Protocol for GDT, Gene Dictionary Tool, to create and implement a gene dictionary across annotated genomes

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

[1]Department of Computer Science, Bioinformatics and Pattern Recognition Group (BIOINFO-CP), Federal University of T

UTFPR, Cornélio Procópio, PR, Brazil

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

[3]Technical contact

[4]Lead contact

[5]These authors contributed equally

*Correspondence: breno.dupin@gmail.com

**Correspondence: msanital@uwo.ca

Summary

GDT - Gene Dictionary Tool - is a protocol for the creation and implementation of a gene dictionary across any type of annotated genomes. The final output, a gene dictionary (.gdt file), is meant to be not only comprehensive, but also versionable, Findable, Accessible, Interoperable, and Reusable (FAIR). The protocol is divided in a Python library and Jupyter notebooks that permit users to investigate their datasets incrementally. Here, we showcase GDT's functionalities by creating a gene dictionary of organelle genes.

Before you begin

The deluge of publicly available Next Generation Sequencing (NGS) transcriptomic data has been dubbed a "goldmine for organelle research"[1], and a similar conclusion can be drawn from DNA-Seq experiments[2]. Greatly because of this data affluence, organelle genomes currently are the most sequenced type of chromosomes[3]. Even though there exist inherent biases in taxonomic sampling[4], comparative analyses of organelle genomes have informed all-encompassing theories of genome evolution[5], shed light on what genome features explain organelle gene retention[6], and shown how selection can act on some of these cytoplasmic genetic systems[7].

Even with such popularity and scientific importance, the field of organelle genomics suffers from the lack of gene nomenclature standardization[8]. Attempts to bring order and consistency in organelle gene nomenclature date back to at least 1983[9], and currently, the HUGO Gene

40 Nomenclature Commitee (HGNC) has a clear taxonomy for human mitochondrial gene
41 names[10]. Still, anyone trying to analyse organelle genomes across different groups will be faced
42 with a cornucopia of inconsistent names that can confound even experts in the field. In fact,
43 recent large-scale comparative organelle genome analyses[6,7,11,12] have repeatedly highlighted,
44 in one way or another, the perils of and challenges brought by organelle gene nomenclature
45 disparities.
46
47 Through our own analyses of organelle genome across eukaryotes, we quickly identified the
48 need of (and opportunity for) consolidating the various gene nomenclature standardization
49 efforts. As Biology increasingly becomes a Big-Data endeavour and morphs into an eScience,
50 we have used the FAIR principles (Findability, Accessibility, Interoperability, and Reusability)[13]
51 in the design of our protocol. Here, we present GDT - Gene Dictionary Tool - a protocol for the
52 creation and implementation of a gene dictionary that is not only comprehensive, but
53 versionable and FAIR-adherent. Compared to previous organelle gene dictionary attempts,
54 GDT exhibits unmatched interoperability. Its main output, a .gdt file, has a parseable syntax
55 and a structure that is inherently iterative (i.e., versionable). Detailed and human-readable log
56 documentation lets any user make sense of the data. The standardized gene nomenclature
57 present in our .gdt files (available at our GitHub repository) is greatly reliant on the well-
58 defined structure of the HGNC-approved mitochondrial gene symbols. The user can either
59 adopt these suggested gene symbols or change them as needed. Despite this high level of
60 customizability, we hope that other researchers will adopt the HGNC-informed symbols for the

sake of consistency. Periodic updates of the .gdt files will be released, as more organelle genomes (with inconsistent gene names) are sequenced and published.

Overview

GDT - Gene Dictionary Tool - is a protocol that serves two main purposes: it creates a comprehensive gene dictionary and, once the dictionary is created, GDT applies the standardized gene nomenclature on any genome of interest. The gene dictionary creation process takes as input GFF3 files (with a tsv file as their index) and, optionally, a pre-built .gdt file (in its “stripped” version). Creating the gene dictionary depends on retrieving identifying information from the respective GFF3 files and NCBI Gene queries, as well as input from the user (**Figure 1**). Note that the GDT does not correct annotations (as it does not rely on BLAST searchers), but rather it creates an expandable and customizable gene dictionary that allows the user to standardize gene names across disparate species.

