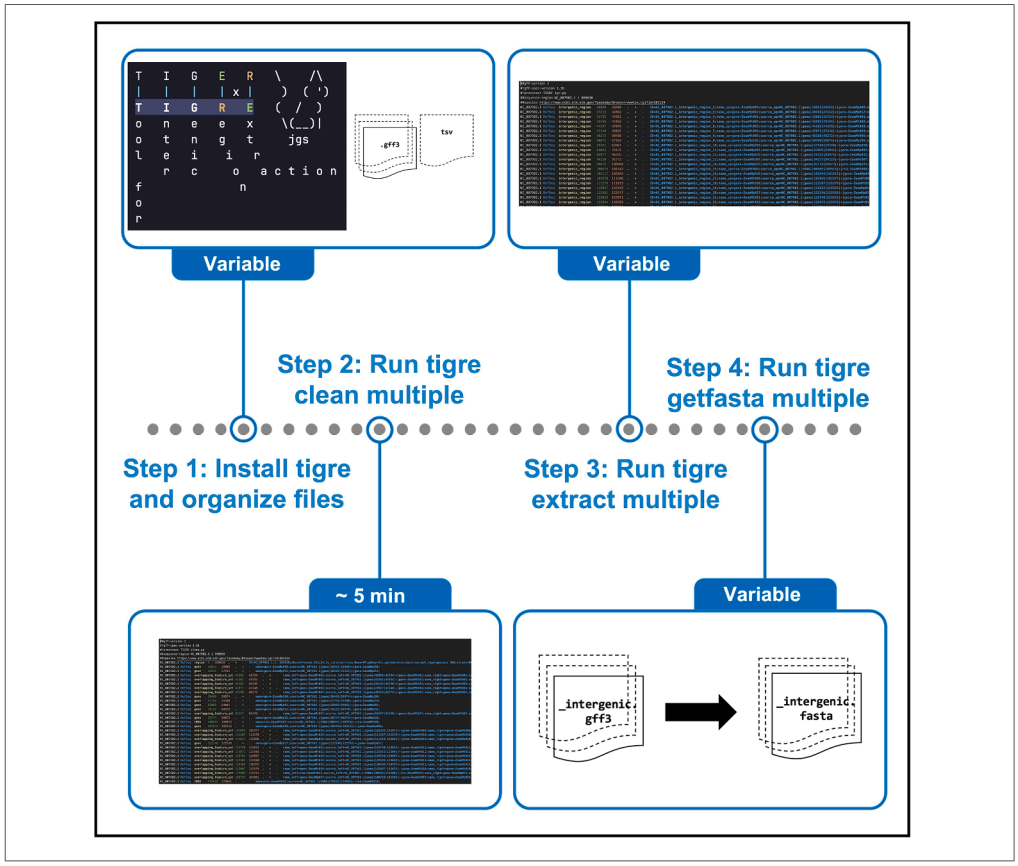
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for extracting intergenic regions from annotated genomes using TIGRE



Here, we present a protocol for extracting intergenic regions from annotated genomes using TIGRE (Tool for Intergenic Region Extraction). This protocol describes steps for installing and running TIGRE locally. Via TIGRE, the user can prepare the input data ("clean" command), retrieve the intergenic sequences ("extract" command), obtain the respective intergenic nucleotide sequences ("getfasta" command), and merge different GFF3 files ("combine" command). TIGRE capabilities are demonstrated on the retrieval of intergenic sequences from 1,207 land plant mitochondrial genomes.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Breno Dupin, Matheus Sanita Lima, Alexandre Rossi Paschoal, David Roy Smith

breno.dupin@gmail.com (B.D.)
msanital@uwo.ca (M.S.L.)

Highlights

Protocol for extracting intergenic regions from annotated genomes

Instructions on how to interact with TIGRE's CLI (command-line interface)

Guidance on how to implement the advanced use of TIGRE for the extraction of introns

Dupin et al., STAR Protocols 6, 104270
December 19, 2025
https://doi.org/10.1016/j.xpro.2025.104270

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for extracting intergenic regions from annotated genomes using TIGRE

Breno Dupin,[1,4,5,*] Matheus Sanita Lima,[2,4,6,*] Alexandre Rossi Paschoal,[1,3] and David Roy Smith[2]

[1]Department of Computer Science, Federal University of Technology-Parana, Cornelio Procopio, Paraná 86300-000, Brazil

