing of Peruvian *Klebsiella pneumoniae* identifies novel plasmid vectors bearing carbapenem resistance gene NDM-1" *Open Forum Inf. Dis.* doi:10.1093/ofid/ofaa266/5866602
- Rothen *et al.* (2020) "A simple, rapid typing method for *Streptococcus agalactiae* based on ribosomal subunit proteins by MALDI-TOF MS" *Sci. Reports* doi:10.1038/s41598-020-65707-5
- Ryngajłło *et al.* (2020) "Towards control of cellulose biosynthesis by *Komagataeibacter* using systems-level and strain engineering strategies: current progress and perspectives" *Appl. Microbil. Biotech.* doi:10.1007/s00253-020-10671-3
- Salgar-Chaparro *et al.* (2020) "Complete Genome Sequence of *Pseudomonas balearica* Strain EC28, an Iron-Oxidizing Bacterium Isolated from Corroded Steel" *Microbiol. Res. Ann.* doi:10.1128/MRA.00275-20
- Salgar-Chaparro *et al.* (2020) " Draft Genome Sequence of *Enterobacter roggenkampii* Strain OS53, Isolated from Corroded Pipework at an Offshore Oil Production Facility" *Microbiol. Res. Ann.* doi:10.1128/MRA.00583-20
- Salgar-Chaparro *et al.* (2020) "Complete Genome Sequence of *Shewanella chilikensis* Strain DC57, Isolated from Corroded Seal Rings at a Floating Oil Production System in Australia" *Microbiol. Res. Announc.* doi:0.1128/MRA.00584-20
- Shen *et al.* (2020) "*Helicobacter monodelphidis* sp. nov. and *Helicobacter didelphidarum* sp. nov., isolated from grey short-tailed opossums (*Monodelphis domestica*) with endemic cloacal prolapses" *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.004424
- Strang (2020) "Genomic Insights and Ecological Adaptations of Deep-Subsurface and Near Subsurface *Thermococcus* Isolates and Near Subsurface *Thermococcus* Isolates" *WWU Graduate School Collection* https://cedar.wwu.edu/wwuet/926
- Taparia *et al.* (2020) "Molecular characterization of *Pseudomonas* from *Agaricus bisporus* caps reveal novel blotch pathogens in Western Europe" *BMC Genomics* doi:10.1186/s12864-020-06905-3
- Tardy *et al.* (2020) "*Mycoplasma bovis* in Nordic European Countries: Emergence and Dominance of a New Clone" *Pathogens* doi:10.3390/pathogens9110875

- Thapa *et al.* (2020) "Genome-wide analyses of Liberibacter species provides insights into evolution, phylogenetic relationships, and virulence factors" *Mol. Plant Path.* doi:10.1111/mpp.12925
- Tian *et al.* (2020) "LINbase: a web server for genome-based identification of prokaryotes as members of crowdsourced taxa" *Nuc. Acids Res.* doi:10.1093/nar/gkaa190
- Tsukimi *et al.* (2020) "Draft Genome Sequences of *Bifidobacterium animalis* Consecutively Isolated from Healthy Japanese Individuals" *J. Genomics* doi:10.7150/jgen.38516
- Vijayan *et al.* (2020) "Bacteria known to induce settlement of larvae of *Hydroides elegans* are rare in natural inductive biofilm" *Aquatic Microb. Ecol.* doi:10.3354/ame01925
- Waleron *et al.* (2020) "*Arthrospiribacter ruber* gen. nov., sp. nov., a novel bacterium isolated from *Arthrospira* cultures" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2020.126072
- Wang *et al.* (2020) "Comparative Genomics Analysis of *Lactobacillus ruminis* from Different Niches" *Genes* doi:10.3390/genes11010070
- Wang *et al.* (2020) "Complete genomic data of Burkholderia glumae strain GX associated with bacterial panicle blight of rice in China" *Plant Dis.* doi:10.1094/PDIS-10-19-2265-A
- Weiser *et al.* (2020) "A Novel Inducible Prophage from *Burkholderia Vietnamiensis* G4 is Widely Distributed across the Species and Has Lytic Activity against Pathogenic Burkholderia" *Viruses* doi:10.3390/v12060601
- Webster *et al.* (2020) "Culturable diversity of bacterial endophytes associated with medicinal plants of the Western Ghats, India" *FEMS Microbiol. Ecol.* doi:10.1093/femsec/fiaa147/5876344
- Wist *et al.* (2020) "Phenotypic and Genotypic Traits of Vancomycin-Resistant *Enterococci* from Healthy Food-Producing Animals" *Microorganisms* doi:10.3390/microorganisms8020261
- Wu *et al.* (2020) "Toward a high-quality pan-genome landscape of *Bacillus subtilis* by removal of confounding strains" *Brief. Bioinf.* doi:10.1093/bib/bbaa013
- Yang *et al.* (2020) "Isolation and Characterization of the Novel Phages vB_VpS_BA3 and vB_VpS_CA8 for Lysing *Vibrio parahaemolyticus*" *Front. Microbiol.* doi:10.3389/fmicb.2020.00259
- Zayulina *et al.* "Complete Genome Sequence of a Hyperthermophilic Archaeon, *Thermosphaera* sp. Strain 3507, Isolated from a Chilean Hot Spring" *Micro. Res. Ann.* doi:10.1128/MRA.01262-20
- Zhang *et al.* (2020) "A novel bacterial thiosulfate oxidation pathway provides a new clue about the formation of zero-valent sulfur in deep sea." *ISME J.* doi:10.1038/s41396-020-0684-5
- Zhang *et al.* (2020) "*Deinococcus detaillensis* sp. nov., isolated from humus soil in Antarctica" *Arch. Microbiol.* doi:10.1007/s00203-020-01920-0
- Zhang *et al.* (2020) "Chloramphenicol biodegradation by enriched bacterial consortia and isolated strain *Sphingomonas* sp. CL5.1: The reconstruction of a novel biodegradation pathway" *Water Res.* doi:10.1016/j.watres.2020.116397
- Zheng *et al.* (2020) "Metagenomic Insight into Environmentally Challenged Methane-Fed Microbial Communities" *Microorganisms* doi:10.3390/microorganisms8101614
- Zhou *et al.* (2020) "Comparative analysis of *Lactobacillus gasseri* from Chinese subjects reveals a new species-level taxa" *BMC Genomics* doi:10.1186/s12864-020-6527-y

2019

- Accetto & Avgustin (2019) "The diverse and extensive plant polysaccharide degradative apparatuses of the rumen and hindgut *Prevotella* species: A factor in their ubiquity?" *Syst. Appl. Microbiol.* doi:j.syapm.2018.10.001
- Acevedo *et al.* (2019) "*Bacillus clarus* sp. nov. is a new *Bacillus cereus* group species isolated from soil" *BioRxiv* doi:10.1101/508077

- Alberoni *et al.* (2019) "*Bifidobacterium xylocopae* sp. nov. and *Bifidobacterium aemilianum *sp. nov., from the carpenter bee (*Xylocopa violacea*) digestive tract" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2018.11.005
- Alex & Antunes (2019) "Whole-Genome Comparisons Among the Genus *Shewanella* Reveal the Enrichment of Genes Encoding Ankyrin-Repeats Containing Proteins in Sponge-Associated Bacteria" *Front. Microbiol.* doi:10.3389/fmicb.2019.00005
- Alex & Antunes (2019) "Comparative Genomics Reveals Metabolic Specificity of *Endozoicomonas* Isolated from a Marine Sponge and the Genomic Repertoire for Host-Bacteria Symbioses" *Microorganisms* doi:10.3390/microorganisms7120635
- Barnier *et al.* (2019) "Description of *Palleronia rufa* sp. nov., a biofilm-forming and AHL-producing *Rhodobacteraceae*, reclassification of *Hwanghaeicola aestuarii* as *Palleronia aestuarii* comb. nov., *Maribius pontilimi* as *Palleronia pontilimi* comb. nov., *Maribius salinus* as *Palleronia salina* comb. nov., *Maribius pelagius* as *Palleronia pelagia* comb. nov. and emended description of the genus *Palleronia*" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2019.126018
- Bayjanov *et al.* (2019) "Whole genome analysis of *Pandoraea* species strains from cystic fibrosis patients" *Future Microbiology* doi:10.2217/fmb-2019-0038
- Botelho *et al.* (2019) "Combining sequencing approaches to fully resolve a carbapenemase-encoding megaplasmid in a Pseudomonas shirazica clinical strain" *Emerg. Microb. Inf.* doi:10.1080/22221751.2019.1648182
- Boukerb *et al.* (2019) "*Campylobacter armoricus sp. nov.*, a novel member of the *Campylobacter lari* group isolated from surface water and stools from humans with enteric infection" *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.003836
- Briand *et al.* (2019) "A rapid and simple method for assessing and representing genome sequence relatedness" *BioRxiv* doi:10.1101/569640
- Cho & Kwak (2019) "Evolution of Antibiotic Synthesis Gene Clusters in the *Streptomyces globisporus* TFH56, Isolated from Tomato Flower" *G3: Genes, Genomes, Genetics* doi:10.1534/g3.119.400037
- Ciok & Dziewit (2019) "Exploring the genome of Arctic *Psychrobacter* sp. DAB_AL32B and construction of novel *Psychrobacter*-specific cloning vectors of an increased carrying capacity" *Arch. Microbiol.* doi:10.1007/s00203-018-1595-y
- D'Souza *et al.* (2019) "Spatiotemporal dynamics of multidrug resistant bacteria on intensive care unit surfaces" *Nat. Comm.* doi:10.1038/s41467-019-12563-1
- do Vale *et al.* (2019) "Draft Genome Sequences of Three Novel *Acinetobacter* Isolates from an Irish Commercial Pig Farm" *Microbiol. Res. Ann.* doi:10.1128/MRA.00919-19
- Doud *et al.* (2019) "Function-driven single-cell genomics uncovers cellulose-degrading bacteria from the rare biosphere" *ISME J.* doi:10.1038/s41396-019-0557-y
- Du *et al.* (2019) "Characterization of a Linezolid- and Vancomycin-Resistant *Streptococcus suis* Isolate That Harbors optrA and vanG Operons" *Front. Microbiol.* doi:10.3389/fmicb.2019.02026
- Esposito *et al.* (2019) "Insights into the genome structure of four acetogenic bacteria with specific reference to the Wood–Ljungdahl pathway" *Microbiol. Open* doi:10.1002/mbo3.938
- Falagan *et al.* (2019) "Acidithiobacillus sulfuriphilus sp. nov.: an extremely acidophilic sulfur-oxidizing chemolithotroph isolated from a neutral pH environment" *Int. J. Syst. Evol. Micro.* doi:0.1099/ijsem.0.003576
- Faoro *et al.* (2019) "Genome comparison between clinical and environmental strains of *Herbaspirillum seropedicae* reveals a potential new emerging bacterium adapted to human hosts" *BMC Genomics* doi:10.1186/s12864-019-5982-9
- Feng *et al.* (2019) "Complete genome sequence of *Hahella* sp. KA22, a prodigiosin-producing algicidal bacterium" *Marine Genomics* doi:10.1016/j.margen.2019.04.003

- Gasparrini *et al.* (2019) "Metagenomic signatures of early life hospitalization and antibiotic treatment in the infant gut microbiota and resistome persist long after discharge" *Nature Microbiol.* doi:10.1038/s41564-019-0550-2
- Ghosh *et al.* (2019) "Reanalysis of *Lactobacillus paracasei* Lbs2 Strain and Large-Scale Comparative Genomics Places Many Strains into Their Correct Taxonomic Position" *Microorganisms* doi:10.3390/microorganisms7110487
- Hollensteiner *et al.* (2019) "Complete Genome Sequence of *Marinobacter* sp. Strain JH2, Isolated from Seawater of the Kiel Fjord" *Micro. Res. Ann.* doi:10.1128/MRA.00596-19
- Hornung *et al.* (2019) "An in silico survey of *Clostridioides difficile* extrachromosomal elements" *BioRxiv* doi:10.1101/651539
- Huang *et al.* (2019) "Genomic differences within the phylum Marinimicrobia: From waters to sediments in the Mariana Trench" *Marine Genomics* doi:10.1016/j.margen.2019.100699
- Ide *et al.* (2019) "Draft Genome Sequence of *Acidovorax* sp. Strain NB1, Isolated from a Nitrite-Oxidizing Enrichment Culture" *Micro. Res. Ann.* doi:10.1128/MRA.00547-19
- Jeong *et al.* (2019) "Chronicle of a Soil Bacterium: *Paenibacillus polymyxa* E681 as a Tiny Guardian of Plant and Human Health" *Front. Microbiol.* doi:10.3389/fmicb.2019.00467
- Kaminsky *et al.* (2019) "Genomic Analysis of γ-Hexachlorocyclohexane-Degrading *Sphingopyxis lindanitolerans* WS5A3p Strain in the Context of the Pangenome of *Sphingopyxis*" *Genes* doi:0.3390/genes10090688
- Khan *et al.* (2019) "Genomic and physiological analyses reveal that extremely thermophilic *Caldicellulosiruptor changbaiensis* deploys unique cellulose attachment mechanisms" *BioRxiv* doi:10.1101/622977
- Kirmiz *et al.* (2019) "Comparative genomics guides elucidation of vitamin B12 biosynthesis in novel human associated *Akkermansia*" *BioRxiv* doi:10.1101/587527
- Kiu *et al.* (2019) "Genomic analysis on broiler-associated *Clostridium perfringens* strains and exploratory caecal microbiome investigation reveals key factors linked to poultry necrotic enteritis" *Animal Microbiome* doi:10.1186/s42523-019-0015-1
- Kiu *et al.* (2019) "Phylogenomic analysis of gastroenteritis-associated *Clostridium perfringens* in England and Wales over a 7-year period indicates distribution of clonal toxigenic strains in multiple outbreaks and extensive involvement of enterotoxin-encoding (CPE) plasmids" *Micro. Genom.* doi:10.1099/mgen.0.000297
- Lozada *et al.* (2019) "Phage vB_BmeM-Goe8 infecting *Bacillus megaterium* DSM319" *Arch. Virol.* doi:10.1007/s00705-019-04513-5
- Kochetkova *et al.* (2019) "*Tepidiforma bonchosmolovskayae* gen. nov., sp. nov., a moderately thermophilic *Chloroflexi* bacterium from a Chukotka hot spring (Arctic, Russia), representing a novel class, *Tepidiformia*, which includes the previously uncultivated lineage OLB14" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.003902
- Kovaleva *et al.* (2019) "*Tautonia sociabilis* gen. nov., sp. nov., a novel thermotolerant planctomycete, isolated from a 4000 m deep subterranean habitat" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.003467
- Labuda *et al.* (2019) "Bloodstream Infections With a Novel Nontuberculous Mycobacterium Involving 52 Outpatient Oncology Clinic Patients—Arkansas, 2018" *Clin. Inf. Dis.* doi:10.1093/cid/ciz1120
- Lan *et al.* (2019) "*Vogesella urethralis sp. nov.*, isolated from human urine, and emended descriptions of *Vogesella perlucida* and *Vogesella mureinivorans*" *Int. J. Syst. Evol. Microbiol.* doi:10.1099/ijsem.0.003802
- Lawson *et al.* (2019) "Breast milk-derived human milk oligosaccharides promote *Bifidobacterium* interactions within a single ecosystem" *ISME J.* doi:0.1038/s41396-019-0553-2
- Ma *et al.* (2019) "First report of *Dickeya fangzhongdai* causing soft rot of onion in New York State" *Plant Dis.* doi:10.1094/PDIS-09-19-1940-PDN

- Matteo-Estrada *et al.* (2019) "Phylogenomics Reveals Clear Cases of Misclassification and Genus-Wide Phylogenetic Markers for *Acinetobacter*" *Genome Biol. Evol.* doi:10.1093/gbe/evz178
- McIntyre *et al.* (2019) "Single-molecule sequencing detection of N6-methyladenine in microbial reference materials" *Nat. Comm.* doi:10.1038/s41467-019-08289-9
- Nordmann *et al.* (2019) "Complete genome sequence of the virus isolate vB_BthM-Goe5 infecting *Bacillus thuringiensis*" *Arch. Virol.* doi:10.1007/s00705-019-04187-z
- Paim *et al.* (2019) "Evaluation of niche adaptation features by genome data mining approach of *Escherichia coli* urinary and gastrointestinal strains" *PeerJ Preprints* doi:10.7287/peerj.preprints.27720v1
- Park *et al* (2019) "Complete genome sequence of acetate-producing *Klebsiella pneumoniae* L5-2 isolated from infant feces" *3Biotech* doi:10.1007/s13205-019-1578-y
- Pedron & van Gijsegem (2019) "Diversity in the Bacterial Genus *Dickeya* Grouping Plant Pathogens and Waterways Isolates" *OBM Genetics* doi:10.21926/obm.genet.1904098
- Portier *et al.* (2019) "Elevation of *Pectobacterium carotovorum* subsp. *odoriferum* to species level as *Pectobacterium odoriferum* sp. nov., proposal of *Pectobacterium brasiliense* sp. nov. and *Pectobacterium actinidiae* sp. nov., emended description of *Pectobacterium carotovorum* and description of *Pectobacterium versatile* sp. nov., isolated from streams and symptoms on diverse plants" *Int. J Syst. Evol. Biol* doi:10.1099/ijsem.0.003611
- Potter *et al.* (2019) "In Silico Analysis of *Gardnerella* Genomospecies Detected in the Setting of Bacterial Vaginosis" *Clin. Chem.* doi:10.1373/clinchem.2019.305474
- Reichler *et al.* (2019) "A century of gray: A genomic locus found in 2 distinct *Pseudomonas* spp. is associated with historical and contemporary color defects in dairy products worldwide" *J. Dairy Sci.* doi:10.3168/jds.2018-16192
- Royo-Llonch *et al.* "Ecological and functional capabilities of an uncultured *Kordia* sp" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2019.126045
- Ruiz *et al.* (2019) "Microbiota of human precolostrum and its potential role as a source of bacteria to the infant mouth" *Sci. Rep.* doi:10.1038/s41598-019-42514-1
- Sant'Anna *et al.* (2019) "Genomic metrics made easy: what to do and where to go in the new era of bacterial taxonomy" *Crit. Rev. Microbiol.* doi:10.1080/1040841X.2019.1569587
- Schmuhl *et al.* (2019) "Comparative Transcriptomic Profiling of *Yersinia enterocolitica* O:3 and O:8 Reveals Major Expression Differences of Fitness- and Virulence-Relevant Genes Indicating Ecological Separation" *mSystems* doi:10.1128/mSystems.00239-18
- Spirina *et al.* (2019) "Draft Genome Sequence of Microbacterium sp. Gd 4-13, Isolated from Gydanskiy Peninsula Permafrost Sediments of Marine Origin" *Microb. Res. Announce.* doi:10.1128/MRA.00889-19
- Stefanic *et al.* (2019) "Intra-species DNA exchange: *Bacillus subtilis* prefers sex with less related strains" *BioRxiv* doi:10.1101/756569
- Stevens *et al.* (2019) "Whole-genome-based phylogeny of *Bacillus cytotoxicus* reveals different clades within the species and provides clues on ecology and evolution" *Sci. Rep.* doi:10.1038/s41598-018-36254-x
- Tanaka *et al.* (2019) "Draft Genome Sequences of *Enterococcus faecalis* Strains Isolated from Healthy Japanese Individuals" *Microb. Res. Announce.* doi:10.1128/MRA.00832-19
- Thorell *et al.* (2019) "Isolates from colonic spirochaetosis in humans show high genomic divergence and carry potential pathogenic features but are not detected by 16S amplicon sequencing using standard primers for the human microbiota" *BioRxiv* doi:doi.org/10.1101/544502
- Tian *et al.* (2019) "LINbase: A Web service for genome-based identification of microbes as members of crowd-sourced taxa" *BioRxiv* doi:10.1101/752212

- Tohno *et al.* (2019) "*Lactobacillus salitolerans* sp. nov., a novel lactic acid bacterium isolated from spent mushroom substrates" *Int. J Syst. Evol. Biol* doi:10.1099/ijsem.0.003224
- Vazquez-Campos *et al.* (2019) "Genomic insights into the Archaea inhabiting an Australian radioactive legacy site" *BioRxiv* doi:10.1101/728089
- Vincent *et al.* (2019) "A Mesophilic *Aeromona salmonicida* Strain Isolated from an Unsuspected Host, the Micratory Bird Pied Avocet" *Microorganisms* doi:10.3390/microorganisms7120592
- Vincent *et al.* (2019) "Investigation of the virulence and genomics of *Aeromonas salmonicida* strains isolated from human patients" *Inf. Genet. Evol.* doi:10.1016/j.meegid.2018.11.019
- Vincent *et al.* (2019) "Revisiting the taxonomy and evolution of pathogenicity of the genus *Leptospira* through the prism of genomics" *PLoS Neg. Trop. Dis.* doi:10.1371/journal.pntd.0007270
- Wallner *et al.* (2019) "Genomic analyses of *Burkholderia cenocepacia* reveal multiple species with differential host-adaptation to plants and humans" *BMC Genomics* doi:10.1186/s12864-019-6186-z
- Wang *et al.* (2019) "Occurrence of CTX-M-123-producing *Salmonella* Indiana in chicken carcasses: a new challenge for the poultry industry and food safety" *J. Antimicrob. Chemo.* doi:10.1093/jac/dkz386
- Webster *et al.* (2019) "Genome Sequences of Two Choline-Utilizing Methanogenic Archaea, *Methanococcoides* spp., Isolated from Marine Sediments" *Microbiol. Res. Ann.* doi:10.1128/MRA.00342-19
- Webster *et al.* (2019) "The Genome Sequences of Three *Paraburkholderia* sp. Strains Isolated from Wood-Decay Fungi Reveal Them as Novel Species with Antimicrobial Biosynthetic Potential" *Microbiol. Res. Ann.* doi:10.1128/MRA.00778-19
- Wiegand *et al.* (2019) "Cultivation and functional characterization of 79 planctomycetes uncovers their unique biology" *Nat. Microbiol.* doi:10.1038/s41564-019-0588-1
- Wittouck *et al.* (2019) " A genome-based species taxonomy of the *Lactobacillus* genus complex" *mSystems* doi:10.1128/mSystems.00264-19
- Yin *et al.* (2019) "A hybrid sub-lineage of *Listeria monocytogenes* comprising hypervirulent isolates" *Nat. Comm.* doi:10.1038/s41467-019-12072-1
- Yin *et al.* (2019) "Genetic Diversity of *Listeria monocytogenes* Isolates from Invasive Listeriosis in China" *Foodborne Path. Dis.* doi:10.1089/fpd.2019.2693
- Zabel *et al.* (2019) "Novel Genes and Metabolite Trends in *Bifidobacterium longum* subsp. *infantis* Bi-26 Metabolism of Human Milk Oligosaccharide 2-fucosyllactose" *Sci. Rep.* doi:10.1038/s41598-019-43780-9
- Zakham *et al.* (2019) "Molecular diagnosis and enrichment culture identified a septic pseudoarthrosis due to an infection with *Erysipelatoclostridium ramosum*" *Int. J. Inf. Dis.* doi:10.1016/j.ijid.2019.02.001
- Zhu *et al.* (2019) "First Report of Integrative Conjugative Elements in *Riemerella anatipestifer* Isolates From Ducks in China" *Front. Vet. Sci.* doi:10.3389/fvets.2019.00128
- Zhu *et al.* (2019) "Pan-genome analysis of *Riemerella anatipestifer* reveals its genomic diversity and acquired antibiotic resistance associated with genomic islands" *Func. Int. Genom* doi:10.1007/s10142-019-00715-x

2018

- Alex & Antunes (2018) "Genus-wide comparison of *Pseudovibrio* bacterial genomes reveal diverse adaptations to different marine invertebrate hosts" *PLoS One* doi:10.1371/journal.pone.0194368
- Beaton *et al.* (2018) "Community-led comparative genomic and phenotypic analysis of the aquaculture pathogen *Pseudomonas baetica* a390T sequenced by Ion semiconductor and Nanopore technologies" *FEMS Micro. Lett.* doi:10.1093/femsle/fny069

- Bogema *et al.* (2018) "Analysis of *Theileria orientalis* draft genome sequences reveals potential species-level divergence of the Ikeda, Chitose and Buffeli genotypes" *BMC Genomics* doi:10.1186/s12864-018-4701-2
- Brand *et al.* (2018) "Niche Differentiation among Three Closely Related *Competibacteraceae* Clades at a Full-Scale Activated Sludge Wastewater Treatment Plant and Putative Linkages to Process Performance" *App. Env. Micro.* doi:10.1128/AEM.02301-18
- Bridel *et al.* (2018) "Comparative Genomics of *Tenacibaculum dicentrarchi* and "*Tenacibaculum finnmarkense*" Highlights Intricate Evolution of Fish-Pathogenic Species" *Genome Biol. Evol.* doi:10.1093/gbe/evy020
- Carlos *et al.* (2018) "Substrate Shift Reveals Roles for Members of Bacterial Consortia in Degradation of Plant Cell Wall Polymers" *Front. Microbiol.* doi:10.3389/fmicb.2018.00364
- Covarrubias *et al.* (2018) "Occurrence, integrity and functionality of *Aca*ML1–like viruses infecting extreme acidophiles of the *Acidithiobacillus* species complex" *Res. Microbiol.* doi:10.1016/j.resmic.2018.07.005
- da Gama *et al.* (2018) "Taxonomic Repositioning of *Xanthomonas campestris* pv. *viticola* (Nayudu 1972) Dye 1978 as *Xanthomonas citri* pv. *viticola* (Nayudu 1972) Dye 1978 comb. nov. and Emendation of the Description of *Xanthomonas citri* pv. *anacardii* to Include Pigmented Isolates Pathogenic to Cashew Plant" *Phytopath.* doi:10.1094/PHYTO-02-18-0037-R
- Ferretti *et al.* (2018) "Mother-to-Infant Microbial Transmission from Different Body Sites Shapes the Developing Infant Gut Microbiome" *Cell Host Microbe* doi:10.1016/j.chom.2018.06.005
- Fontana *et al.* (2018) "Genetic Signatures of Dairy *Lactobacillus casei* Group" *Front. Microbiol.* doi:10.3389/fmicb.2018.02611
- Freschi *et al.* (2018) "The *Pseudomonas aeruginosa* Pan-Genome Provides New Insights on Its Population Structure, Horizontal Gene Transfer, and Pathogenicity" *Genome Biol. Evol.* doi:10.1093/gbe/evy259
- Gillis *et al.* (2018) "Role of plasmid plasticity and mobile genetic elements in the entomopathogen *Bacillus thuringiensis* serovar *israelensis*" *FEMS Micro. Rev.* doi:10.1093/femsre/fuy034
- Gragna-Miraglia *et al.* (2018) "Phylogenomics picks out the par excellence markers for species phylogeny in the genus *Staphylococcus*" *PeerJ* doi:10.7717/peerj.5839
- Hubbard *et al.* (2018) "Comparison of the first whole genome sequence of '*Haemophilus quentini*' with two new strains of '*Haemophilus quentini*' and other species of *Haemophilus*" *Genome* doi:10.1139/gen-2017-0195
- Issotta *et al.* (2018) "Insights into the biology of acidophilic members of the *Acidiferrobacteraceae* family derived from comparative genomic analyses" *Res. Microbiol.* doi:10.1016/j.resmic.2018.08.001
- Jangam *et al.* (2018) "Draft Genome Sequence of *Vibrio parahaemolyticus* Strain VP14, Isolated from a *Penaeus vannamei* Culture Farm" *Micro. Res. Ann.* doi:10.1128/genomeA.00149-18
- Jarett *et al.* (2018) "Single-cell genomics of co-sorted *Nanoarchaeota* suggests novel putative host associations and diversification of proteins involved in symbiosis" *Microbiome* doi:10.1186/s40168-018-0539-8
- Jung *et al.* (2018) "Complete genome sequence of *Bifidobacterium choerinum* FMB-1, a resistant starch-degrading bacterium" *J. Biotech.* doi:10.1016/j.jbiotec.2018.03.009
- Lazarte *et al.* (2018) "*Bacillus wiedmannii* biovar *thuringiensis*: A Specialized Mosquitocidal Pathogen with Plasmids from Diverse Origins" *Genome Biol. Evol.* doi:10.1093/gbe/evy211
- Li *et al.* (2018) "A Novel Strategy for Detecting Recent Horizontal Gene Transfer and Its Application to *Rhizobium* Strains" *Front. Microbiol.* doi:10.3389/fmicb.2018.00973
- Lima *et al.* "Genome sequencing and functional characterization of the non-pathogenic *Klebsiella pneumoniae* KpGe bacteria* *Microbes Inf.* doi:10.1016/j.micinf.2018.04.001
- McCann *et al.* (2018) "Viromes of one year old infants reveal the impact of birth mode on microbiome diversity" *PeerJ* doi:10.7717/peerj.4694