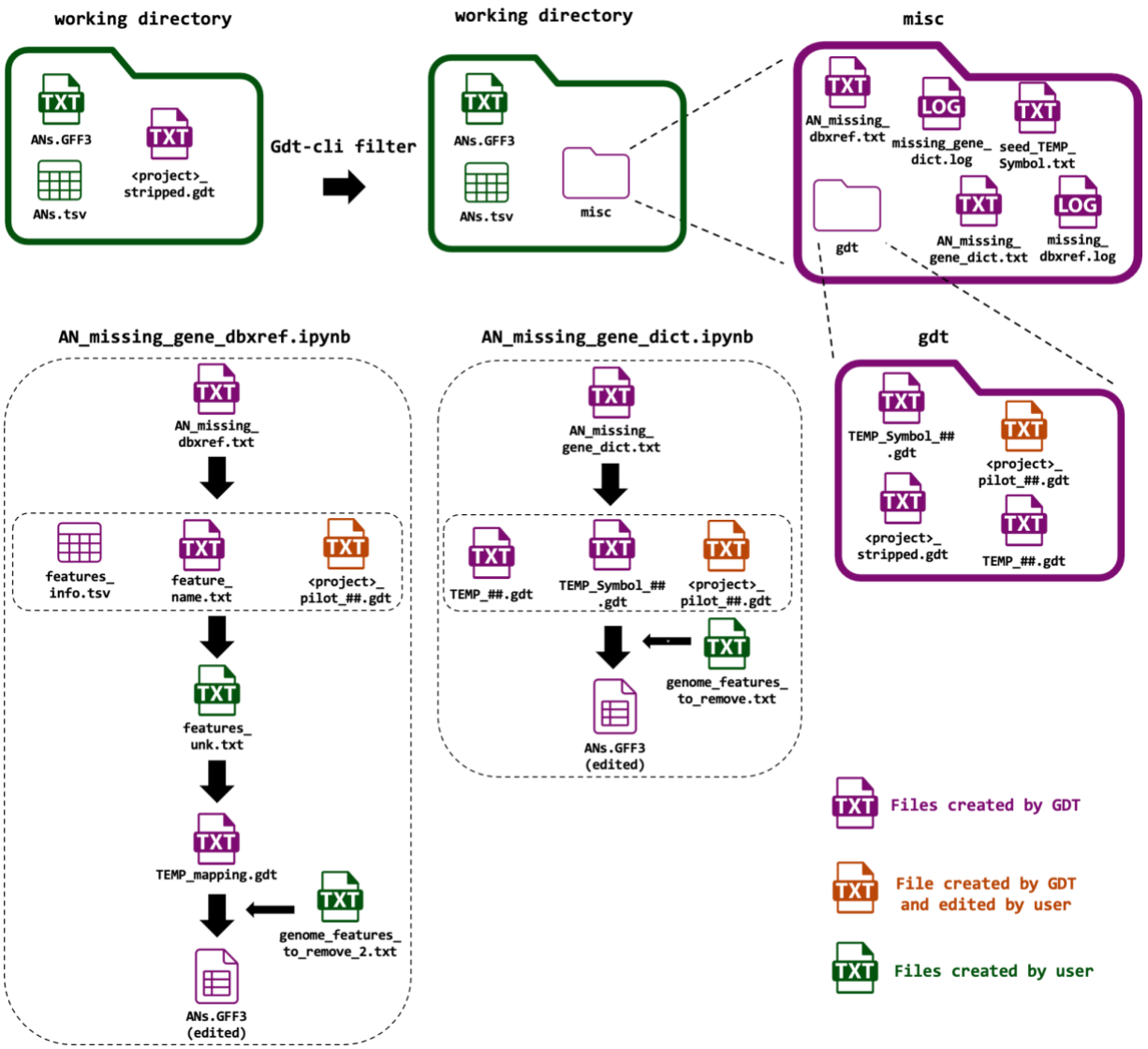

Figure 1. Overview of the GDT protocol.

Creating and implementing a gene dictionary is a multipronged process that relies on extensive user input. This diagram depicts all intermediate and final output files that can be generated by the protocol.

Download the software and prerequisites

GDT is composed of four Python files, accompanied by two accessory Jupyter Notebooks. All files can be downloaded from https://github.com/brenodupin/gdt. The Python files comprise the gdt library that allows the user to interface with the .gdt files (**Figure 2**), whereas the accompanying Jupyter Notebooks let the user iterate over NCBI Gene queries and create new .gdt versions. The library can be installed locally using pip (see Step-by-Step method details), which will automatically install all necessary dependencies. For the Jupyter Notebooks, the user needs to install the necessary dependencies (biopython, pandas, gdt) separately. The pipeline requires a Unix environment with Python 3.10 (or above).

> ***Note:*** specific hardware requirements, such as RAM memory and storage capacity, will depend on the size of the dataset at hand. In principle, a common personal computer should have enough power to run gdt and analyse even thousands of GFF3 files (of small genomes).

```
#! version 0.0.2
#! Fungi_mt
#! 2025-04-09 17:54 - Conversion from gdt to gdt2
#! 2025-06-05 18:11 - Stripped GDT version from original GDT file
Fungi_mt.gdt
#! 2025-06-11 20:37 - Data added from TEMP 01
#! 2025-06-11 20:50 - Data added from TEMP Symbol 01
#! 2025-06-14 16:00 - Data added from 'Automated insertion of gene_ids
with known features from features_info.tsv'
#! 2025-06-14 16:02 - Data added from 'AN_missing_dbxref_GeneID
matching 'child + parent' best feature pair'

[MIT--ATP6]
ATP6 #gd MANUAL
atp6 #gd NCBI
Atp6 #gd NCBI
ATP6 protein #gd NCBI
Atp6p #gd NCBI

...

gene-AFUA_m0110 #gn JQ346808.1 #c insertion from missing_dbxref_GeneID
feature mapping, source: ATP synthase subunit 6 | type: CDS
gene-atp6 #gn HE983611.1 #c automated insertion from
missing_dbxref_GeneID best feature

...

gene-FDY65_mgp14 #dx NC_042229.1:40135497 #c gff_gene: atp6
gene-ICI09_mgp12 #dx NC_050344.1:58907367 #c gff_gene: atp6
gene-N4M62_mgp09 #dx NC_066216.1:74863486 #c gff_gene: atp6
gene-Q0085 #dx NC_001224.1:854601 #c gff_gene: ATP6
```

header containing metadata

standardized gene label (based on HGNC style)

genome feature descriptions equivalent to the gene label above

#gd = gene description

#gn = gene generic

#dx = gene dbxref

#c = commentary

Figure 2. Example of a .gdt (Gene Dictionary) file.

In this example, this .gdt is in its final version after multiple iterations through the protocol. A .gdt file is a text file, and it is heavily inspired by the TOML structure. As a FAIR-informed file type, .gdt starts with a header containing metadata for traceability. Version number, time stamp, and the origin of the file are generated automatically by GDT. Group name (Fungi_mt in the example) is inputted by the user directly in the file. The actual dictionary starts right below the header and is read top-to-bottom, organized by gene labels (e.g., [MT-ATP6]). For each gene label, genome features are classified (and sorted) according to the origin of their identifying information. The classification scheme is as such: “#gd” = GeneDescription, “#gn” = GeneGeneric, and “#dx” = DbxrefGeneID. The .gdt file also takes in commentaries in the “#c” field.

Materials and Equipment

The GDT tool is fully written in Python 3. The gdt library is composed of four Python scripts: cli.py, gdt_impl.py, gff3_utils.py, and logger_setup.py. The CLI (command-line interface) lets the user interact with the gdt library in a user-friendly way (**Figure 3**). The two accompanying Jupyter Notebooks (AN_missing_gene_dict.ipynb and AN_missing_dbxref_GeneID.ipynb) are meant to give the user the functionalities necessary to build their own .gdt files iteratively. pandas and biopython are the only major external dependencies. NCBI Gene Database can be queried extensively or not, depending on what type of information the input dataset has. All necessary dependencies are listed in the **KEY RESOURCES TABLE**. GDT's source code can be found at https://github.com/brenodupin/gdt.

Key Resources Table

Software and Algorithms

REAGENT or RESOURCE	SOURCE	IDENTIFIER
gdt library, AN_missing_gene_dict.ipynb and AN_missing_Dbxref_GeneID.ipynb	This manuscript	N/A
Python 3.10	Python Software Foundation	https://www.python.org
Biopython 1.85	Cock et al.[14]	https://biopython.org/
Pandas 2.2.3	The Pandas Development Team	https://pandas.pydata.org/
Jupyter Notebook	Project Jupyter	https://jupyter.org/

NCBI-Gene	Sayers et al.[15]	https://www.ncbi.nlm.nih.gov/gene/

Deposited Data

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Curated gene dictionary (.gdt) files	This manuscript	Will be provided upon publication
Examples of edited GFF3 files	This manuscript	Will be provided upon publication

A

```
❯ gdt-cli --help
usage: gdt-cli [-h] [--debug] [--log LOG] [--quiet] [--version] {filter,stripped,standardize} ...

GDT
Gene Dictionary Tool

🧬 Standardizing gene names across organelle genomes 🧬
Version: 0.1.3

positional arguments:
  {filter,stripped,standardize}
    filter              Filter GFF3 files indexed through TSV file.
    stripped            Create a stripped version of a GDT file
    standardize         Standardize gene names in GFF3 files.

options:
  -h, --help            show this help message and exit
  --debug               Enable TRACE level in file, and DEBUG on console. Default: DEBUG level on file and INFO on console.
  --log LOG             Path to the log file. If not provided, a default log file will be created.
  --quiet               Suppress console output. Default: console output enabled.
  --version             Show the version of the gdt package.

Source ~ https://github.com/brenodupin/gdt
```