[2]Department of Biology, Western University, London, ON N6A 5B7, Canada

[3]Rosalind Franklin Institute, Harwell Science Campus, Didcot OX11 0QX, UK

[4]These authors contributed equally

[5]Technical contact

[6]Lead contact

*Correspondence: breno.dupin@gmail.com (B.D.), msanital@uwo.ca (M.S.L.)
https://doi.org/10.1016/j.xpro.2025.104270

SUMMARY

Here, we present a protocol for extracting intergenic regions from annotated genomes using TIGRE (Tool for Intergenic Region Extraction). This protocol describes steps for installing and running TIGRE locally. Via TIGRE, the user can prepare the input data ("clean" command), retrieve the intergenic sequences ("extract" command), obtain the respective intergenic nucleotide sequences ("getfasta" command), and merge different GFF3 files ("combine" command). TIGRE capabilities are demonstrated on the retrieval of intergenic sequences from 1,207 land plant mitochondrial genomes.

BEFORE YOU BEGIN

In molecular evolution, few topics are more controversial than noncoding DNA. The two camps (functional vs junk DNA) have been at loggerheads for decades.[1] Between the catchy article titles[2] and hyperbolic media coverage,[3] three facts are settled: i) population genetics can easily explain the accumulation of noncoding DNA via nonadaptive mechanisms[4]; ii) pervasive transcription, although largely spurious,[5] is found in multiple systems[6,7]; and iii) noncoding RNAs have essential regulatory roles across the Tree of Life.[7–9]

With controversy comes scrutiny, and dozens of research groups have peered through noncoding nucleotides of prokaryotes and eukaryotes alike. Of particular interest is the evolution of intergenic regions (also named intergenic spacers or intergenic sequences), which make up most of the noncoding portion of any genome.[10] Most studies of intergenic regions gloss over the methods used to extract such sequences, but bioinformatics tools and protocols dedicated (exclusively or not) to the extraction of intergenic regions have been published.[9,11–14]

Innovation

With a user-centered focus, TIGRE has been designed with scalability, interoperability, and customizability in mind. Contrary to traditional intergenic extraction strategies (such as bedtools complement), TIGRE maintains the identifying information of the delimiting genes and handles circular genomes. For even greater consistency and interoperability, the user can implement the Gene Dictionary Tool (GDT) and protocol[15] to standardize gene names across the study sample. This protocol facilitates the extraction and analysis of intergenic regions from any type of annotated genome.

Overview

TIGRE - Tool for InterGenic Region Extraction – can extract intergenic regions from any annotated genome in a scalable manner. The tool has three main functions: clean, extract, and

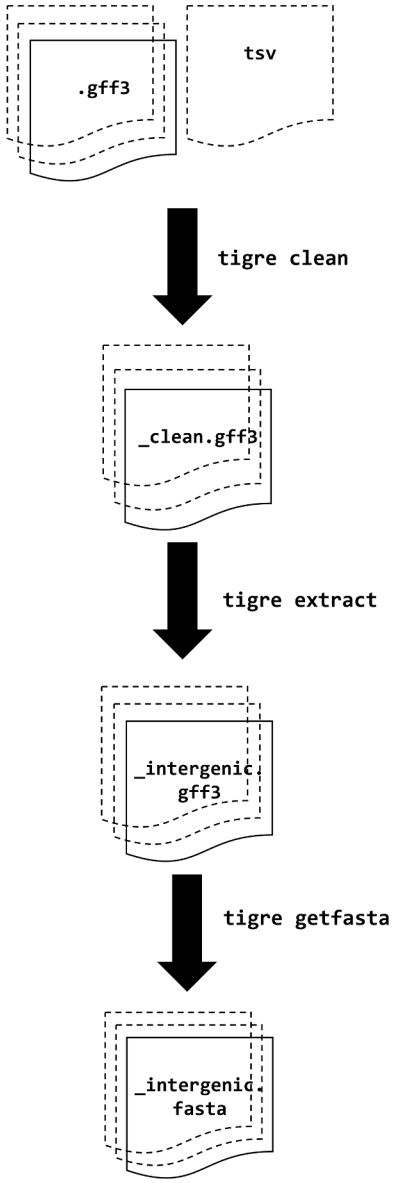


Figure 1. Overview of the TIGRE protocol

TIGRE's pipeline is composed of three main commands - clean, extract, and getfasta. All commands can operate on a single or multiple GFF3 files. When processing multiple genomes, the genome annotations must be indexed in a TSV file. tigre clean does the "heavy lifting", as it standardizes the input GFF3 files for the extraction of intergenic regions. tigre extract generates GFF3 files containing the extracted intergenic regions, including regions spanning the genome boundaries of circular genomes. tigre getfasta retrieves the corresponding nucleotide sequences of the intergenic regions. tigre combine is an auxiliary command, so it is not depicted in the main pipeline. Details on the implementation of tigre combine can be found under Advanced Usage in https://github.com/brenodupin/tigre.

