The Journal of Open Source Software

Opfi: A Python package for identifying gene clusters in large genomics and metagenomics data sets

Alexis M. Hill[*1], **James R. Rybarski**[†2], **Kuang Hu**[1,2], **Ilya J. Finkelstein**[2,3], **and Claus O. Wilke**[1]

1 Department of Integrative Biology, The University of Texas at Austin, Austin, Texas 78712, USA **2** Department of Molecular Biosciences, The University of Texas at Austin, Austin, Texas 78712, USA **3** Center for Systems and Synthetic Biology, The University of Texas at Austin, Austin, Texas, 78712, USA

DOI: 10.21105/joss.03678

Software

- Review
- Repository
- Archive

Editor: Charlotte Soneson

Reviewers:

- @Thomieh73
- @afrubin

Submitted: 20 August 2021
Published: 27 October 2021

Summary

Gene clusters are sets of co-localized, often contiguous genes that together perform specific functions, many of which are relevant to biotechnology. There is a need for software tools that can extract candidate gene clusters from vast amounts of available genomic data. Therefore, we developed Opfi: a modular pipeline for identification of arbitrary gene clusters in assembled genomic or metagenomic sequences. Opfi contains functions for annotation, de-deduplication, and visualization of putative gene clusters. It utilizes a customizable rule-based filtering approach for selection of candidate systems that adhere to user-defined criteria. Opfi is implemented in Python, and is available on the Python Package Index and on Bioconda (Grüning et al., 2018).

Statement of need

Gene clusters have been successfully repurposed for a number of biotechnical applications, including biofuel production, organic compound synthesis, and gene editing (Fischbach & Voigt, 2010). Despite the broad utility of known gene clusters, identification of novel gene clusters remains a challenging task. While there are many tools available for annotation of singular genes (or protein domains) in biological sequence data (Buchfink et al., 2021; Camacho et al., 2009; Steinegger & Söding, 2017), these programs do not identify whole gene clusters out of the box. In many cases, researchers must combine bioinformatics tools ad hoc, resulting in one-off pipelines that can be difficult to reproduce. Several software packages have been developed for the discovery of specific types of gene clusters (Blin et al., 2019; Santos-Aberturas et al., 2019; van Heel et al., 2018), but these tools may not be sufficiently flexible to identify clusters of an arbitrary genomic composition. To address these gaps, we developed a modular pipeline that integrates multiple bioinformatics tools, providing a flexible, uniform computational framework for identification of arbitrary gene clusters. In a recent study, we used Opfi to uncover novel CRISPR-associated transposons (CASTs) in a large metagenomics dataset (Rybarski et al., 2021).

*co-first author, corresponding author
†co-first author

Implementation

Opfi is implemented in Python, and uses several bioinformatics tools for feature annotation (Buchfink et al., 2021; Camacho et al., 2009; Edgar, 2007; Shi & Liang, 2019; Steinegger & Söding, 2017). Users can install Opfi and all of its dependencies through Bioconda (Grüning et al., 2018). Opfi consists of two major components: Gene Finder, for discovery of gene clusters, and Operon Analyzer, for rule-based filtering, deduplication, and visualization of gene clusters identified by Gene Finder. All modules generate output in a comma-separated (CSV) format that is common to the entire package.

Example Gene Finder usage

The following example script searches for putative CRISPR-Cas loci in the genome of *Rippkaea orientalis PCC 8802*. Information about the biological significance of this example, as well as data inputs and descriptions, can be found in the `tutorials` directory in the project GitHub repository. The example illustrates the use of the `Pipeline` class for setting up a gene cluster search. First, `add_seed_step` specifies a step to annotate *cas1* genes, using protein BLAST (BLASTP) (Camacho et al., 2009) and a database of representative Cas1 protein sequences. 10,000 bp regions directly up- and downstream of each putative *cas1* gene are selected for further analysis, and all other regions are discarded. Next, `add_filter_step` adds a step to annotate candidate regions for additonal *cas* genes. Candidates that do not have at least one additional *cas* gene are discarded from the master list of putative systems. Finally, `add_crispr_step` adds a step to search remaining candidates for CRISPR arrays, i.e. regions of alternating ~30 bp direct repeat and variable sequences, using the PILER-CR repeat finding software (Edgar, 2007).

```
from gene_finder.pipeline import Pipeline
import os

genomic_data = "GCF_000024045.1_ASM2404v1_genomic.fna.gz"
job_id = "r_orientalis"

p = Pipeline()
p.add_seed_step(db="cas1", name="cas1", e_val=0.001, blast_type="PROT")
p.add_filter_step(db="cas_all", name="cas", e_val=0.001, blast_type="PROT")
p.add_crispr_step()

p.run(job_id=job_id, data=genomic_data, span=10000, gzip=True)
```

Running this code creates the CSV file `r_orientalis_results.csv`, which contains information about each system identified; in this example, that is two canonical CRISPR-Cas systems, and one locus with weak homology to *cas* genes. Each line in the file represents a single putative feature in a candidate locus. Features from the same candidate are grouped together in the CSV. Detailed information about the output format can be found in the Opfi documentation.