- Morales-Covarrubias (2018) "*Streptococcus penaeicida* sp. nov., isolated from a diseased farmed Pacific white shrimp (*Penaeus vannamei*)" *Int. J Syst. Evol. Biol* doi:10.1099/ijsem.0.002693
- Munoz-Villagran *et al.* (2018) "Comparative genomic analysis of a new tellurite-resistant *Psychrobacter* strain isolated from the Antarctic Peninsula" *PeerJ* doi:10.7717/peerj.4402
- Nascimento *et al.* (2018) "From plants to nematodes: *Serratia grimesii* BXF1 genome reveals an adaptation to the modulation of multi-species interactions" *Microb. Genom.* doi:10.1099/mgen.0.000178
- Orr *et al.* (2018) "De novo assembly of the *Pasteuria penetrans* genome reveals high plasticity, host dependency, and BclA-like collagens" *BioRxiv* doi:10.1101/485748
- Pinto *et al.* (2018) "Draft Genome Sequences of Novel *Pseudomonas*, *Flavobacterium*, and *Sediminibacterium* Strains from a Freshwater Ecosystem" *Micro. Res. Ann.* doi:10.1128/genomeA.00009-18
- Potter *et al.* (2018) "Population Structure, Antibiotic Resistance, and Uropathogenicity of *Klebsiella variicola*" *mBio* doi:10.1128/mBio.02481-18
- Potter *et al.* (2018) "*Superficieibacter electus* gen. nov., sp. nov., an Extended-Spectrum β-Lactamase Possessing Member of the Enterobacteriaceae Family, Isolated From Intensive Care Unit Surfaces" *Front. Microbiol.* doi:10.3389/fmicb.2018.01629
- Samad *et al.* (2017) "Comparative genome analysis of the vineyard weed endophyte *Pseudomonas viridiflava* CDRTc14 showing selective herbicidal activity" *Sci. Rep.* doi:10.1038/s41598-017-16495-y
- Sant'Anna *et al.* (2018) "Genome-based reclassification of *Paenibacillus dauci* as a later heterotypic synonym of *Paenibacillus shenyangensis*" *Int. J. Syst. Evol. Micro.* doi:10.1099/ijsem.0.003127
- Schilling *et al.* (2018) "Genomic Analysis of the Recent Viral Isolate vB_BthP-Goe4 Reveals Increased Diversity of ϕ29-Like Phages" *Viruses* doi:10.3390/v10110624
- Stevens *et al.* (2018) "Massive Diversity in Whole-Genome Sequences of *Streptococcus suis* Strains from Infected Pigs in Switzerland" *Microbiol. Res. Ann.* doi:10.1128/MRA.01656-18
- Tanizawa *et al.* (2018) "Lactobacillus paragasseri sp. nov., a sister taxon of Lactobacillus gasseri, based on whole-genome sequence analyses" *Int. J Syst. Evol. Biol* doi:10.1099/ijsem.0.003020
- Vincent & Charette (2018) "Completion of genome of *Aeromonas salmonicida* subsp. *salmonicida* 01-B526 reveals how sequencing technologies can influence sequence quality and result interpretations" *New Microb. New Inf.* doi:10.1016/j.nmni.2018.05.007
- Wilhelm (2018) "Following the terrestrial tracks of *Caulobacter* - redefining the ecology of a reputed aquatic oligotroph" *ISME J* doi:10.1038/s41396-018-0257-z
- Wittwer *et al.* (2018) "Population Genomics of *Francisella tularensis* subsp. *holarctica* and its Implication on the Eco-Epidemiology of Tularemia in Switzerland" *Front. Cell. Inf. Microbiol.* doi:10.3389/fcimb.2018.00089
- Zhang *et al.* (2018) "Draft Genome Sequence of *Komagataeibacter maltaceti* LMG 1529T, a Vinegar-Producing Acetic Acid Bacterium Isolated from Malt Vinegar Brewery Acetifiers" *Micro. Res. Ann.* doi:10.1128/genomeA.00330-18

2017

- Anderson *et al.* (2017) "Genomic variation in microbial populations inhabiting the marine subseafloor at deep-sea hydrothermal vents" *Nat. Comm.* doi:10.1038/s41467-017-01228-6
- Ding *et al.* (2017) "Loss of the ssrA genome island led to partial debromination in the PBDE respiring *Dehalococcoides mccartyi* strain GY50" *Env. Micro.* doi:10.1111/1462-2920.13817
- Edgington *et al.* (2017) "Genome Sequences of Chancellor, Mitti, and Wintermute, Three Subcluster K4 Phages Isolated Using *Mycobacterium smegmatis* mc^{2}155" *Microbiol. Res. Ann.* doi:10.1128/genomeA.01070-17

- Esposito *et al.* (2017) "Evolution of *Stenotrophomonas maltophilia* in Cystic Fibrosis Lung over Chronic Infection: A Genomic and Phenotypic Population Study" *Front. Microbiol.* doi:10.3389/fmicb.2017.01590
- Jeukens *et al.* (2017) "A Pan-Genomic Approach to Understand the Basis of Host Adaptation in *Achromobacter*" *Genome Biol. Evol.* doi:10.1093/gbe/evx061
- Ke *et al.* (2017) "Comparative genomics of *Vibrio campbellii* strains and core species of the *Vibrio Harveyi* clade" *Sci. Rep.* doi:10.1038/srep41394
- Kumar *et al.* (2017) "Draft Genome Sequence of the Luminescent Strain *Vibrio campbellii* LB102, Isolated from a Black Tiger Shrimp (*Penaeus monodon*) Broodstock Rearing System" *Micro. Res. Ann.* doi:10.1128/genomeA.00342-17
- Pelve *et al.* (2017) "Bacterial Succession on Sinking Particles in the Ocean's Interior" *Front. Microbiol.* doi:10.3389/fmicb.2017.02269
- Poehlein *et al.* (2017) "Microbial solvent formation revisited by comparative genome analysis" *Biotech. Biofuels* doi:10.1186/s13068-017-0742-z
- Ruiz-Valdeviezo *et al.* (2017) "Complete Genome Sequence of a Novel Nonnodulating *Rhizobium* Species Isolated from *Agave americana* L. Rhizosphere" *Micro. Res. Ann.* doi:10.1128/genomeA.01280-17
- Tada *et al.* (2017) "Revealing the genomic differences between two subgroups in *Lactobacillus gasseri*" *Biosci. Microb. Food Health* doi:10.12938/bmfh.17-006
- Tanizawa *et al.* (2017) "Genomic characterization reconfirms the taxonomic status of *Lactobacillus parakefiri*" *Biosci. Microb. Food Health* doi:10.12938/bmfh.16-026
- Tohno *et al.* (2017) "*Lactobacillus silagincola* sp. nov. and *Lactobacillus pentosiphilus* sp. nov., isolated from silage" *Int. J Syst. Evol. Biol* doi:10.1099/ijsem.0.002196
- Vincent *et al.* (2017) "Study of mesophilic *Aeromonas salmonicida* A527 strain sheds light on the species' lifestyles and taxonomic dilemma" *FEMS Micro. Lett.* doi:10.1093/femsle/fnx239
- Vollmers *et al.* (2017) "Untangling Genomes of Novel *Planctomycetal* and *Verrucomicrobial* Species from Monterey Bay Kelp Forest Metagenomes by Refined Binning" *Front. Microbiol.* doi:10.3389/fmicb.2017.00472
- Wang *et al.* (2017) "Genomic sequence of 'Candidatus *Liberibacter solanacearum*' haplotype C and its comparison with haplotype A and B genomes" *PLoS One* doi:10.1371/journal.pone.0171531

2016

- Burstein *et al.* (2016) "New CRISPR–Cas systems from uncultivated microbes" *Nature* doi:10.1038/nature21059
- Gupta *et al.* (2016) "Comparative genomic analysis of novel *Acinetobacter* symbionts: A combined systems biology and genomics approach" *Sci. Rep.* doi:10.1038/srep29043
- Haack *et al.* (2016) "Molecular Keys to the *Janthinobacterium* and *Duganella* spp. Interaction with the Plant Pathogen *Fusarium graminearum*" *Front. Microbiol.* doi:10.3389/fmicb.2016.01668
- Maeno *et al.* (2016) "Genomic characterization of a fructophilic bee symbiont *Lactobacillus kunkeei* reveals its niche-specific adaptation" *Syst. Appl. Microbiol.* doi:10.1016/j.syapm.2016.09.006
- Pritchard *et al.* (2016) "Genomics and taxonomy in diagnostics for food security: soft-rotting enterobacterial plant pathogens" *Anal. Methods* doi:10.1039/C5AY02550H
- Rodriguez-Rojas *et al.* (2016) "Draft Genome Sequence of a Multi-Metal Resistant Bacterium *Pseudomonas putida* ATH-43 Isolated from Greenwich Island, Antarctica" *Front. Microbiol.* doi:10.3389/fmicb.2016.01777
- Tanizawa *et al.* (2016) "DFAST and DAGA: web-based integrated genome annotation tools and resources" *Biosci. Microb. Food Health* doi:10.12938/bmfh.16-003

- Zheng *et al.* (2016) "Metabolism of Toxic Sugars by Strains of the Bee Gut Symbiont *Gilliamella apicola*" *mBio* doi:10.1128/mBio.01326-16

5.2 About `pyani`

`pyani` is a Python package and standalone program for calculation of whole-genome similarity measures. It is designed to be used with draft or complete prokaryote (bacterial and archaeal) genomes, and implements the following methods:

- ANIb (average nucleotide identity using `BLAST+`)
- ANIblastall (average nucleotide identity using `legacy BLAST`)
- ANIm (average nucleotide identity using `MUMmer`)
- fastANI (average nucleotide identity using `fastANI`)
- TETRA (4-mer sequence profiles)

5.2.1 Funding and Support

The development of `pyani` has been made possible by generous funding from a number of sources, including:

- **The Scottish Government**
  - 2011-2016 workpackages
  - 2016-2019 workpackage RD2.1.4

As is the case with much bioinformatics software, ongoing development does not receive continuous direct funding support. If you use `pyani`, please do cite this tool in your publications and outputs - this helps us justify future funding applications to maintain and improve the software.

You can find more information about how to cite *pyani* on the *Citing pyani* page.

5.3 QuickStart Guide

5.3.1 Installation

To use `pyani` you will need to install it on a local machine (your laptop, desktop, server or cluster). Installation is easiest using one of the two more popular Python package managers:

1. `conda`

`pyani` is available through the `bioconda` channel of Anaconda:

```
conda install -c bioconda pyani
```

2. `PyPI`

`pyani` is available *via* the PyPI package manager for Python:

```
pip install pyani
```

Tip: `pyani` can also be installed directly from source, or run as a `Docker` image. More detailed, and alternative, installation instructions can be found on the *Installation Guide* page.

5.3.2 `pyani` Walkthrough

The general procedure for any `pyani` analysis is:

1. Collect genomes for analysis (and index them)43
2. Create a database to hold genome data and analysis results
3. Perform ANI/TETRA/etc. analysis
4. Report and visualise analysis results
5. Generate species hypotheses (classify the input genomes) using the analysis results

To see options available for the `pyani` program, use the `-h` (help) option:

```
pyani -h
```

An example ANIm (ANI with `MUMmer`) analysis using `pyani` is provided as a walkthrough below. The sequence of commands used is:

```
pyani download --email my.email@my.domain -t 203804 C_blochmannia
pyani createdb
pyani anim C_blochmannia C_blochmannia_ANIm \
    --name "C. blochmannia run 1" \
    --labels C_blochmannia/labels.txt --classes C_blochmannia/classes.txt
pyani report --runs C_blochmannia_ANIm/ --formats html,excel,stdout
pyani report --run_results 1 --formats html,excel,stdout C_blochmannia_ANIm/
pyani report --run_matrices 1 --formats html,excel,stdout C_blochmannia_ANIm/
pyani plot --formats png,pdf --method seaborn C_blochmannia_ANIm 1
```

Tip: If you have the `pyani` source code, you can run the walkthrough commands by executing `make walkthrough` at the command-line, in the repository root. You can clean up the walkthrough output with `make clean_walkthrough`.

1. Collect genomes

It is possible to use genomes you have already placed into a local directory with *pyani*, but for this walkthrough a new set of genomes will be obtained from GenBank, using the `pyani download` command.

Tip: To read more about using local files with `pyani`, please see the *Indexing Genomes* documentation. To read more about downloading genomes from NCBI, please see the *Downloading Genomes from NCBI* documentation.

Attention: To use their online resources programmatically, NCBI require that you provide your email address for contact purposes if jobs go wrong, and for their own usage statistics. This should be specified with the `--email <EMAIL ADDRESS>` argument of `pyani download`.

Using the `pyani download` subcommand, we download all available genomes for *Candidatus Blochmannia* from NCBI. The taxon ID for this grouping is `203804`, and this ID is passed as the `-t` argument. The final (compulsory) argument is the path to the directory into which the genome data will be downloaded.

```
pyani download --email my.email@my.domain -t 203804 C_blochmannia
```

This creates a new directory (`C_blochmannia`) with the following contents:

```
$ tree C_blochmannia
C_blochmannia
├── GCF_000011745.1_ASM1174v1_genomic.fna
├── GCF_000011745.1_ASM1174v1_genomic.fna.gz
├── GCF_000011745.1_ASM1174v1_genomic.fna.md5
[...]
├── GCF_000973545.1_ASM97354v1_hashes.txt
├── classes.txt
└── labels.txt
```

Each downloaded genome is represented by four files: the genome sequence (FASTA: `*.fna`, compressed: `*.fna.gz`), an NCBI hashes file (`*_hashes.txt`) and an MD5 hash of the genome sequence file (`*.md5`).

Two additional files are created, summarising all genomes in the subdirectory:

- `classes.txt`: defines a *class* to which each input genome belongs. This is used for determining membership of groups for each genome, and annotating graphical output.
- `labels.txt`: provides text which will be used to label each input genome in the graphical output from `pyani`

2. Create database

`pyani` uses a local SQLite3 database to store genome data and analysis results. Existing databases can be re-used. For this walkthrough we create a new, empty database by executing the command:

```
pyani createdb
```

Tip: This creates the new database in a default location (`.pyani/pyanidb`), but the name and location of this database can be controlled with the `pyani createdb` command (see the *Creating a Local pyani Database* documentation). The path to the database can be specified in each of the subsequent commands, to enable maintenance and sharing of multiple analysis runs.

3. Conduct ANIm analysis

We run ANIm on the downloaded genomes by specifying first the directory containing the genome data (here, `C_blochmannia`) then the path to a directory which will contain the analysis results (`C_blochmannia_ANIm` for this walkthrough).

We also provide a name for the analysis (`--name`, for later human-readable reference), with optional files defining labels for each genome to be used when plotting output (`--labels`) and a set of classes to which each genome belongs (`--classes`) for downstream analysis:

```
pyani anim C_blochmannia C_blochmannia_ANIm \
    --name "C. blochmannia run 1" \
    --labels C_blochmannia/labels.txt --classes C_blochmannia/classes.txt
```

This command runs ANIm analysis on the genomes in the specified `C_blochmannia` directory. As we did not specifiy a database, the analysis results will be stored in the default database we created earlier (`.pyani/pyanidb`), where they will be identified by the name `C. blochmannia run 1`. The comparison result files will be written to the `C_blochmannia_ANIm` directory.

4. Reporting Analyses and Analysis Results

We can list all the runs contained in the (default) database, using the command:

```
pyani report --runs C_blochmannia_ANIm/ --formats html,excel,stdout
```

This will report the relevant information to new files in the `C_blochmannia_ANIm` directory.

```
$ tree -L 1 C_blochmannia_ANIm/
C_blochmannia_ANIm/
├── nucmer_output
├── runs.html
├── runs.tab
└── runs.xlsx
```

Tip: By default the `pyani report` command will create a tab-separated text file with the `.tab` suffix, but by using the `--formats` option, we have also created an HTML file, and an Excel file with the same data. The `stdout` option also prints the output table to the terminal window.

By inspecting the `runs.tab` file (or any of the other `runs.*` files) we see that our walkthrough analysis has run ID `1`. So we can use this ID to get tables of specific information for that run, such as:

the genomes that were analysed in all runs

```
pyani report --runs_genomes --formats html,excel,stdout C_blochmannia_ANIm/
```

the complete set of pairwise comparison results for a single run (listed by comparison)

```
pyani report --run_results 1 --formats html,excel,stdout C_blochmannia_ANIm/
```

comparison results as matrices (percentage identity and coverage, number of aligned bases and "similarity errors", and a Hadamard matrix of identity multiplied by coverage).

```
pyani report --run_matrices 1 --formats html,excel,stdout C_blochmannia_ANIm/
```

Attention: The `--run_results` and `--run_matrices` options take a single run ID or a comma-separated list of IDs (such as `1,3,4,5,9`) as an argument, and will produce output for each specified run ID.

Graphical output

Graphical output is obtained by executing the `pyani plot` subcommand, specifying the output directory and run ID. Optionally, output file formats and the graphics drawing method can be specified.

```
pyani plot --formats png,pdf --method seaborn C_blochmannia_ANIm 1
```

Supported output methods are:

- `seaborn`
- `mpl` (`matplotlib`)
- `plotly`

and each generates five plots corresponding to the matrices that `pyani report` produces:

- percentage identity of aligned regions
- percentage coverage of each genome by aligned regions
- number of aligned bases on each genome
- number of "similarity errors" on each genome
- a Hadamard matrix of percentage identity multiplied by percentage coverage for each comparison

Several graphics output formats are available, including `.png`, `.pdf` and `.svg`.

5.4 Requirements

The `pyani` package requires several other programs, packages and tools to run and develop. Many of these are automatically installed alongside `pyani`, but some packages and tools must be installed separately.

This page describes requirements for `pyani` and how/why they are used.

Tip: For more information about installation of specific packages, please see the *Installation Guide* page.

5.4.1 `Python3`

`pyani` is written in `Python`, and the modern version of `Python` is `Python3`. The legacy version of `Python` will not be maintained past 2020. `pyani` is written to use many features of `Python3` and will not run on `Python2`.

- Python3

5.4.2 `NCBI-BLAST+`

To carry out ANIb (average nucleotide identity using `BLAST`) analysis, genome sequences are compared using the `BLAST+` tool, provided by NCBI. The `BLAST+` tool is the current, maintained version, and is completely rewritten with respect to the legacy `BLAST` package (see below).

- NCBI-BLAST+

5.4.3 `MUMmer` v3.23

To carry out ANIm (average nucleotide identity using `MUMmer`) analysis, genome sequences are compared using the `nucmer` tool, part of the `MUMmer` package. Currently, `pyani` uses an older version of `MUMmer` for this analysis, pinned at version 3.23. `pyani` has not yet been tested with `MUMmer` 4.x.

- MUMmer3

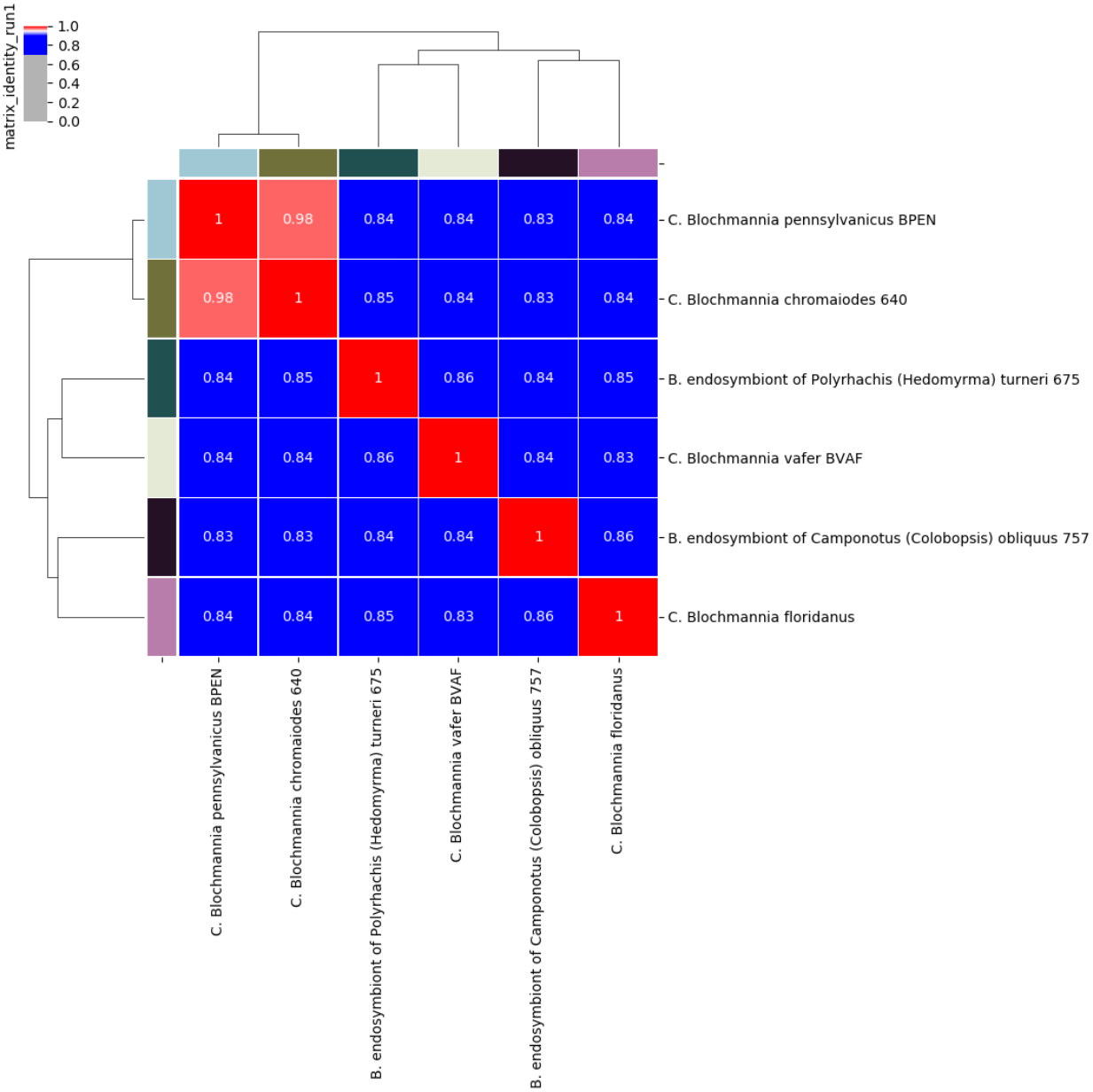

Fig. 1: Percentage identity matrix for *Candidatus Blochmannia* ANIm analysis

Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in the cell is the pairwise identity *of aligned regions*. The dendrograms are single-linkage clustering trees generated from the matrix of pairwise identity results. The default colour scheme colours cells with identity > 0.95 as red, and those with < 0.95 as blue. This division corresponds to a widely-used convention for bacterial species boundaries.

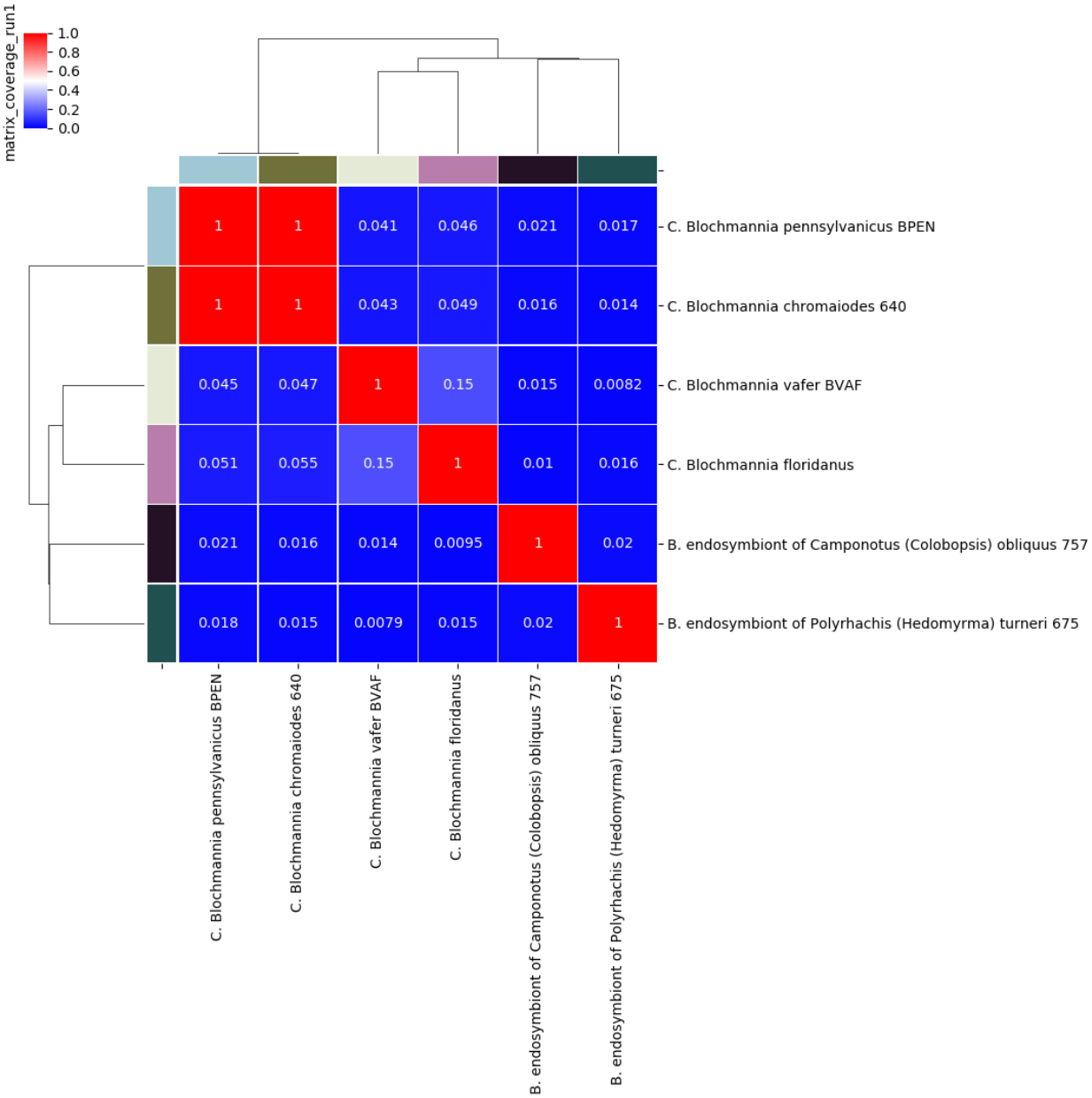

Fig. 2: Percentage coverage matrix for *Candidatus Blochmannia* ANIm analysis

Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in the cell is pairwise coverage of each genome by aligned regions in the comparison. The dendrograms are single-linkage clustering trees generated from the matrix of pairwise coverage results. The default colour scheme colours cells with identity > 0.50 as red, and those with < 0.50 as blue. This division corresponds to a strict majority of each genome in the comparison being alignable (a plausible minimum requirement for two sequences being considered "the same thing").

- MUMmer4

5.4.4 Legacy `NCBI-BLAST`

An alternative implementation of ANIb (average nucleotide identity using `BLAST`), included for compatibility checks with other ANI calculation software is provided in `pyani` through the legacy script `average_nucleotide_identity.py`. The use of the legacy `aniblastall` analysis is not recommended, and NCBI do not recommend use of the legacy `NCBI-BLAST` tool. However, the legacy software can still be downloaded and installed, for the curious and those who wish to test legacy compatibility.

- Legacy NCBI-BLAST (not supported)

5.4.5 `fastANI` v1.32

To carry out fastANI (average nucleotide identity using `fastANI`) analysis, genome sequences are compared using the `fastANI` tool.

- fastANI

5.4.6 SQLite3

The output generated by `pyani` analyses is stored in a local database, provided by `SQLite3`, for rapid querying and recovery. This allows for persistent storage of results without the need to keep the original alignment files, and for incremental addition of new analyses. `SQLite` is installed with `Python`

- SQLite

5.4.7 Open Grid Scheduler

When running on a cluster, `pyani` currently schedules jobs using the Sun Grid Engine/Open Grid Engine/Open Grid Scheduler syntax. Your cluster will require a compatible scheduler for `pyani` to distribute jobs appropriately:

- Open Grid Scheduler

5.4.8 Python Packages

`pyani` relies on functionality provided by a number of additional Python packages, and we gratefully acknowledge their contribution:

- Biopython: for working with biological data formats
- Matplotlib: for graphical output
- NetworkX: for graph calculations and representation
- Numpy: for matrix calculations
- OpenPyXL: for MicroSoft Excel output compatibility
- Pandas: for dataframe operations
- Pillow: for graphics manipulation and rendering
- SciPy: for scientific computing operations
- Seaborn: for graphical output

- SQLAlchemy: (pinned at v1.2.18 for compatibility reasons) for interaction with `SQLite3`
- tqdm: provides progress bars for user interaction

Development

We rely on a number of additional packages to aid `pyani` development, and if you set up a development environment as recommended in *Contributing to pyani*, then the following Python packages will be installed or expected to be present:

- bandit: to check for security issues in the codebase
- black: to enforce consistent, opinionated code formatting
- codecov: to generate code coverage output for the codecov.io service
- coverage: to generate code coverage output for local inspection
- doc8: to check docstring formatting syntax
- flake8: for code linting
- jinja2: for output/docfile templating
- pre-commit: for checking code style and quality prior to `git` commit
- pylint: for code linting
- pytest: to manage and run automated testing
- pytest-cov: to integrate `pytest` with `codecov` and `coverage`
- pytest-ordering: to ensure `pytest` test ordering
- sphinx: to generate documentation
- sphinx-rtd-theme: to provide local `ReadTheDocs` style formatting

5.5 Installation Guide

We support four ways to install and run `pyani` on your system:

1. Installation with Anaconda (i.e. the `conda` package manager) **[Recommended]**
2. Installation *via* `pip` (i.e. from PyPI)
3. Installation from source (i.e. download from GitHub)
4. Installation of a Docker image

Note: If you wish to contribute to development of `pyani`, you will require developer tools and packages that are not included in these installation instructions. To set up a local environment suitable for developing `pyani`, please refer to the *Contributing to pyani* page.