getfasta (Figure 1 and step-by-step method details). The main inputs are a TSV file indexing the genome(s) of interest and their corresponding GFF3 files. Should the user want to extract the nucleotide sequences of the intergenic regions, then the respective FASTA files of each genome are also necessary. TIGRE has a fourth auxiliary command called combine. This command is used only in the Advanced Usage of TIGRE for the extraction of introns (see expected outcomes).

Table 1. Performance metrics for tigre clean, extract, and getfasta in two machines: Ubuntu 22.04.5 LTS and macOS Sonoma 14.6.1

Command	Operating system	Elapsed time (s)	Max memory use (MB)
clean	Ubuntu 22.04.5 LTS	7.88	144.8
	macOS Sonoma 14.6.1	27.05	156.2
extract	Ubuntu 22.04.5 LTS	1.46	104.6
	macOS Sonoma 14.6.1	6.93	93
getfasta	Ubuntu 22.04.5 LTS	1.38	106.3
	macOS Sonoma 14.6.1	5.67	125.3

All commands share a unified logging system and two modes: simple vs multiple. The log flags (-v, –log, –no-log-file, –quiet) give the user autonomy to determine the level of detail to be logged in any TIGRE command. Unless –no-log-file is passed, a log will always be saved, at least according to the default parameters.

For added safety, TIGRE has checkpoints that ensure whether the input (and output) files already exist. Unless the flag –overwrite (which is shared across all TIGRE commands) is specified, output files will not be overwritten on already existing files with the same name. Just as defensively, TIGRE commands will create empty output files when the input files are empty. For instance, tigre clean might create a clean GFF containing overlapping feature sets juxtaposed with other genome features that leave no room for intergenic regions. In that case, tigre extract will generate an empty GFF (containing only the respective file header), and tigre getfasta will generate an empty file (without any header). These empty files function as a "cushion" preventing the program from crashing when the checkpoints (listed above) are called.

TIGRE's potential for scalability is inherently designed in its built-in multiprocessing capabilities. The modes single and multiple allow the user to choose between running a quick test in one genome (single) or a large-scale analysis in thousands of genomes (multiple). TIGRE's multiprocessing framework also supports the integration of GDT,[15] regardless of the size of the gene dictionary files (.gdict) and/or the number of analyzed genomes (Table 1).

The three commands are intertwined via a group of shared flags that allows the user to create customized pipelines (Figure 2). Depending on the genome features specified via the –query-string flag (in tigre clean), the user can extract not only intergenic regions but any region(s) that falls between any set(s) of genome feature(s). For example, if the user specifies –query-string as being "type in ('tRNA', 'region')", tigre clean will filter out all other genome features (such as genes, rRNAs, etc.) and provide tigre extract with a clean GFF file that has only tRNAs as the delimiting elements of the soon-to-be-extracted "inter-tRNA" regions. The several –gff-<> and –fasta-<> flags, combined with –feature-type, give users the flexibility to customize their pipeline according to their interest.

Optional flags such as –keep-orfs (for tigre clean), –add-region (for tigre extract), and –bedtools-compatible (for tigre getfasta) were designed to meet the needs and challenges usually encountered when analyzing organelle intergenic regions across disparate species. Detailed documentation of all three commands and their multiple flags, can be found at https://github.com/brenodupin/tigre.

Download the software and prerequisites

TIGRE is a pure Python (>=3.12) CLI tool developed in Unix environment. TIGRE was tested extensively on Ubuntu and validated for use on MacOS. The tool can be downloaded and installed directly from PyPI (https://pypi.org/project/tigre/). The only mandatory dependency is pandas. Biopython and GDT are the two other optional dependencies: tigre getfasta relies on Biopython to extract FASTA sequences of the intergenic regions; and tigre clean implements GDT, if –gdict is used, to standardize gene names when formatting its output (<AN>_clean.gff3).