Example Operon Analyzer usage

In the previous example, passing systems must meet the relatively permissive criterion of having at least one *cas1* gene co-localized with one additional *cas* gene. This is sufficient to identify CRISPR-Cas loci, but may also capture regions that do not contain functional CRISPR-Cas systems, but rather consist of open reading frames (ORFs) with weak homology to *cas* genes.

These improbable systems could be eliminated during the homology search by making the match acceptance threshold more restrictive (i.e., by decreasing the e-value), however, this could result in the loss of interesting, highly diverged systems. Therefore, we implemented a module that enables post-homology search filtering of candidate systems, using flexible rules that can be combined to create sophisticated elimination functions. This allows the user to first perform a broad homology search with permissive parameters, and then apply rules to cull unlikely candidates without losing interesting and/or novel systems. Additionally, rules may be useful for selecting candidates with a specific genomic composition for downstream analysis. It should be noted that the use of the term "operon" throughout this library is an artifact from early development of Opfi. At this time, Opfi does not predict whether a candidate system represents a true operon, that is, a set of genes under the control of a single promoter. Although a candidate gene cluster may certainly qualify as an operon, it is currently up to the user to make that distinction.

Rule-based filtering is illustrated with the following example. The sample script takes the output generated by the previous example and reconstructs each system as an `Operon` object. Next, the `RuleSet` class is used to assess each candidate; here, passing systems must contain two cascade genes (*cas5* and *cas7*) no more than 1000 bp apart, and at least one *cas3* (effector) gene. For a complete list of rules, see the Opfi documentation.

```
from operon_analyzer import analyze, rules

rs = rules.RuleSet()
rs.contains_group(["cas5", "cas7"], max_gap_distance_bp = 1000)
rs.require("cas3"))

with open("r_orientalis_results.csv", "r") as input_csv:
    with open("filtered_output.csv", "w") as output_csv:
        analyze.evaluate_rules_and_reserialize(input_csv, rs, output_csv)
```

After running this code, the file `filtered_output.csv` contains only high-confidence type-I CRISPR-Cas systems (re-serialized to CSV format) that passed all rules in the rule set.

Candidate visualization

Opfi integrates the `DNAFeaturesViewer` package (Zulkower & Rosser, 2020) to create gene diagrams of candidate systems. Each input system is visualized as a single PNG image. The sample script below reads in output from the previous example, and generates two gene diagram images, one for each CRISPR-Cas system present in *Rippkaea orientalis*. One image is provided for reference in Figure 1.

```
from operon_analyzer import load, rules, visualize

feature_colors = { "cas1": "lightblue",
                   "cas2": "seagreen",
                   "cas3": "gold",
                   "cas4": "springgreen",
                   "cas5": "darkred",
                   "cas6": "thistle",
                   "cas7": "coral",
                   "cas8": "red",
                   "cas9": "palegreen",
                   "cas10": "blue",
                   "cas11": "tan",
```

```
                    "cas12": "orange",
                    "cas13": "saddlebrown",
                    "CRISPR array": "purple"
                    }

fs = rules.FilterSet().pick_overlapping_features_by_bit_score(0.9)
with open("filtered_output.csv", "r") as operon_data:
    operons = [operon for operon in load.load_operons(operon_data)]
    for operon in operons:
      fs.evaluate(operon)
    visualize.plot_operons(operons, output_directory=".", \
      plot_ignored=False, feature_colors=feature_colors)
```

The `FilterSet` class is used to resolve features with sequences that overlap by more than 90%. Specifically, only the overlapping feature with the highest bitscore value (a quantity that describes the overall quality of an alignment) is rendered when `pick_overlapping_features_by_bit_score` is applied. Note that is not a requirement for candidate visualization, but can improve gene diagram clarity.