5.5.1 1. Installation with `Anaconda`

Anaconda <https://www.anaconda.com/> is a `Python` language distribution that includes the `conda` package manager. Several *channels*, themed collections of packages, are available throught he `conda` package manager. The latest release of `pyani` should always be available from the bioconda channel.

We recommend installation of `pyani` using the Anaconda3 or Miniconda3 distributions, and the `conda` package manager. This provides a straightforward way of managing third-party tool (e.g. `MUMmer` and `NCBI-BLAST`) installation, and allows for installation of `pyani` in a virtual environment, protected from other installations on the system.

1.1 Install `Anaconda` or `Miniconda`

If not already available on your system, install `Anaconda3` or `Miniconda3` on your system, following the instructions for your system on their respective websites:

- Anaconda3 download
- Miniconda3 download

After installation, you can check whether `conda` is available by issuing the following command in your terminal:

```
$ conda -V
conda 4.7.10
```

1.2 (optional) Create a new `conda` environment

Creation of a new `conda` environment is not necessary for installation of `pyani`, but it can be sometimes useful to specify the available version of `Python`, or to keep the versions of third party tools and Python packages separate from the system-level install. This may be particularly useful if, for example, the default system-level `Python` is `Python2`, as `Python3` is required for `pyani`.

The following commands will create and activate a new `conda` environment called `pyani`, using `Python 3.7`:

```
conda create --name pyani --yes python=3.7  # --yes accepts all suggestions
conda activate pyani
```

If successful, you should see the prefix `(pyani)` before your terminal prompt. To exit the current `conda` environment, issue the following command:

```
conda deactivate
```

- Managing conda environments

1.3 Add required `conda` channels

`pyani` and many of the packages it requires are provided in the `conda` *channel* called bioconda. *Channels* are locations where `conda` will look for packages, and they are typically grouped by topic area.

To install `pyani`, you will need to make available the `defaults`, `conda-forge` and `bioconda` channels, with the following commands:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

- Managing conda channels

1.4 Install `pyani`

The `bioconda` distribution of `pyani` will install all necessary packages and software required to run the software, including `NCBI-BLAST+` and `MUMmer`, with the following command:

```
conda install --yes pyani
```

When installation is complete, you can check for the availability of the `pyani` programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

Attention: If you wish to use the `ANIblastall` legacy ANIb method, then the legacy `NCBI-BLAST` tools need to be installed. These are not available through `conda` and must be installed manually for your system, as described below.

5.5.2 2. Installation with `pip`

PyPI is the Python Packaging Index, a repository of software for the `Python` language. Packages from `PyPI` can be installed using the `pip` package installer, which should come preinstalled with your system's `Python3`. The latest release of `pyani` should always be available from `PyPI`.

2.1 (optional) Create and activate a new virtual environment

As with the `conda` installation route above, it can be useful to separate the version of `Python` you use for `pyani`, and any installed packages and tools, from the system-level `Python` installation. There are multiple tools available to do this, and for convenience we list some below:

- Anaconda distribution
- pipenv
- pyenv
- virtualenv

2.2 Install third-party tools

Two third-party software tools are needed to perform ANIm and ANIb analysis:

- MUMmer3 for ANIm
- NCBI-BLAST+ for ANIb

These tools are not part of the PyPI distribution of pyani, and should be installed according to the instructions on their respective websites.

Tip: If you are using a conda environment for pyani, you can install both tools with a single command: `conda install --yes blast mummer`.

Attention: If you wish to use the ANIblastall legacy ANIb method, then the legacy NCBI-BLAST tools need to be installed. These are not available through conda and must be installed manually for your system, as described below.

2.3 Install pyani

The pyani programs, and their Python dependencies, can be installed with the command:

```
pip install pyani
```

When installation is complete, you can check for the availability of the pyani programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

5.5.3 3. Installation from source

The source code for the current pyani release can always be found on GitHub:

- Current pyani release source

3.1 (optional) Create and activate a new virtual environment

As with the conda installation route above, it can be useful to separate the version of Python you use for pyani, and any installed packages and tools, from the system-level Python installation. There are multiple tools available to do this, and for convenience we list some below:

- Anaconda distribution
- pipenv
- pyenv
- virtualenv

3.2 Install third-party tools

Two third-party software tools are needed to perform ANIm and ANIb analysis:

- MUMmer3 for ANIm

- NCBI-BLAST+ for ANIb

These tools are not part of the PyPI distribution of pyani, and should be installed according to the instructions on their respective websites.

Tip: If you are using a conda environment for pyani, you can install both tools with a single command: `conda install --yes blast mummer`.

Attention: If you wish to use the ANIblastall legacy ANIb method, then the legacy NCBI-BLAST tools need to be installed. These are not available through conda and must be installed manually for your system, as described below.

3.3 Download and extract the pyani source code

Click on one of the two links on the pyani releases page, or use a download tool such as curl or wget to download the link to a convenient location. For example:

```
wget https://github.com/widdowquinn/pyani/archive/v0.2.9.tar.gz
```

Then extract the source code archive. This will create a new directory called pyani-<CURRENT_VERSION> in your current location:

```
$ tar -zxf v0.2.9.tar.gz
$ ls
[...]
pyani-0.2.9/
[...]
```

3.4 Install pyani

Change directory in the terminal to the source code directory, e.g.:

```
$ cd pyani-0.2.9
$ ls
CHANGES.md                                  LICENSE                                    _
↪config.yml                                  requirements.txt
CITATIONS                                   MANIFEST.in                               bin/
↪                                          setup.cfg
CONTRIBUTORS.md                             Makefile                                  ↪
↪pyani/                                       setup.py
Dockerfile-average_nucleotide_identity      README-docker.md                          ↪
↪requirements-dev.txt                         test-requirements.txt
Dockerfile-genbank_get_genomes_by_taxon     README.md                                 ↪
↪requirements-pip.txt                         tests/
```

pyani can be installed using Python3's setup tools, using the command:

```
python setup.py install
```

This will download and install the Python packages that pyani needs to run. When installation is complete, you can check for the availability of the pyani programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

5.5.4 4. Installation with `Docker`

`Docker` is a platform for running applications that uses *containerisation* to share virtual machines that come pre-installed with all required tools and dependencies. The latest `pyani` programs should always be available as separate `Docker` containers from `DockerHub`:

- average_nucleotide_identity.py
- genbank_get_genomes_by_taxon.py

In order to use the containerised versions of `pyani`, you must have `Docker` installed and working on your system. To do so, please follow the instructions at the `Docker` website:

- Docker website

4.1 Running `pyani` with `Docker`

To pull (if necessary) and run the `pyani` programs in a `Docker` image on your local system, use the following commands (with the `Docker` daemon running):

```
docker run -v ${PWD}:/host_dir leightonpritchard/average_nucleotide_identity:v
↪<REQUIRED_VERSION>
docker run -v ${PWD}:/host_dir leightonpritchard/genbank_get_genomes_by_taxon:v
↪<REQUIRED_VERSION>
docker run -v ${PWD}:/host_dir leightonpritchard/pyani:v<REQUIRED_VERSION>
```

Tip: If no tag is specified, then `Docker` will attempt to use the `:latest` tag, which may not exist.

Note: The `-v ${PWD}:/host_dir` links the `Docker` image to the current working directory, and enables the `pyani` programs to see files below the current execution point in your filesystem. This is necessary for the analysis to proceed.

5.6 Basic Use

5.6.1 Downloading Genomes from NCBI

This page describes some typical use cases for downloading genomes from NCBI using the `pyani download` subcommand. This command downloads all assembled genomes found beneath a given taxon identifier, at the NCBI GenBank database. You will need to know the NCBI taxon ID for each taxon you wish to download.

Attention: To use their online resources programmatically, NCBI require that you provide your email address for contact purposes if jobs go wrong, and for their own usage statistics. This should be specified with the `--email <EMAIL ADDRESS>` argument of `pyani download`.

For more information about the `pyani download` subcommand, please see the *pyani download* page, or issue the command `pyani download -h` to see the inline help.

Download all genomes in a single taxon

The basic form of the command is:

```
pyani download --email my.email@my.domain -t <TAXON_ID> -o <OUTPUT_DIRECTORY>
```

This instructs `pyani` to use the `download` subcommand to obtain all available genome assemblies below the taxon ID `<TAXON_ID>`, passed with the `-t` argument, and place the downloaded files - along with label and class information files created by `pyani` in the subdirectory `<OUTPUT_DIRECTORY>`, passed with the `-o` argument.

For example, if we wished to download all available assemblies for the bacterium *Pseudomonas flexibilis* we would identify the taxon ID to be 706570, and use this as the argument to `-t`, placing the output in a convenient subdirectory (e.g. `genomes`, the argument to `-o`), with the command:

```
$ pyani download --email my.email@my.domain -t 706570 -o genomes
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:00, 3224293.90it/s]
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪110097041.36it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:00, 3125724.89it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪55139621.76it/s]
GCA_001312105.1_ASM131210v1_genomic.fna.gz: 1048576it [00:00, 1519534.52it/s]
GCA_001312105.1_ASM131210v1_hashes.txt: 1048576it [00:00, 272426072.29it/s]
GCF_000806415.1_ASM80641v1_genomic.fna.gz: 2097152it [00:00, 3097667.41it/s]
GCF_000806415.1_ASM80641v1_hashes.txt: 1048576it [00:00, 129632638.05it/s]
GCF_000802425.1_ASM80242v1_genomic.fna.gz: 2097152it [00:00, 2973018.02it/s]
GCF_000802425.1_ASM80242v1_hashes.txt: 1048576it [00:00, 233590743.10it/s]
```

This displays each assembly as a download is attempted, and places all output in the named subdirectory:

```
$ tree genomes/
genomes/
├── GCA_001312105.1_ASM131210v1_genomic.fna
├── GCA_001312105.1_ASM131210v1_genomic.fna.gz
├── GCA_001312105.1_ASM131210v1_genomic.fna.md5
├── GCA_001312105.1_ASM131210v1_hashes.txt
├── GCF_000802425.1_ASM80242v1_genomic.fna
├── GCF_000802425.1_ASM80242v1_genomic.fna.gz
├── GCF_000802425.1_ASM80242v1_genomic.fna.md5
├── GCF_000802425.1_ASM80242v1_hashes.txt
├── GCF_000806415.1_ASM80641v1_genomic.fna
├── GCF_000806415.1_ASM80641v1_genomic.fna.gz
├── GCF_000806415.1_ASM80641v1_genomic.fna.md5
├── GCF_000806415.1_ASM80641v1_hashes.txt
├── GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna
├── GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz
```

(continues on next page)

(continued from previous page)

```
├── GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.md5
├── GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt
├── GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna
├── GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz
├── GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.md5
├── GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt
├── classes.txt
└── labels.txt
```

Each genome is downloaded in compressed format (`.fna.gz` files) and expanded in-place to give the FASTA file (`.fna`) files. The MD5 hash of each FASTA file is also calculated (`.md5`). This will be used by `pyani` to uniquely identify that assembly throughout the analysis process.

```
$ head genomes/GCA_001312105.1_ASM131210v1_genomic.fna.md5
e55cd3d913a198ac60afd8d509c02ab4    genomes/GCA_001312105.1_ASM131210v1_genomic.fna
```

`pyani` also creates two files:

- `classes.txt`: each genome is assigned a *class* which is used to annotate genomes in the graphical output. `pyani` attempts to infer genus and species as the default class
- `labels.txt`: each genome is assigned a text label, which is used to label genomes in the graphical output. `pyani` attempts to infer genus, species, and strain ID as the default label

```
$ head genomes/classes.txt
1ac4941c569f32f044eba0a8540d4704    GCF_900155995.1_IMG-taxon_2681812811_annotated_
↪assembly_genomic Pseudomonas flexibilis
8664341798070f1d70b2569a5b3a2320    GCF_900101515.1_IMG-taxon_2596583557_annotated_
↪assembly_genomic Pseudomonas flexibilis
e55cd3d913a198ac60afd8d509c02ab4    GCA_001312105.1_ASM131210v1_genomic     ␣
↪Pseudomonas flexibilis
9b9719eb78bf7cf6dd0146a3f9426f60    GCF_000806415.1_ASM80641v1_genomic      ␣
↪Pseudomonas flexibilis
2bdfffd867d843f970e4dfd388d5332a    GCF_000802425.1_ASM80242v1_genomic      ␣
↪Pseudomonas flexibilis
$ head genomes/labels.txt
1ac4941c569f32f044eba0a8540d4704    GCF_900155995.1_IMG-taxon_2681812811_annotated_
↪assembly_genomic P. flexibilis ATCC 29606
8664341798070f1d70b2569a5b3a2320    GCF_900101515.1_IMG-taxon_2596583557_annotated_
↪assembly_genomic P. flexibilis CGMCC 1.1365
e55cd3d913a198ac60afd8d509c02ab4    GCA_001312105.1_ASM131210v1_genomic     P.␣
↪flexibilis JCM 14085
9b9719eb78bf7cf6dd0146a3f9426f60    GCF_000806415.1_ASM80641v1_genomic      P.␣
↪flexibilis JCM 14085
2bdfffd867d843f970e4dfd388d5332a    GCF_000802425.1_ASM80242v1_genomic      P.␣
↪flexibilis ATCC 29606
```

These files are used to associate labels and classes to the genome files in the `pyani` database, specific to the analysis run.

Both `classes.txt` and `labels.txt` can be edited to suit the user's classification and labelling scheme.

Download all genomes from multiple taxa

To download genomes from more than one taxon, you can provide a comma-separated list of taxon IDs to the `pyani download` subcommand, e.g.:

```
pyani download --email my.email@my.domain -t <TAXON_ID1>,<TAXON_ID2>,... -o <OUTPUT_
↪DIRECTORY>
```

The following command can be used to download assemblies from three different *Pseudomonas* taxa (*P. flexibilis*: 706570, *P. mosselli*: 78327, and *P. fulva*: 47880):

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 -o multi_taxa
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:00, 3081776.59it/s]
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪63489526.95it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:00, 3194885.99it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪77838775.82it/s]
```

Dry-run test (identify, but do not download, files)

If you only want to see which genomes will be downloaded from NCBI with a given `pyani download` subcommand, but not download them, then you can use the `--dry-run` option. For example:

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 -o multi_taxa --dry-
↪run
WARNING: Dry run only: will not overwrite or download
WARNING: (dry-run) skipping download of GCF_900155995.1
WARNING: (dry-run) skipping download of GCF_900101515.1
WARNING: (dry-run) skipping download of GCA_001312105.1
WARNING: (dry-run) skipping download of GCF_000806415.1
[...]
```

Download genomes for compilation of a custom `Kraken` database

Kraken is a bioinformatics tool that assigns taxonomic identities to short DNA sequences, such as Illumina or Nanopore reads. Several wide-ranging `Kraken` databases are available for download, and around the community, but it can sometimes be useful to construct a custom local `Kraken` database specific for your organism or taxon of interest (e.g. for filtering out contaminating or suspect reads).

The `pyani download` command can prepare downloaded genome files for immediate use with the `Kraken` database-building tools, by specifying the `--kraken` option:

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 -o genomes_kraken --
↪kraken
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:00, 3085741.03it/s]
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪140958511.30it/s]
WARNING: Modifying downloaded sequence for Kraken compatibility
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz: 2097152it␣
↪[00:01, 1902023.68it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt: 1048576it [00:00,␣
↪31428985.47it/s]
WARNING: Modifying downloaded sequence for Kraken compatibility
[...]
$ head multi_taxa/GCA_001312105.1_ASM131210v1_genomic.fna
```

(continues on next page)

(continued from previous page)

```
>BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001,␣
↪whole genome shotgun sequence
ACCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGGTAGCGGCCGAGCGGCGCGTC
GCCGCGCTCGCCCGCCAGCTCCTCGGCCATCTGCGTGTAGCCGAGCATGCTGGTCAGCAGGTTGTTGAAGTCGTGGGCAA
TGCCGCCGGTCAGGTGGCCGATGGCTTCCATGCGCTGCGCCTGGCGCAGCTGCTGTTCCAGCGCCGCCCGCTCCACCTCG
$ head genomes_kraken/GCA_001312105.1_ASM131210v1_genomic.fna
>BBCY01000001.1|kraken:taxid|706570 BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085␣
↪DNA, contig: JCM14085.contig00001, whole genome shotgun sequence
ACCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGG
TAGCGGCCGAGCGGCGCGTCGCCGCGCTCGCCCGCCAGCTCCTCGGCCATCTGCGTGTAG
CCGAGCATGCTGGTCAGCAGGTTGTTGAAGTCGTGGGCAATGCCGCCGGTCAGGTGGCCG
ATGGCTTCCATGCGCTGCGCCTGGCGCAGCTGCTGTTCCAGCGCCGCCCGCTCCACCTCG
```

Using this option does affects downstream performance or use of `pyani` only in that the two different input files for the same genome will have distinct hashes:

```
$ head multi_taxa/GCA_001312105.1_ASM131210v1_genomic.fna.md5
e55cd3d913a198ac60afd8d509c02ab4    multi_taxa/GCA_001312105.1_ASM131210v1_genomic.fna
$ head genomes_kraken/GCA_001312105.1_ASM131210v1_genomic.fna.md5
053fd98d8c9ab30de46f56fd601ef529    genomes_kraken/GCA_001312105.1_ASM131210v1_
↪genomic.fna
```

and so will not be considered to be the "same sequence" when repeating comparisons.

5.6.2 Indexing Genomes

Indexing genomes is a necessary step in `pyani` to prepare input genomes for analysis.

Attention: If you use the `pyani download` subcommand (see *Downloading Genomes from NCBI*) to obtain genomes for analysis, then indexing is carried out automatically. However, if you collect a local set of genomes (e.g. from your own sequencing project), then you will need to index the genomes with the `pyani index` subcommand.

For more information about the `pyani index` subcommand, please see the *pyani index* page, or issue the command `pyani index -h` to see the inline help.

What does indexing do?

In the context of `pyani`, *indexing* refers to generating an index code that is unique to each input genome FASTA file in the input directory. The index code is the MD5 hash for the FASTA file.

This MD5 index code is used to identify each specific input genome sequence (and associated metadata) so that duplicate comparisons can be readily identified, and previous results reused from the `pyani` database, if they are available.

Indexing also generates two files (see *Downloading Genomes from NCBI*):

- `classes.txt`: each genome is assigned a *class* which is used to annotate genomes in the graphical output. `pyani` attempts to infer genus and species as the default class
- `labels.txt`: each genome is assigned a text label, which is used to label genomes in the graphical output. `pyani` attempts to infer genus, species, and strain ID as the default label

These files are used to associate labels and classes to the genome files in the `pyani` database, specific to the analysis run. Both `classes.txt` and `labels.txt` can be edited to suit the user's classification and labelling scheme.

Index a directory of FASTA files

The basic form of the command is:

```
pyani index -i <GENOME_DIRECTORY>
```

This instructs `pyani` to search `<GENOME_DIRECTORY>` for files with a standard FASTA suffix (`.fna`, `.fasta`, `.fa`, `.fas`, `.fsa_nt`). For each file found, it calculates the MD5 hash and writes it to an accompanying file with extension `.md5`. The hash is then associated with a genome label and a genome class, written to the two files `labels.txt` and `classes.txt` (see above).

For example, if we have a directory called `unindexed` that contains some FASTA format genome sequence files:

```
$ tree unindexed
unindexed
├── GCA_001312105.1_ASM131210v1_genomic.fna
├── GCF_000834555.1_ASM83455v1_genomic.fna
└── GCF_005796105.1_ASM579610v1_genomic.fna
```

We could run the `pyani index` command:

```
$ pyani index -i unindexed/
$ tree unindexed
unindexed
├── GCA_001312105.1_ASM131210v1_genomic.fna
├── GCA_001312105.1_ASM131210v1_genomic.fna.md5
├── GCF_000834555.1_ASM83455v1_genomic.fna
├── GCF_000834555.1_ASM83455v1_genomic.fna.md5
├── GCF_005796105.1_ASM579610v1_genomic.fna
├── GCF_005796105.1_ASM579610v1_genomic.fna.md5
├── classes.txt
└── labels.txt
```

This creates an `.md5` file for each genome, and corresponding `classes.txt` and `labels.txt` files:

```
$ head unindexed/GCA_001312105.1_ASM131210v1_genomic.fna
>BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001,␣
↪whole genome shotgun sequence
ACCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGGTAGCGGCCGAGCGGCGCGTC
GCCGCGCTCGCCCGCCAGCTCCTCGGCCATCTGCGTGTAGCCGAGCATGCTGGTCAGCAGGTTGTTGAAGTCGTGGGCAA
$ head unindexed/GCA_001312105.1_ASM131210v1_genomic.fna.md5
e55cd3d913a198ac60afd8d509c02ab4    unindexed/GCA_001312105.1_ASM131210v1_genomic.fna
$ head unindexed/classes.txt
527f35b3eb9dd371d8d5309b6043dd9f    GCF_000834555.1_ASM83455v1_genomic      ␣
↪Pseudomonas fulva strain MEJ086 contig_1, whole genome shotgun sequence
b00c5b1f636b8083b68b128e7ee28a40    GCF_005796105.1_ASM579610v1_genomic     ␣
↪Pseudomonas mosselii strain SC006 Scaffold1, whole genome shotgun sequence
e55cd3d913a198ac60afd8d509c02ab4    GCA_001312105.1_ASM131210v1_genomic     ␣
↪Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome␣
↪shotgun sequence
$ head unindexed/labels.txt
527f35b3eb9dd371d8d5309b6043dd9f    GCF_000834555.1_ASM83455v1_genomic      ␣
↪Pseudomonas fulva strain MEJ086 contig_1, whole genome shotgun sequence
```

(continues on next page)

(continued from previous page)

```
b00c5b1f636b8083b68b128e7ee28a40    GCF_005796105.1_ASM579610v1_genomic
↪Pseudomonas mosselii strain SC006 Scaffold1, whole genome shotgun sequence
e55cd3d913a198ac60afd8d509c02ab4    GCA_001312105.1_ASM131210v1_genomic
↪Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome
↪shotgun sequence
```

Tip: The class and label information produced by `pyani index` is different to that generated with `pyani download`. Genus, species and strain identifiers can reliably be obtained from NCBI metadata when downloading genomes, but with user-provided sequences the information may not be encoded in the sequence description line in a standard manner.

As a result, when using `pyani index` it is often useful to edit the `classes.txt` and `labels.txt` directly, or generate these files in some other way.

5.6.3 Creating a Local `pyani` Database

`pyani` stores genome information and analysis results in a persistent local SQLite3 database. This allows for reuse of previous comparisons and reanalysis of datasets without having to rerun the analysis. It also means that the genome comparison results don't have to be stored in full on disk, saving space.

Note: To conduct a `pyani` analysis, there needs to be an existing database in-place.

To create a new, empty database you can use the `pyani createdb` command.

For more information about the `pyani createdb` subcommand, please see the *pyani createdb* page, or issue the command `pyani download -h` to see the inline help.

Create a new empty `pyani` database

The basic form of the command is:

```
pyani createdb
```

This instructs `pyani` to create a new, empty database for analysis at the default location.

Note: The default location for the `pyani` database is in a hidden directory: `.pyani/pyanidb`. All other `pyani` subcommands will look in this location for the database, unless told otherwise using the `--dbpath` option.

For example:

```
$ls .pyani
ls: .pyani: No such file or directory
$ pyani createdb
$ ls .pyani
pyanidb
```

Create an empty `pyani` database at a specific location

Tip: If you use `pyani` for a number of distinct taxa, it can be convenient to create a new database for each project, to avoid performance issues as the database grows in size, filled by data that does not contribute to the analysis.

The following command can be used to specify the location of the newly-created `pyani` database:

```
pyani createdb --dbpath <PATH_TO_DATABASE>
```

where `<PATH_TO_DATABASE>` is the intended location of the database. For instance, to create a new database specific for an analysis we'll call `multitaxa`, we could use the command:

```
$ ls .pyani
pyanidb
$ pyani createdb --dbpath .pyani/multitaxadb
$ ls .pyani
multitaxadb pyanidb
```

The new database can then be specified in other `pyani` subcommands, using the `--dbpath` option.

5.6.4 Running ANIm analysis

`pyani` implements average nucleotide identity analysis using MUMmer3 (*ANIm*) as defined in Richter & Rosselló-Móra (2009) (doi:10.1073/pnas.0906412106). To run ANIm on a set of input genomes, use the `pyani anim` subcommand.

In brief, the analysis proceeds as follows for a set of input prokaryotic genomes:

1. MUMmer3 is used to perform pairwise comparisons between each possible pair of input genomes, to identify homologous (alignable) regions.
2. For each comparison, the alignment output is parsed, and the following values are calculated:
    - total number of aligned bases on each genome
    - fraction of each genome that is aligned (the *coverage*)
    - the proportion of all aligned regions that is identical in each genome (the *ANI*)
    - the number of unaligned or non-identical bases (the *similarity errors*)
    - the product of *coverage* and *ANI*

The output values are recorded in the `pyani` database.

Note: A single `MUMmer` comparison is performed between each pair of genomes. Input genomes are sorted into alphabetical order by filename, and the query sequence is the genome that occurs earliest in the list; the subject sequence is the genome that occurs latest in the list.

Tip: The `MUMmer` comparisons are embarrasingly parallel, and can be distributed across cores on an Open Grid Scheduler-compatible cluster, using the `--scheduler SGE` option.

Attention: `pyani anim` requires that a working copy of *MUMmer3* is available. Please see *Installation Guide* for information about installing this package.

For more information about the `pyani anim` subcommand, please see the *pyani anim* page, or issue the command `pyani anim -h` to see the inline help.

Perform ANIm analysis

The basic form of the command is:

```
pyani anim -i <INPUT_DIRECTORY> -o <OUTPUT_DIRECTORY>
```

This instructs `pyani` to perform ANIm on the genome FASTA files in `<INPUT_DIRECTORY>`, which is passed to the `-i` argument, and write any output files to `<OUTPUT_DIRECTORY>`, which is passed to the `-o` argument. For example, the following command performs ANIm on genomes in the directory `genomes` and writes output to a new directory `genomes_ANIm`:

```
pyani anim -i genomes -o genomes_ANIm
```

Note: While running, `pyani anim` will show progress bars unless these are disabled with the option `--disable_tqdm`

This command will write the intermediate `nucmer`/`MUMmer` output to the directory `genomes_ANIm`, in a subdirectory called `nucmer_output`, where the results can be inspected if required.