Figure 2. TIGRE's command-line interface (CLI)

TIGRE's command-line interface is extensively documented to help users make sense of the pipeline. Further implementation details of each command can be found at https://github.com/brenodupin/tigre. Detailed

Figure 2. *Continued*
explanation of how TIGRE handles circular genomes is presented in https://github.com/brenodupin/tigre/blob/master/PROCESSING.md. The logo contains the ASCII art from Joan G. Stark (https://oldcompcz.github.io/jgs/joan_stark/index-2.html).

Note: TIGRE's built-in optimization strategies allow for a wide range of hardware specifications. Any modern computer (desktop or laptop) with at least 8 GB of RAM should be able to use TIGRE for analyzing thousands of organelle genomes. For reference, TIGRE's performance was tested using a dataset with 1,207 land plant mitochondrial genomes in two machines: Ubuntu 22.04.5 LTS laboratory server (dual Intel Xeon E5-2640 v4 @ 2.40 GHz; 20 cores / 40 threads); 384 GB RAM, and macOS Sonoma 14.6.1 (Intel Core i9-8950HK @ 2.90 GHz; 6 cores / 12 threads; 32 GB RAM) (Table 1). All tests used the maximal available number of threads.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Updated gene dictionary for land plant mitochondrial genomes (.gdt) files	Dupin et al.[15]	https://zenodo.org/records/17177240
Original GFF3 and FASTA files of 1.2K land plant mitochondrial genomes	NCBI Nucleotide	https://www.ncbi.nlm.nih.gov/nucleotide/ and https://zenodo.org/records/17475986
Output GFF3 and FASTA files of 1.2K land plant mitochondrial genomes	This manuscript	https://zenodo.org/records/17475986
Software and algorithms		
gdt library	Dupin et al.[15]	https://github.com/brenodupin/gdt
Python 3.12	Python Software Foundation	https://www.python.org
Biopython 1.85	Cock et al.[16]	https://biopython.org/
pandas 2.2.3	The Pandas Development Team	https://pandas.pydata.org/

MATERIALS AND EQUIPMENT

As a pure Python CLI tool, TIGRE has a user-friendly interface, permitting users to implement flexibly the different commands (clean, extract, getfasta, and combine). TIGRE's functionalities are organized in their own Python files: cli.py is the actual CLI, so it orchestrates all tigre commands and arguments passed by the user; gff3_utils.py is derived from the gdt library and possess several functionalities to treat GFF files; log_setup.py organizes all the detailed logging system of TIGRE; clean.py, and clean_gdt.py make up the tigre clean command in its various flavours (i.e., single vs multiple and with or without GDT); biopython_wrapper.py is the script for the tigre getfasta command; igr.py is the cornerstone of TIGRE, as this file encapsulates the intergenic region extraction rationale and mechanics; combine.py contains the source code for the auxiliary command that merges GFF3 files. The annotated genomes must be inputted as GFF3 files. The mandatory and optional dependencies are listed in the key resources table. The source code for TIGRE can be found at https://github.com/brenodupin/tigre.

STEP-BY-STEP METHOD DETAILS

Preparing the environment and files

Timing: Minutes to hours (depending on dataset size)

This section sets up the environment and organizes the input data for the extraction of intergenic sequences.

1. Install TIGRE, and possibly biopython and gdt, using pip:

```
>pip install "tigre[all]''
```

Note: "tigre[all]" will automatically install the two optional dependencies gdt and biopython, which are needed to analyze the example dataset. These extras can be installed separately with "tigre[gdt]" and "tigre[bio]". You can also install tigre without any optional dependency via ">pip install tigre".

Note: There is an example dataset in https://github.com/brenodupin/tigre/tree/master/examples. If you would like to test this protocol using the example dataset (along with the commands below), you should download the plants_mit.tar.zst (compressed) file and decompress it with >tar -xf plants_mit.tar.zst. Then you should enter the folder with >cd plants_mit and run TIGRE from it. If you follow these steps, you can skip to step 4 (See troubleshooting for details on how to use the example dataset for benchmarking).

2. Create a TSV file listing the accession numbers of the genomes you want to analyze.
3. Create a separate folder for each genome listed in the TSV, in the working directory containing your TSV file.

Note: The example dataset contains only annotations from NCBI, so the TSV file lists NCBI Accession Numbers (as in NC_XXXX.X). Each folder must be named exactly as the corresponding genome annotation listed in the TSV - e.g., folder as "NC_007982.1", and inside it the GFF3 file as "NC_007982.1.gff3".

