gFACs Documentation

Release 1.0.0

**Madison Caballero, Jill Wegrzyn**

Nov 14, 2019

Getting started

gFACs is a filtering, analysis, and conversion tool to unify genome annotations across alignment and gene prediction frameworks. It was developed by Madison Caballero and Dr. Jill Wegrzyn of the Plant Computational Genomics Lab at the University of Connecticut.

Find gFACs on GitLab and read about it on bioRxiv!

Current version: 1.1

Comments? Questions? Email me at Madison.Caballero@uconn.edu
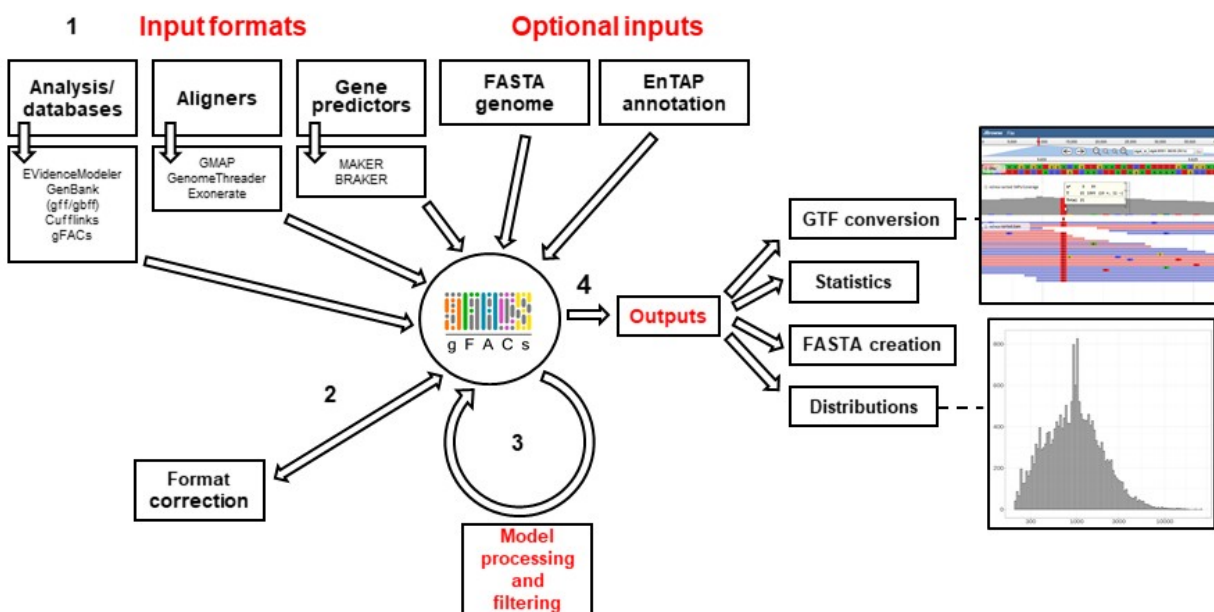
Installation

Installing gFACs is quite easy since this program operates almost exclusively through perl scripts. So long as the files are copied over, basic gFACs should be operational so long as perl and bioperl libraries are available. gFACs is also very light so feel free to have multiple copies wherever you need it!

You can find gFACs on GitLab.

In order to download gFACs, follow these basic steps:

1. Obtain the directory for gFACs from GitLab in whatever format you prefer:
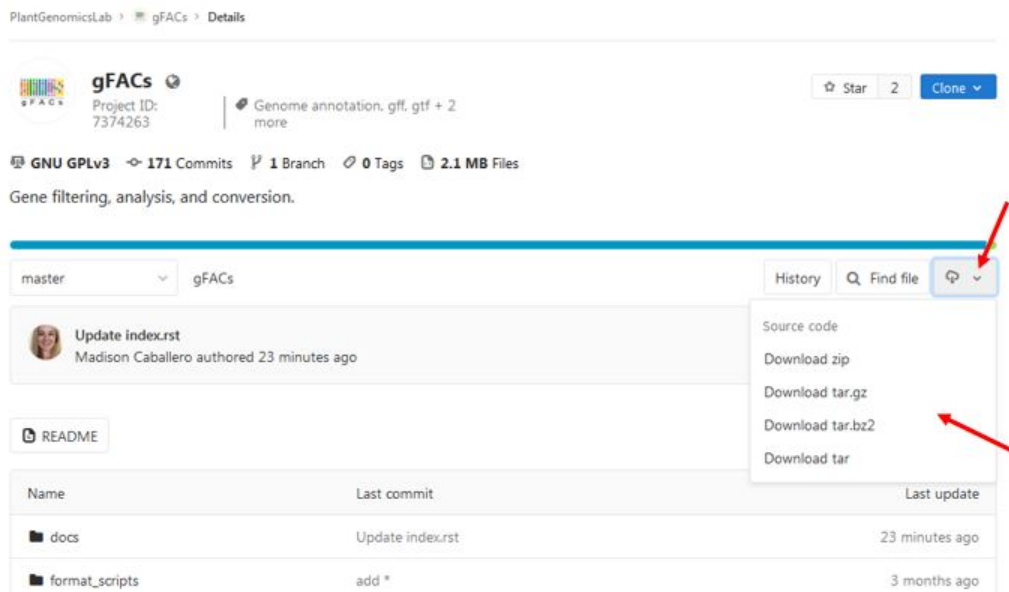
2. Place the entire folder onto your system and extract components. You should have a screen that looks like this:

3. To test to see if it works, perform the simple command of:

```
perl gFACs.pl
```

The command line manual should appear. If so, congratulations! If not, a common error that occurs is that bioperl libraries are not loaded. If you have issues, feel free to send an issue request on GitLab!

Supported Input Formats

There are many different programs that utilize a gff or gtf style format. Each has its own inclusion rules and formatting guidelines. Therefore, a true universal gff/gff3/gtf file converter that requires no user input may be an impossible task. gFACs requires the user to define the application source and version. If you do not find you input format, look into the support script format_diagnosis.pl. If your file is made up of many different formats merged together, I suggest breaking it apart. If you want a format added (especially if it is a well-known one) let me know and I will likely create one!

The format is specified in the gFACs command by a **-f [code]** flag. It is a **mandatory** flag and the code will fail without it. These codes are listed out below with notes and can also be seen in the command line manual. For an example of the command with a proper format flag, see any of the sample runs!

BRAKER: braker_2.05_gtf braker_2.05_gff braker_2.05_gff3 braker_2.0_gff3 braker_2.0_gff braker_2.0_gtf

MAKER: maker_2.31.9_gff

Note: Maker may provide other information such as blastx and protein2genome information. Currently, only maker models of genes and exons will be considered.

PROKKA: prokka_1.11_gff

GMAP: gmap_2017_03_17_gff3

GENOME THREADER: genomethreader_1.6.6_gff3

GFFREAD: gffread_0.9.12_gff3

EXONERATE: exonerate_2.4.0_gff

EVIDENCE MODELER: EVM_1.1.1_gff3

GFACS: gFACs_gene_table gFACs_gtf

Note: You can input a gene table from gFACs, any version. However, the prefix on the input will NOT be retained.

NCBI: refseq_gff

Note: CDS only taken.

---

genbank_gbff

Note: UTR is removed. (CDS sequences only considered)

> **Warning:** Gbff is undergoing repair! Results may not be accurate!

For those who are curious, each format has a special conversion script that transitions the input into the gene table. These are the scripts found in the format_scripts folder that comes along with gFACs. If you are feeling adventurous, you can make your own conversion script that creates the gene table and simply run gFACs with the gene table format code.

About

gFACs was created during the annotation of the megagenome of loblolly pine. In dealing with gene models created from different softwares and alignment tools, we needed a way to filter and merge these models. Unfortunately, no such system existed so gFACs was designed and developed to fill this niche. The applications that generate gene model evidence include aligners and ab initio gene prediction software. These programs report their predictions and alignments in a similar structured gene transfer format (gtf) or general file format (gff) however there is little consistency across these standards. You can read more about the general structures here. gFACs will filter and select final gene models based upon user provided filters regarding their structural attributes. In addition, gFACs can optionally consider functional annotation from the EnTAP application as an additional filter to define true models.