```
$ ls genomes_ANIm/
nucmer_output
```

Attention: To view the output ANIm results, you will need to use the `pyani report` or `pyani plot` subcommands. Please see *pyani report* and *pyani plot* for more details.

Perform ANIm analysis with Open Grid Scheduler

The `MUMmer` comparison step of ANIm is embarrassingly parallel, and `nucmer` jobs can be distributed across cores in a cluster using the *Open Grid Scheduler*. To enable this during the analysis, use the `--scheduler SGE` option:

```
pyani anim --scheduler SGE -i genomes -o genomes_ANIm
```

Note: Jobs are submitted as *array jobs* to keep the scheduler queue short.

Note: If `--scheduler SGE` is not specified, all `MUMmer` jobs are run locally with `Python`'s `multiprocessing` module.

Controlling parameters of Open Grid Scheduler

It is possible to control the following features of Open Grid Scheduler *via* the `pyani anim` subcommand:

- The array job size (by default, comparison jobs are batched in arrays of 10,000)
- The prefix string for the job, as reported in the scheduler queue
- Arguments to the `qsub` job submission command

These allow for useful control of job execution. For example, the command:

```
pyani anim --scheduler SGE --SGEgroupsize 5000 -i genomes -o genomes_ANIm
```

will batch `MUMmer` jobs in groups of 500 for the scheduler. The command:

```
pyani anim --scheduler SGE --jobprefix My_Ace_Job -i genomes -o genomes_ANIm
```

will prepend the string `My_Ace_Job` to your job in the scheduler queue. And the command:

```
pyani anim --scheduler SGE --SGEargs "-m e -M my.name@my.domain" 5000 -i genomes -o␣
↪genomes_ANIm
```

will email `my.name@my.domain` when the jobs finish.

References

- Richter & Rosselló-Móra (2009) Proc Natl Acad Sci USA 106: 19126-19131 doi:10.1073/pnas.0906412106.

5.6.5 Interpreting the Graphical Output

The Output

`pyani plot` generates five heatmaps corresponding to the matrices that `pyani report` produces:

- percentage identity across all aligned regions (*Average Nucleotide Identity*, ANI)
- percentage coverage of each genome by aligned regions (*Coverage*, or *Aligned Fraction* (AF))
- number of bases from each genome contributing to the aligned regions
- number of "similarity errors" on each genome
- a Hadamard matrix of percentage identity multiplied by percentage coverage for each comparison

For each heatmap, a pair of plots describing the distributions of values in the heatmap/matrix are also generated. These show a histogram (left) and a KDE with rugplot (right) of the values in the heatmap.

In addition, a scatterplot of ANI vs Coverage/AF for each pairwise comparison is produced.

Heatmaps

Percentage Identity/ANI

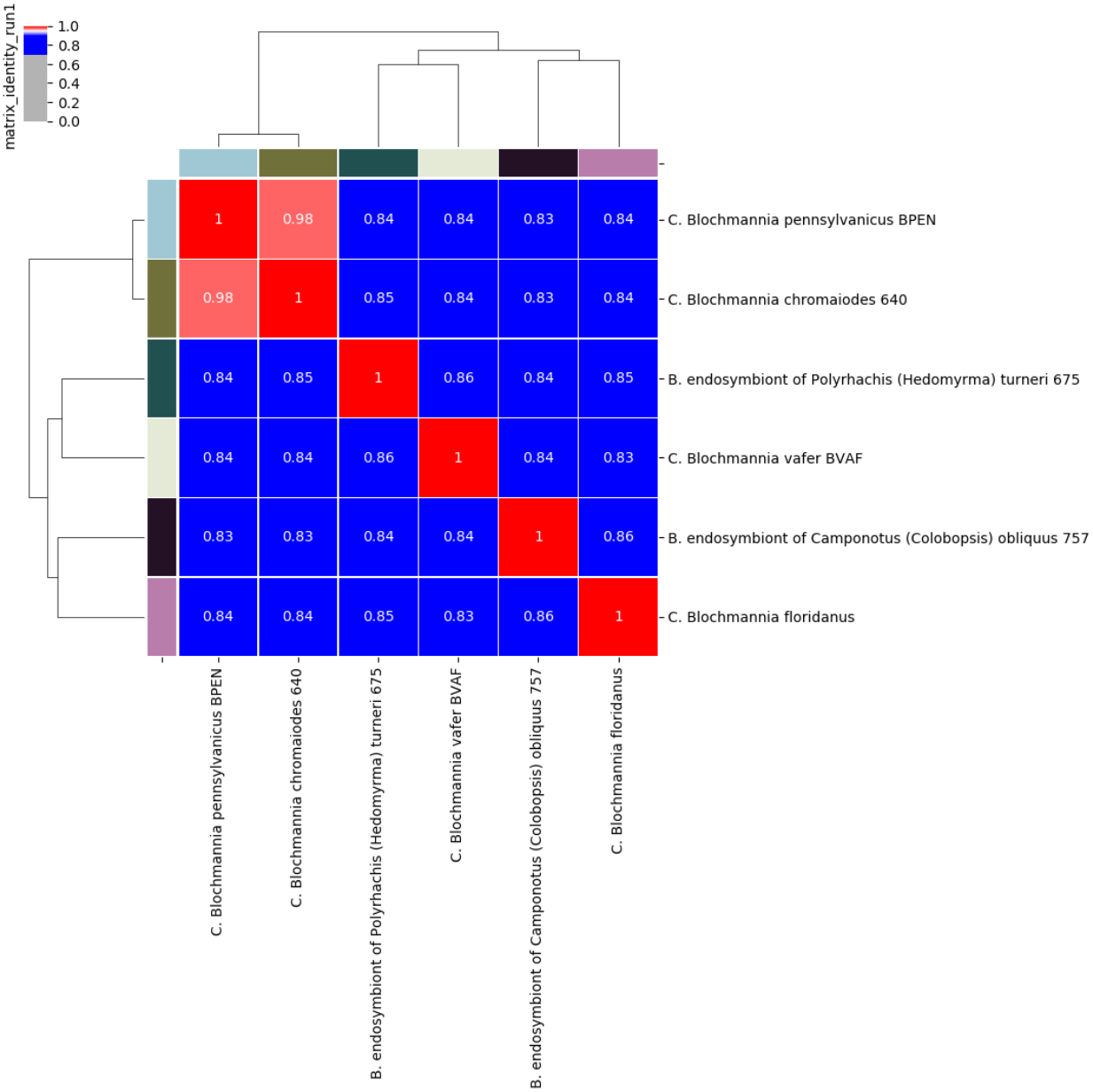

Fig. 3: Percentage identity matrix for *Candidatus Blochmannia* ANIm analysis

Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in each cell is the pairwise identity *of all aligned regions*. The dendrograms are produced by single-linkage hierarchical clustering trees from the matrix of pairwise identity results. The default colour scheme colours cells with identity > 0.95 as red, and those with < 0.95 as blue. This division corresponds to a widely-used convention for bacterial species boundaries.

Note: No single ANI threshold should be considered universally applicable to distinguish between species for all bacterial genomes.

We can often take the red blocks on the main diagonal of the heatmap to indicate groups of genomes that are coherent with each other and exclude all other genomes in the analysis, which is one of the criteria we would use to delineate a biological species or other taxon. As a rule of thumb, red squares on the main diagonal are a good approximation to species.

Taking the 95% threshold between red and blue cells to be equivalent to a species boundary, an interpretation of this figure would be that:

- the two genomes BPEN and 640 could be classified as the same species
- the remaining four genomes each represent a distinct species

In particular, we can see that the off-diagonal identity values are all around 85%, consistent with the limit of detection for homologous nucleotide regions.

Note: ANI reports the average percentage identity for the *aligned regions* only. If the total aligned proportion of either genome is not large, then ANI is not a reliable measure of overall genome similarity, and interpretation of percentage identity thresholds as species boundaries becomes less reliable. Percentage identity should always be considered in conjunction with coverage/aligned fraction.

Coverage/Aligned Fraction

Note: There is no widely-accepted convention for interpreting coverage/aligned fraction in its own right. Coverage should always be considered in conjunction with percentage identity and other measures.

The default graphical representation of coverage/AF distinguishes between alignments that cover more than 50% of the query genome (red) from those that cover less than 50% (blue). This is an *ad hoc* boundary with no particular biological meaning, but which does have a useful property that guides interpretation of the output.

If two genomes A and B align over less than 50% of genome A, then it is possible that the *majority* of genome A aligns to a different genome than genome B, with which genome B shares no homology. For instance, if the alignment of A and B has 30% coverage of genome A, it is possible that 70% of genome A is identical to another genome - C - which shares no sequence at all with genome B. In this case, it is perhaps not reasonable to assert that genomes A and B "are the same thing" or, in technical terms, *belong to the same taxon*.

However, if two genomes A and B align over more than 50% of their genomes - a majority of each genome - it is possibly reasonable to assert that the two genomes are, in some way, "the same thing" (and possibly correspond to the same taxon).

Note: The 50% coverage threshold can be considered as a line of "caution." Where coverage is less than 50%, there is the possibility that the two genomes are not in the same taxon. However, this is not diagnostic.

Here, taking the 50% coverage threshold between red and blue cells to indicate an approximate boundary between genera, we would consider that BPEN and 640 are the same genus, but that each of the other genomes is representative of a distinct genus.

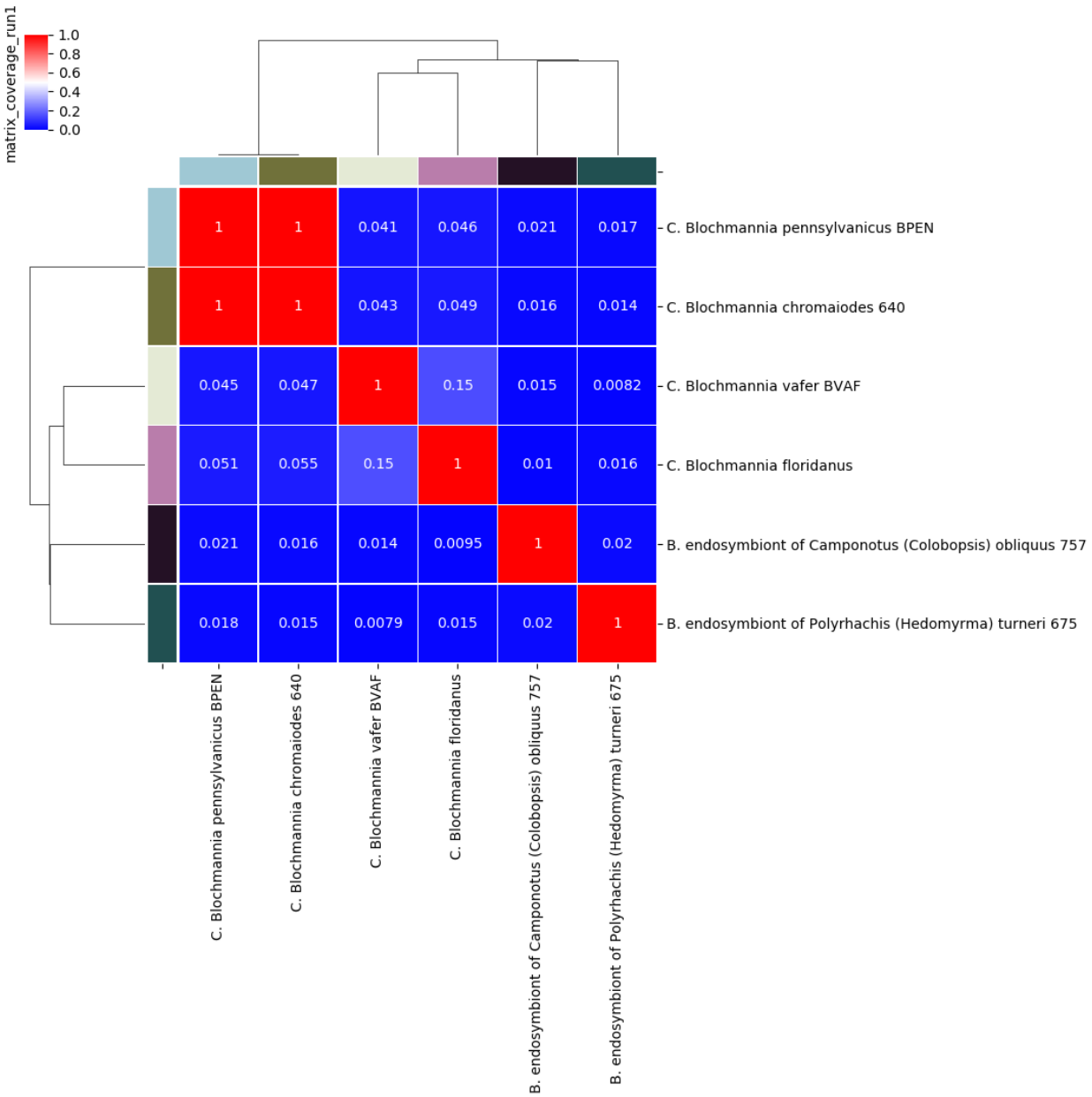

Fig. 4: Percentage coverage matrix for *Candidatus Blochmannia* ANIm analysis

Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in each cell is the pairwise coverage of each genome by aligned regions in the comparison. The dendrograms are generated by single-linkage hierarchical clustering from the matrix of pairwise coverage results. The default colour scheme colours cells with identity > 0.50 as red, and those with < 0.50 as blue. This division corresponds to a strict majority of each genome in the comparison being alignable (a plausible *ad hoc* minimum requirement for two sequences being considered "the same thing").

Note: There is no agreed, universal coverage threshold corresponding to genus boundaries, so we should always consider the actual coverage/AF values

In this case, nearly all the off-diagonal values in the blue cells are below 5%. This indicates that the proportion of each genome in the off-diagonal alignments is very small, and we are safe to assert that these organisms come from different genera. The exception to this is the comparison between BVAF and floridianus: their coverage is higher, at 15%. This may indicate a common plasmid or mobile element, or it may indicate a more recent common ancestor than the other comparisons; they may or may not validly be in the same genus - we would need to investigate further to understand their relationship.

The BPEN/640 comparison is conclusive, however. Their coverage/AF is essentially 100%, so these are closely-related, highly sequence-homologous organisms.

Distribution Plots

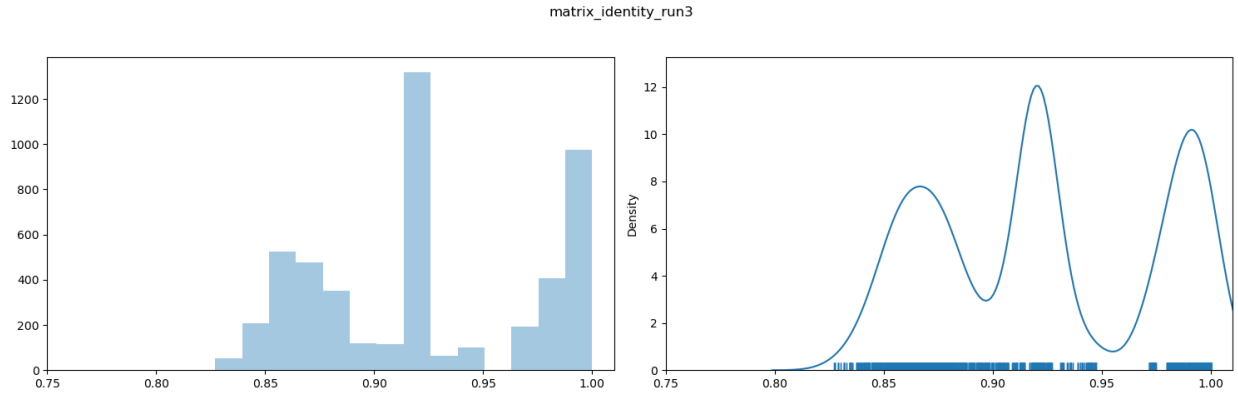

Fig. 5: ANI value distribution plot
In the distribution plots for each matrix, two figures are shown. On the left, a histogram of cell values is presented, representing binned values. On the right, a rug plot of individual matrix cell values and corresponding KDE plot (smoothed curve modelling the density as Gaussian distributions) is shown.

Note: *Discontinuities* in the distribution of ANI values have been associated with taxonomically-useful boundaries, especially species boundaries (between 94-96% depending on lineage). It is common to see these as gaps in the rug plot.

By inspection, we can see a discontinuity (i.e. a gap) in the rug plot that spans 95%. This is consistent with many prior observations that species boundaries coincide with a 94-96% ANI threshold. This plot provides some support for the assertion that comparisons to the right of the gap (red in the heatmap) are within-species comparisons, and those to the left (blue in the heatmap) are between-species comparisons.

Other gaps/discontinuities are visible. The interpretations of these are highly context-dependent and it is not always clear whether they are taxonomically meaningful (e.g. subspecies or genus boundary), or reflect sampling biases. Further investigation and evidential support is necessary.

Scatterplots

Comparisons with high ANI but relatively low coverage for the dataset in question (these appear *below* the main population) may suggest the presence of a significant proportion of mobile elements in the sequenced genome.

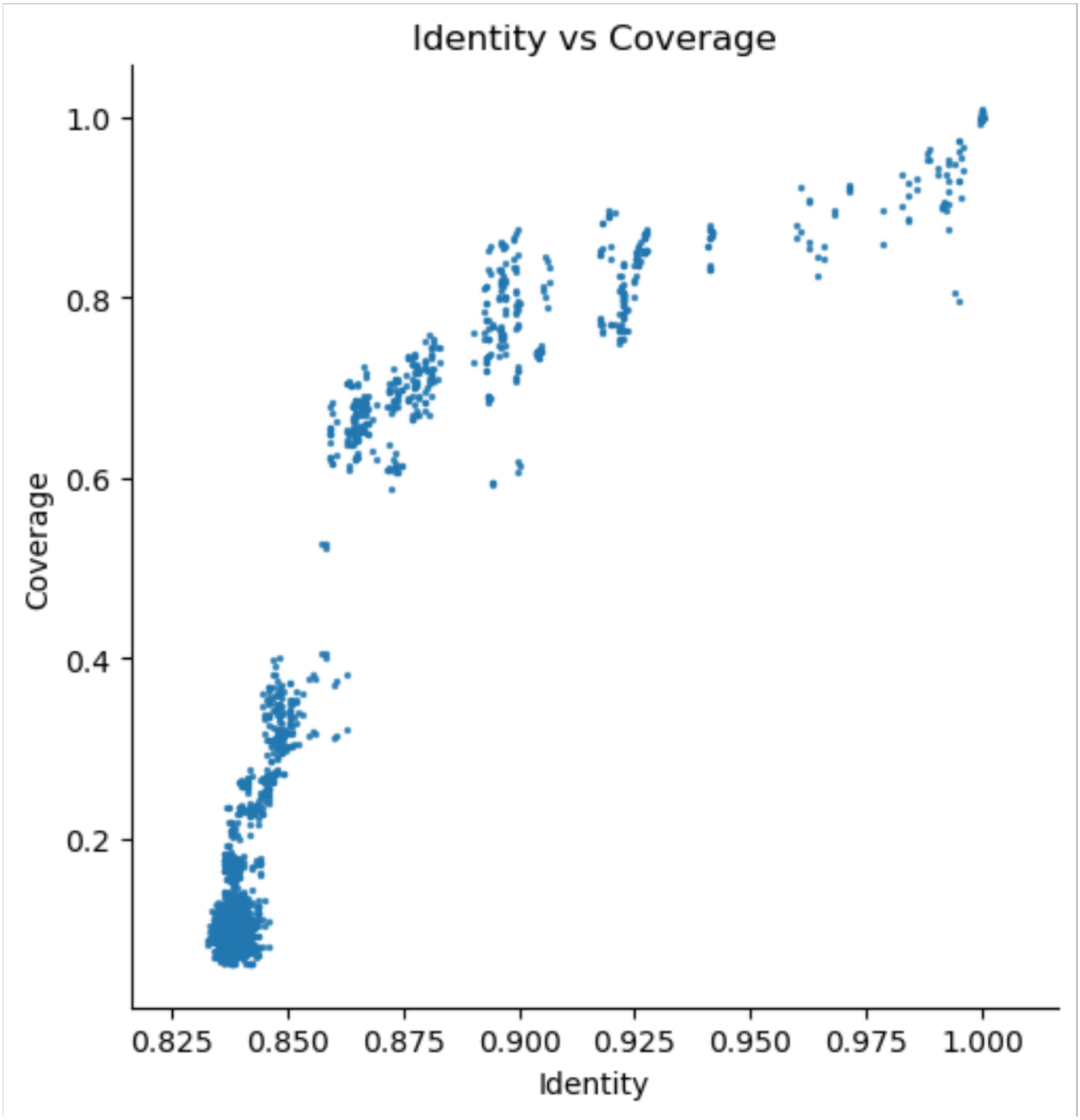

Fig. 6: Scatterplot of coverage/aligned fraction vs ANI
Plotting coverage/aligned fraction (y-axis) against ANI (x-axis) can be informative. Here, as is often the case for larger comparisons, there is a clear piecewise linear appearance to the plot. There is a relatively shallow gradient for high (>50%) coverage comparisons, and a steep gradient for low (<50%) coverage comparisons. There is a discontinuity on the coverage axis between 40% and 60% coverage, corresponding to a shift between the two piecewise linear regimes. This is often interpretable as a genus boundary, but requires further evidence and support to be certain.

Often a vertical banding can be seen, due to discontinuities in the distribution of ANI values. Here, the discontinuity at around 95% ANI is consistent with a division between within-species (right of the gap) and between-species (left of the gap) comparisons. Vertical bands to the left of the 95% line may indicate comparisons between particular pairs of species that are more or less recently diverged, but likely fall within the same genus.

The bulk of comparisons at the lower left of the plot likely indicate comparisons between relatively unrelated genomes, possibly from different genera.

Plot Asymmetry

Note: Each ANI method in *pyani* calculates results by a different method. The difference between methods is usually that alternative third-party alignment tools are used. However, there may also be differences between the ways those alignment outputs are used. Please see the relevant documentation for details of each method.

Average nucleotide identity is a measure of similarity between two genomes. Depending on the ANI method used, this may be symmetrical: comparing genome A to genome B is the same as comparing genome B to genome A; or asymmetrical: the result of comparing genome A with genome B can be different from comparing genome B with genome A.

Asymmetry can arise as a consequence of the way the sequence alignment algorithm used for calculating genome alignments works. For instance, the initial seed alignment for a pair of genomes may be very similar, but not identical, and this difference may propagate through an extension step into differences in the final alignment. Alternatively, an aspect of the ANI algorithm may introduce asymmetry. For instance, the genome fragmentation step in ANIb may break each participating genome in different ways.

pyani provides both symmetrical and asymmetrical ANI methods:

- ANIm — symmetrical
- FastANI — asymmetrical (only available in version 0.3.0-alpha)
- ANIb — asymmetrical
- ANIblastall — asymmetrical
- TETRA — symmetrical (though please note that this is not strictly an ANI method)

Alignment coverage is the proportion of the query genome that aligns against the reference genome. This can be asymmetrical even when the alignment itself is symmetrical, as the genomes participating in a pairwise alignment may have differing amounts of genomic sequence that do not contribute to the alignment. In general, comparing genome A to genome B will give different coverage values for A and B.

- in ANIm this is `alignment_length / genome_length` (asymmetrical)
- in fastANI this is `matched_fragments / all_fragments` (asymmetrical)
- in ANIb this is `alignment_length / query_genome_length` (asymmetrical)
- in ANIblastall this is `alignment_length / query_genome_length` (asymmetrical)

Alignment length is the count of bases contributed by each genome to the pairwise alignment between those genomes.

- in ANIm this is calculated as `reference_positions_in_alignment + insertions - deletions`
- in fastANI this is `matched_fragments * fragment_length`
- in ANIb this is `alignment_length - gaps`
- in ANIblastall this is `alignment_length - gaps`

The **similarity errors** graph shows a measure of the number of bases/positions that do not match exactly.

- in ANIm this is `non-identities + insertions + deletions`
- in fastANI this is `all_fragments - matched_fragents`
- in ANIb this is `gaps + mismatches`
- in ANIblastall this is `gaps + mismatches`

The **Hadamard** ouptut is the elementwise product (identity x coverage), as described at Hadamard product of identity and coverage. It's meant to provide a measure that allows you to interpret identity and coverage simultaneously.

- this is always `ANI * coverage`, but as the plot is not symmetric, coverage may differ for query and reference genomes

`pyani plot` also outputs a scatterplot of **Average nucleotide identity** versus **Alignment coverage** (calculated as described above).

5.7 Examples

5.7.1 Using non-NCBI genomes

It is usual to want to include or work only with genomes that have been generated locally, or that were not downloaded from NCBI using *pyani download*. To use these genomes with the *pyani* analysis subcommands, the genomes must be *indexed*[1].

To *index* a set of genomes, use the *pyani index* subcommand on the input directory, which is passed to the `-i` argument. To index the directory `mygenomes`, for example:

```
pyani index -i mygenomes
```

This will create a `.md5` file (containing the *hash*) for each genome, as well as class and label files listing all the input genomes.

All genomes in the `mygenomes` directory will now be available for use in *pyani*.

`class` and `labels` files

Using the `pyani download` command will create two files, by default `classes.txt` and `labels.txt` containing identifiers for each input genome that are used in later analysis and visualisation. These files are also created when `pyani index` is used as above.

The location of the labels and classes files may be changed using the `--labels` and `--classes` arguments, for example:

```
pyani index -i mygenomes --classes myclasses.txt --labels mylabels.txt
```

5.8 `pyani` subcommands

`pyani` has a subcommand structure, where the command `pyani` is followed immediately by a subcommand to let the software know which action you wish it to perform, in the format `pyani <subcommand>`. For example, to create a database you would use the `createdb` subcommand:

[1] *indexing* here refers to constructing a *hash* of the genome: a short representation of the entire genome's contents that can be used to identify it uniquely

```
pyani createdb
```

This document links out to detailed instructions for each of the `pyani` subcommands.

5.8.1 `pyani download`

The `download` subcommand controls download of genome files from the NCBI Assembly database for input to `pyani`.

```
usage: pyani download [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm] [--version]
                      [--citation] -o OUTDIR -t TAXON --email EMAIL
                      [--api_key API_KEYPATH] [--retries RETRIES]
                      [--batchsize BATCHSIZE] [--timeout TIMEOUT] [-f]
                      [--noclobber] [--labels LABELFNAME] [--classes CLASSFNAME]

                      [--kraken] [--dry-run]
```

Positional arguments

outdir The `outdir` argument should be the path to a directory into which genome files will be downloaded. If the directory exists, a warning will be given and the download will not proceed, to avoid overwriting existing data. To force writing into an existing directory, use the `-f` option.

Flagged arguments

--api_key PATH_TO_API_KEY The program will attempt to use an NCBI API key (see here) located at `PATH_TO_API_KEY`. Default: `~/.ncbi/api_key`

--batchsize BATCHSIZE The download process will attempt to download assemblies in multiples of `BATCHSIZE`. Default: 10000

--classes CLASSFNAME Write a set of labels (one per downloaded genome) to the file `CLASSFNAME` in `outdir`. Default: `classes.txt`

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--dry-run Perform all actions of the download process except for downloading files.

--email EMAIL COMPULSORY. Provide the email address `EMAIL` to NCBI so that they can track problems.

-f, --force Force use of the `OUTDIR` directory when downloaded genomes, even if it already exists.

-h, --help Display usage information for `pyani download`.

--kraken Add taxonomy information to the FASTA file headers of downloaded genomes. This allows the genomes to be readily used to construct databases for the Kraken software package.

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

--labels LABELFNAME Write a set of labels (one per downloaded genome) to the file `LABELFNAME` in `outdir`. Default: `labels.txt`

--noclobber Do not overwrite individual files in the `outdir` directory, when used with `-f`.

-o OUTDIR, --outdir OUTDIR The `OUTDIR` argument should be the path to a directory into which genome files will be downloaded. If the directory exists, a warning will be given and the download will not proceed, to avoid overwriting existing data. To force writing into an existing directory, use the `-f` option.

--retries RETRIES The download process will attempt to download each batch of assemblies a maximum of `RETRIES` times. Default: 20

-t TAXON, --taxon TAXON COMPULSORY. All genomes below taxon ID `TAXON` of a node in the NCBI Taxonomy database will be downloaded to the location specified by `outdir`.

--timeout TIMEOUT The download process will wait a amaximum of `TIMEOUT` seconds before abandoning a URL connection attempt. Default: 10

-v, --verbose Provide verbose output to `STDOUT`

5.8.2 pyani index

The `index` subcommand will index the genome files it finds the passed directory `INDIR`, generating label and class files, and files that contain an MD5 hash of the nucleotide sequence of each genome.

```
usage: pyani index [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm] [--version]
                   [--citation] -i INDIR [--labels LABELFNAME]
                   [--classes CLASSFNAME]
```

Flagged arguments

--classes CLASSFNAME Write a set of labels (one per genome sequence file) to the file `CLASSFNAME` in `INDIR`. Default: `classes.txt`

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

-h, --help Display usage information for `pyani index`.

-i INDIR, --indir INDIR The `INDIR` argument should be the path to a directory containing genome sequence data as FASTA files (one per genome assembly).

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

--labels LABELFNAME Write a set of labels (one per genome sequence file) to the file `LABELFNAME` in `INDIR`. Default: `labels.txt`

-v, --verbose Provide verbose output to `STDOUT`

5.8.3 pyani createdb

The `createdb` subcommand creates a new, empty database for `pyani` to use in subsequent analysis runs.

```
usage: pyani createdb [-h] [-l LOGFILE] [-v] [--disable_tqdm]
                      [--dbpath DBPATH] [-f]
```

Flagged arguments

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--dbpath DBPATH Path to the location where the database will be created. Default: `.pyani/pyanidb`

-h, --help Display usage information for `pyani createdb`.

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

-v, --verbose Provide verbose output to `STDOUT`

5.8.4 pyani anim

The `anim` subcommand will carry out ANIm analysis using genome files contained in the `INDIR` directory, writing result files to the `OUTDIR` directory, and recording data about each comparison and run in a local SQLite3 database.