⚠ CRITICAL: The column in the TSV file must be named. The (default) name used in the example dataset is "AN" (for Accession Number), but you can name this column in any other way. The correct column name must be passed to the –an-column flag, whenever this flag is needed. Similarly, the GFF files can have a file extension different than.gff3; the correct extension must be passed to –gff-in-ext. If you wish to maintain consistency across your dataset, the flag –gff-out-ext can be used in certain commands to produce output files with the same extension. Note that TIGRE does not convert any file types - i.e., the GFF files will always be GFF files regardless of the chosen extension.

Running tigre clean

Timing: ~5 min (depending on dataset size)

This section standardizes the inputted GFF3 files in a way that allows the extraction of intergenic regions from the genomes of interest.

4. Run tigre clean multiple from your working direct:

```
>tigre clean multiple -v --log tigre_clean.log --gdict plants_mit.gdict --tsv plants_mit.tsv
--overwrite
```

Note: In the example commands, the flags -v, –log, and –gdict are used to showcase TIGRE's functionalities. These flags are optional and can be used independent of each other. The -v flag will provide the user with details for later debugging, –gdict implements the gene

dictionary capabilities from GDT, and –log will point to the file where the log will be saved. As default the log is saved in the working directory as tigre_<command>_<timestamp>.log. There also exist –quiet and –no-log-file; the first one will not print any message onto stdout (standard output), whereas the second one will not create a log file.

Note: tigre clean will do the heavy lifting in TIGRE. This command will standardize the GFF files, so they can serve as input for the next commands. The standardization happens in five main ways: i) filtering of chosen genome features that have been determined by the user via the –query-string flag; ii) genome features are sorted (in ascending order) according to their start coordinates; iii) overlapping features are merged into "overlapping_feature_set" while retaining the identifying information of their delimiting components; iv) genome boundaries are treated in a way that allows tigre extract and tigre getfasta to obtain intergenic regions that might be wrapping around the start/end region of the genome (https://github.com/brenodupin/tigre/blob/master/PROCESSING.md); v) the names and positions of the delimiting genes are maintained in the attributes column for traceability. By default, gene names are retrieved from the "ID=" field in the attributes column of the original GFF file. Should the user want to standardize gene names across genomes, they can implement GDT - Gene Dictionary Tool - in this step. TIGRE supports gene dictionary files (.gdict) from the gdt library, and it has been vastly optimized for large input datasets and gene dictionaries.

⚠ CRITICAL: As an output, tigre clean will generate "clean" GFF3 files that contain only the genome features specified in the –query-string. In the example command, this flag is not passed, which means that the default value ("type in ('gene', 'tRNA', 'rRNA', 'region')") is used. You can change this value to include your genome features of interest. If you are not using the default of –query-string, make sure to maintain 'region' in your new –query-string list - e.g., –query-string "type in ('gene', 'region')". The genome features that have remained in the "clean" GFF will serve as signposts representing the genome regions that codify something (typically proteins). The subsequent steps of tigre will extract everything that falls outside (or in between) these sign posts (Figure 3).

Running tigre extract

Timing: Minutes (depending on the size of the dataset)

This section retrieves the intergenic regions from the "clean" GFF3 files. The intergenic regions are not exactly "extracted" but are "annotated" as intergenic regions.

5. Run tigre extract to obtain the intergenic regions from the clean GFF3 files:

```
>tigre extract multiple -v --log tigre_extract.log --tsv plants_mit.tsv --overwrite
```

Note: The output of tigre extract is the "<AN>_intergenic.gff3" files. These files are the "negative" of the clean GFF3 files. In the attributes column, each intergenic region is annotated with the identifying information of its delimiting genes (Figure 4). All intergenic regions are annotated as having positive polarity and they receive the name "intergenic region" under the type column.

⚠ CRITICAL: By default, the "region" element is not added to <AN>_intergenic.gff3. If you want to add this element to your intergenic GFF3 files, you should pass the –add-region flag.