B

```
❯ gdt-cli filter --help
usage: gdt-cli filter [-h] [--debug] [--log LOG] [--quiet] --tsv TSV [--AN-column AN_COLUMN] [--gdt GDT] [--keep-orfs]
                      [--workers WORKERS] [--gff-suffix GFF_SUFFIX] [--query-string QUERY_STRING]

Filter GFF3 files that are indexed via TSV file, creating AN_missing_dbxref.txt and/or AN_missing_gene_dict.txt based on
the provided GDT file.

options:
  -h, --help            show this help message and exit
  --debug               Enable TRACE level in file, and DEBUG on console. Default: DEBUG level on file and INFO on
                        console.
  --log LOG             Path to the log file. If not provided, a default log file will be created.
  --quiet               Suppress console output. Default: console output enabled.
  --tsv TSV             TSV file with indexed GFF3 files to filter.
  --AN-column AN_COLUMN
                        Column name for NCBI Accession Number inside the TSV. Default: AN
  --gdt GDT             GDT file to use for filtering. If not provided, an empty GeneDict (i.e GDT file) will be used.
  --keep-orfs           Keep ORFs. Default: exclude ORFs
  --workers WORKERS     Number of workers to use. Default: 0 (use all available cores: 12)
  --gff-suffix GFF_SUFFIX
                        Suffix for GFF files. Default: '.gff3'
  --query-string QUERY_STRING
                        Query string that pandas filter features in GFF. Default: 'type in ('gene', 'tRNA', 'rRNA')'
```

C

```
❯ gdt-cli standardize --help
usage: gdt-cli standardize [-h] [--debug] [--log LOG] [--quiet] (--gff GFF | --tsv TSV) --gdt GDT [--AN-column AN_COLUMN]
                           [--gff-suffix GFF_SUFFIX] [--query-string QUERY_STRING] [--check] [--second-place] [--gdt-tag GDT_TAG]
                           [--error-on-missing] [--save-copy]

Standardize gene names across features in GFF3 files using a GDT file.

options:
  -h, --help            show this help message and exit
  --debug               Enable TRACE level in file, and DEBUG on console. Default: DEBUG level on file and INFO on console.
  --log LOG             Path to the log file. If not provided, a default log file will be created.
  --quiet               Suppress console output. Default: console output enabled.
  --gff GFF             GFF3 file to standardize.
  --tsv TSV             TSV file with indexed GFF3 files to standardize.
  --gdt GDT             GDT file to use for standardization.
  --AN-column AN_COLUMN
                        Column name for NCBI Accession Number inside the TSV. Default: AN
  --gff-suffix GFF_SUFFIX
                        Suffix for GFF files. Default: '.gff3'
  --query-string QUERY_STRING
                        Query string that pandas filter features in GFF. Default: 'type in ('gene', 'tRNA', 'rRNA')'
  --check               Just check for standardization issues, do not modify the GFF3 file. Default: False
  --second-place        Add gdt-tag pair to the second place in the GFF3 file, after the ID. Default: False (add to the end of the
                        attributes field).
  --gdt-tag GDT_TAG     Tag to use for the GDT key/value pair in the GFF3 file. Default: 'gdt_label='.
  --error-on-missing    Raise an error if a feature is missing in the GDT file. Default: False (just log a warning and skip the
                        feature).
  --save-copy           Save a copy of the original GFF3 file with a .original suffix. Default: False (change inplace).
```

125

Figure 3. GDT's command-line interface (CLI).

(A) GDT's CLI help page.

(B) gdt-cli filter –help output.

(C) gdt-cli standardize –help output.

GDT's CLI lets the user run the gdt library commands "filter", "stripped", and "standardize" without the need to edit the source code. This interface is extensively documented with detailed help messages to facilitate the use of GDT. The flag "–help" should be used anytime more information on GDT's main functionalities is needed.

Step-by-step method details

Preparing the environment and files

Timing: ~ 30 min

1. Install GDT and biopython using pip, as shown in the following command:

```
>pip install gdt biopython
```

2. Download the Jupyter Notebooks from https://github.com/brenodupin/gdt to your chosen working directory.

Note: you will need an IDE (Integrated Development Enviroment) to run the Jupyter Notebooks. For more information, consult Jupyter's official documentation at: https://docs.jupyter.org/en/latest/#where-do-i-start.

3. Compile a tsv file with the NCBI Accession Numbers of your genomes of interest. Save this tsv file in your chosen working directory, and download all the corresponding GFF3 files into this folder.

CRITICAL: We have abbreviated Accession Number to AN (in the tsv file), but any other name can be used to refer to Accession Number, as long as the chosen name is passed to the argument –AN-column in gdt-cli filter. The GFF3 files must be named as <AN.gff3>, where AN = "Accession Number" (as written in your tsv file). The .gff3 extension can be changed (e.g., .gff), as long as the correct file extension is added to the flag –gff-suffix in GDT's CLI, and Jupyter Notebooks. Consistency is key.

Optional: Should you decide to build a dictionary from our stripped version (or from a previous version of any dictionary), the .gdt file should also be added to the same working directory. gdt-cli filter command will automatically move this .gdt file to the right subfolder for downstream analyses.

Running gdt-cli filter –tsv –gdt

Timing: ~ 3 min (depending on dataset size)

4. Inside your working direct, run gdt-cli filter command (**Figure 4**):

```
>gdt-cli filter --tsv fungi_mt_model2.tsv --gdt fungi_mt_model2_stripped.gdt --debug
```

Note: For a smoother experience, the command gdt-cli filter should be run in the terminal console from the working directory where the GFF3 and tsv files are located. Alternatively, gdt-cli filter can be called from elsewhere, but –tsv and –gdt flags will take in either the absolute or relative paths to the respective files. In this case, the tsv file's parent folder will be considered the working directory. gdt-cli filter has a –debug functionality that will facilitate and improve your debugging process. This flag is optional.

For each Accession Number listed in the tsv file, gdt-cli filter will scan through the respective GFF3 file and try to match its genome features to the genome features listed in the inputted .gdt. Assuming that you start your analysis with an incomplete gene dictionary (e.g., a stripped .gdt), some of your GFF3 files will likely have genome features that are not listed in your initial .gdt. There are then two possibilities: a given Accession Number will be added to AN_missing_gene_dict.txt only when all genome features of the GFF3 file have Dbxref

GeneIDs. If at least one genome feature does not have Dbxref GeneID, the corresponding Accession Number will be listed in AN_missing_dbxref_GeneID.txt. These two files will be used subsequently in the accompanying Jupyter Notebooks (each named after the gdt-cli filter output file names).