The flow of gFACs.pl is controlled by the master script gFACs.pl. Flags and input files are processed by the master script, and a series of task-specific scripts are called upon to edit and filter gene or alignment models.
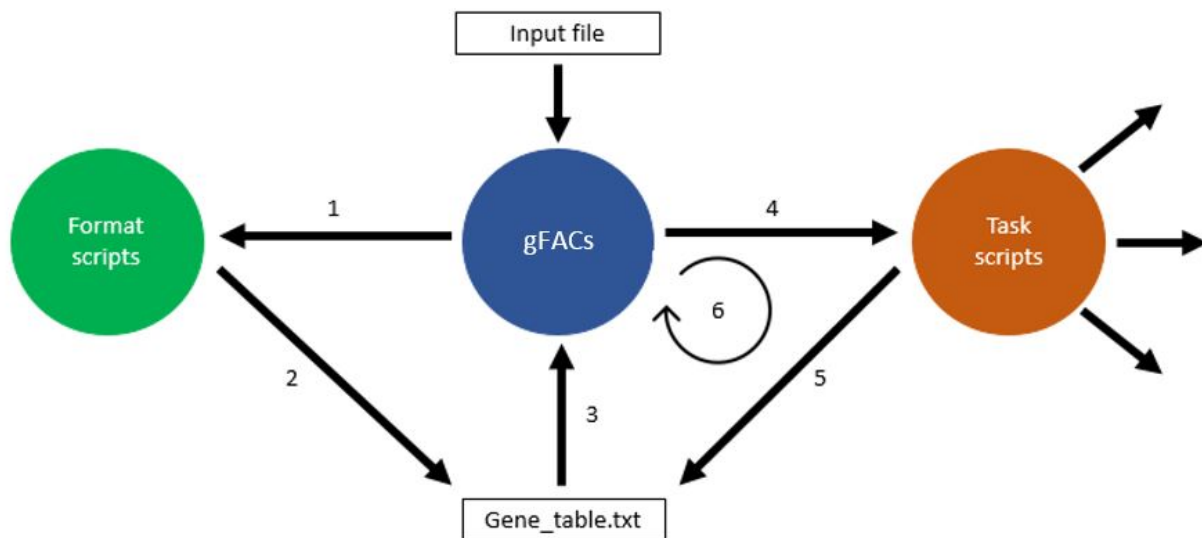
The primary input is an annotation file, either in gene transfer format (gtf) or general feature format (gff/gff3). Since these file types are variable across the applications that generate them, formats designed to fit a particular software's

output must be created. Specific scripts in the folder format_scripts/ are used to convert the input into a median file type called gene_table.txt.The user is able to input their specific file type but they must inform gFACs about the format of the file. For specific formats and how to provide this information, see the supported input section on it.
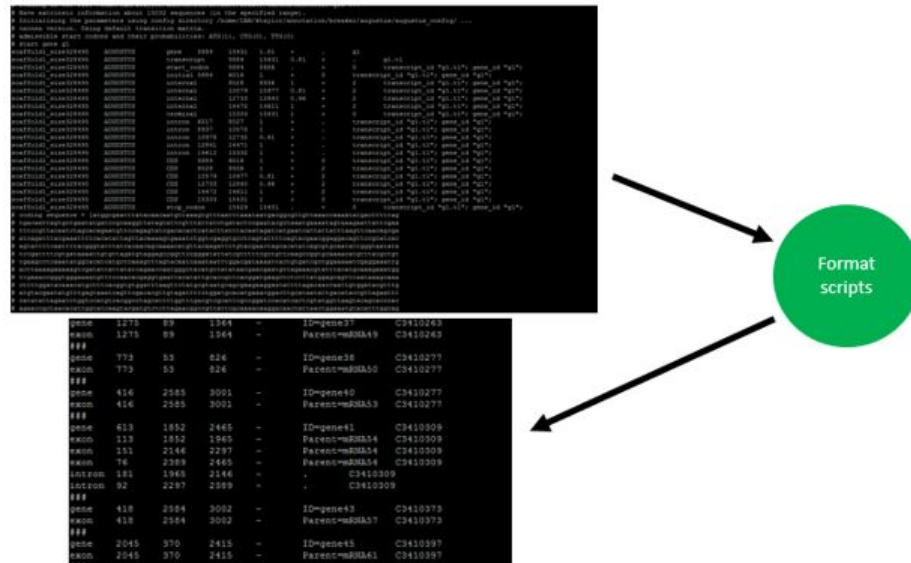
3.1 The gene table

gene_table.txt (referred to also as the gene table) is specfiic to gFACs and is created to hold the minimum amount of information needed to uniformly apply the filtering options.