```
usage: pyani anim [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm] [--version]
                  [--citation] [--scheduler {multiprocessing,SGE}]
                  [--workers WORKERS] [--SGEgroupsize SGEGROUPSIZE]
                  [--SGEargs SGEARGS] [--jobprefix JOBPREFIX] [--name NAME]
                  [--classes CLASSES] [--labels LABELS] [--recovery] -i INDIR
                  -o OUTDIR [--dbpath DBPATH] [--nucmer_exe NUCMER_EXE]
                  [--filter_exe FILTER_EXE] [--maxmatch] [--nofilter]
```

Flagged arguments

--classes CLASSFNAME Use the set of classes (one per genome sequence file) found in the file `CLASSFNAME` in `INDIR`. Default: `classes.txt`

--dbpath DBPATH Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--filter_exe FILTER_EXE Path to the `MUMmer delta-filter` executable. Default: `delta-filter`

-i INDIR, --indir INDIR Path to the directory containing indexed genome files to be used for the analysis.

-h, --help Display usage information for `pyani anim`.

--jobprefix JOBPREFIX Use the string `JOBPREFIX` as a prefix for SGE job submission names. Default: `PYANI`

--labels LABELFNAME Use the set of labels (one per genome sequence file) found in the file `LABELFNAME` in `INDIR`. Default: `labels.txt`

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

--maxmatch Use the `MUMmer --maxmatch` option to include all `nucmer` matches.

--name NAME Use the string `NAME` to identify this ANIm run in the `pyani` database.

--nofilter Do not use `delta-filter` to restrict `nucmer` output to 1:1 matches.

--nucmer_exe NUCMER_EXE Path to the `MUMmer nucmer` executable. Default: `nucmer`

-o OUTDIR, --outdir OUTDIR Path to a directory where comparison output files will be written.

--recovery Use existing `NUCmer` comparison output if available, e.g. if recovering from a failed job submission. Using this option will not generate a new comparison if the old output files exist.

--scheduler {multiprocessing, SGE} Specify the job scheduler to be used when parallelising genome comparisons: one of `multiprocessing` (use many cores on the current machine) or `SGE` (use an SGE or OGE job scheduler). Default: `multiprocessing`.

--SGEargs SGEARGS Pass additional arguments `SGEARGS` to `qsub` when running the SGE-distributed jobs.

--SGEgroupsize SGEGROUPSIZE Create SGE arrays containing SGEGROUPSIZE comparison jobs. Default: 10000

-v, --verbose Provide verbose output to `STDOUT`

--workers WORKERS Spawn WORKERS worker processes with the `--scheduler multiprocessing` option. Default: 0 (use all cores)

5.8.5 pyani anib

The `anib` subcommand will carry out ANIb analysis using genome files contained in the `indir` directory, writing result files to the `outdir` directory, and recording data about each comparison and run in a local SQLite3 database.

```
usage: pyani.py anib [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm] [--version]
                     [--citation] [--scheduler {multiprocessing,SGE}]
                     [--workers WORKERS] [--SGEgroupsize SGEGROUPSIZE]
                     [--SGEargs SGEARGS] [--jobprefix JOBPREFIX] [--name NAME]
                     [--classes CLASSES] [--labels LABELS] [--recovery] -i INDIR -o
                     OUTDIR [--dbpath DBPATH] [--blastn_exe BLASTN_EXE]
                     [--format_exe FORMAT_EXE] [--fragsize FRAGSIZE]
```

Flagged arguments

--blastn_exe BLASTN_EXE Path to the `blastn` executable. Default: `blastn`

--classes CLASSFNAME Use the set of classes (one per genome sequence file) found in the file `CLASSFNAME` in `INDIR`. Default: `classes.txt`

--dbpath DBPATH Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm Disable the `tqdm` progress bar while the ANIb process runs. This is useful when testing to avoid aesthetic problems with test output.

--format_exe FORMAT_EXE Path to the `blastn` executable. Default: `makeblastdb`

--fragsize FRAGSIZE Fragment size to use in analysis. (default: 1020)

-h, --help Display usage information for `pyani anib`.

-i INDIR, --input INDIR Path to the directory containing indexed genome files to be used for the analysis.

--jobprefix JOBPREFIX Use the string `JOBPREFIX` as a prefix for SGE job submission names. Default: `PYANI`

--labels LABELFNAME Use the set of labels (one per genome sequence file) found in the file `LABELFNAME` in `INDIR`. Default: `labels.txt`

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the ANIb process will be written.

--name NAME Use the string `NAME` to identify this `ANIb` run in the `pyani` database.

-o OUTDIR, --outdir OUTDIR Path to a directory where comparison output files will be written.

--recovery Use existing ANIb comparison output if available, e.g. if recovering from a failed job submission. Using this option will not generate a new comparison if the old output files exist.

--scheduler {multiprocessing, SGE} Specify the job scheduler to be used when parallelising genome comparisons: one of multiprocessing (use many cores on the current machine) or SGE (use an SGE or OGE job scheduler). Default: multiprocessing.

--SGEargs SGEARGS Pass additional arguments SGEARGS to qsub when running the SGE-distributed jobs.

--SGEgroupsize SGEGROUPSIZE Create SGE arrays containing SGEGROUPSIZE comparison jobs. Default: 10000

-v, --verbose Provide verbose output to STDOUT.

--workers WORKERS Spawn WORKERS worker processes with the --scheduler multiprocessing option. Default: 0 (use all cores)

5.8.6 pyani report

The report subcommand reports the contents of a local SQLite3 database, located at dbpath. Output files will be written to the outdir directory.

usage: pyani report [-h] [-l LOGFILE] [-v] [–debug] [–disable_tqdm] [–version] [–citation] -o OUTDIR [–dbpath DBPATH] [–runs] [–genomes] [–runs_genomes] [–genomes_runs] [–run_results RUN_ID [RUN_ID . . .]] [–run_matrices RUN_ID [RUN_ID . . .]] [–no_matrix_labels] [–formats FORMAT [FORMAT . . .]]

Flagged arguments

--dbpath DBPATH Path to the location of the local pyani database to be used. Default: .pyani/pyanidb

--disable_tqdm Disable the tqdm progress bar while the report process runs. This is useful when testing to avoid aesthetic problems with test output.

--formats FORMAT [FORMAT ...] Space-separated list of output formats (in addition to .tab); possible values: {html, excel,stdout}. (default: None)

--genomes Report table of genomes in database. (default: False)

--genomes_runs Report table of all runs in which each genome in the database participates. (default: False)

--no_matrix_labels Turn off row/column labels in output matrix files (default: False)

-o OUTDIR, --outdir OUTDIR Directory where output analysis reports will be written.

--runs Report table of analysis runs in database. (default: False)

--runs_genomes Report table of all genomes for each run in database. (default: False)

--run_matrices RUN_ID [RUN_ID ...] Report matrices of results for space-separated list of runs. (default: None)

--run_results RUN_ID [RUN_ID ...] Report table of results for space-separated list of runs. (default: None)

-v, --verbose Provide verbose output to STDOUT.

5.8.7 pyani plot

The plot subcommand will plot the results of an ANI analysis, specified by run_id, contained within a local SQLite3 database, located at dbpath. Plot files will be written to the outdir directory.

```
usage: pyani plot [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm] [--version]
                  [--citation] -o OUTDIR --run_ids RUN_ID [RUN_ID ...]
                  [--dbpath DBPATH] [--formats FORMAT [FORMAT ...]]
                  [--method METHOD] [--workers WORKERS]
```

Flagged arguments

--dbpath DBPATH Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm Disable the `tqdm` progress bar while the plotting process runs. This is useful when testing to avoid aesthetic problems with test output.

--formats FORMAT [FORMAT ...] Graphics output format(s); more than one can be specified. Valid options are: (pdf/png/svg/jpg). (default: png)

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the plotting process will be written.

--method METHOD Graphics method to use for plotting; options (seaborn, mpl, plotly). (default: seaborn)

-o OUTDIR, --o outdir OUTDIR Path to a directory where comparison plot files will be written.

--run_ids RUN_ID [RUN_ID ...] Unique database ID of the runs to be plotted.

5.8.8 `pyani classify`

The `classify` subcommand identifies cliques (k-complete) graphs of genomes from a larger set. This is helpful for circumscribing potentially meaningful groups of genomes that can not be described using traditional taxonomy.

```
usage: pyani.py classify [-h] [-l LOGFILE] [-v] [--debug] [--disable_tqdm]
                         [--version] [--citation] -o OUTDIR --run_id RUN_ID
                         [--dbpath DBPATH] [--cov_min COV_MIN] [--id_min ID_MIN]
                         [--min_id MIN_ID] [--max_id MAX_ID]
                         [--resolution RESOLUTION] [--show_all]
```

Flagged arguments

--cov_min COV_MIN Minimum percent coverage required to draw an edge between two nodes (genomes). (default: 0.5)

--dbpath DBPATH Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output. (default: False)

--formats FORMATS Graphics output format(s); more than one can be specified. Valid options are: (pdf/png/svg/jpg). (default: png)

--id_min ID_MIN Minimum percent identity required to draw an edge between two nodes (genomes). (default: 0.8)

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

--max_id MAX_ID Maximum identity threshold to test. (default: None)

--method {seaborn,mpl,plotly} Graphics method to use for plotting. (default: seaborn)

--min_id MIN_ID Minimum identity threshold to test. (default: None)

-o OUTDIR, --outdir OUTDIR Path to a directory where comparison output files will be written.

--resolution RESOLUTION Number of identity thresholds to test. (default: 0.0001)

--run_id RUN_ID Unique database ID of the run to be plotted.

--show_all Report all intervals in log. (default: only intervals where all subgraphs are k-complete) (default: False)

-v, --verbose Provide verbose output to `STDOUT`.

5.8.9 `pyani listdeps`

The `listdeps` subcommand writes an account of the local platform, the installed Python version, and installed dependencies (Python packages and third-party software tools) to `STDOUT`.

```
usage: pyani listdeps [-h] [-l LOGFILE] [-v]
```

Flagged arguments

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. Does nothing.

-h, --help Display usage information for `pyani listdeps`.

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the subcommand output will be written.

-v, --verbose Provide verbose output to `STDOUT`. This actually provides no additional information.

5.8.10 `pyani fastani`

The `fastani` subcommand will carry out fastANI analysis using genome files contained in the `indir` directory, writing result files to the `outdir` directory, and recording data about each comparison and run in a local SQLite3 database.

```
usage: pyani fastani [-h] [-l LOGFILE] [-v] [--debug]
                     [--disable_tqdm] [--citation] [--scheduler {multiprocessing,SGE}]
                     [--workers WORKERS] [--SGEgroupsize SGEGROUPSIZE]
                     [--SGEargs SGEARGS] [--jobprefix JOBPREFIX] [--name NAME]
                     [--classes CLASSES] [--labels LABELS] [--recovery]
                     [--dbpath DBPATH] [--fastani_exe FASTANI_EXE] [-k KMERSIZE]
                     [--fragLen FRAGLEN] [-t THREADS] [--minFraction MINFRACTION]
                     -i INDIR -o OUTDIR
```

Flagged arguments

-i, --input INDIR Path to the directory containing indexed genome files to be used for the analysis.

-o, --outdir OUTDIR Path to a directory where comparison output files will be written.

--classes CLASSFNAME Use the set of classes (one per genome sequence file) found in the file `CLASSFNAME` in `INDIR`. Default: `classes.txt`

--dbpath DBPATH Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--fastani_exe FASTANI_EXE Path to the `fastANI` executable. Default: `fastANI`

--fragLen FRAGLEN Fragment length to use in analysis. (default: 3000)

-h, --help Display usage information for `pyani fastani`.

--jobprefix JOBPREFIX Use the string `JOBPREFIX` as a prefix for SGE job submission names. Default: `PYANI`

-k, --kmer KMERSIZE Set the kmer size <= 16; can not be greater than 16. (default: 16)

--labels LABELFNAME Use the set of labels (one per genome sequence file) found in the file `LABELFNAME` in `INDIR`. Default: `labels.txt`

-l LOGFILE, --logfile LOGFILE Provide the location `LOGFILE` to which a logfile of the download process will be written.

--minFraction MINFRACTION Minimum fraction of genome that must be shared for trusting ANI. If reference and query genome size differ, smaller one among the two is considered. (default: 0.2)

--name NAME Use the string `NAME` to identify this `fastANI` run in the `pyani` database.

--recovery Use existing `fastANI` comparison output if available, e.g. if recovering from a failed job submission. Using this option will not generate a new comparison if the old output files exist.

--scheduler {multiprocessing, SGE} Specify the job scheduler to be used when parallelising genome comparisons: one of `multiprocessing` (use many cores on the current machine) or `SGE` (use an SGE or OGE job scheduler). Default: `multiprocessing`.

--SGEargs SGEARGS Pass additional arguments `SGEARGS` to `qsub` when running the SGE-distributed jobs.

--SGEgroupsize SGEGROUPSIZE Create SGE arrays containing SGEGROUPSIZE comparison jobs. Default: 10000

-v, --verbose Provide verbose output to `STDOUT`.

--workers WORKERS Spawn WORKERS worker processes with the `--scheduler multiprocessing` option. Default: 0 (use all cores)

5.9 Use With a Scheduler

5.9.1 Sun/Open Grid Engine

The `--scheduler SGE` argument allows one to use `pyani` with an an SGE-type scheduler.

In order for this work, one must be able to submit jobs using the `qsub` command. By default, this will batch the pairwise comparisons in array jobs of 10,000, in order to avoid clogging the scheduler queue. Each comparison will be run as a single-core task in an array job.

Arguments assigned by Pyani

The following arguments will be automatically set:

```
-N job_name  # this is the value passed to `--name`
 -cwd
 -o ./stdout  # cwd/ + "stdout"
 -e ./stderr  # cwd/ + "stderr"
```

Modifiable arguments

The number of pairwise comparisons submitted per chunk can be modified using:

```
--SGEgroupsize *number*
```

The job prefix to use can be modified using:

```
--jobprefix *prefix*
```

Specifying additional arguments

Additional SGE arguments may be specified with:

```
--SGEargs "<your arguments here>"
```

Additional arguments must be specified as a string which includes all flags and their values. For instance, to specify a value for the memory resource:

```
"-l h_mem=64G"
```

Or for both memory and run time:

```
"-l h_mem=64G -l h_rt=02:00:00"
```

An alternative to listing all desired options on the command line is to pass an `optionfile`:

```
"-@ optionfile"
```

This file only contains comments and the flags to be passed (do not include the `#$` in front of the arguments):

```
# Memory to assign to the job
 -l h_mem=64G

 # Time to allow for job HH:MM:SS
 -l h_rt=10:20:00

 # Notification email address
 -M email@domain.com

 # Send notifications when job 'b'egins, 'a'borts (or is rescheduled), 'e'nds, or is
↪'s'uspended
 -m baes
```

For more information on using an SGE/OGE scheduler, see:

- Open Grid Scheduler

5.9.2 SLURM

Support for the SLURM scheduler is forthcoming.

5.10 Testing

We are currently writing tests formatted for the pytest package, for testing `pyani`.

Warning: Some tests are still targeted at nosetests, which is in maintenance mode and, although we are still in transition, our plan is to change the test framework completely to use pytest.

5.10.1 Test directory structure

All tests, supporting input and target data, are to be found in the `tests/` subdirectory of the `pyani` repository.

- Input data for tests is located under the `tests/test_input` directory, in subdirectories named for the general operation that is being tested.
- Target data (known correct values) for tests is located under the `tests/test_targets` directory, in subdirectories named for the general operation being tested.
- Test output is written to the `tests/test_output` directory, in subdirectories named for the general operation being tested.

Tip: If you wish to write new tests for `pyani`, we ask that your test data and operations conform to this structure

The `tests/test_failing_data` directory contains data that is known to cause problems for `pyani` in that at least two of the input genomes have no appreciable sequence identity.

5.10.2 Running tests

To run tests with `nosetests`, change directory to the root of the `pyani` repository, and invoke a `nosetests` command.

Run all tests locally

To run all tests locally on your machine, issue the following command from the repository root:

```
nosetests -v
```

This will cause `nose` to run all tests under the `tests/` subdirectory.

Run individual tests

Tests are grouped in files with filenames that match `test_*.py`. We aim to write tests as classes that subclass `unittest.TestCase`, as described in the nosetests documentation. An example of this style can be found in the `tests/test_anim.py` test file.

This style allows us to run tests at several levels of granularity, specifying all tests (see above), all tests within a module (e.g. `test_anim.py`), or all tests within a class in that test file.

For example, to run all ANIm-related tests, we can issue:

```
nosetests -v tests/test_anim.py
```

To run all tests of `nucmer` command line generation, we can specify a single class within that file using the command:

```
nosetests -v tests/test_anim.py:TestNUCmerCmdline
```

And to test only "multiple command generation" we can issue the following:

```
nosetests -v tests/test_anim.py:TestNUCmerCmdline.test_multi_cmd_generation
```

5.11 Contributing to `pyani`

5.11.1 Reporting bugs and errors

If you find a bug, or an error in the code or documentation, please report this by raising an issue at the GitHub issues page for `pyani`:

- GitHub issues page

5.11.2 Contributing code or documentation

We gratefully accept code and other contributions. A list of contributors can be found via the Github contributors link.

You are welcome to help develop `pyani`, fix a bug, improve documentation, or contribute in any other way. To make everyone's lives easier in this process, we ask that you please follow the guidelines for developers below:

Pre-commit checks and style guide

So far as is possible, we aim to follow the coding conventions as described in PEP8 and PEP257, but we have adopted `black` code styling, which does vary from the PEPs in places.

We use the `flake8` tool for style checks, and this can be installed as follows (with two useful plugins):

```
pip install flake8 flake8-docstrings flake8-blind-except
```

`flake8` can then be run directly on the codebase with

```
flake8 bin/
flake8 pyani/
```

We use the `black` tool for code style checking, which can be installed with:

```
pip install black
```

The `flake8` and ``black` styles can be enforced as pre-commit hooks using the pre-commit package (included in `requirements.txt`).

The `black` and `flake8` hooks are defined in `.pre-commit-config.yaml`; custom settings for `flake8` are held in `.flake8` (all files are under version control).

To enable pre-commit checks in the codebase on your local machine (once `pre-commit` has been installed), execute the following command in the root directory of this repository:

```
pre-commit install
```

Checking changes to the documentation

Much of the repository documentation is written in Markdown files, but the main documentation (which you are reading) is prepared for ReadTheDocs, which uses reStructuredText and `Sphinx`. The `Sphinx` configuration is described in `docs/conf.py` (under version control).

So long as `Sphinx` is installed on your machine, you can check your documentation changes locally, building inplace by changing to the `docs/` directory and issuing:

```
make html
```

This will place a compiled version of the documentation under `_build/html`, which you can inspect before committing to the repository.

Tip: To build the documentation in `ReadTheDocs` style, you will need to install the corresponding theme with `pip install sphinx_rtd_theme` or `conda install sphinx_rtd_theme`

For now, docstrings in the source code are not required to be in any controlled syntax, such as reStructuredText, but this may change.

Making changes and pull requests

1. Fork the `pyani` repository under your account at GitHub.
2. Clone your fork to your development machine.

- To be able to edit *pyani* and have changes you make take effect immediately without a reinstall (useful for testing), you can run *pip install -e .* inside the local cloned repository.

3. Create a new branch in your forked repository with an informative name like `fix_issue_107`, using `git` (e.g. with the command `git checkout -b fix_issue_107`).
4. Make the changes you need and commit them to your local branch.
5. Run the repository tests (see the *Testing* documentation for more details).
6. If the tests all pass, push the changes to your fork, and submit a pull request against the original repository.
7. Indicate one of the `pyani` developers as an assignee to review your pull request when you submit your pull request.

The assigned developer will then review your pull request, and merge it or continue the conversation, as appropriate.

5.11.3 Suggestions for improvement

If you would like to make a suggestion for how we could improve `pyani`, we welcome contributions. If you have a specific problem, or a concrete suggestion, you can submit these at the GitHub issues page. If you would like to discuss an idea with the maintainers and the `pyani` community, this can be done at the Github discussions page.

5.12 Licensing

Unless otherwise indicated, all code is subject to the following agreement:

> (c) The James Hutton Institute 2014-2019 (c) The University of Strathclyde 2019 Author: Leighton Pritchard
>
> Contact: leighton.pritchard@strath.ac.uk
>
> **Address:** Leighton Pritchard, Strathclyde Institute of Pharmacy and Biomedical Sciences, University of Strathclyde, 161 Cathedral Street, Glasgow, G4 0RE, Scotland, UK

5.12.1 The MIT License

Copyright (c) 2014-2019 The James Hutton Institute Copyright (c) 2019 The Univeristy of Strathclyde

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.13 pyani package

Module with main code for pyani application/package.

exception pyani.**PyaniException**
Bases: Exception

General exception for pyani.

5.13.1 Subpackages

pyani.scripts package

Modules to support use of pyani as a script.

exception pyani.scripts.**PyaniScriptException**(*msg='Error in pyani.py script'*)
Bases: Exception

General exception for pyani.py script.

pyani.scripts.**make_outdir**(*outdir: pathlib.Path*, *force: bool*, *noclobber: bool*) → None
Create output directory (allows for force and noclobber).

Parameters

- **outdir** – Path, path to output directory
- **force** – bool, True if an existing directory will be reused
- **noclobber** – bool, True if existing files are not overwritten

The intended outcomes are: outdir doesn't exist: create outdir outdir exists: raise exception outdir exists, –force only: remove the directory tree outdir exists, –force –noclobber: continue with existing directory tree

So long as the outdir is created with this function, we need only check for args.noclobber elsewhere to see how to proceed when a file exists.

Subpackages

pyani.scripts.parsers package

Module providing command-line parser definitions.

pyani.scripts.parsers.**parse_cmdline**(*argv: Optional[List[T]] = None*) → argparse.Namespace

Parse command-line arguments for script.

Parameters **argv** – Namespace, command-line arguments

The script offers a single main parser, with subcommands for the actions:

- **download** download all available NCBI assemblies below the passed taxonomy ID
- **index** index genome sequence files in a subdirectory, for analysis
- **createdb** generate SQLite database for data and analysis results
- **anim** conduct ANIm analysis
- **anib** conduct ANIb analysis
- **aniblastall** conduct ANIblastall analysis
- **fastani** conduct fastANI analysis
- **report** generate output describing analyses, genomes, and results
- **plot** generate graphical output describing results
- **classify** produce graph-based classification of genomes on the basis of ANI analysis

Submodules

pyani.scripts.parsers.anib_parser module

Provides parser for anib subcommand.

pyani.scripts.parsers.anib_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the anib subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The terminology may be confusing, but in practice the main parser collects command-line arguments that are then available to this parser, which inherits options from the parsers in *parents* in addition to those defined below.

pyani.scripts.parsers.aniblastall_parser module

Provides parser for aniblastall subcommand.

pyani.scripts.parsers.aniblastall_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the aniblastall subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

pyani.scripts.parsers.anim_parser module

Provides parser for anim subcommand.

pyani.scripts.parsers.anim_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the anim subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

pyani.scripts.parsers.classify_parser module

Provides parser for classify subcommand.

pyani.scripts.parsers.classify_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the classify subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The classify subcommand takes specific arguments:

--cov_min	(minimum coverage threshold for an edge)
--id_min	(minimum identity threshold for an edge)
--resolution	(number of identity thresholds to test)

pyani.scripts.parsers.common_parser module

Provides parser for arguments common to all subcommands.

pyani.scripts.parsers.common_parser.**build**() → argparse.ArgumentParser
Return the common argument parser for all script subcommands.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

Common arguments are:

-l, –logfile (specify logfile output path) -v, –verbose (produce verbose output on STDOUT)

pyani.scripts.parsers.createdb_parser module

Provides parser for createdb subcommand.

pyani.scripts.parsers.createdb_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None
Return a command-line parser for the createdb subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

pyani.scripts.parsers.download_parser module

Provides parser for download subcommand.

pyani.scripts.parsers.download_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None
Return a command-line parser for the download subcommand.

Parameters

- **subps** – ArgumentParser.subparser
- **parents** – additional Parser objects

The download subcommand takes specific arguments:

-t, –taxon (NCBI taxonomy IDs - comma-separated list, or one ID) –email (email for providing to Entrez services) –api_key (path to file containing personal API key for Entrez) –retries (number of Entrez retry attempts to make) –batchsize (number of Entrez records to download in a batch) –timeout (how long to wait for Entrez query timeout) -f, –force (allow existing directory overwrite) –noclobber (don't replace existing files) –labels (path to write labels file) –classes (path to write classes file)

pyani.scripts.parsers.index_parser module

Provides parser for index subcommand.

pyani.scripts.parsers.index_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the index subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The index subcommand takes a single positional argument:

- indir (directory containing input genome sequence files)

pyani.scripts.parsers.plot_parser module

Provides parser for plot subcommand.

pyani.scripts.parsers.plot_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the plot subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The plot subcommand takes specific arguments:

--method (graphics method to use)

pyani.scripts.parsers.report_parser module

Provides parser for report subcommand.

pyani.scripts.parsers.report_parser.**build**(*subps: argparse._SubParsersAction*, *parents: Optional[List[argparse.ArgumentParser]] = None*) → None

Return a command-line parser for the report subcommand.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

pyani.scripts.parsers.run_common_parser module

Provides parser for arguments common to analysis run subcommands.

pyani.scripts.parsers.run_common_parser.**build**() → argparse.ArgumentParser

Return the common argument parser for analysis run subcommands.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

Common arguments are:

--name	(human-readable name for the run)
--labels	(genome labels for this run)
--classes	(genome classes for this run)

pyani.scripts.parsers.scheduling_parser module

Provides parser for arguments common to job scheduling.

pyani.scripts.parsers.scheduling_parser.**build**() → argparse.ArgumentParser
Return the common argument parser for job scheduling.

Parameters

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

Common arguments are:

pyani.scripts.subcommands package

Module providing subcommands for pyani scripts.

Submodules

pyani.scripts.subcommands.subcmd_anib module

Provides the anib subcommand for pyani.

pyani.scripts.subcommands.subcmd_anib.**fragment_fasta_file**(*inpath: pathlib.Path*, *outdir: pathlib.Path*, *fragsize: int*) → Tuple[pathlib.Path, str]
Return path to fragmented sequence file and JSON of fragment lengths.

Parameters

- **inpath** – Path to genome file
- **outdir** – Path to directory to hold fragmented files
- **fragsize** – size of genome fragments

Returns a tuple of `(path, json)` where `path` is the path to the fragment file and `json` is a JSON-ified dictionary of fragment lengths, keyed by fragment sequence ID.

pyani.scripts.subcommands.subcmd_anib.**generate_joblist**(*comparisons: List[T]*, *existingfiles: List[T]*, *fragfiles: List[T]*, *fraglens: List[T]*, *args: argparse.Namespace*) → NotImplementedError
Return list of ComparisonJobs.

Parameters

- **comparisons** – list of (Genome, Genome) tuples for which comparisons are needed
- **existingfiles** – list of pre-existing BLASTN+ outputs
- **fragfiles** –
- **fraglens** –
- **args** – Namespace, command-line arguments

pyani.scripts.subcommands.subcmd_anib.**subcmd_anib**(*args: argparse.Namespace*) → None

Perform ANIb on all genome files in an input directory.

Parameters **args** – Namespace, command-line arguments

Finds ANI by the ANIb method, as described in Goris J, Konstantinidis KT, Klappenbach JA, Coenye T, Vandamme P, et al. (2007) DNA-DNA hybridization values and their relationship to whole-genome sequence similarities. Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0.

All FASTA format files (selected by suffix) in the input directory are fragmented into (by default 1020nt) consecutive sections, and a BLAST+ database constructed from the whole genome input. The BLAST+ blastn tool is then used to query each set of fragments against each BLAST+ database, in turn.

For each query, the BLAST+ .tab output is parsed to obtain alignment length, identity and similarity error count. Alignments below a threshold are not included in the calculation (this introduces systematic bias with respect to ANIm). The results are processed to calculate the ANI percentages, coverage, and similarity error.

The calculated values are stored in the local SQLite3 database.

pyani.scripts.subcommands.subcmd_aniblastall module

Provides the aniblastall subcommand for pyani.

pyani.scripts.subcommands.subcmd_aniblastall.**subcmd_aniblastall**(*args: argparse.Namespace*)

Perform ANIblastall on all genome files in an input directory.

Parameters

- **args** –
- **logger** –

pyani.scripts.subcommands.subcmd_anim module

Provides the anim subcommand for pyani.

class pyani.scripts.subcommands.subcmd_anim.**ComparisonJob**

Bases: tuple

Pairwise comparison job for the SQLAlchemy implementation.

filtercmd
Alias for field number 2

job
Alias for field number 5

nucmercmd
Alias for field number 3

outfile
Alias for field number 4

query
Alias for field number 0

subject
Alias for field number 1

class pyani.scripts.subcommands.subcmd_anim.**ComparisonResult**
Bases: tuple

Convenience struct for a single nucmer comparison result.

aln_length
Alias for field number 2

pid
Alias for field number 4

qcov
Alias for field number 7

qid
Alias for field number 0

qlen
Alias for field number 5

scov
Alias for field number 8

sid
Alias for field number 1

sim_errs
Alias for field number 3

slen
Alias for field number 6

class pyani.scripts.subcommands.subcmd_anim.**ProgData**
Bases: tuple

Convenience struct for comparison program data/info.

program
Alias for field number 0

version
Alias for field number 1

class pyani.scripts.subcommands.subcmd_anim.**ProgParams**
Bases: tuple

Convenience struct for comparison parameters.

Use default of zero for fragsize or else db queries will not work as SQLite/Python nulls do not match up well

fragsize
Alias for field number 0

maxmatch
Alias for field number 1

class pyani.scripts.subcommands.subcmd_anim.**RunData**
Bases: tuple

Convenience struct describing an analysis run.

cmdline
Alias for field number 3

date
Alias for field number 2

method
Alias for field number 0

name
Alias for field number 1

pyani.scripts.subcommands.subcmd_anim.**generate_joblist**(*comparisons: List[Tuple], existingfiles: List[pathlib.Path], args: argparse.Namespace*) → List[pyani.scripts.subcommands.subcmd_anim.Comp

Return list of ComparisonJobs.

Parameters

- **comparisons** – list of (Genome, Genome) tuples
- **existingfiles** – list of pre-existing nucmer output files
- **args** – Namespace of command-line arguments for the run

pyani.scripts.subcommands.subcmd_anim.**run_anim_jobs**(*joblist: List[pyani.scripts.subcommands.subcmd_anim.Comparis args: argparse.Namespace*) → None

Pass ANIm nucmer jobs to the scheduler.

Parameters

- **joblist** – list of ComparisonJob namedtuples
- **args** – command-line arguments for the run

pyani.scripts.subcommands.subcmd_anim.**subcmd_anim**(*args: argparse.Namespace*) → None

Perform ANIm on all genome files in an input directory.

Parameters **args** – Namespace, command-line arguments

Finds ANI by the ANIm method, as described in Richter et al (2009) Proc Natl Acad Sci USA 106: 19126-19131 doi:10.1073/pnas.0906412106.

All FASTA format files (selected by suffix) in the input directory are compared against each other, pairwise, using NUCmer (whose path must be provided).

For each pairwise comparison, the NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment between the two organisms, as represented by sequences in the FASTA files. These are processed to calculated aligned sequence lengths, average nucleotide identity (ANI) percentages, coverage (aligned percentage of whole genome - forward direction), and similarity error count for each pairwise comparison.

The calculated values are deposited in the SQLite3 database being used for the analysis.

For each pairwise comparison the NUCmer output is stored in the output directory for long enough to extract summary information, but for each run the output is gzip compressed. Once all runs are complete, the outputs for each comparison are concatenated into a single gzip archive.

pyani.scripts.subcommands.subcmd_anim.**update_comparison_results**(*joblist: List[pyani.scripts.subcommands.subcm run, session, nucmer_version: str, args: argparse.Namespace*) → None

Update the Comparison table with the completed result set.

Parameters

- **joblist** – list of ComparisonJob namedtuples
- **run** – Run ORM object for the current ANIm run
- **session** – active pyanidb session via ORM
- **nucmer_version** – version of nucmer used for the comparison
- **args** – command-line arguments for this run

The Comparison table stores individual comparison results, one per row.

pyani.scripts.subcommands.subcmd_classify module

Provides the classify subcommand for pyani.

class pyani.scripts.subcommands.subcmd_classify.**SubgraphData**

Bases: tuple

Subgraph clique/classification output.

cliqueinfo
Alias for field number 2

graph
Alias for field number 1

interval
Alias for field number 0

pyani.scripts.subcommands.subcmd_classify.**subcmd_classify**(*args: argparse.Namespace*) → int

Generate classifications for an analysis.

Parameters **args** – Namespace, command-line arguments

pyani.scripts.subcommands.subcmd_classify.**trimmed_graph_sequence**(*ingraph: networkx.classes.graph.Graph, args: argparse.Namespace, attribute: str = 'identity'*) → Generator[T_co, T_contra, V_co]

Return graphs trimmed from lowest to highest attribute value.

Parameters

- **ingraph** – nx.Graph of genomes as nodes, having edges weighted by the property named in attribute
- **args** – Namespace, parsed command-line arguments
- **attribute** – str, name of the property by which the graph edges should be trimmed

A generator which, starting from the initial graph, yields in sequence a series of graphs from which the edge(s) with the lowest threshold value attribute were removed. The generator returns a tuple of:

(threshold, graph, analyse_cliques(graph))

This will be slow with moderate-large graphs

pyani.scripts.subcommands.subcmd_createdb module

Provides the createdb subcommand for pyani.

pyani.scripts.subcommands.subcmd_createdb.**subcmd_createdb**(*args: argparse.Namespace*) → int

Create an empty pyani database.

Parameters

- **args** – Namespace, command-line arguments
- **logger** – logging object

pyani.scripts.subcommands.subcmd_download module

Provides the download subcommand for pyani.

class pyani.scripts.subcommands.subcmd_download.**Skipped**

Bases: tuple

Convenience struct for holding information about skipped genomes.

accession
Alias for field number 1

dltype
Alias for field number 5

organism
Alias for field number 2

strain
Alias for field number 3

taxon_id
Alias for field number 0

url
Alias for field number 4

pyani.scripts.subcommands.subcmd_download.**configure_entrez**(*args: argparse.Namespace*) → Optional[str]

Configure Entrez email, return API key.

Parameters **args** – Namespace, command-line arguments

Returns None if no API key found

pyani.scripts.subcommands.subcmd_download.**dl_info_to_str**(*esummary*, *uid_class*) → str

Return descriptive string for passed download data.

Parameters

- **esummary** –
- **uid_class** –

pyani.scripts.subcommands.subcmd_download.**download_data**(*args: argparse.Namespace, api_key: Optional[str], asm_dict: Dict[str, List[T]]*) → Tuple[List[T], List[T], List[T]]

Download the accessions indicated in the passed dictionary.

Parameters

- **args** – Namespace of command-line arguments
- **api_key** – str, API key for NCBI downloads
- **asm_dict** – dictionary of assembly UIDs to download, keyed by taxID

Returns lists of information about downloaded genome classes and labels, and a list of skipped downloads (as Skipped objects).

pyani.scripts.subcommands.subcmd_download.**download_genome**(*args: argparse.Namespace*, *filestem: str*, *tid: str*, *uid: str*, *uid_class*)

Download single genome data to output directory.

Parameters

- **args** – Namespace, command-line arguments
- **filestem** – str, output filestem
- **tid** – str, taxonID
- **uid** – str, assembly UID
- **uid_class** –

pyani.scripts.subcommands.subcmd_download.**extract_genomes**(*args: argparse.Namespace*, *dlstatus: pyani.download.DLStatus*, *esummary*) → None

Extract genome files in passed dlstatus.

Parameters

- **args** – Namespace of command-line arguments
- **dlstatus** –
- **esummary** –

pyani.scripts.subcommands.subcmd_download.**get_tax_asm_dict**(*args: argparse.Namespace*) → Dict[str, List[T]]

Return dictionary of assembly UIDs to download, keyed by taxID.

Parameters **args** – Namespace of command-line arguments

pyani.scripts.subcommands.subcmd_download.**hash_genomes**(*args: argparse.Namespace*, *dlstatus: pyani.download.DLStatus*, *filestem: str*, *uid_class*) → Tuple[str, str]

Hash genome files in passed dlstatus.

Parameters

- **args** – Namespace of command-line arguments
- **dlstatus** –
- **filestem** – str, filestem for output
- **uid_class** –

pyani.scripts.subcommands.subcmd_download.**parse_api_key**(*args: argparse.Namespace*) → Optional[str]

Returns NCBI API key if present, None otherwise.

Parameters **args** – Namespace of command-line arguments

Checks for key in args.api_keypath.

pyani.scripts.subcommands.subcmd_download.**subcmd_download**(*args: argparse.Namespace*) → int

Download assembled genomes in subtree of passed NCBI taxon ID.

Parameters **args** – Namespace, command-line arguments

pyani.scripts.subcommands.subcmd_index module

Provides the index subcommand for pyani.

pyani.scripts.subcommands.subcmd_index.**subcmd_index**(*args: argparse.Namespace*) → int

Generate a file with the MD5 hash for each genome in an input directory.

Parameters

- **args** – Namespace, received command-line arguments
- **logger** – logging object

Identify the genome files in the input directory, and generate a single MD5 for each so that <genome>.fna produces <genome>.md5

Genome files (FASTA) are identified from the file extension.

pyani.scripts.subcommands.subcmd_plot module

Provides the plot subcommand for pyani.

pyani.scripts.subcommands.subcmd_plot.**subcmd_plot**(*args: argparse.Namespace*) → int
Produce graphical output for an analysis.

Parameters **args** – Namespace of command-line arguments

This is graphical output for representing the ANI analysis results, and takes the form of a heatmap, or heatmap with dendrogram.

pyani.scripts.subcommands.subcmd_plot.**write_distribution**(*run_id: int, matdata: pyani.pyani_tools.MatrixData, outfmts: List[str], args: argparse.Namespace*) → None
Write distribution plots for each matrix type.

Parameters

- **run_id** – int, run_id for this run
- **matdata** – MatrixData object for this distribution plot
- **args** – Namespace for command-line arguments
- **outfmts** – list of output formats for files

pyani.scripts.subcommands.subcmd_plot.**write_heatmap**(*run_id: int, matdata: pyani.pyani_tools.MatrixData, result_labels: Dict[KT, VT], result_classes: Dict[KT, VT], outfmts: List[str], args: argparse.Namespace*) → None
Write a single heatmap for a pyani run.

Parameters

- **run_id** – int, run_id for this run
- **matdata** – MatrixData object for this heatmap
- **result_labels** – dict of result labels
- **result_classes** – dict of result classes
- **args** – Namespace for command-line arguments
- **outfmts** – list of output formats for files

pyani.scripts.subcommands.subcmd_plot.**write_run_plots**(*run_id: int, session, outfmts: List[str], args: argparse.Namespace*) → None
Write all heatmaps for a specified run to file.

Parameters

- **run_id** – int, run identifier in database session
- **session** – Session, active SQLite session
- **outfmts** – list of output format types
- **args** – Namespace, command line arguments

pyani.scripts.subcommands.subcmd_plot.**write_scatter**(*run_id: int, matdata1: pyani.pyani_tools.MatrixData, matdata2: pyani.pyani_tools.MatrixData, result_labels: Dict[KT, VT], result_classes: Dict[KT, VT], outfmts: List[str], args: argparse.Namespace*) → None

Write a single scatterplot for a pyani run.

Parameters

- **run_id** – int, run_id for this run
- **matdata1** – MatrixData object for this scatterplot
- **matdata2** – MatrixData object for this scatterplot
- **result_labels** – dict of result labels
- **result_classes** – dict of result classes
- **args** – Namespace for command-line arguments
- **outfmts** – list of output formats for files

pyani.scripts.subcommands.subcmd_report module

Provides the report subcommand for pyani.

class pyani.scripts.subcommands.subcmd_report.**ReportParams**

Bases: tuple

Report query/header data.

headers
Alias for field number 2

name
Alias for field number 0

statement
Alias for field number 1

pyani.scripts.subcommands.subcmd_report.**process_formats**(*args: argparse.Namespace*) → List[str]

Return processed list of output formats for writing reports.

Parameters **args** – Namespace of command-line arguments

pyani.scripts.subcommands.subcmd_report.**report**(*args: argparse.Namespace, session, formats: List[str], params: pyani.scripts.subcommands.subcmd_report.ReportParams*) → None

Write tabular report of pyani runs from database.

Parameters

- **args** – Namespace of command-line arguments
- **session** – SQLAlchemy database session
- **formats** – list of output formats

- **params** – ReportParams namedtuple

pyani.scripts.subcommands.subcmd_report.**subcmd_report**(*args: argparse.Namespace*) → int

Present report on ANI results and/or database contents.

Parameters **args** – Namespace, command-line arguments

The report subcommand takes any of several long options that do one of two things:

1. perform a single action.
2. set a parameter/format

These will typically take an output path to a file or directory into which the report will be written (whatever form it takes). By default, text output is written in plain text format, but for some outputs this can be modified by an 'excel' or 'html' format specifier, which writes outputs in that format, where possible.

Submodules

pyani.scripts.average_nucleotide_identity module

Script that calculates ANI measures for a directory of genomes.

This script calculates Average Nucleotide Identity (ANI) according to one of a number of alternative methods described in, e.g.

Richter M, Rossello-Mora R (2009) Shifting the genomic gold standard for the prokaryotic species definition. Proc Natl Acad Sci USA 106: 19126-19131. doi:10.1073/pnas.0906412106. (ANI1020, ANIm, ANIb)

Goris J, Konstantinidis KT, Klappenbach JA, Coenye T, Vandamme P, et al. (2007) DNA-DNA hybridization values and their relationship to whole-genome sequence similarities. Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0.

ANI is proposed to be the appropriate in silico substitute for DNA-DNA hybridisation (DDH), and so useful for delineating species boundaries. A typical percentage threshold for species boundary in the literature is 95% ANI (e.g. Richter et al. 2009).

All ANI methods follow the basic algorithm:

- Align the genome of organism 1 against that of organism 2, and identify the matching regions
- Calculate the percentage nucleotide identity of the matching regions, as an average for all matching regions

Methods differ on: (1) what alignment algorithm is used, and the choice of parameters (this affects the aligned region boundaries); (2) what the input is for alignment (typically either fragments of fixed size, or the most complete assembly available); (3) whether a reciprocal comparison is necessary or desirable.

ANIm: uses MUMmer (NUCmer) to align the input sequences. ANIb: uses BLASTN to align 1000nt fragments of the input sequences TETRA: calculates tetranucleotide frequencies of each input sequence

This script takes as main input a directory containing a set of correctly-formatted FASTA multiple sequence files. All sequences for a single organism should be contained in only one sequence file. The names of these files are used for identification, so it would be advisable to name them sensibly.

Output is written to a named directory. The output files differ depending on the chosen ANI method.

ANIm: MUMmer/NUCmer .delta files, describing the sequence alignment; tab-separated format plain text tables describing total alignment lengths, and total alignment percentage identity

ANIb: FASTA sequences describing 1000nt fragments of each input sequence; BLAST nucleotide databases - one for each set of fragments; and BLASTN output files (tab-separated tabular format plain text) - one for each pairwise comparison of input sequences. There are potentially a lot of intermediate files.

TETRA: Tab-separated text file describing the Z-scores for each tetranucleotide in each input sequence.

In addition, all methods produce a table of output percentage identity (ANIm and ANIb) or correlation (TETRA), between each sequence.

If graphical output is chosen, the output directory will also contain PDF files representing the similarity between sequences as a heatmap with row and column dendrograms.

DEPENDENCIES

o Biopython (http://www.biopython.org)

o BLAST+ executable in the $PATH, or available on the command line (ANIb) (ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/)

o MUMmer executables in the $PATH, or available on the command line (ANIm) (http://mummer.sourceforge.net/)

For graphical output

o R with shared libraries installed on the system, for graphical output (http://cran.r-project.org/)

o Rpy2 (http://rpy.sourceforge.net/rpy2.html)

pyani.scripts.average_nucleotide_identity.**calculate_anim**(*args: argparse.Namespace, infiles: List[pathlib.Path], org_lengths: Dict[KT, VT]*) → pyani.pyani_tools.ANIResults

Return ANIm result dataframes for files in input directory.

Parameters

- **args** – Namespace, command-line arguments
- **logger** – logging object
- **infiles** – list of paths to each input file
- **org_lengths** – dict, input sequence lengths, keyed by sequence

Finds ANI by the ANIm method, as described in Richter et al (2009) Proc Natl Acad Sci USA 106: 19126-19131 doi:10.1073/pnas.0906412106.

All FASTA format files (selected by suffix) in the input directory are compared against each other, pairwise, using NUCmer (which must be in the path). NUCmer output is stored in the output directory.

The NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment between the two organisms, as represented by the sequences in the FASTA files.

These are processed to give matrices of aligned sequence lengths, average nucleotide identity (ANI) percentages, coverage (aligned percentage of whole genome), and similarity error cound for each pairwise comparison.

pyani.scripts.average_nucleotide_identity.**calculate_tetra**(*infiles: List[pathlib.Path]*) → pandas.core.frame.DataFrame

Calculate TETRA for files in input directory.

Parameters

- **logger** – logging object
- **infiles** – list, paths to each input file

Calculates TETRA correlation scores, as described in:

Richter M, Rossello-Mora R (2009) Shifting the genomic gold standard for the prokaryotic species definition. Proc Natl Acad Sci USA 106: 19126-19131. doi:10.1073/pnas.0906412106.

and

Teeling et al. (2004) Application of tetranucleotide frequencies for the assignment of genomic fragments. Env. Microbiol. 6(9): 938-947. doi:10.1111/j.1462-2920.2004.00624.x

pyani.scripts.average_nucleotide_identity.**compress_delete_outdir**(*outdir: pathlib.Path*, *logger: logging.Logger*) → None

Compress the contents of the passed directory to .tar.gz and delete.

pyani.scripts.average_nucleotide_identity.**draw**(*args: argparse.Namespace, filestems: List[str], gformat: str*) → None

Draw ANIb/ANIm/TETRA results.

Parameters

- **args** – Namespace, command-line arguments
- **logger** – logging object
- **filestems** –
  - filestems for output files
- **gformat** –
  - the format for output graphics

pyani.scripts.average_nucleotide_identity.**get_method**(*args: argparse.Namespace*) → Tuple

Return function and config for the chosen method.

Parameters

- **args** – Namespace of command-line arguments
- **logger** – logging object

The dictionary defines pairs of method function and configurations, keyed by method name.

pyani.scripts.average_nucleotide_identity.**last_exception**() → str

Return last exception as a string, or use in logging.

pyani.scripts.average_nucleotide_identity.**make_outdirs**(*args: argparse.Namespace*)

Make the output directory, if required.

Parameters

- **args** – Namespace of command-line options
- **logger** – logging object

If the output directory already exists and args.force is not set True, stop with an error.

If args.force is set... If args.noclobber is not set True, delete the output directory tree If args.noclobber is set True, use the existing output directory, and keep any existing output

pyani.scripts.average_nucleotide_identity.**make_sequence_fragments**(*args: argparse.Namespace, logger: logging.Logger, infiles: List[pathlib.Path], blastdir: pathlib.Path*) → Tuple[List[T], Dict[KT, VT]]

Return tuple of fragment files, and fragment sizes.

Parameters

- **args** – Namespace of command-line arguments
- **logger** – logging object
- **infiles** – iterable of sequence files to fragment
- **blastdir** – path of directory to place BLASTN databases of fragments

Splits input FASTA sequence files into the fragments (a requirement for ANIb methods), and writes BLAST databases of these fragments, and fragment lengths of sequences, to local files.

pyani.scripts.average_nucleotide_identity.**parse_cmdline**(*argv: Optional[List[T]] = None*) → argparse.Namespace

Parse command-line arguments for script.

Parameters **argv** – list of arguments from command-line

pyani.scripts.average_nucleotide_identity.**process_arguments**(*args: Optional[argparse.Namespace]*) → argparse.Namespace

Process command-line arguments.

Parameters **args** – Namespace of command-line arguments

Either returns parsed arguments or - if only the script name is used, shows the version and exits.

pyani.scripts.average_nucleotide_identity.**run_blast**(*args: argparse.Namespace, logger: logging.Logger, infiles: List[pathlib.Path], blastdir: pathlib.Path*) → Tuple

Run BLAST commands for ANIb methods.

Parameters

- **args** – Namespace of command-line options

- **logger** – logging object
- **infiles** – iterable of sequence files to compare
- **blastdir** – path of directory to fragment BLASTN databases

Runs BLAST database creation and comparisons, returning the cumulative return values of the BLAST tool subprocesses, and the fragment sizes for each input file

pyani.scripts.average_nucleotide_identity.**run_main**(*argsin: Optional[argparse.Namespace] = None*) → int

Run main process for average_nucleotide_identity.py script.

Parameters

- **argsin** – Namespace, command-line arguments
- **logger** – logging object

pyani.scripts.average_nucleotide_identity.**subsample_input**(*args: argparse.Namespace, logger: logging.Logger, infiles: List[pathlib.Path]*) → List[pathlib.Path]

Return a random subsample of the passed input files.

Parameters

- **args** – Namespace, command-line arguments
- **logger** – logging object
- **infiles** – list of input files for analysis

pyani.scripts.average_nucleotide_identity.**test_class_label_paths**(*args: argparse.Namespace, logger: logging.Logger*) → None

Raise error and exit if label and class files exist.

Parameters

- **args** – Namespace of command-line arguments
- **logger** – logging object

Exits if class and label files are not found

pyani.scripts.average_nucleotide_identity.**test_scheduler**(*args: argparse.Namespace, logger: logging.Logger*) → None

Test if the specified scheduler can be used.

Parameters

- **args** – Namespace of command-line arguments
- **logger** – logging object

Exits if the scheduler is invalid

pyani.scripts.average_nucleotide_identity.**unified_anib**(*args: argparse.Namespace, infiles: List[pathlib.Path], org_lengths: Dict[str, int]*) → pyani.pyani_tools.ANIResults

Calculate ANIb for files in input directory.

Parameters

- **args** – Namespace of command-line options
- **logger** – logging object
- **infiles** – iterable of paths to each input file
- **org_lengths** – dict of input sequence lengths keyed by sequence name

Calculates ANI by the ANIb method, as described in Goris et al. (2007) Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0. There are some minor differences depending on whether BLAST+ or legacy BLAST (BLASTALL) methods are used.

All FASTA format files (selected by suffix) in the input directory are used to construct BLAST databases, placed in the output directory. Each file's contents are also split into sequence fragments of length options.fragsize, and the multiple FASTA file that results written to the output directory. These are BLASTNed, pairwise, against the databases.

The BLAST output is interrogated for all fragment matches that cover at least 70% of the query sequence, with at least 30% nucleotide identity over the full length of the query sequence. This is an odd choice and doesn't correspond to the twilight zone limit as implied by Goris et al. We persist with their definition, however. Only these qualifying matches contribute to the total aligned length, and total aligned sequence identity used to calculate ANI.

The results are processed to give matrices of aligned sequence length (aln_lengths.tab), similarity error counts (sim_errors.tab), ANIs (perc_ids.tab), and minimum aligned percentage (perc_aln.tab) of each genome, for each pairwise comparison. These are written to the output directory in plain text tab-separated format.

pyani.scripts.average_nucleotide_identity.**write**(*args: argparse.Namespace, results: pandas.core.frame.DataFrame*) → None

Write ANIb/ANIm/TETRA results to output directory.

Parameters

- **args** – Namespace, command-line arguments
- **logger** – logging object
- **results** – Results object from analysis

Each dataframe is written to an Excel-format file (if args.write_excel is True), and plain text tab-separated file in the output directory. The order of result output must be reflected in the order of filestems.

pyani.scripts.delta_filter_wrapper module

Wrapper for MUMmer 3.23 delta-filter script.

It is required in order to catch STDOUT ahead of the SGE job runner so that pyani can run on SGE/OGE scheduling systems.

The wrapper does not modify the output of delta-filter, and is called in exactly the same way, passing arguments through directly. The only departure from this is that the first argument is the path to delta-filter, and the final argument denotes the output filtered delta file path, so that redirection (which no longer works with SGE/OGE) is not necessary.

For example, the delta-filter command

delta-filter [options] <delta file> > <filtered delta file>

becomes

delta_filter_wrapper.py delta-filter [options] <delta file> <filtered delta file>

This wrapper is not very robust, but will be improved in later versions of pyani.

pyani.scripts.delta_filter_wrapper.**run_main**() → int
Run main process for delta_filter_wrapper.py.

pyani.scripts.genbank_get_genomes_by_taxon module

Script to download from NCBI all genomes in a specified taxon subtree.

This script takes an NCBI taxonomy identifier (or string, though this is not always reliable for taxonomy tree subgraphs...) and downloads all genomes it can find from NCBI in the corresponding taxon subgraph that has the passed argument as root.

exception pyani.scripts.genbank_get_genomes_by_taxon.**NCBIDownloadException**
Bases: Exception

General exception for failed NCBI download.

pyani.scripts.genbank_get_genomes_by_taxon.**entrez_batch_webhistory**(*args*, *record*, *expected*, *batchsize*, **fnargs*, ***fnkwargs*)
Recover Entrez data from a prior NCBI webhistory search.

Parameters

- **args** – Namespace, command-line arguments
- **record** – Entrez webhistory record
- **expected** – int, number of expected search returns
- **batchsize** – int, number of search returns to retrieve in each batch
- ***fnargs** – tuple, arguments to Efetch
- ****fnkwargs** – dict, keyword arguments to Efetch

Recovery is performed in in batches of defined size, using Efetch. Returns all results as a list.

pyani.scripts.genbank_get_genomes_by_taxon.**entrez_retry**(*args*, *func*, **fnargs*, ***fnkwargs*)
Retry the passed function a defined number of times.

Parameters

- **args** – Namespace, command-line arguments
- **func** – func, Entrez function to attempt
- ***fnargs** – tuple, arguments to the Entrez function
- ****fnkwargs** – dict, keyword arguments to the Entrez function

pyani.scripts.genbank_get_genomes_by_taxon.**extract_archive**(*archivepath*)
Return path to extracted gzip file.

Parameters **archivepath** – Path, path to gzipped file with ".tar.gz" suffix

pyani.scripts.genbank_get_genomes_by_taxon.**extract_filestem**(*data*)
Extract filestem from Entrez eSummary data.

Parameters **data** – Entrez eSummary

Function expects esummary['DocumentSummarySet']['DocumentSummary'][0]

Some illegal characters may occur in AssemblyName - for these, a more robust regex replace/escape may be required. Sadly, NCBI don't just use standard percent escapes, but instead replace certain characters with underscores: white space, slash, comma, hash, brackets.

pyani.scripts.genbank_get_genomes_by_taxon.**get_asm_uids**(*args*, *taxon_uid*)
Return a set of NCBI UIDs associated with the passed taxon.

Parameters

- **args** – Namespace, command-line arguments
- **taxon_uid** – str, NCBI taxon ID

This query at NCBI returns all assemblies for the taxon subtree rooted at the passed taxon_uid.

pyani.scripts.genbank_get_genomes_by_taxon.**get_ncbi_asm**(*args*, *asm_uid*, *fmt='fasta'*)
Return the NCBI AssemblyAccession and AssemblyName for an assembly.

Parameters

- **args** – Namespace, command-line arguments
- **asm_uid** – NCBI assembly UID
- **fmt** – str, format to retrieve assembly information

Returns organism data for class/label files also, as well as accession, so we can track whether downloads fail because only the most recent version is available..

AssemblyAccession and AssemblyName are data fields in the eSummary record, and correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GC{[}AF{]}/nnn/nnn/nnn/<AA>_<AN> where <AA> is AssemblyAccession, and <AN> is AssemblyName, and the choice of GCA vs GCF, and the three values of nnn are taken from <AA>

pyani.scripts.genbank_get_genomes_by_taxon.**last_exception**()
Return last exception as a string, or use in logging.

pyani.scripts.genbank_get_genomes_by_taxon.**logreport_downloaded**(*accn*, *skiplist*, *accndict*, *uidaccndict*)
Report to logger if alternative assemblies were downloaded.

Parameters

- **accn** –
- **skiplist** –
- **accndict** –
- **uidaccndict** –

pyani.scripts.genbank_get_genomes_by_taxon.**make_outdir**(*args: argparse.Namespace*) → None

Make the output directory, if required.

Parameters **args** – Namespace, command-line arguments

This is a little involved. If the output directory already exists, we take the safe option by default, and stop with an error. We can, however, choose to force the program to go on, in which case we can either clobber the existing directory, or not. The options turn out as the following, if the directory exists: DEFAULT: stop and report the collision FORCE: continue, and remove the existing output directory NOCLOBBER+FORCE: continue, but do not remove the existing output

pyani.scripts.genbank_get_genomes_by_taxon.**parse_cmdline**(*argv=None*)

Parse command-line arguments.

Parameters **argv** – list of command-line arguments

pyani.scripts.genbank_get_genomes_by_taxon.**retrieve_asm_contigs**(*args*, *filestem*, *ftpstem='ftp://ftp.ncbi.nlm.nih.gov/genom*
fmt='fasta')

Download assembly sequence to a local directory.

Parameters

- **args** – Namespace, command-line arguments
- **filestem** – str, filestem for output file
- **ftpstem** – str, URI stem for NCBI FTP site
- **fmt** – str, format for output file

The filestem corresponds to <AA>_<AN>, where <AA> and <AN> are AssemblyAccession and AssemblyName: data fields in the eSummary record. These correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GC{[}AF{]}/nnn/nnn/nnn/<AA>_<AN>/ where <AA> is AssemblyAccession, and <AN> is AssemblyName. The choice of GCA vs GCF, and the values of nnn, are derived from <AA>

The files in this directory all have the stem <AA>_<AN>_<suffix>, where suffixes are: assembly_report.txt assembly_stats.txt feature_table.txt.gz genomic.fna.gz genomic.gbff.gz genomic.gff.gz protein.faa.gz protein.gpff.gz rm_out.gz rm.run wgsmaster.gbff.gz

This function downloads the genomic_fna.gz file, and extracts it in the output directory name specified when the script is called.

pyani.scripts.genbank_get_genomes_by_taxon.**run_main**(*args=None*)

Run main process for average_nucleotide_identity.py script.

Parameters **args** – Namespace, command-line arguments

pyani.scripts.genbank_get_genomes_by_taxon.**set_ncbi_email**(*args: argparse.Namespace*) → None

Set contact email for NCBI.

Parameters **args** – Namespace, command-line arguments

pyani.scripts.genbank_get_genomes_by_taxon.**write_contigs**(*args*, *asm_uid*, *contig_uids*, *batchsize=10000*)

Write assembly contigs to a single FASTA file.

Parameters

- **args** – Namespace, command-line arguments
- **asm_uid** – str, NCBI assembly UID
- **contig_uids** –
- **batchsize** – int

FASTA records are returned, as GenBank and even GenBankWithParts format records don't reliably give correct sequence in all cases.

The script returns two strings for each assembly, a 'class' and a 'label' string - this is for use with, e.g. pyani.

pyani.scripts.logger module

pyani.scripts.pyani_script module

Implements the pyani script for classifying prokaryotic genomes.

pyani.scripts.pyani_script.**add_log_headers**()
Add headers to log output.

pyani.scripts.pyani_script.**run_main**(*argv: Optional[List[str]] = None*) → int
Run main process for pyani.py script.

Parameters **argv** –

pyani.scripts.tools module

5.13.2 Submodules

pyani.anib module

Code to implement the ANIb average nucleotide identity method.

Calculates ANI by the ANIb method, as described in Goris et al. (2007) Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0.

From Goris et al.

'"The genomic sequence from one of the genomes in a pair (the query) was cut into consecutive 1020 nt fragments. The 1020 nt cut-off was used to correspond with the fragmentation of the genomic DNA to approximately 1 kb fragments during the DDH experiments. [...] The 1020 nt fragments were then used to search against the whole genomic sequence of the other genome in the pair (the reference) by using the BLASTN algorithm; the best BLASTN match was saved for further analysis. The BLAST algorithm was run using the following settings: X=150 (where X is the drop-off value for gapped alignment), q=-1 (where q is the penalty for nucleotide mismatch) and F=F (where F is the filter for repeated sequences); the rest of the parameters were used at the default settings. These settings give better sensitivity than the default settings when more distantly related genomes are being compared, as the latter target sequences that are more similar to each other. [...] The ANI between the query genome and the reference genome was calculated as the mean identity of all BLASTN matches that showed more than 30% overall sequence identity (recalculated to an identity along the entire sequence) over an alignable region of at least 70% of their length. This cut-off is above the 'twilight zone' of similarity searches in which an inference of homology is error prone because of low levels of Reverse searching, i.e. in which the reference genome is used as the query, was also performed to provide reciprocal values."'

All input FASTA format files are used to construct BLAST databases. Each file's contents are also split into sequence fragments of length options.fragsize, and the multiple FASTA file that results written to the output directory. These are BLASTNed, pairwise, against the databases.

BLAST output is interrogated for all fragment matches that cover at least 70% of the query sequence, with at least 30% nucleotide identity over the full length of the query sequence. This is an odd choice and doesn't correspond to the twilight zone limit as implied by Goris et al. We persist with their definition, however. Only these qualifying matches contribute to the total aligned length, and total aligned sequence identity used to calculate ANI.

exception pyani.anib.**PyaniANIbException**
Bases: *pyani.PyaniException*

ANIb-specific exception for pyani.

pyani.anib.**build_db_jobs**(*infiles: List[pathlib.Path], blastcmds: pyani.pyani_tools.BLASTcmds*) → Dict[KT, VT]
Return dictionary of db-building commands, keyed by dbname.

Parameters

- **infiles** –
- **blastcmds** –

pyani.anib.**construct_blastall_cmdline**(*fname1: pathlib.Path, fname2: pathlib.Path, outdir: pathlib.Path, blastall_exe: pathlib.Path = PosixPath('blastall')*) → str
Return single blastall command.

Parameters

- **fname1** –
- **fname2** –
- **outdir** –
- **blastall_exe** – str, path to BLASTALL executable

pyani.anib.**construct_blastn_cmdline**(*fname1: pathlib.Path, fname2: pathlib.Path, outdir: pathlib.Path, blastn_exe: pathlib.Path = PosixPath('blastn')*) → str
Return a single blastn command.

Parameters

- **fname1** –
- **fname2** –
- **outdir** –
- **blastn_exe** – str, path to blastn executable

pyani.anib.**construct_formatdb_cmd**(*filename: pathlib.Path, outdir: pathlib.Path, blastdb_exe: pathlib.Path = PosixPath('formatdb')*) → Tuple[str, pathlib.Path]
Return formatdb command and path to output file.

Parameters

- **filename** – Path, input filename
- **outdir** – Path, path to output directory
- **blastdb_exe** – Path, path to the formatdb executable

pyani.anib.**construct_makeblastdb_cmd**(*filename: pathlib.Path, outdir: pathlib.Path, blastdb_exe: pathlib.Path = PosixPath('makeblastdb')*) → Tuple[str, pathlib.Path]

Return makeblastdb command and path to output file.

Parameters

- **filename** – Path, input filename
- **outdir** – Path, directory for output
- **blastdb_exe** – Path, path to the makeblastdb executable

pyani.anib.**fragment_fasta_files**(*infiles: List[pathlib.Path], outdirname: pathlib.Path, fragsize: int*) → Tuple[List[T], Dict[KT, VT]]

Chop sequences of the passed files into fragments, return filenames.

Parameters

- **infiles** – collection of paths to each input sequence file
- **outdirname** – Path, path to output directory
- **fragsize** – Int, the size of sequence fragments

Takes every sequence from every file in infiles, and splits them into consecutive fragments of length fragsize, (with any trailing sequences being included, even if shorter than fragsize), writing the resulting set of sequences to a file with the same name in the specified output directory.

All fragments are named consecutively and uniquely (within a file) as fragNNNNN. Sequence description fields are retained.

Returns a tuple `(filenames, fragment_lengths)` where `filenames` is a list of paths to the fragment sequence files, and `fragment_lengths` is a dictionary of sequence fragment lengths, keyed by the sequence files, with values being a dictionary of fragment lengths, keyed by fragment IDs.

pyani.anib.**generate_blastdb_commands**(*filenames: List[pathlib.Path], outdir: pathlib.Path, blastdb_exe: Optional[pathlib.Path] = None, mode: str = 'ANIb'*) → List[Tuple[str, pathlib.Path]]

Return list of makeblastdb command-lines for ANIb/ANIblastall.

Parameters

- **filenames** – a list of paths to input FASTA files
- **outdir** – path to output directory
- **blastdb_exe** – path to the makeblastdb executable
- **mode** – str, ANIb analysis type (ANIb or ANIblastall)

pyani.anib.**generate_blastn_commands**(*filenames: List[pathlib.Path], outdir: pathlib.Path, blast_exe: Optional[pathlib.Path] = None, mode: str = 'ANIb'*) → List[str]

Return a list of blastn command-lines for ANIm.

Parameters

- **filenames** – a list of paths to fragmented input FASTA files
- **outdir** – path to output directory
- **blastn_exe** – path to BLASTN executable
- **mode** – str, analysis type (ANIb or ANIblastall)

Assumes that the fragment sequence input filenames have the form ACCESSION-fragments.ext, where the corresponding BLAST database filenames have the form ACCESSION.ext. This is the convention followed by the fragment_FASTA_files() function above.

pyani.anib.**get_fraglength_dict**(*fastafiles: List[pathlib.Path]*) → Dict[KT, VT]
Return dictionary of sequence fragment lengths, keyed by query name.

Parameters **fastafiles** – list of paths to FASTA input whole sequence files

Loops over input files and, for each, produces a dictionary with fragment lengths, keyed by sequence ID. These are returned as a dictionary with the keys being query IDs derived from filenames.

pyani.anib.**get_fragment_lengths**(*fastafile: pathlib.Path*) → Dict[KT, VT]
Return dictionary of sequence fragment lengths, keyed by fragment ID.

Parameters **fastafile** –

Biopython's SeqIO module is used to parse all sequences in the FASTA file.

NOTE: ambiguity symbols are not discounted.

pyani.anib.**get_version**(*blast_exe: pathlib.Path = PosixPath('blastn')*) → str
Return BLAST+ blastn version as a string.

Parameters **blast_exe** – path to blastn executable

We expect blastn to return a string as, for example