Note: The default genome feature types that will be searched in each GFF file are "gene", "tRNA", and "rRNA". Should you want to change the default (e.g., to search "gene" only), you can change the –query-string flag as in: "type == 'gene'" or "type in ('gene', 'tRNA')".

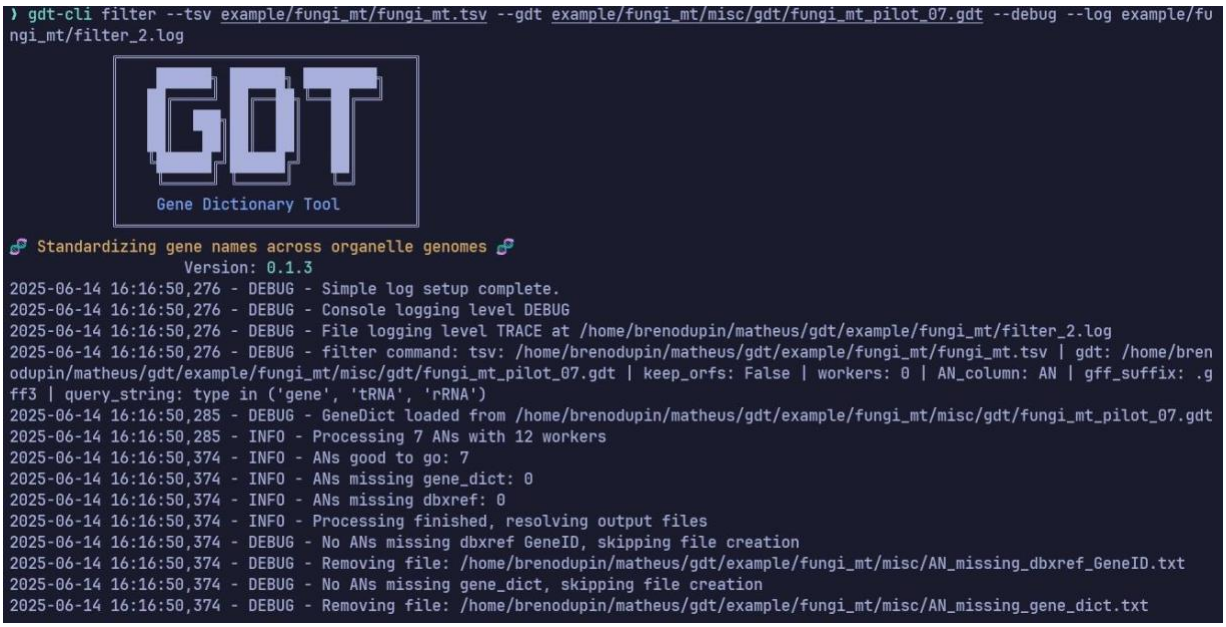

Figure 4. gdt-cli filter command example.

In this example, the –debug flag is used. Note the detailed (and user-friendly) logging that is provided when –debug is active. The default mode of –debug will print onto the console screen log information at DEBUG level and save log information into a separate log file at TRACE level.

Immediately after running gdt-cli filter, the user knows how many genomes (i.e., ANs = Accession Numbers) are "good to go" (i.e., all chosen genome features in these genomes are already listed in the inputted .gdt), how many genomes have genome features that are not listed in the gene dictionary, but have Dbxref GeneIDs ("ANs missing gene_dict:"), and how many genomes have genome features that are not in the gene dictionary and lack Dbxref GeneIDs ("ANs missing dbxref:").

Running AN_missing_gene_dict.ipynb

Timing: ~ hours to days (depending on dataset size and time spent in manual curation)

5. Run the first cell (Imports and functions) to import all necessary dependencies, and instantiate all required global variables and related functions.

   Note: In both Jupyter Notebooks, we recommend running each cell sequentially, one by one, instead of running all at once. This will ensure that processing errors are caught when needed, and it will avoid your program to crash.

6. In the cell A. of the Setup section, edit the following variables as necessary: DATA_DIR, most_recent_gdt_filename, global_query_string, remove_orfs, gct, gff_sufix, Entrez.email, and Entrez.api_key.

a. global_query_string is pre-defined in the gdt library with two possible values: QS_GENE ("type == 'gene'") or QS_GENE_TRNA_RRNA ("type in ('gene', 'tRNA', 'rRNA')"). This variable will determine what type(s) of genome features will be analyzed. If you would like to add a new genome feature (e.g., exon, ncRNA, etc), you should add the new type name to the list the defines this variable.

CRITICAL: If you change global_query_string, you must restart the process. The new global_query_string should be passed to gdt-cli filter as the flag –query-string. Consistency is key.

b. most_recent_gdt_filename represents the latest iteration of the gdt creation process. This variable must be updated, as you iterate through the protocol and update your working .gdt. It is highly recommended that you start your analysis with the _stripped.gdt version (of the appropriate organism group) provided in the GitHub page.

CRITICAL: If you are using the stripped version of the gdt, the file name must contain the suffix "stripped" (as in fungi_mt_stripped.gdt), otherwise the protocol will crash. The "most_recent_gdt_filename =" variable contains just the name of the respective file inside the gdt folder. most_recent_gdt is not mandatory only in the first pass of either AN_missing_gene_dictionary or

AN_missing_dbxref_GeneID. Whenever a new version of the .gdt file is being created, you must update “most_recent_gdt_filename =” to the latest iteration of your .gdt and run cell A. of the Setup section again. This applies to both Jupyter Notebooks.

c. gct stands for “genetic compartment”. In its current version, we have adopted “MIT” for mitochondria and “PLT” for plastid. These acronyms are added as a prefix to the gene labels as in [MIT-ATP6]. You can change this prefix, as you see fit.

d. Entrez.email and Entrez.api_key are private and specific to each user. These values are obtained from the user’s NCBI account. The email is mandatory, regardless the size of the dataset, whereas the NCBI API key is highly recommended when analysing large(r) datasets. The NCBI API key gives the user a higher API request rate.

7. In cell B. of the Setup section, you must set up a mandatory logger for debugging. Log files are stored in the misc folder. If you wish not to have a log file, you can opt out by changing “print_to_console=” and “save_to_file=” to False:

8. In the section “TEMP First Pass”, run all cells (from A. to E.).

a. Cell A.: the AN_missing_dict.txt will be loaded.

b. Cell B.: the .gdt file (stored in the most_recent_gdt_filename) will be loaded onto the variable gene_dict.

c. Cell C: the next cell composes the core process of this step. Briefly, each GFF3 file will be loaded, and scanned for the chosen features (e.g., “gene”). If the corresponding feature ID (found in “ID=”) is already present in the loaded .gdt, the feature is skipped. Otherwise, the values of “gene=” (or “Dbxref=GeneID”) will be used to match the respective feature to .gdt’s gene labels.