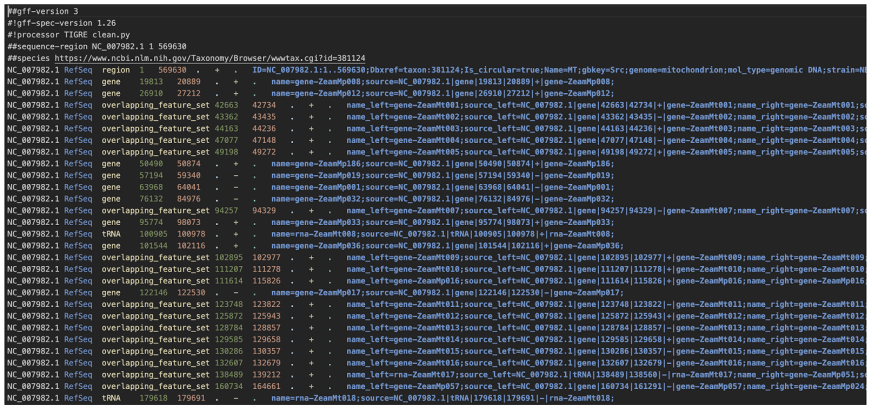


Figure 3. TIGRE's <>_clean.gff3 output file. TIGRE's <>_clean.gff3 file is a conventional gff3 file, but with a tailored attributes column

The information in the attributes column (i.e., name/name_left/name_right and source/source_left/source_right) is derived from the elements delimiting each intergenic region. This contextual information is carried over to the subsequent steps, so the user can pinpoint where each intergenic comes from (based not only on genomic coordinates, but also on flanking elements). Detailed explanation of the values from the attributes column can be found in the https://github.com/brenodupin/tigre/blob/master/PROCESSING.md.

> *Note:* tigre extract was designed to work with the clean GFF3 files generated by tigre clean. It is not certain whether tigre extract will produce proper results with GFF3 files that have not been formatted by tigre clean.

Running tigre getfasta

Timing: Seconds to minutes (depending on the size of the dataset)

This section fetches the nucleotide sequences of the annotated intergenic regions. This step requires the FASTA files of corresponding genomes.

6. Run tigre getfasta to generate FASTA files containing the intergenic regions from <AN>_intergenic.gff3:

```
>tigre getfasta multiple -v --log tigre_getfasta.log --tsv plants_mit.tsv --bedtools-compatible --overwrite
```

> *Note:* tigre getfasta is optional and requires as input the respective FASTA files containing the genomes being analyzed. Although this step is optional, most users might implement it to obtain the intergenic nucleotide sequences for downstream analyses. tigre getfasta relies on the properly annotated <AN>_intergenic.gff3 files (generated by tigre extract) to handle circular genomes that possess intergenic regions spanning the end/start of the genome.

> ⚠ CRITICAL: Even though biopython is implemented to extract the fasta sequences, the flag –bedtools-compatible generates fasta files that are compatible with BEDtools (i.e., the nucleotide sequences will be 0-based). This functionality will not alter the nucleotide sequence itself but only the header information presented in the header (after the ">" sign).

EXPECTED OUTCOMES

The expected outcomes of this protocol are the annotation and extraction of intergenic regions from any annotated genome. tigre clean will generate, with the optional help of GDT, "clean" GFF3 files

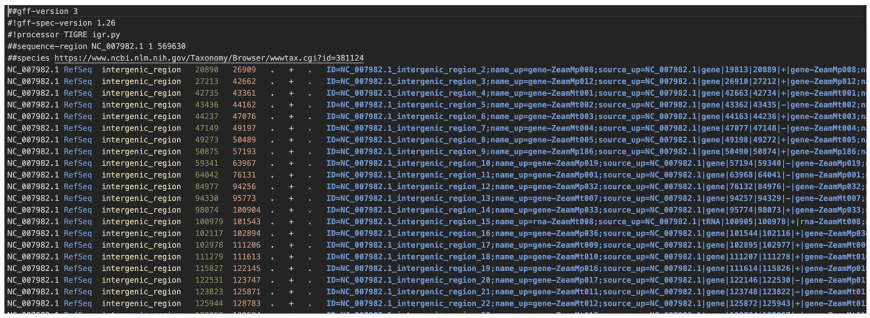


Figure 4. TIGRE's <>_intergenic.gff3 output file. TIGRE's <>_intergenic.gff3 file is a conventional GFF3 file containing only the intergenic regions of the inputted genome

Note that, just like in. <>_clean.gff3, the attributes column of <>_intergenic.gff3 is customized and carries the contextual information necessary for localizing each intergenic region. https://github.com/brenodupin/tigre/blob/master/PROCESSING.md contains further details on how TIGRE deals with elements (an intergenic region or otherwise) spanning the boundary of circular genomes.

formatted for the subsequent commands (tigre extract and tigre getfasta). The clean GFF3 files serve as templates to reveal where the intergenic regions are situated, including the complicated case of intergenic regions wrapping around the end/start boundaries of circular genomes. tigre extract generates "<>_intergenic.gff3" files that have the intergenic regions annotated and ready to be extracted by tigre getfasta. Via biopython, tigre getfasta can extract hundreds of thousands of intergenic sequences in a matter of seconds. With its user-centered design along with its multiprocessing capabilities, TIGRE is meant to facilitate large-scale comparative analyses of intergenic regions across disparate species.