```
$ blastn -version
blastn: 2.9.0+
Package: blast 2.9.0, build Jun 10 2019 09:40:53
```

This is concatenated with the OS name.

The following circumstances are explicitly reported as strings

- no executable at passed path
- non-executable file at passed path (this includes cases where the user doesn't have execute permissions on the file)
- no version info returned

pyani.anib.**make_blastcmd_builder**(*mode: str*, *outdir: pathlib.Path*, *format_exe: Optional[pathlib.Path] = None*, *blast_exe: Optional[pathlib.Path] = None*, *prefix: str = 'ANIBLAST'*) → pyani.pyani_tools.BLASTcmds
Return BLASTcmds object for construction of BLAST commands.

Parameters

- **mode** – str, the kind of ANIb analysis (ANIb or ANIblastall)
- **outdir** –
- **format_exe** –
- **blast_exe** –
- **prefix** –

pyani.anib.**make_job_graph**(*infiles: List[pathlib.Path]*, *fragfiles: List[pathlib.Path]*, *blastcmds: pyani.pyani_tools.BLASTcmds*) → List[pyani.pyani_jobs.Job]
Return job dependency graph, based on the passed input sequence files.

Parameters

- **infiles** – list of paths to input FASTA files
- **fragfiles** – list of paths to fragmented input FASTA files
- **blastcmds** –

By default, will run ANIb - it *is* possible to make a mess of passing the wrong executable for the mode you're using.

All items in the returned graph list are BLAST executable jobs that must be run *after* the corresponding database creation. The Job objects corresponding to the database creation are contained as dependencies. How those jobs are scheduled depends on the scheduler (see run_multiprocessing.py, run_sge.py)

pyani.anib.**parse_blast_tab**(*filename: pathlib.Path, fraglengths: Dict[KT, VT], mode: str = 'ANIb'*) → Tuple[int, int, int]

Return (alignment length, similarity errors, mean_pid) tuple.

Parameters

- **filename** – Path, path to .blast_tab file
- **fraglengths** – Optional[Dict], dictionary of fragment lengths for each genome.
- **mode** – str, analysis type (ANIb or ANIblastall)

Calculate the alignment length and total number of similarity errors (as we would with ANIm), as well as the Goris et al.-defined mean identity of all valid BLAST matches for the passed BLASTALL alignment .blast_tab file.

'"ANI between the query genome and the reference genome was calculated as the mean identity of all BLASTN matches that showed more than 30% overall sequence identity (recalculated to an identity along the entire sequence) over an alignable region of at least 70% of their length. '"

pyani.anib.**process_blast**(*blast_dir: pathlib.Path, org_lengths: Dict[KT, VT], fraglengths: Dict[KT, VT], mode: str = 'ANIb', logger: Optional[logging.Logger] = None*) → pyani.pyani_tools.ANIResults

Return tuple of ANIb results for .blast_tab files in the output dir.

Parameters

- **blast_dir** – Path, path to the directory containing .blast_tab files
- **org_lengths** – Dict, the base count for each input sequence
- **fraglengths** – dictionary of query sequence fragment lengths, only needed for BLASTALL output
- **mode** – str, analysis type (ANIb or ANIblastall)
- **logger** – a logger for messages

Returns the following pandas dataframes in an ANIResults object; query sequences are rows, subject sequences are columns:

- alignment_lengths - non-symmetrical: total length of alignment
- percentage_identity - non-symmetrical: ANIb (Goris) percentage identity
- alignment_coverage - non-symmetrical: coverage of query
- similarity_errors - non-symmetrical: count of similarity errors

May throw a ZeroDivisionError if one or more BLAST runs failed, or a very distant sequence was included in the analysis.

pyani.anim module

Code to implement the ANIm average nucleotide identity method.

Calculates ANI by the ANIm method, as described in Richter et al (2009) Proc Natl Acad Sci USA 106: 19126-19131 doi:10.1073/pnas.0906412106.

All input FASTA format files are compared against each other, pairwise, using NUCmer (binary location must be provided). NUCmer output will be stored in a specified output directory.

The NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment. These are processed to give matrices of aligned sequence lengths, similarity error counts, average nucleotide identity (ANI) percentages, and minimum aligned percentage (of whole genome) for each pairwise comparison.

exception pyani.anim.**PyaniANImException**

Bases: *pyani.PyaniException*

ANIm-specific exception for pyani.

pyani.anim.**construct_nucmer_cmdline**(*fname1: pathlib.Path*, *fname2: pathlib.Path*, *outdir: pathlib.Path = PosixPath('.')*, *nucmer_exe: pathlib.Path = PosixPath('nucmer')*, *filter_exe: pathlib.Path = PosixPath('delta-filter')*, *maxmatch: bool = False*) → Tuple[str, str]

Return a tuple of corresponding NUCmer and delta-filter commands.

Parameters

- **fname1** – path to query FASTA file
- **fname2** – path to subject FASTA file
- **outdir** – path to output directory
- **nucmer_exe** –
- **filter_exe** –
- **maxmatch** – Boolean flag indicating whether to use NUCmer's -maxmatch

option. If not, the -mum option is used instead

The split into a tuple was made necessary by changes to SGE/OGE. The delta-filter command must now be run as a dependency of the NUCmer command, and be wrapped in a Python script to capture STDOUT.

NOTE: This command-line writes output data to a subdirectory of the passed outdir, called "nucmer_output".

pyani.anim.**generate_nucmer_commands**(*filenames: List[pathlib.Path]*, *outdir: pathlib.Path = PosixPath('.')*, *nucmer_exe: pathlib.Path = PosixPath('nucmer')*, *filter_exe: pathlib.Path = PosixPath('delta-filter')*, *maxmatch: bool = False*) → Tuple[List[T], List[T]]

Return list of NUCmer command-lines for ANIm.

Parameters

- **filenames** – a list of paths to input FASTA files
- **outdir** – path to output directory
- **nucmer_exe** – location of the nucmer binary
- **maxmatch** – Boolean flag indicating to use NUCmer's -maxmatch option

The first element returned is a list of NUCmer commands, and the second a corresponding list of delta_filter_wrapper.py commands. The NUCmer commands should each be run before the corresponding delta-filter command.

TODO: This return value needs to be reworked as a collection.

Loop over all FASTA files generating NUCmer command lines for each pairwise comparison.

pyani.anim.**generate_nucmer_jobs**(*filenames: List[pathlib.Path], outdir: pathlib.Path = PosixPath('.'), nucmer_exe: pathlib.Path = PosixPath('nucmer'), filter_exe: pathlib.Path = PosixPath('delta-filter'), maxmatch: bool = False, jobprefix: str = 'ANINUCmer'*)

Return list of Jobs describing NUCmer command-lines for ANIm.

Parameters

- **filenames** – Iterable, Paths to input FASTA files
- **outdir** – str, path to output directory
- **nucmer_exe** – str, location of the nucmer binary
- **filter_exe** –
- **maxmatch** – Boolean flag indicating to use NUCmer's -maxmatch option
- **jobprefix** –

Loop over all FASTA files, generating Jobs describing NUCmer command lines for each pairwise comparison.

pyani.anim.**get_fasta_files**(*dirname: pathlib.Path = PosixPath('.')*) → Iterable[T_co]

Return iterable of FASTA files in the passed directory.

Parameters **dirname** – str, path to input directory

pyani.anim.**get_version**(*nucmer_exe: pathlib.Path = PosixPath('nucmer')*) → str

Return NUCmer package version as a string.

Parameters **nucmer_exe** – path to NUCmer executable

We expect NUCmer to return a string on STDERR as