Pause point: after having run cell C., you should pause and check whether any errors have been listed at the end of your log file. These errors can be of type “File not found” or “Entrez.read”. The log messages will help you troubleshoot.

d. Cells D. and E.: the next two cells write and save the temporary gene dictionary (temp_gene_dict) and the latest iteration of the gene dictionary (gene_dict) (**Figure 5**).

```
2025-06-14 15:28:02,088 - DEBUG - Simple log setup complete.
2025-06-14 15:28:02,093 - DEBUG - Console logging level DEBUG
2025-06-14 15:28:02,099 - DEBUG - File logging level TRACE at
/home/brenodupin/matheus/gdt/example/fungi_mt/misc/01_missing_gene_dict.log
2025-06-14 15:28:02,100 - DEBUG - Running from notebook AN_missing_gene_dict.ipynb
2025-06-14 15:28:03,886 - INFO - Found 4 ANs in /home/brenodupin/matheus/gdt/example/fungi_mt/misc/AN_missing_gene_dict.txt
2025-06-14 15:28:03,888 - TRACE - ANs: ['NC_042229.1', 'NC_050344.1', 'NC_066216.1', 'NC_001224.1']
2025-06-14 15:28:04,825 - INFO - GeneDict loaded from fungi_mt_stripped.gdt
2025-06-14 15:28:04,826 - DEBUG - path: /home/brenodupin/matheus/gdt/example/fungi_mt/misc/gdt/fungi_mt_stripped.gdt
2025-06-14 15:28:04,827 - INFO - Header:
2025-06-14 15:28:04,828 - INFO -            version 0.0.2
2025-06-14 15:28:04,829 - INFO -            Fungi_mt
2025-06-14 15:28:04,829 - INFO -            2025-04-09 17:54 - Conversion from gdt to gdt2
2025-06-14 15:28:04,830 - INFO -            2025-06-05 18:11 - Stripped GDT version from original GDT file Fungi_mt.gdt
2025-06-14 15:28:04,830 - INFO - GDT Info:
2025-06-14 15:28:04,831 - INFO -            Labels: 54
2025-06-14 15:28:04,832 - INFO -            Total Entries   : 509
2025-06-14 15:28:04,832 - INFO -            GeneDescriptions: 509
2025-06-14 15:28:04,833 - INFO -            GeneGenerics    : 0
2025-06-14 15:28:04,833 - INFO -            DbxrefGeneIDs   : 0
2025-06-14 15:28:06,502 - INFO -  ---- [Starting 'AN_missing_gene_dict parsing gene= + NCBI Description'] ----
2025-06-14 15:28:06,504 - DEBUG - -- [Processing: NC_042229.1] --
2025-06-14 15:28:06,515 - DEBUG - Found 44 feature(s) with gene= in NC_042229.1, not in gene_dict.
2025-06-14 15:28:06,517 - DEBUG - gene_id: gene-FDY65_mgp44 | dbxref: 40135512 | att: ID=gene-
FDY65_mgp44;Dbxref=GeneID:40135512;Name=hyp3;gbkey=Gene;gene=hyp3;gene_biotype=protein_coding;locus_tag=FDY65_mgp44
2025-06-14 15:28:06,518 - TRACE -           check_var = 'hyp3' | use_gene = True | use_NCBI_symbol = False
2025-06-14 15:28:06,518 - TRACE -           row.other_aliases = 'no_other_aliases' | row.desc = 'no_description' | row.gene_symbol =
'no_gene_symbol'
2025-06-14 15:28:06,518 - TRACE -           [1st F]check_var not found in gene_dict | checking temp_gene | Label: hyp3
2025-06-14 15:28:06,518 - DEBUG -           [2nd F]check_var not found in temp_gene, new label |MIT-TEMP-1| adding: gene-FDY65_mgp44 #dx
NC_042229.1:40135512 #c gff_gene: hyp3
```

Figure 5. initial output log file of AN_missing_gene_dict.ipynb

Human-readable and machine-parseable log output that shows the user how each genome feature was parsed. Note the detailed header with information on what .gdt iteration is being loaded (stripped or pilot_XX), how many genomes are being analysed (in this case, 4 Accession Numbers listed in AN_missing_gene_dict.txt) and the structure of the loaded dictionary: its number of entries (gene dictionary length), number of standardized gene labels, and number of #gd, #gn, and #dx entries. The genome features that are left unmatched will be added to the TEMP_Description_XX.gdt file (see next step).

9. Now manually parse the TEMP_Description_##.gdt (**Figure 6**).
    (a) If you can classify the genome feature with the information obtained from "gff_gene" and/or "ncbi_desc", you should copy the respective line(s) from TEMP_Description_##.gdt to your latest .gdt.

296 (b) If you cannot classify the genome feature with the provided information, you
297 should copy the line(s) from TEMP_Description_##.gdt to a new file named
298 seed_TEMP_Symbol.txt.
299
300 ***Note:*** For better traceability, we suggest that you maintain the output files as they are
301 created and create/save new files as you progress through the analysis. As you iterate
302 through the protocol and manually add new entries to your .gdt, we highly recommend
303 you to save each new pilot_##.gd and name each version incrementally as in
304 pilot_01.gdt, pilot_02.gdt, etc.
305

```
#! version 0.0.2
#! TEMP_Description - 1
#! 2025-06-14 15:28 - Automatically generated from 'AN_missing_gene_dict parsing
gene= + NCBI Description'

[MIT-TEMP-1]
hyp3 #gd gff_gene
gene-FDY65_mgp44 #dx NC_042229.1:40135512 #c gff_gene: hyp3

...

[MIT-TEMP-26]
trnN-ATT #gd gff_gene
gene-N4M62_mgt03 #dx NC_066216.1:74863481 #c gff_gene: trnN-ATT
rna-N4M62_mgt03 #dx NC_066216.1:74863481 #c gff_gene: trnN-ATT
...

[MIT-TEMP-58]
OLI1 #gd gff_gene
gene-Q0130 #dx NC_001224.1:854584 #c gff_gene: OLI1
```

306
307 **Figure 6. Excerpts of our TEMP_Description_01.gdt and its respective log information.**

308 (A) Just like any other .gdt file, the TEMP_Description_XX.gdt has a time-stamped
309 incremental header that indicates when and how this file was created (i.e.,
310 automatically from AN_missing_gene_dict.txt).
311
312 10. Now update the most_recent_gdt_filename to the latest .gdt you just edited. Then,
313 run the "Setup" (cell A.) again.
314
315 11. If you end up having the seed_TEMP_Symbol.txt, run "TEMP Second Pass" section.
316
317 (a) This process follows the same rationale as described in item 8. The only
318 difference is that the identifying information is obtained from "NCBI gene
319 symbol" field, instead of "NCBI gene description" field.
320 (b) The two final cells of this section will write (and save) a temporary gene
321 dictionary (temp_symbol_gene_dict) and automatically update the latest gene
322 dictionary (gene_dict).
323
324 12. Now manually parse the TEMP_Symbol_##.gdt.
325 a) The rational here is the same as in item 9. However, unclassifiable genome
326 features should either be added to a new file called to_exclude.txt or the whole
327 genome should be removed from the dataset.
328

13. If you end up creating to_exclude.txt, run cells A. and B. of the final section ("Gene exclusion of to_remove.txt).

Note: This step will add (directly into the original GFF3 files) the tag "discard-" before the genome feature(s) to be excluded. This will guarantee that these features will not be included in any future iteration of GDT.