One extra outcome of this protocol is the possible advanced usage of TIGRE to extract intronic regions. As introns represent another major component in the noncoding portions of genomes, the command tigre combine has been developed to be used as a workaround to extract introns. In a staggered approach via tigre clean and tigre combine, the user can create a "clean_to_introns" GFF3 file that will serve as input for the extraction of introns via tigre extract. See troubleshooting (and Advanced Usage in https://github.com/brenodupin/tigre) for more details. Note that this usage is experimental, as it is a process for the extraction (and not the identification) of introns from exonic annotations.

LIMITATIONS

TIGRE has two main limitations. The first is that this protocol was designed with GFF3 files that closely resemble the GFF3's from NCBI RefSeq. Essential parts of TIGRE rely on the existence (or absence) of elements common to these files, such as "region" being the feature type of the first line after the header, the presence of "is_circular=true" to confirm the circularity of a genome, and the field "ID=" in the column attributes to provide the names of the genes that delimit intergenic regions (regardless whether GDT is implemented or not). One could argue that TIGRE should have been designed with other annotation files in mind, such as BED and GTF, but these files do not have all the necessary information for the type of downstream analyses that TIGRE permits. GTF's (which are identical to GFF2 - https://gmod.org/wiki/GFF2) contain hierarchical information up to only two levels (exons to transcripts and transcripts to genes). Meanwhile, intervening sequences can be easily extract from BED files with BEDtools via bedtools complement (https://bedtools.readthedocs.io/en/latest/content/tools/complement.html). But again, no identifying information of the delimiting features can be extracted from a BED file.

Regarding file structure, _intergenic.gff3 files are not readily "mergeable" to a conventional GFF3 file. This is because tigre extract outputs GFF3 files that follow the format created by tigre clean. Particularly, the clean GFF3 attributes column is very different than the attributes column found in a conventional GFF3 (from having information such as name_up/source_up and name_dw/source_dw to having every line ending in ";", which is not always the case for regular GFF3 files).

TROUBLESHOOTING

Problem 1

What if I want to extract introns as well?

Potential solution

With the _intergenic.gff3 in hand, merge the _intergenic.gff3 with the original GFF using the tigre combine command. In the combined file(s), rerun tigre clean with the following modified flags.

```
--query-string "type in ('exon', 'intergenic region', 'tRNA', 'rRNA')''

--gff-in-suffix "_combined''

--gff-out-suffix "_clean_to_introns''

--extended-filtering
```

This will create a clean GFF3 suitable for the extraction of introns only (via tigre extract and, possibly, tigre getfasta).

> *Note:* –gff-in-suffix as "_combined" refers to the default output of tigre combine. –gff-out-suffix as "_clean_to_introns" is a recommendation for the output suffix, just so the output of this tigre clean iteration does not get mixed up with previous outputs of tigre clean. Note that "_clean_to_introns" must be passed as –gff-in-suffix for tigre extract (to extract the intronic regions). More details of this implementation can be found at https://github.com/brenodupin/tigre under Advanced Usage.

Problem 2

I followed this protocol using the example dataset (plants_mit). How can I compare my results with the results stored in the expected outcome folder (https://github.com/brenodupin/tigre/tree/master/examples)?

Potential solution

You should download plants_mit_expected_outome.tar.zst (preferably in the same directory where the plants_mit folder is stored). Then, you should decompress the expected outcome folder with >tar -xf plants_mit_expected_outcome.tar.zst. After that, you can compare your results with the expected outcomes by running >diff -rq plants_mit/plants_mit_expected_outcome/. If your results are identical to the expected outcome, the only differences should appear in the log files (as they are time stamped).

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Matheus Sanita Lima (msanital@uwo.ca).

Technical contact

Technical questions on executing this protocol should be directed to and will be answered by the technical contact, Breno Dupin (breno.dupin@gmail.com).

Materials availability

Not applicable.

Data and code availability

The datasets and code generated during this study are available at GitHub: https://github.com/brenodupin/tigre and Zenodo: https://zenodo.org/records/17475986.