```
$ nucmer
NUCmer (NUCleotide MUMmer) version 3.1
```

we concatenate this with the OS name.

The following circumstances are explicitly reported as strings

- no executable at passed path
- non-executable file at passed path (this includes cases where the user doesn't have execute permissions on the file)
- no version info returned

pyani.anim.**parse_delta**(*filename: pathlib.Path*) → Tuple[int, int]

Return (alignment length, similarity errors) tuple from passed .delta.

Parameters **filename** – Path, path to the input .delta file

Extracts the aligned length and number of similarity errors for each aligned uniquely-matched region, and returns the cumulative total for each as a tuple.

Similarity errors are defined in the .delta file spec (see below) as non-positive match scores. For NUCmer output, this is identical to the number of errors (non-identities and indels).

Delta file format has seven numbers in the lines of interest: see http://mummer.sourceforge.net/manual/ for specification

- start on query
- end on query
- start on target
- end on target
- error count (non-identical, plus indels)
- **similarity errors (non-positive match scores)** [NOTE: with PROmer this is equal to error count]
- stop codons (always zero for nucmer)

To calculate alignment length, we take the length of the aligned region of the reference (no gaps), and process the delta information. This takes the form of one value per line, following the header sequence. Positive values indicate an insertion in the reference; negative values a deletion in the reference (i.e. an insertion in the query). The total length of the alignment is then:

reference_length + insertions - deletions

For example:

A = ABCDACBDCAC$ B = BCCDACDCAC$ Delta = (1, -3, 4, 0) A = ABC.DACBDCAC$ B = .BCC-DAC.DCAC$

A is the reference and has length 11. There are two insertions (positive delta), and one deletion (negative delta). Alignment length is then 11 + 1 = 12.

pyani.anim.**process_deltadir**(*delta_dir: pathlib.Path, org_lengths: Dict[KT, VT], logger: Optional[logging.Logger] = None*) → pyani.pyani_tools.ANIResults

Return tuple of ANIm results for .deltas in passed directory.

Parameters

- **delta_dir** – Path, path to the directory containing .delta files
- **org_lengths** – dictionary of total sequence lengths, keyed by sequence

Returns the following pandas dataframes in an ANIResults object; query sequences are rows, subject sequences are columns:

- alignment_lengths - symmetrical: total length of alignment
- percentage_identity - symmetrical: percentage identity of alignment
- alignment_coverage - non-symmetrical: coverage of query and subject
- similarity_errors - symmetrical: count of similarity errors

May throw a ZeroDivisionError if one or more NUCmer runs failed, or a very distant sequence was included in the analysis.

pyani.blast module

Code for handling BLAST output files.

pyani.blast.**parse_blasttab**(*fhandle: TextIO*) → List[List[str]]

Return the passed BLAST tab output file as a list of lists.

Parameters **fhandle** – TextIO, filehandle containing BLAST output file

This is used when testing for conserved BLAST output, as the exact format of the BLAST result can depend on the software version. For instance, the locally-installed version may be BLASTN+ 2.6.0, which reports match identity to 3sf, and the version in CI may be BLASTN+ 2.2.28, which reports to 2sf.

Returning a list of lines, parsed into the appropriate data type, allows for direct comparison of line content independent of formatting.

pyani.download module

Module providing functions useful for downloading genomes from NCBI.

class pyani.download.**ASMIDs**
Bases: tuple

Matching Assembly ID information for a query taxID.

asm_ids
Alias for field number 2

query
Alias for field number 0

result_count
Alias for field number 1

class pyani.download.**Classification**
Bases: tuple

Taxonomic classification for an isolate.

genus
Alias for field number 1

organism
Alias for field number 0

species
Alias for field number 2

strain
Alias for field number 3

class pyani.download.**DLFileData**
Bases: tuple

Convenience struct for file download data.

filestem
Alias for field number 0

ftpstem
Alias for field number 1

suffix
Alias for field number 2

class pyani.download.**DLStatus**(*url: str*, *hashurl: str*, *outfname: pathlib.Path*, *outfhash: pathlib.Path*, *skipped: bool*, *error: Optional[str] = None*)
Bases: object

Download status data.

exception pyani.download.**FileExistsException**(*msg: str = 'Specified file exists'*)
Bases: Exception

A specified file exists.

class pyani.download.**Hashstatus**
Bases: tuple

Status report on file hash comparison.

filehash
Alias for field number 2

localhash
Alias for field number 1

passed
Alias for field number 0

exception pyani.download.**NCBIDownloadException**(*msg: str = 'Error downloading file from NCBI'*)
Bases: Exception

General exception for failed NCBI download.

exception pyani.download.**PyaniIndexException**
Bases: Exception

General exception for indexing with pyani

pyani.download.**check_hash**(*fname: pathlib.Path*, *hashfile: pathlib.Path*) → pyani.download.Hashstatus
Check MD5 of passed file against downloaded NCBI hash file.

Parameters

- **fname** – Path, path to local hash file
- **hashfile** – Path, path to NCBI hash file

pyani.download.**compile_url**(*filestem: str*, *suffix: str*, *ftpstem: str*) → Tuple[str, str]
Compile download URLs given a passed filestem.

Parameters

- **filestem** –
- **suffix** –
- **ftpstem** –

The filestem corresponds to <AA>_<AN>, where <AA> and <AN> are AssemblyAccession and AssemblyName: data fields in the eSummary record. These correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GC{[}AF{]}/nnn/nnn/nnn/<AA>_<AN>/ where <AA> is AssemblyAccession, and <AN> is AssemblyName. The choice of GCA vs GCF, and the values of nnn, are derived from <AA>

The files in this directory all have the stem <AA>_<AN>_<suffix>, where suffixes are: assembly_report.txt assembly_stats.txt feature_table.txt.gz genomic.fna.gz genomic.gbff.gz genomic.gff.gz protein.faa.gz protein.gpff.gz rm_out.gz rm.run wgsmaster.gbff.gz

pyani.download.**construct_output_paths**(*filestem: str*, *suffix: str*, *outdir: pathlib.Path*) → Tuple[pathlib.Path, pathlib.Path]
Construct paths to output files for genome and hash.

Parameters

- **filestem** – str, output filename stem
- **suffix** – str, output filename suffix
- **outdir** – Path, path to output directory

pyani.download.**create_hash**(*fname: pathlib.Path*) → str
Return MD5 hash of the passed file contents.

Parameters **fname** – Path, path to file for hashing

We can ignore the Bandit B303 error as we're not using the hash for cryptographic purposes.

pyani.download.**create_labels**(*classification: pyani.download.Classification*, *filestem: str*, *genomehash: str*) → Tuple[str, str]
Return class and label text from UID classification.

Parameters

- **classification** – Classification named tuple (org, genus, species, strain)
- **filestem** – str, filestem of input genome file
- **genomehash** – str, MD5 hash of genome data

The 'class' data is the organism as provided in the passed Classification named tuple; the 'label' data is genus, species and strain information from the same tuple. The label is intended to be human-readable, the class data to be a genuine class identifier.

Returns a tuple of two strings: (label, class).

The two strings are tab-separated strings: <HASH>t<FILE>t<CLASS/LABEL>. The hash is used to help uniquely identify the genome in the database (label/class is unique by a combination of hash and run ID).

pyani.download.**download_genome_and_hash**(*outdir: pathlib.Path*, *timeout: int*, *dlfiledata: pyani.download.DLFileData*, *dltype: str = 'RefSeq'*, *disable_tqdm: bool = False*) → namedlist.namedlist
Download genome and accompanying MD5 hash from NCBI.

Parameters

- **args** – Namespace for command-line arguments
- **outdir** – Path to output directory for downloads
- **timeout** – int: timeout for download attempt
- **dlfiledata** – namedtuple of info for file to download
- **dltype** – reference database to use: RefSeq or GenBank
- **disable_tqdm** – disable progress bar

This function tries the (assumed to be passed) RefSeq FTP URL first and, if that fails, then attempts to download the corresponding GenBank data.

We attempt to gracefully skip genomes with download errors.

pyani.download.**download_url**(*url: str*, *outfname: pathlib.Path*, *timeout: int*, *disable_tqdm: bool = False*) → None
Download remote URL to a local directory.

Parameters

- **url** – URL of remote file for download
- **outfname** – Path, path to write output

- **timeout** –
- **disable_tqdm** – Boolean, show tqdm progress bar?

This function downloads the contents of the passed URL to the passed filename, in buffered chunks

pyani.download.**entrez_batch**(*func*)
Decorator to compile batches from the wrapped function into a single set of results.

The entrez_batch decorator should go outside the entrez_retry decorator.

pyani.download.**entrez_batched_webhistory**(**args*, *expected=None*, *batchsize=None*, ***kwargs*)

pyani.download.**entrez_esearch**(**args*, *retries=1*, ***kwargs*)

pyani.download.**entrez_esummary**(**args*, *retries=1*, ***kwargs*)

pyani.download.**entrez_retry**(*func*)
Decorator to retry the wrapped function up to 'retries' times.

pyani.download.**extract_contigs**(*fname: pathlib.Path*, *ename: pathlib.Path*) → subprocess.CompletedProcess
Extract contents of fname to ename using gunzip.

Parameters

- **fname** – str, path to input compressed file
- **ename** – str, path to output uncompressed file

Returns status of subprocess call

pyani.download.**extract_filestem**(*esummary*) → str
Extract filestem from Entrez eSummary data.

Parameters **esummary** –

Function expects esummary['DocumentSummarySet']['DocumentSummary'][0]

Some illegal characters may occur in AssemblyName - for these, a more robust regex replace/escape may be required. Sadly, NCBI don't just use standard percent escapes, but instead replace certain characters with underscores: white space, slash, comma, hash, brackets.

pyani.download.**extract_hash**(*hashfile: pathlib.Path*, *name: str*) → str
Return MD5 hash from file of name:MD5 hashes.

Parameters

- **hashfile** – Path, path to file containing name:MD5 pairs
- **name** – str, name associated with hash

pyani.download.**get_asm_uids**(*taxon_uid: str*, *retries: int*) → pyani.download.ASMIDs
Return set of NCBI UIDs associated with the passed taxon UID.

Parameters

- **taxon_uid** – str, NCBI taxID for taxon to download
- **retries** – int, number of download retry attempts

This query at NCBI returns all assemblies for the taxon subtree rooted at the passed taxon_uid.

pyani.download.**get_ncbi_classification**(*esummary*) → pyani.download.Classification
Return organism, genus, species, strain info from eSummary data.

Parameters **esummary** –

pyani.download.**get_ncbi_esummary**(*asm_uid*, *retries*, *api_key=None*) → Tuple
Obtain full eSummary info for the passed assembly UID.

Parameters

- **asm_uid** –
- **retries** –
- **api_key** –

pyani.download.**last_exception**() → str
Return last exception as a string.

pyani.download.**make_asm_dict**(*taxon_ids: List[str], retries: int*) → Dict[KT, VT]
Return a dict of assembly UIDs, keyed by passed taxon IDs.

Parameters

- **taxon_ids** –
- **retries** –

Takes the passed list of taxon IDs and calls get_asm_uids to generate a dictionary linking each taxon ID to a list of assembly IDs at NCBI.

pyani.download.**retrieve_genome_and_hash**(*filestem: str*, *suffix: str*, *ftpstem: str*, *outdir: pathlib.Path*, *timeout: int*, *disable_tqdm: bool = False*) → pyani.download.DLStatus
Download genome contigs and MD5 hash data from NCBI.

Parameters

- **filestem** –
- **suffix** –
- **ftpstem** –
- **outdir** –
- **timeout** –
- **disable_tqdm** – Boolean, show tqdm progress bar?

pyani.download.**set_ncbi_email**(*email: str*) → None
Set contact email for NCBI.

Parameters **email** – str, email address to give to Entrez at NCBI

pyani.download.**split_taxa**(*taxa: str*) → List[str]
Return list of taxon ids from the passed comma-separated list.

Parameters **taxa** – str, comma-separated list of valid NCBI taxonomy IDs

The function checks the passed taxon argument against a regular expression that permits comma-separated numerical symbols only.

pyani.nucmer module

Code for handling NUCmer output files.

class pyani.nucmer.**DeltaAlignment**(*refstart: int*, *refend: int*, *qrystart: int*, *qryend: int*, *errs: int*, *simerrs: int*, *stops: int*)
Bases: object

Represents a single alignment region and scores for a pairwise comparison.

class pyani.nucmer.**DeltaComparison**(*header: pyani.nucmer.DeltaHeader, alignments: List[pyani.nucmer.DeltaAlignment]*)

Bases: object

Represents a comparison between two sequences in a .delta file.

add_alignment(*aln: pyani.nucmer.DeltaAlignment*) → None
Add passed alignment to this object.

Parameters **aln** – DeltaAlignment object

class pyani.nucmer.**DeltaData**(*name: str*, *handle: TextIO = None*)

Bases: object

Class to hold MUMmer/nucmer output "delta" data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- **header: first line of the .delta file, naming the two input comparison files; stored** as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- program: name of the MUMmer program that produced the output
- query: path to the query sequence file
- subject: path to the subject sequence file

comparisons
Comparisons in the .delta file.

from_delta(*handle: TextIO*) → None
Populate the object from the passed .delta or .filter filehandle.

metadata
Metadata from the .delta file.

program
The MUMmer program used for the comparison.

query
Query file for the MUMmer comparison.

reference
Reference file for the MUMmer comparison.

class pyani.nucmer.**DeltaHeader**(*reference: pathlib.Path*, *query: pathlib.Path*, *reflen: int*, *querylen: int*)

Bases: object

Represents a single sequence comparison header from a MUMmer .delta file.

class pyani.nucmer.**DeltaIterator**(*handle: TextIO*)

Bases: object

Iterator for MUMmer .delta files.

Returns a stream of DeltaMetadata, DeltaComparison and DeltaAlignment objects when iterated over a filehandle

The .delta file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

class pyani.nucmer.**DeltaMetadata**
Bases: object

Represents the metadata header for a MUMmer .delta file.

pyani.pyani_classify module

Module providing functions to generate clusters/species hypotheses.

class pyani.pyani_classify.**Cliquesinfo**
Bases: tuple

Summary of clique structure.

all_k_complete
Alias for field number 2

n_nodes
Alias for field number 0

n_subgraphs
Alias for field number 1

pyani.pyani_classify.**all_components_k_complete**(*graph: networkx.classes.graph.Graph*) → bool
Return True if all components in passed graph are k-complete.

Parameters **graph** – NetworkX Graph object

pyani.pyani_classify.**analyse_cliques**(*graph: networkx.classes.graph.Graph*) → pyani.pyani_classify.Cliquesinfo
Return Cliquesinfo NamedTuple describing clique data for a graph.

Parameters **graph** – NetworkX Graph object

pyani.pyani_classify.**build_graph_from_results**(*results, label_dict: Dict[int, str], cov_min: float = 0, id_min: float = 0*) → networkx.classes.graph.Graph
Return undirected graph representing the passed ANIResults object.

The passed ANIResults object is converted to an undirected graph where nodes on the graph represent genomes, and edges represent pairwise comparisons having the minimum coverage and identity indicated.

Parameters

- **results** –
  - Run object from pyani_orm
- **label_dict** – dictionary of genome labels for result matrices the dict is keyed by the index/column values for the results matrices
- **cov_min** –
  - minimum coverage for an edge
- **id_min** –
  - minimum identity for an edge

pyani.pyani_classify.**k_complete_component_status**(*graph: networkx.classes.graph.Graph*) → List[bool]

Return list of Booleans of whether connected components of the graph are k-complete.

Parameters **graph** – NetworkX Graph object

For each component in the passed graph, a list of Booleans is calculated, representing whether each node has property P: the degree of the node is equal to the number of nodes in that component, minus 1.

The all() gives a Boolean indicating whether all nodes in that component have property P.

pyani.pyani_classify.**remove_low_weight_edges**(*graph: networkx.classes.graph.Graph*, *threshold: float*, *attribute: str = 'identity'*) → Tuple[networkx.classes.graph.Graph, List[T]]

Return graph and edgelist where edges having weight < threshold are removed.

Parameters

- **graph** – NetworkX Graph
- **threshold** – float, minimum edge weight
- **attribute** – String, attribute to use as weight

pyani.pyani_config module

Configuration settings for the pyani package.

pyani.pyani_config.**get_colormap**(*dataframe: pandas.core.frame.DataFrame*, *matname: str*) → Tuple[str, Any, Any]

Return colormap parameters for a dataframe.

Parameters

- **dataframe** –
- **matname** –

The colormap is dependent on the type of analysis that was done.

pyani.pyani_config.**params_mpl**(*dfm: pandas.core.frame.DataFrame*) → Dict[str, Tuple[str, Any, Any]]

Return dict of matplotlib parameters, dependent on dataframe.

Parameters **dfm** –

DEPRECATED FROM v0.3 onwards

pyani.pyani_files module

Code to handle files for average nucleotide identity calculations.

exception pyani.pyani_files.**PyaniFilesException**

Bases: *pyani.PyaniException*

Exception raised by pyani when file interaction goes bad.

pyani.pyani_files.**collect_existing_output**(*dirpath: pathlib.Path*, *program: str*, *args: argparse.Namespace*) → List[pathlib.Path]

Return a list of existing output files at dirpath.

Parameters

- **dirpath** – Path, path to existing output directory
- **program** – str, name of program to use for comparisons
- **args** – Namespace, command-line arguments for the run

pyani.pyani_files.**get_fasta_and_hash_paths**(*dirname: pathlib.Path = PosixPath('.')*) → List[Tuple[pathlib.Path, pathlib.Path]]

Return a list of (FASTA file, hash file) tuples in passed directory.

Parameters **dirname** – Path, path to input directory

Raises an IOError if the corresponding hash for a FASTA file does not exist

pyani.pyani_files.**get_fasta_files**(*dirname: pathlib.Path = PosixPath('.')*) → List[pathlib.Path]

Return a list of FASTA files in the passed directory.

Parameters **dirname** – Path, path to input directory

pyani.pyani_files.**get_fasta_paths**(*dirname: pathlib.Path = PosixPath('.')*, *extlist: Optional[List[T]] = None*) → List[pathlib.Path]

Return a list of paths to files matching a list of FASTA extensions.

Parameters

- **dirname** – Path, path to directory containing input FASTA files
- **extlist** – List, file suffixes for FASTA files

Returns sorted list of the full path to each file.

pyani.pyani_files.**get_input_files**(*dirname: pathlib.Path*, **ext*) → List[pathlib.Path]

Return sorted files in passed directory, filtered by extension.

Parameters

- **dirname** – Path, path to input directory
- ***ext** – optional iterable of arguments describing permitted file extensions

pyani.pyani_files.**get_sequence_lengths**(*fastafilenames: Iterable[pathlib.Path]*) → Dict[str, int]

Return dictionary of sequence lengths, keyed by organism.

Parameters **fastafilenames** – Iterable[Path], paths to input FASTA files

Biopython's SeqIO module is used to parse all sequences in the FASTA file corresponding to each organism, and the total base count in each is obtained.

NOTE: ambiguity symbols are not discounted.

pyani.pyani_files.**load_classes_labels**(*path: pathlib.Path*) → Dict[str, str]

Return a dictionary of genome classes or labels keyed by hash.

Parameters **path** – Path, path to classes or labels file

The expected format of the classes and labels files is: <HASH>t<FILESTEM>t<CLASS>|<LABEL>, where <HASH> is the MD5 hash of the genome data (this is not checked); <FILESTEM> is the path to the genome file (this is intended to be a record for humans to audit, it's not needed for the database interaction; and <CLASS>|<LABEL> is the class or label associated with that genome.

pyani.pyani_files.**read_fasta_description**(*filename: pathlib.Path*) → str

Return the first description string from a FASTA file.

Parameters **filename** – Path, path to FASTA file

pyani.pyani_files.**read_hash_string**(*filename: pathlib.Path*) → Tuple[str, str]
Return the hash and file strings from the passed hash file.

Parameters **filename** – Path, path to file containing hash string

pyani.pyani_graphics module

Code to implement graphics output for ANI analyses.

class pyani.pyani_graphics.**Params**(*params: Tuple*, *labels: Optional[Dict[KT, VT]] = None*, *classes: Optional[Dict[KT, VT]] = None*)
Bases: object

Convenience class to hold heatmap rendering parameters.

vdiff
Return difference between max and min values for presentation.

pyani.pyani_jobs module

Code to manage jobs for pyani.

In order to be a little more consistent behind the scenes for schedulers, and to allow for a fairly hacky approach to scheduing on SGE, a job dependency graph is used.

Commands to be run are stored in Jobs. A Job's dependency is stored so that the Job will not be executed until its dependency is executed.

When used in ANI analysis, the way jobs are used depends on the scheduler.

With multiprocessing, we place all root jobs in a single pool; then all first-level dependencies will go in a second (dependent) pool that is not run until the first is completed, and so on. It's not very efficient, but should work equivalently to the original code that handled asynchronous pools directly.

With SGE, the dependencies can be managed independently, and effectively interleaved by the scheduler with no need for pools.

This code is essentially a frozen and cut-down version of pysge (https://github.com/widdowquinn/pysge)

class pyani.pyani_jobs.**Job**(*name: str*, *command: str*, *queue: Optional[str] = None*)
Bases: object

Individual job to be run, with list of dependencies.

add_dependency(*job*) → None
Add passed job to the dependency list for this Job.

Parameters **job** – Job to be added to the Job's dependency list

This Job should not execute until all dependent jobs are completed.

remove_dependency(*job*) → None
Remove passed job from this Job's dependency list.

Parameters **job** – Job to be removed from the Job's dependency list

wait(*interval: float = 0.01*) → None
Wait until the job finishes, and poll SGE on its status.

Parameters **interval** – float, number of seconds to wait before polling SGE

class pyani.pyani_jobs.**JobGroup**(*name: str*, *command: str*, *queue: Optional[str] = None*, *arguments: Optional[Dict[str, List[Any]]] = None*)

Bases: object

Class that stores a group of jobs, permitting parameter sweeps.

add_dependency(*job*) → None
Add the passed job to the dependency list for this JobGroup.

Parameters **job** – Job, job to be added to the JobGroup's dependency list

This JobGroup should not execute until all dependent jobs are completed

generate_script() → None
Create the SGE script that will run the jobs in the JobGroup.

remove_dependency(*job*) → None
Remove passed job from this JobGroup's dependency list.

Parameters **job** – Job, job to be removed from the JobGroup's dependency list

wait(*interval: float = 0.01*) → None
Wait for a defined period, then poll SGE for job status.

Parameters **interval** – int, seconds to wait before polling SGE

pyani.pyani_orm module

Module providing useful functions for manipulating pyani's SQLite3 db.

This SQLAlchemy-based ORM replaces the previous SQL-based module

class pyani.pyani_orm.**BlastDB**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Describes relationship between genome, run, source BLAST database and query fragments.

Each genome and run combination can be assigned a single BLAST database for the comparisons

- fragpath path to fragmented genome (query in ANIb)
- dbpath path to source genome database (subject in ANIb)
- fragsizes JSONified dict of fragment sizes
- dbcmd command used to generate database

blastdb_id

dbcmd

dbpath

fragpath

fragsizes

genome

genome_id

run

run_id

class pyani.pyani_orm.**Comparison**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Describes a single pairwise comparison between two genomes.

aln_length

comparison_id

cov_query

cov_subject

fragsize

identity

kmersize

maxmatch

minmatch

program

query

query_id

runs

sim_errs

subject

subject_id

version

class pyani.pyani_orm.**Genome**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Describes an input genome for a pyani run.

- **genome_id** primary key
- **genome_hash** MD5 hash of input genome file (in path)
- **path** path to FASTA genome file
- **length** length of genome (total bases)
- **description** genome description

blastdbs

description

genome_hash

genome_id

labels

length

path

query_comparisons

runs

subject_comparisons

class pyani.pyani_orm.**Label**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Describes relationship between genome, run and genome label.

Each genome and run combination can be assigned a single label

class_label

genome

genome_id

label

label_id

run

run_id

class pyani.pyani_orm.**LabelTuple**
Bases: tuple

Label and Class for each file.

class_label
Alias for field number 1

label
Alias for field number 0

exception pyani.pyani_orm.**PyaniORMException**
Bases: *pyani.PyaniException*

Exception raised when ORM or database interaction fails.

class pyani.pyani_orm.**Run**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Describes a single pyani run.

blastdbs

cmdline

comparisons

date

df_alnlength

df_coverage

df_hadamard

df_identity

df_simerrors

genomes

labels

method

name

run_id

status

pyani.pyani_orm.**add_run**(*session*, *method*, *cmdline*, *date*, *status*, *name*)
Create a new Run and add it to the session.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **method** – string describing analysis run type
- **cmdline** – string describing pyani command-line for run
- **date** – datetime object describing analysis start time
- **status** – string describing status of analysis
- **name** – string - name given to the analysis run

Creates a new Run object with the passed parameters, and returns it.

pyani.pyani_orm.**add_run_genomes**(*session*, *run*, *indir: pathlib.Path*, *classpath: pathlib.Path*, *labelpath: pathlib.Path*, ***kwargs*) → List[T]
Add genomes for a run to the database.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **run** – Run object describing the parent pyani run
- **indir** – path to the directory containing genomes
- **classpath** – path to the file containing class information for each genome
- **labelpath** – path to the file containing class information for each genome

This function expects a single directory (indir) containing all FASTA files for a run, and optional paths to plain text files that contain information on class and label strings for each genome.

If the genome already exists in the database, then a Genome object is recovered from the database. Otherwise, a new Genome object is created. All Genome objects will be associated with the passed Run object.

The session changes are committed once all genomes and labels are added to the database without error, as a single transaction.

pyani.pyani_orm.**create_db**(*dbpath: pathlib.Path*) → None
Create an empty pyani SQLite3 database at the passed path.

Parameters **dbpath** – path to pyani database

pyani.pyani_orm.**filter_existing_comparisons**(*session*, *run*, *comparisons*, *program*, *version*, *fragsize: Optional[int] = None*, *maxmatch: Optional[bool] = False*, *kmersize: Optional[int] = None*, *minmatch: Optional[float] = None*) → List[T]
Filter list of (Genome, Genome) comparisons for those not in the session db.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **run** – Run object describing parent pyani run
- **comparisons** – list of (Genome, Genome) query vs subject comparisons

- **program** – program used for comparison
- **version** – version of program for comparison
- **fragsize** – fragment size for BLAST databases
- **maxmatch** – maxmatch used with nucmer comparison

When passed a list of (Genome, Genome) comparisons as comparisons, check whether the comparison exists in the database and, if so, associate it with the passed run. If not, then add the (Genome, Genome) pair to a list for returning as the comparisons that still need to be run.

pyani.pyani_orm.**get_comparison_dict**(*session: Any*) → Dict[Tuple, Any]
Return a dictionary of comparisons in the session database.

Parameters **session** – live SQLAlchemy session of pyani database

Returns Comparison objects, keyed by (_.query_id, _.subject_id, _.program, _.version, _.fragsize, _.maxmatch) tuple

pyani.pyani_orm.**get_matrix_classes_for_run**(*session: Any*, *run_id: int*) → Dict[str, List[T]]
Return dictionary of genome classes, keyed by row/column ID.

Parameters

- **session** – live SQLAlchemy session
- **run_id** – the Run.run_id value for matrices

The class labels should be valid for identity, coverage and other complete matrix results accessed via the .df_* attributes of a run

Labels are returned keyed by the string of the genome ID, for compatibility with matplotlib.

pyani.pyani_orm.**get_matrix_labels_for_run**(*session: Any*, *run_id: int*) → Dict[KT, VT]
Return dictionary of genome labels, keyed by row/column ID.

Parameters

- **session** – live SQLAlchemy session
- **run_id** – the Run.run_id value for matrices

The labels should be valid for identity, coverage and other complete matrix results accessed via the .df_* attributes of a run.

Labels are returned keyed by the string of the genome ID, for compatibility with matplotlib.

pyani.pyani_orm.**get_session**(*dbpath: pathlib.Path*) → Any
Connect to an existing pyani SQLite3 database and return a session.

Parameters **dbpath** – path to pyani database

pyani.pyani_orm.**update_comparison_matrices**(*session*, *run*) → None
Update the Run table with summary matrices for the analysis.

Parameters

- **session** – active pyanidb session via ORM
- **run** – Run ORM object for the current ANIm run

pyani.pyani_report module

Module providing functions for presenting analysis/db output.

pyani.pyani_report.**colour_coverage**(*series: pandas.core.series.Series*, *threshold: float = 0.95*, *colour: str = '#FF2222'*) → List[str]
Highlight percent coverage over a threshold.

Parameters

- **series** –
- **threshold** – float, threshold for cell highlighting
- **colour** – str, hex colour for highlighted cells

pyani.pyani_report.**colour_identity**(*series: pandas.core.series.Series*, *threshold: float = 0.95*, *colour: str = '#FF2222'*) → List[str]
Highlight percentage identities over a threshold.

Parameters

- **series** –
- **threshold** – float, threshold for cell highlighting
- **colour** – str, hex colour for highlighted cells

pyani.pyani_report.**colour_numeric**(*val: float*, *threshold: float = 0.95*, *colour: str = '#FF2222'*) → str
Highlight numeric values over a threshold.

Parameters

- **val** –
- **threshold** – float, threshold for cell highlighting
- **colour** – str, hex colour for highlighted cell

pyani.pyani_report.**colour_rows**(*series: pandas.core.series.Series*, *even_colour: str = '#DDECF5'*, *odd_colour: str = '#6CB6E4'*) → List[str]
Return alternating colours for rows in a dataframe.

Parameters

- **series** – pd.Series
- **even_colour** – str, hex colour for even rows
- **odd_colour** – str, hex colour for odd rows

pyani.pyani_report.**header_font**() → Dict[str, Any]
Return header HTML font style.

pyani.pyani_report.**hover_highlight**(*hover_colour: str = '#FFFF99'*) → Dict[str, Any]
Return HTML style to colour dataframe row when hovering.

Parameters **hover_colour** – str, hex colour for hover highlight

pyani.pyani_report.**table_padding**() → Dict[str, Any]
Return HTML for table cell padding.

pyani.pyani_report.**write_dbtable**(*dfm: pandas.core.frame.DataFrame*, *path: pathlib.Path*, *formats: Sequence[str] = ('tab',)*, *show_index: bool = True*, *colour_num: bool = False*) → None
Write database result table to output file in named format.

Parameters

- **dfm** – pd.Dataframe
- **path** – Path to output file

- **formats** – tuple of str, output file formats
- **show_index** – output row and column labels
- **colour_num** – use colours for values in HTML output

colours are used for identity/coverage tables

pyani.pyani_report.**write_styled_html**(*path: pathlib.Path*, *dfm: pandas.core.frame.DataFrame*, *index: Optional[str] = None*, *colour_num: bool = False*) → None

Add CSS styling to a dataframe and write as HTML.

Parameters

- **path** – path to write output file
- **dfm** – dataframe to be written out
- **index** – column to be set as index (if necessary)

pyani.pyani_report.**write_to_stdout**(*stem: str*, *dfm: pandas.core.frame.DataFrame*, *show_index: bool = False*, *line_width: float = None*) → None

Write dataframe in tab-separated form to STDOUT.

Parameters

- **stem** – str
- **dfm** – pd.Dataframe
- **show_index** – Boolean, include index in output table
- **line_width** –

pyani.pyani_tools module

Code to support pyani.

class pyani.pyani_tools.**ANIResults**(*labels: List[str], mode: str*)

Bases: object

Holds ANI dataframe results.

add_coverage(*qname: str*, *sname: str*, *qcover: float*, *scover: Optional[float] = None*) → None

Add percentage coverage values to self.alignment_coverage.

Parameters

- **qname** –
- **sname** –
- **value** –
- **sym** –

add_pid(*qname: str*, *sname: str*, *value: float*, *sym: bool = True*) → None

Add a percentage identity value to self.percentage_identity.

Parameters

- **qname** –
- **sname** –

- **value** –
- **sym** –

add_sim_errors(*qname: str*, *sname: str*, *value: float*, *sym: bool = True*) → None
Add a similarity error value to self.similarity_errors.

Parameters

- **qname** –
- **sname** –
- **value** –
- **sym** –

add_tot_length(*qname: str*, *sname: str*, *value: float*, *sym: bool = True*) → None
Add a total length value to self.alignment_lengths.

Parameters

- **qname** –
- **sname** –
- **value** –
- **sym** –

data
Return list of (dataframe, filestem) tuples.

hadamard
Return Hadamard matrix (identity * coverage).

class pyani.pyani_tools.**BLASTcmds**(*funcs: pyani.pyani_tools.BLASTfunctions*, *exes: pyani.pyani_tools.BLASTexes*, *prefix: str*, *outdir: pathlib.Path*)
Bases: object

Class for construction of BLASTN and database formatting commands.

build_blast_cmd(*fname: pathlib.Path*, *dbname: pathlib.Path*)
Return BLASTN command.

Parameters

- **fname** – Path to query file
- **dbname** – Path to database

build_db_cmd(*fname: pathlib.Path*) → str
Return database format/build command.

Parameters **fname** –

get_db_name(*fname: pathlib.Path*) → str
Return database filename.

Parameters **fname** –

class pyani.pyani_tools.**BLASTexes**
Bases: tuple

Convenience structure to hold BLAST executables.

blast_exe
Alias for field number 1

format_exe
Alias for field number 0

class pyani.pyani_tools.**BLASTfunctions**
Bases: tuple

Convenience structure to hold BLAST functions.

blastn_func
Alias for field number 1

db_func
Alias for field number 0

class pyani.pyani_tools.**Dependencies**
Bases: tuple

Convenience struct for third-party dependency presence.

blast
Alias for field number 0

legacy_blast
Alias for field number 1

mummer
Alias for field number 2

class pyani.pyani_tools.**MatrixData**
Bases: tuple

Convenience struct for matrix data returned by ORM.

data
Alias for field number 1

graphic_args
Alias for field number 2

name
Alias for field number 0

pyani.pyani_tools.**get_genome_length**(*filename: pathlib.Path*) → int
Return total length of all sequences in a FASTA file.

Parameters **filename** – path to FASTA file

pyani.pyani_tools.**get_labels**(*filename: pathlib.Path*, *logger: Optional[logging.Logger] = None*) → Dict[KT, VT]
Return dictionary of alternative sequence labels, or None.

Parameters

- **filename** – path to file containing tab-separated table of labels
- **logger** – logging object

Input files should be formatted as <hash>t<key>t<label>, one pair per line.

pyani.pyani_tools.**has_dependencies**() → pyani.pyani_tools.Dependencies
Return NamedTuple indicating if 3rd dependencies are available.

pyani.pyani_tools.**label_results_matrix**(*matrix: pandas.core.frame.DataFrame, labels: Dict[KT, VT]*) → pandas.core.frame.DataFrame

Return results matrix dataframe with labels.

Parameters

- **matrix** – results dataframe deriving from Run object
- **labels** – dictionary of genome labels labels must be keyed by index/col values from matrix

Applies the labels from the dictionary to the dataframe in matrix, and returns the result.

pyani.pyani_tools.**termcolor**(*logstr: str, color: Optional[str] = None, bold: Optional[bool] = False*) → str

Return the passed logstr, wrapped in terminal colouring.

pyani.run_multiprocessing module

Code to run a set of command-line jobs using multiprocessing.

For parallelisation on multi-core desktop/laptop systems, etc. we use Python's multiprocessing module to distribute command-line jobs.

pyani.run_multiprocessing.**multiprocessing_run**(*cmdlines: List[T], workers: Optional[int] = None, logger: Optional[logging.Logger] = None*) → int

Distributes passed command-line jobs using multiprocessing.

Parameters

- **cmdlines** – iterable, command line strings
- **workers** – int, number of workers to use for multiprocessing

Returns the sum of exit codes from each job that was run. If all goes well, this should be 0. Anything else and the calling function should act accordingly.

Sends a warning to the logger if a comparison fails; the warning consists of the specific command line that has failed.

pyani.run_multiprocessing.**populate_cmdsets**(*job: pyani.pyani_jobs.Job, cmdsets: List[T], depth: int*) → List[T]

Create list of jobsets at different depths of dependency tree.

Parameters

- **job** –
- **cmdsets** –
- **depth** –

This is a recursive function (is there something quicker in the itertools module?) that descends each 'root' job in turn, populating each

pyani.run_multiprocessing.**run_dependency_graph**(*jobgraph, workers: Optional[int] = None, logger: Optional[logging.Logger] = None*) → int

Create and run pools of jobs based on the passed jobgraph.

Parameters

- **jobgraph** – list of jobs, which may have dependencies.

- **workers** – int, number of workers to use with multiprocessing
- **logger** – a logger module logger (optional)

The strategy here is to loop over each job in the list of jobs (jobgraph), and create/populate a series of Sets of commands, to be run in reverse order with multiprocessing_run as asynchronous pools.

pyani.run_sge module

Code to run a set of command-line jobs using SGE/Grid Engine.

For parallelisation on multi-node system, we use some custom code to submit jobs.

pyani.run_sge.**build_and_submit_jobs**(*root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None*) → None

Submit passed iterable of Job objects to SGE.

Parameters

- **root_dir** – root directory for SGE and job output
- **jobs** – list of Job objects, describing each job to be submitted
- **sgeargs** – str, additional arguments to qsub

This places SGE's output in the passed root directory

pyani.run_sge.**build_directories**(*root_dir: pathlib.Path*) → None

Construct the subdirectories output, stderr, stdout, and jobs.

Parameters **root_dir** – path of root directory in which to place output

Subdirectories are created in the passed root directory. These subdirectories have the following roles:

jobs Stores the scripts for each job stderr Stores the stderr output from SGE stdout Stores the stdout output from SGE output Stores output (if the scripts place the output here)

- root_dir Path to the top-level directory for creation of subdirectories

pyani.run_sge.**build_job_scripts**(*root_dir: pathlib.Path, jobs: List[T]*) → None

Construct script for each passed Job in the jobs iterable.

Parameters

- **root_dir** – Path to output directory
- **jobs** –

pyani.run_sge.**build_joblist**(*jobgraph*) → List[T]

Return a list of jobs, from a passed jobgraph.

Parameters **jobgraph** –

pyani.run_sge.**compile_jobgroups_from_joblist**(*joblist: List[T], jgprefix: str, sgegroupsize: int*) → List[T]

Return list of jobgroups, rather than list of jobs.

Parameters

- **joblist** –
- **jgprefix** – str, prefix for SGE jobgroup
- **sgegroupsize** – int, number of jobs in each SGE jobgroup

pyani.run_sge.**extract_submittable_jobs**(*waiting: List[T]*) → List[T]
Obtain list of jobs that are able to be submitted from pending list.

Parameters **waiting** – list of Job objects

pyani.run_sge.**populate_jobset**(*job: pyani.pyani_jobs.Job, jobset: Set[T], depth: int*) → Set[T]
Create set of jobs reflecting dependency tree.

Parameters

- **job** –
- **jobset** –
- **depth** –

The set contains jobs at different depths of the dependency tree, retaining dependencies as strings, not Jobs.

pyani.run_sge.**run_dependency_graph**(*jobgraph*, *jgprefix: str = 'ANIm_SGE_JG'*, *sgegroupsize: int = 10000*, *sgeargs: Optional[str] = None*) → None
Create and runs SGE scripts for jobs based on passed jobgraph.

Parameters

- **jobgraph** – list of jobs, which may have dependencies.
- **verbose** – flag for multiprocessing verbosity
- **jgprefix** – a prefix for the submitted jobs, in the scheduler
- **sgegroupsize** – the maximum size for an array job submission
- **sgeargs** – additional arguments to qsub

The strategy here is to loop over each job in the dependency graph and, because we expect a single main delta-filter (wrapped) job, with a single nucmer dependency for each analysis, we can split the dependency graph into two lists of corresponding jobs, and run the corresponding nucmer jobs before the delta-filter jobs.

pyani.run_sge.**split_seq**(*iterable: Iterable[T_co], size: int*) → Generator[T_co, T_contra, V_co]
Split a passed iterable into chunks of a given size.

Parameters

- **iterable** – iterable
- **size** – int, number of items to retun in each chunk

pyani.run_sge.**submit_jobs**(*root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None*) → None
Submit passed jobs to SGE server with passed directory as root.

Parameters

- **root_dir** – path to output directory
- **jobs** – list of Job objects
- **sgeargs** – str, additional arguments for qsub

pyani.run_sge.**submit_safe_jobs**(*root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None*) → None
Submit passed list of jobs to SGE server with dir as root for output.

Parameters

- **root_dir** – path to output directory
- **jobs** – iterable of Job objects

- **sgeargs** – str, additional arguments for qsub

pyani.tetra module

Code to implement the TETRA average nucleotide identity method.

Provides functions for calculation of TETRA as described in:

Richter M, Rossello-Mora R (2009) Shifting the genomic gold standard for the prokaryotic species definition. Proc Natl Acad Sci USA 106: 19126-19131. doi:10.1073/pnas.0906412106.

and

Teeling et al. (2004) Application of tetranucleotide frequencies for the assignment of genomic fragments. Env. Microbiol. 6(9): 938-947. doi:10.1111/j.1462-2920.2004.00624.x

pyani.tetra.**calculate_correlations**(*tetra_z: Dict[str, Dict[str, float]]*) → pandas.core.frame.DataFrame
Return dataframe of Pearson correlation coefficients.

Parameters **tetra_z** – dict, Z-scores, keyed by sequence ID

Calculates Pearson correlation coefficient from Z scores for each tetranucleotide. This is done longhand here, which is fast enough, but for robustness we might want to do something else... (TODO).

Note that we report a correlation by this method, rather than a percentage identity.

pyani.tetra.**calculate_tetra_zscore**(*filename: pathlib.Path*) → Dict[str, float]
Return TETRA Z-score for the sequence in the passed file.

Parameters **filename** – path to sequence file

Calculates mono-, di-, tri- and tetranucleotide frequencies for each sequence, on each strand, and follows Teeling et al. (2004) in calculating a corresponding Z-score for each observed tetranucleotide frequency, dependent on the mono-, di- and tri- nucleotide frequencies for that input sequence.

pyani.tetra.**calculate_tetra_zscores**(*infilenames: Iterable[T_co]*) → Dict[str, Dict[str, float]]
Return dictionary of TETRA Z-scores for each input file.

Parameters **infilenames** – iterable of paths to input sequence files

pyani.tetra.**tetra_clean**(*instr: str*) → bool
Return True if string contains only unambiguous IUPAC nucleotide symbols.

Parameters **instr** – str, nucleotide sequence

We are assuming that a low frequency of IUPAC ambiguity symbols doesn't affect our calculation.

5.14 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W