Running AN_missing_dbxref_GeneID.ipynb

Timing: ~ hours to days (depending on dataset size and time spent in manual curation)

14. If gdt-cli filter created AN_missing_dbxref_GeneID.txt, run AN_missing_dbxref_GeneID.ipynb to parse features that do not have a Dbxref GeneID.
    a) As before, you will import the necessary dependencies and instantiate the regular expressions (regex) variables and necessary functions in the first cell (Imports and functions).

15. In the Setup section, define other essential variables and the logger just as you did at the beginning of AN_missing_gene_dict.ipynb. See items 6 and 7 of "Running AN_missing_gene_dict.ipynb" for a refresher on what which variable means.

16. Now, run cells A to D of the "Features Extraction" section.

350 a. Briefly, the AN_missing_dbxref_GeneID.txt is read, the latest .gdt file is loaded,
351 and the respective GFF3 files (of the accession numbers listed in
352 AN_missing_dbxref_GeneID.txt) are analyzed. The attributes column of each
353 genome feature will be broken down according to a series of regular expressions
354 that is defined at the beginning of this script. The output will be
355 “features_info.tsv” and “feature_names.txt”. The last column of
356 “features_info.tsv” (named “best_feature”) will contain the attribute
357 information deemed to be most relevant for subsequent parsing. This column
358 will give rise to “feature_names.txt”.
359
360 ***Note:*** “best_feature” derives from the variable “features_order”, which is
361 defined in “Imports and functions” section of this script. features_order is a list
362 of attributes ranked according to their discriminatory power. Based on our
363 experience with organelle genomes, we have chosen the following order to be
364 the default: ‘features_order = ["gene", "product", "description", "name",
365 "note", "gene_synonym"]’. You have the flexibility to change the order and/or
366 to add new attributes to this variable. More detailed guidelines are described in
367 the “Imports and functions” section.
368

Pause point: After having run cell C. in “Features Extraction”, pause and check whether an error message appears. Detailed log messages will guide you through the troubleshooting.

17. Now manually parse “feature_names.txt”.
    a. Check whether any entries can be added to your latest .gdt. Once you add these entries to .gdt, save the new .gdt version. If you have entries that are still unidentifiable, you should create a new txt file in the misc folder (named feature_unks.txt) and add the unidentifiable entries to it.
    b. After having created and saved your new .gdt, update most_recent_gdt_filename in the Setup section and rerun the respective cell.

18. In the “Automated Insertion from features_info.txt” section, run cell A.
    a. This cell will check whether any entries from features_info.tsv, that are not in feature_unks.txt, are missing in the latest .gdt. If that is the case, you will receive a warning (in the log and/or screen) telling you what features should be double checked.

Note: The warning message will alert you of three possibilities: either you forgot to add the respective entries to your latest .gdt, or you forgot to reload the latest .gdt, or you forgot to add the entries to “feature_unks.txt”. After

addressing the issues identified in this step, run this cell again to ensure that you are good to continue.

19. Run cells B. and C. of this section.
    a. Cell B.: this step will scan features_info.tsv and add to your latest .gdt all genome feature IDs that have a “best feature” that is not listed in “feature_unks.txt”. Genome feature IDs will be added under the corresponding gene labels according to their “best_feature”.
    b. Cell C.: a new .gdt iteration is saved with an updated header containing the respective traceable information (i.e., a stime stamp, iteration number, etc).

    CRITICAL: more than ever before, you must ensure that your most_recent_gdt_filename is updated in the Setup section, and rerun this section before moving to the next step.

20. Run the section “TEMP Mapping” from A. to D.
    a) Cell A: will reload the latest .gdt. and will extract the species and their corresponding feature IDs that need to be more investigated. This extraction happens through the filtering of “best_feature” that are found in “feature_unks.txt”.
    b) Cell B: as one last attempt to find descriptive information about these features, the step will (try to) map child features to their corresponding parent features.

Through this mapping, the descriptive information of the child feature will be relayed to the parent feature. Once mapped, the parent feature is compared against the latest .gdt, and added to it, if there is a match. Otherwise, the parent feature will be added to TEMP_Mapping for manual parsing. The features that could not be mapped to any child are also added to TEMP_Mapping, but are marked with the -UNKNOWN label.

c) Cells C. and D: TEMP_Mapping and the new iteration of .gdt will be saved, if applicable.

21. Manually parse TEMP_Mapping and add, to the best of your knowledge, any entries to your .gdt. This parsing is similar to the process described in items 9 and 12. The entries that remain unidentifiable should be added to “to_remove_2.txt”.

Note: “to_remove_2.txt” should not be confounded with “to_remove.txt” (which was created at the end of AN_missing_gene_dict.ipynb). Both files serve the same purpose, but for different steps of the protocol. Just as with “to_remove.txt”, you should create “to_remove_2.txt” as an empty file in the misc folder.

22. If you end up creating to_exclude_2.txt, run cells A. and B. in the final section (“Gene exclusion of to_remove_2.txt).

Note: this step is similar to step 13.

Optionally: to verify whether there is any unaccounted gene ID left in your dataset, run gdt-cli filter one last time using your most complete (i.e., latest iteration of) .gdt. If there is any unidentified genome feature, gdt-cli filter will create the appropriate file(s) (either AN_missing_gene_dict.txt or AN_missing_dbxref_GeneID.txt) and the process cycles over again.