ACKNOWLEDGMENTS

We would like to kindly thank Dr. Igor Fesenko for swiftly responding to our inquiries when we started investigating organelle intergenic sequences. TIGRE started off as an adaptation of Dr. Fesenko's protocol for the extraction of bacterial intergenic sequences. We would also like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fundação Araucária and CNPq for financial support granted to D.R.S. (via a NSERC Discovery Grant) and to A.R.P. (via NAPI Bioinformática 66.2021 and 440412/2022-6; Fundação Araucária and CNPq, respectively).

AUTHOR CONTRIBUTIONS

Conceptualization, M.S.L., B.D., D.R.S., and A.R.P.; software and methodology, B.D.; data curation, M.S.L. and B.D.; writing – original draft, M.S.L. and B.D.; writing – review and editing, M.S.L., B.D., D.R.S., and A.R.P.; funding acquisition and supervision, D.R.S. and A.R.P.

DECLARATION OF INTERESTS

The authors declare no competing interests.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work, the authors used Claude AI and ChatGPT for script optimization and documentation, code formatting, GitHub backend configuration, tool logo generation, and alike. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

REFERENCES

1. Eddy, S.R. (2013). The ENCODE project: Missteps overshadowing a success. Curr. Biol. *23*, R259–R261.

2. Graur, D., Zheng, Y., Price, N., Azevedo, R.B.R., Zufall, R.A., and Elhaik, E. (2013). On the immortality of television sets: "function" in the human genome according to the evolution-free gospel of ENCODE. Genome Biol. Evol. *5*, 578–590.

3. Doolittle, W.F. (2013). Is junk DNA bunk? A critique of ENCODE. Proc. Natl. Acad. Sci. USA *110*, 5294–5300.

4. Lynch, M., and Conery, J.S. (2003). The origins of genome complexity. Science *302*, 1401–1404.

5. Xu, H., Li, C., Xu, C., and Zhang, J. (2023). Chance promoter activities illuminate the origins of eukaryotic intergenic transcriptions. Nat. Commun. *14*, 1826.

6. Sanita Lima, M., and Smith, D.R. (2017). Pervasive transcription of mitochondrial, plastid, and nucleomorph genomes across diverse plastid-bearing species. Genome Biol. Evol. *9*, 2650–2657.

7. Werner, A., Kanhere, A., Wahlestedt, C., and Mattick, J.S. (2024). Natural antisense transcripts as versatile regulators of gene expression. Nat. Rev. Genet. *25*, 730–744.

8. Attallah, C., Conti, G., Zuljan, F., Zavallo, D., and Ariel, F. (2025). Noncoding RNAs as tools for advancing translational biology in plants. Plant Cell *37*, koaf054.

9. Narunsky, A., Higgs, G.A., Torres, B.M., Yu, D., de Andrade, G.B., Kavita, K., and Breaker, R.R. (2024). The discovery of novel noncoding RNAs in 50 bacterial genomes. Nucleic Acids Res. *52*, 5152–5165.

10. Lynch, M. (2006). Streamlining and simplification of microbial genome architecture. Annu. Rev. Microbiol. *60*, 327–349.

11. Wu, P., Xue, N., Yang, J., Zhang, Q., Sun, Y., and Zhang, W. (2025). OGU: a toolbox for better utilising organelle genomic data. Mol. Ecol. Resour. *25*, e14044.

12. Fesenko, I., Sahakyan, H., Dhyani, R., Shabalina, S.A., Storz, G., and Koonin, E.V. (2025). The hidden bacterial microproteome. Mol. Cell *85*, 1024–1041.e6.

13. Thorpe, H.A., Bayliss, S.C., Hurst, L.D., and Feil, E.J. (2017). Comparative analyses of selection operating on nontranslated intergenic regions of diverse bacterial species. Genetics *206*, 363–376.

14. Tsai, C.-H., Liao, R., Chou, B., Palumbo, M., and Contreras, L.M. (2015). Genome-wide analyses in bacteria show small-RNA enrichment for long and conserved intergenic regions. J. Bacteriol. *197*, 40–50.

15. Dupin, B., Sanita Lima, M., Rossi Paschoal, A., and Smith, D.R. (2025). Protocol for GDT, Gene Dictionary Tool, to create and implement a gene dictionary across annotated genomes. STAR Protocols 6, 104187. https://doi.org/10.1016/j.xpro.2025.104187.

16. Cock, P.J.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M.J.L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics *25*, 1422–1423.