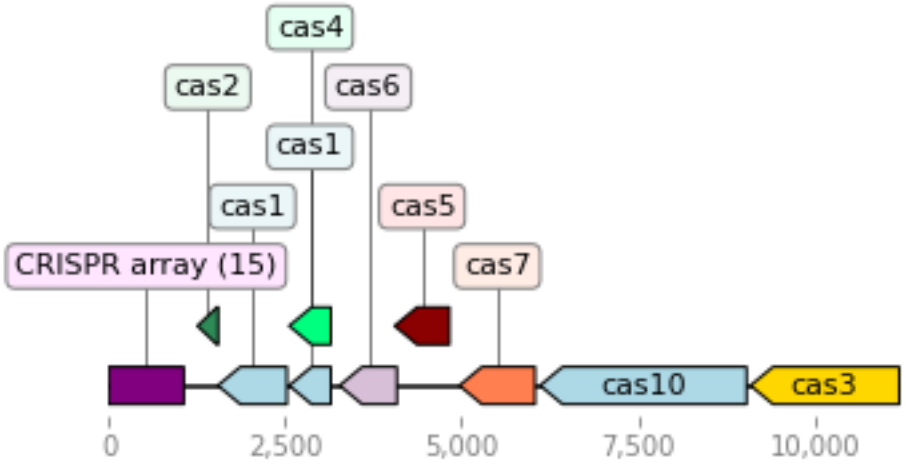

Figure 1: One of two type-I CRISPR-Cas systems present in the genome of *Rippkaea orientalis PCC 8802*. Note that the ORF beginning at position ~2500 has homology with both *cas1* and *cas4*. These alignments have identical bitscores (i.e., the goodness of alignments is quivalent, using this metric), so both annotations appear in the diagram, even though `pick_overlapping_features_by_bit_score` was applied.

Acknowledgements

The authors would like to thank the staff of the Texas Advanced Computing Center for providing computational resources, and members of the Finkelstein and Wilke labs for helpful discussions. This work was supported by an NIGMS grant R01GM124141 (to I.J.F.), the Welch Foundation grant F-1808 (to I.J.F.), NIGMS grant R01 GM088344 (to C.O.W.), and the College of Natural Sciences Catalyst Award for seed funding.

References

Blin, K., Shaw, S., Steinke, K., Villebro, R., Ziemert, N., Lee, S. Y., Medema, M. H., & Weber, T. (2019). antiSMASH 5.0: updates to the secondary metabolite genome mining pipeline. *Nucleic Acids Research*, *47*(W1), W81–W87. https://doi.org/10.1093/nar/gkz310

Buchfink, B., Reuter, K., & Drost, H.-G. (2021). Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature Methods*, *18*(4), 366–368. https://doi.org/10.1038/s41592-021-01101-x

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: Architecture and applications. *BMC Bioinformatics*, *10*(1), 421. https://doi.org/10.1186/1471-2105-10-421

Edgar, R. C. (2007). PILER-CR: Fast and accurate identification of CRISPR repeats. *BMC Bioinformatics*, *8*(1), 18. https://doi.org/10.1186/1471-2105-8-18

Fischbach, M., & Voigt, C. A. (2010). Prokaryotic gene clusters: A rich toolbox for synthetic biology. *Biotechnology Journal*, *5*(12), 1277–1296. https://doi.org/10.1002/biot.201000181

Grüning, B., Dale, R., Sjödin, A., Chapman, B., Rowe, J., Tomkins-Tinch, CH., Valieris, R., Köster, J., & The Bioconda Team. (2018). Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, *15*(7), 475–476. https://doi.org/10.1038/s41592-018-0046-7

Rybarski, J. R., Hu, K., Hill, A. M., Wilke, C. O., & Finkelstein, I. J. (2021). Metagenomic discovery of CRISPR-associated transposons. *bioRxiv*. https://doi.org/10.1101/2021.08.16.456562

Santos-Aberturas, J., Chandra, G., Frattaruolo, L., Lacret, R., Pham, T. H., Vior, N. M., Eyles, T. H., & Truman, A. W. (2019). Uncovering the unexplored diversity of thioamidated ribosomal peptides in Actinobacteria using the RiPPER genome mining tool. *Nucleic Acids Research*, *47*(9), 4624–4637. https://doi.org/10.1093/nar/gkz192

Shi, J., & Liang, C. (2019). Generic Repeat Finder: A High-Sensitivity Tool for Genome-Wide De Novo Repeat Detection. *Plant Physiology*, *180*(4), 1803–1815. https://doi.org/10.1104/pp.19.00386

Steinegger, M., & Söding, J. (2017). MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, *35*(11), 1026–1028. https://doi.org/10.1038/nbt.3988

van Heel, A. J., de Jong, A., Song, C., Viel, J. H., Kok, J., & Kuipers, O. P. (2018). BAGEL4: a user-friendly web server to thoroughly mine RiPPs and bacteriocins. *Nucleic Acids Research*, *46*(W1), W278–W281. https://doi.org/10.1093/nar/gky383

Zulkower, V., & Rosser, S. (2020). DNA Features Viewer: a sequence annotation formatting and plotting library for Python. *Bioinformatics*, *36*(15), 4350–4352. https://doi.org/10.1093/bioinformatics/btaa213