Optionally: if you want to add the standardized gene labels to your GFF3 files, run gdt-cli standardize (**Figure 7**).

```
❯ gdt-cli standardize --help
usage: gdt-cli standardize [-h] [--debug] [--log LOG] [--quiet] (--gff GFF | --tsv TSV) --gdt GDT [--AN-column AN_COLUMN]
                           [--gff-suffix GFF_SUFFIX] [--query-string QUERY_STRING] [--check] [--second-place] [--gdt-tag GDT_TAG]
                           [--error-on-missing] [--save-copy]

Standardize gene names across features in GFF3 files using a GDT file.

options:
  -h, --help            show this help message and exit
  --debug               Enable TRACE level in file, and DEBUG on console. Default: DEBUG level on file and INFO on console.
  --log LOG             Path to the log file. If not provided, a default log file will be created.
  --quiet               Suppress console output. Default: console output enabled.
  --gff GFF             GFF3 file to standardize.
  --tsv TSV             TSV file with indexed GFF3 files to standardize.
  --gdt GDT             GDT file to use for standardization.
  --AN-column AN_COLUMN
                        Column name for NCBI Accession Number inside the TSV. Default: AN
  --gff-suffix GFF_SUFFIX
                        Suffix for GFF files. Default: '.gff3'
  --query-string QUERY_STRING
                        Query string that pandas filter features in GFF. Default: 'type in ('gene', 'tRNA', 'rRNA')'
  --check               Just check for standardization issues, do not modify the GFF3 file. Default: False
  --second-place        Add gdt-tag pair to the second place in the GFF3 file, after the ID. Default: False (add to the end of the
                        attributes field).
  --gdt-tag GDT_TAG     Tag to use for the GDT key/value pair in the GFF3 file. Default: 'gdt_label='.
  --error-on-missing    Raise an error if a feature is missing in the GDT file. Default: False (just log a warning and skip the
                        feature).
  --save-copy           Save a copy of the original GFF3 file with a .original suffix. Default: False (change inplace).
```

Figure 7. gdt-cli standardize help page.

The user can either pass one GFF3 file or a list of GFF3 files indexed through a tsv file. All files must be in the same working directory. Just as before, pay attention to the need

of consistency in customizable flags such as gff-suffix and query-string. These attributes must match with those utilized in the rest of the protocol.

Expected outcomes

The main outcome of this protocol is a comprehensive, and versionable gene dictionary saved as a .gdt file. The dictionary is not only intuitive and human readable, but easily parseable by a computational agent. Along with the .gdt file, this protocol generates intermediate files (both of type .gdt and .log) that produce a detailed view of the analysis, and permit the user to act on the data as needed. The protocol also gives the option to add to the inputted GFF3 files the standardized gene dictionary labels for downstream analyses. Note that the user can interact with our protocol in two main ways: i) users can create their own gene dictionaries from their genomes of interest (with the possibility of creating their dictionary gene labels); or ii) users can apply the gene dictionaries created by us and implement the protocol as a way to bring consistency to the gene nomenclature across (organelle) genomes of their choosing. Regardless of whether the user chooses to create their own dictionary or deploy ours, another contribution of this protocol is the launching of the gdt library. This Python library, with its user-centered design, enable users to manipulate and integrate .gdt files into other pipelines seamlessly. We hope that these outcomes will contribute to a more consistent and FAIR-guided gene nomenclature ecosystem.

475 ## Limitations

476 Most (if not all) of the limitations we have identified in our analyses are applicable to any type
477 of annotated genome. The first limitation pertains to the time spent in manual curation.
478 Although the dataset (i.e., the number of GFF3 files to be analysed) might increase linearly, the
479 final number of genome features to be parsed is poised to grow much faster. This is because
480 each genome will likely add several genome features to the dataset. This limitation is
481 ammeliorated as the user iterates through the pipeline and creates a more complete version
482 of their working .gdt. In an attempt to provide users with a head start on the analysis of
483 organelle genomes, our own .gdt files (both stripped and complete versions for mitochondria
484 and plastids) are available in our GitHub page. As more genomes are annotated and made
485 available in public repositories, we intend to release updates of our dictionaries to cover new
486 organelle genome features IDs.
487
488 Another group of limitations comes from the inherent structure of GFF3 files. The Generic
489 Feature Format (GFF) file has become a staple in genome analysis because major stakeholders
490 like NCBI, GENCODE, and Ensembl release their genome annotations in the GFF3 format
491 (notwithstanding other formats like BED). The difficulties (and frustration) that come with
492 trying to analyse GFF3 files from disparate sources has been shared in popular media outlets
493 like Medium (https://medium.com/@shiansu/the-not-so-standard-gff3-standards-
494 957c54d848af) and in the very own The-Sequence-Ontology GittHub Specifications page
495 (https://github.com/the-sequence-ontology/specifications/blob/master/gff3.md). Upon

496 reading the current specifications (version 1.26, released on 18 August 2020), one can clearly
497 find two potential pitfalls: the first one coming from the fact that IDs are mandatory just for
498 features that possess children features (such as protein-coding genes). As GDT's framework
499 relies on unique genome feature IDs, annotations without IDs cannot be indexed in the .gdt
500 file and are rendered incompatible with our protocol. The second pitfall pertains to the fact
501 that "IDs for each feature must be unique within the scope of the GFF file", so these IDs are
502 guaranteed to be unique only within each GFF3 file. In our analyses of thousands of organelle
503 genomes, we have never found two identical IDs that referred to different genome features.
504 But, technically, a single ID can refer to a different genome feature (e.g., the same ID can refer
505 to ATP9 in one genome and to ATP8 in another). We expect this potential pitfall to be more
506 likely to happen, if the user analyses different annotation versions of the same genome (as in
507 NC_??######.1 and NC_??######.2, where .1 and .2 represent different versions of the same
508 genome). This issue also becomes more probable in very large datasets. Similarly to the
509 phenomenon of hash collision that plagues hash tables, as the number of GFF's increases, the
510 "genome feature ID universe" shrinks and "genome feature ID collisions" might become a
511 reality. See the Troubleshooting section for how the protocol handles a somewhat similar
512 collision issue.
513
514 Having tested our protocol only on organelle genomes can be seen either as an asset or as a
515 blatant limitation. Organelle genomes are notorious for their unconventional genomic
516 attributes and inconsistent annotation. Arguably, we might have picked the toughest case
517 scenario, which have forced us to adopt defensive programming from top to bottom. But we

518 can only script defensively against what we have encountered (and/or predicted), so testing
519 GDT in other types of genomes is an immediate next step in the development of our protocol.
520 In its current form, GDT relies only on NCBI Gene queries when searching (externally) for
521 further information on the annotation of the genome feature at hand. We do that through the
522 unique NCBI gene IDs that are assigned to "Dbxref=" in the GFF3 attributes column. Another
523 further improvement of this protocol is to give the user the possibility to tweak the cross-
524 reference search to a database of their choosing.
525
526 After having run this protocol in thousands of organelle genomes, some clear trends within the
527 "hard-to-identify" genome features have surfaced. Features annotated as "hypothetical
528 protein" and "orf" are discussed in the Troubleshooting section, but tRNAs deserve their own
529 mentioning within limitations. Given the myriad of gene naming styles, you should expect (with
530 almost certainty) that you will be manually parsing tRNA names. Issues vary from "trnA vs
531 tRNA" (the first referring to tRNA-Ala and the second referring to a generic tRNA) to "gene-
532 trnF(GAA) vs gene-trnF (GAA)" and anything in between. To certain extent, consistency can be
533 distilled from organelle tRNA names, so much so that the use of regular expressions is
534 commonplace when dealing with tRNAs. But variability and discordance within organelle tRNA
535 nomenclature has been extensively highlighted[8], even though clear guidelines have been
536 proposed over four decades ago[9]. Therefore, another future improvement of our protocol is
537 to automate, as much as possible, the parsing of tRNAs.
538

539 # Troubleshooting

540 **Problem 1**

541 What if I do not want to adopt the gene labels proposed in your protocol?

542

543 **Potential solution**

544 You have full autonomy to create gene labels that you find more appropriate to your dataset.
545 This protocol lets the user create a comprehensive gene dictionary regardless of the how
546 dictionary labels look like.

547

548 **Problem 2**

549 I have hundreds of "hypothetical protein" features in my TEMP_Description_##.gdt. What
550 should I do? Similarly, what should I do with very generic/cryptic entries like "rna", "trna",
551 "hyp", or "oi"?

552

553 **Potential solution**

554 We have two suggestions: you can either create an "umbrella label" for all these problematic
555 genome features or you can divert them directly to the TEMP Symbol processing to obtain
556 more descriptive information. In our own experience, we have found some interesting genome
557 features buried in the "hypothetical protein" pile. Sending these features to Temp Symbol will
558 likely clutter your TEMP_Symbol.gdt, so feel free to use to_remove.txt option as well.

559

560 **Problem 3**

561 What do [1st T], [1st F], [2nd T], and [2nd F] mean in the log file?

562

563 **Potential solution**

564 [1st T], [1st F], [2nd T], and [2nd F] serve as sign posts in the staggered parsing that happens
565 when the data_process function is called. The question being asked is always "is this genome
566 feature already in my .gdt"?, but data_process asks this question (three times) in slightly
567 different ways depending on where you are in the analysis.

568 [1st T]: genome feature ID's check_var is in the current gene dictionary (i.e., gene_dict), where
569 check_var can be the information from gene, NCBI gene symbol, or NCBI gene description
570 (NCBI gene description is the default, when use_gene and use_NCBI_symbol are both False).
571 Then, the genome feature ID is added to the current gene dictionary immediately under the
572 check_var's gene label.

573 [1st F]: check_var is not in gene_dict (so go to the question below).

574 [2nd T]: check_var is in TEMP_Description_##.gdt (or TEMP_Symbol_##.gdt).

575 [2nd F]: check_var is not in TEMP_Description_##.gdt (or TEMP_Symbol_##.gdt).

576

577 **Problem 4**

578 After having run cell B. in "Automated Insertion from features_info.txt", I got the error
579 'Multiple best_features with different labels'. What should I do?

580

Potential solution

This error occurs when the script identifies a gene_id that has multiple 'best_features', and these features, when queried in the .gdt, points to different labels. Although the use of the same feature ID in different GFF3 files is somewhat common (e.g., "ID=gene-atp9;") for files without Dbxref GeneID, this ID collision becomes a problem when the IDs refer to incongruent features {e.g., "ID=rps;"). The solution is to manually edit one of the GFFs to diferentiate one of the gene_IDs, then edit the features_info.tsv to reflect the change (or just run the entire AN_missing_dbxref_GeneID.ipynb again). It is also possible to just remove the offending genome from your dataset.

Acknowledgments

The authors wish to acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fundação Araucária for funding granted to D.R.S. (Discovery Grant, NSERC) and to A.R.P. (NAPI Bioinformática 66.2021, Fundação Araucária).

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 D.B. Writing – original draft: M.S.L. and B.D. Writing – review & editing: M.S.L., B.D., D.R.S. and A.R.P. Funding acquisition and Supervision: D.R.S. and A.R.P.

601 ## Declaration of interests

602 The authors declare no competing interests.

603

604 ## References

605 1. Smith, D.R. (2013). RNA-Seq data: a goldmine for organelle research. Brief. Funct. Genom.
606 12, 454-456.

607 2. Smith, D.R. (2012). Not seeing the genomes for the DNA. Brief. Funct. Genom. 11, 289-290.

608 3. Sanita Lima, M., Woods, L.C., Cartwright, M.W., and Smith, D.R. (2016). The (in)complete
609 organelle genome: exploring the use and non-use of available technologies for characterizing
610 mitochondrial and plastid chromosomes. Mol. Ecol. Resour. 16, 1279-1286.

611 4. Smith, D.R., and Keeling, P.J. (2015). Mitochondrial and plastid genome architecture:
612 reoccurring themes, but significant differences at the extremes. Proc. Natl. Acad. Sci. USA 112,
613 10177-10184.

614 5. Lynch, M., Koskella, B., and Schaack, S. (2006). Mutation pressure and the evolution of
615 organelle genomic architecture. Science. 311, 1727-1730.

616 6. Giannakis, K., Arrowsmith, S.J., Richards, L., Gasparini, S., Chustecki, J.M., Røyrvik, E.C., and
617 Johnston, I.G. (2022). Evolutionary inference across eukaryotes identifies universal features
618 shaping organelle gene retention. Cell Syst. 13, 874-884.

619 7. Shtolz, N., and Mishmar, D. (2023). The metazoan landscape of mitochondrial DNA gene
620 order and content is shaped by selection and affects mitochondrial transcription. Commun.
621 Biol. 6, 93.

622 8. Boore, J.L. (2006). Requirements and standards for organelle genome databases. OMICS: A
623 Journal of Integrative Biology. 10, 119-126

624 9. Hallick, R.B., and Bottomley, W. (1983). Proposals for the naming of chloroplast genes. Plant
625 Mol. Biol. Rep. 1, 38-43.

626 10. Seal, R.L., Braschi, B., Gray, K., Jones, T.E.M., Tweedie, S., Haim-Vilmovsky, L., and Bruford,
627 E.A. Genemames.org: the HGNC resources in 2023. Nucleic Acids Res. 51, D1003-D1009.

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

630 12. Pinto Rabelo, L., Sodré, D., Petry Corrêa de Sousa, R., Watanabe, L., Gomes, G., Sampaio, I.,
631 and Vallinoto, M. (2024). SynGenes: a Python class for standardizing nomenclatures of
632 mitochondrial and chloroplast genes and a web form for enhancing searches for evolutionary
633 analyses. BMC Bioinformatics. 25, 160.

634 13. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A.,
635 Blomberg, N., Boiten, J-W., Bonino da Silva Santos, L., Bourne, P.E., et al. (2016). The FAIR
636 Guiding Principles for scientific data management and stewardship. Sci. Data. 3, 160018.

637 14. Cock, P.J.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I.,
638 Hamelryck, T., Kauff, F. Wilczynski, B., et al. (2009). Biopython: freely available Python tools for
639 computational molecular biology and bioinformatics. Bioinformatics. 25, 1422-1423.

640 15. Sayers, E.W., Beck, J., Bolton, E.E., Brister, J.R., Chan, J., Connor, R., Feldgarden, M., Fine,
641 A.M., Funk, K., Hoffman, J., et al. (2025). Database resources of the national center for
642 biotechnology information. Nucleic Acids Res. D1, D20-D29.