The gene table is the most important file for this program as it is used and edited in every step. Each flag or task in evaluation needs the format of the gene table to work successfully. The gene table will always have the gene models or alignments that are retained. Here is an example of gene table format:

```
###
gene     1206    144168   145373   -        ID=g28960.t1;geneID=g28960      scaffold124501
exon     250     144168   144417   -        Parent=g28960.t1         scaffold124501
exon     373     144565   144937   -        Parent=g28960.t1         scaffold124501
exon     286     145088   145373   -        Parent=g28960.t1         scaffold124501
intron   147     144418   144564   -        .        scaffold124501
intron   150     144938   145087   -        .        scaffold124501
###
gene     375     76279    76653    +        ID=g28961.t1;geneID=g28961      scaffold124508
exon     375     76279    76653    +        Parent=g28961.t1         scaffold124508
###
```

The columns go:

1. Gene part
2. Length
3. Start
4. Stop
5. Strand
6. ID (8th column from input file)

7. Scaffold/chromosome (needed for fasta commands)

Further scripts expect the gene table to be in the output directory called gene_table.txt. If you are using a prefix, it will look for the file with the prefix. The task-scripts also create their own files, notably if things are being separated, such as potential splice variants. However, the retained genes will always be renamed or concatenated into the master gene table.

3.2 Intron prediction

The gene table provides intron information that is not always found in the input file. Even if the input format does provide introns, they will be recalculated based on the positions of predicted exons. In the format step, a temporary file is created that will terminated upon step completion. The purpose is to add a divider between gene families. The ### line in the gene table allows for a clear break between different genes. The script then revisits the temp file, calculates introns, and makes final formatting shifts.

Here is an example of the temporary file:

```
###
scaffold124501  AUGUSTUS      gene    5717    6235    .       -       .       ID=g28956.t2;geneID=g28956
scaffold124501  AUGUSTUS      mRNA    5717    6235    .       -       .       ID=g28956.t2;geneID=g28956
scaffold124501  AUGUSTUS      exon    5717    6235    0.61    -       .       Parent=g28956.t2
scaffold124501  AUGUSTUS      CDS     5717    6235    .       -       0       Parent=g28956.t2
###
scaffold124501  AUGUSTUS      gene    6831    8759    .       +       .       ID=g28957.t1;geneID=g28957
scaffold124501  AUGUSTUS      mRNA    6831    8759    .       +       .       ID=g28957.t1;geneID=g28957
scaffold124501  AUGUSTUS      exon    6831    7127    0.97    +       .       Parent=g28957.t1
scaffold124501  AUGUSTUS      exon    7301    7392    1.00    +       .       Parent=g28957.t1
scaffold124501  AUGUSTUS      exon    8400    8435    0.56    +       .       Parent=g28957.t1
scaffold124501  AUGUSTUS      exon    8579    8759    0.32    +       .       Parent=g28957.t1
scaffold124501  AUGUSTUS      CDS     6831    7127    .       +       0       Parent=g28957.t1
scaffold124501  AUGUSTUS      CDS     7301    7392    .       +       0       Parent=g28957.t1
scaffold124501  AUGUSTUS      CDS     8400    8435    .       +       1       Parent=g28957.t1
scaffold124501  AUGUSTUS      CDS     8579    8759    .       +       1       Parent=g28957.t1
###
```

Introns are the sequence between exons. Lengths and start and stop coordinates are calculated based on exon information. To accomplish this, exon lengths are pushed into an array and called by position. This method is more universally reliable but prone to errors involving overlapping exons. This can be resolved with a flag –splice-rescue which I recommend.

3.3 Task scripts

Once the formatting has completed, and the gene table has been created, filtering as designated by flags is done cyclically on the gene table until the final set of genes is produced. The order of filtering flags is **pre-determined** although this does not change the final result. For example, whether or not monoexonic genes are removed first or last will not change how many monoexonics appear in the final iteration of the gene table (spoiler alert: none).

Once all filtering is done, other commands are activated that involve processing or analyzing the final sets. This includes statistics, analysis of splice types, or distributions.
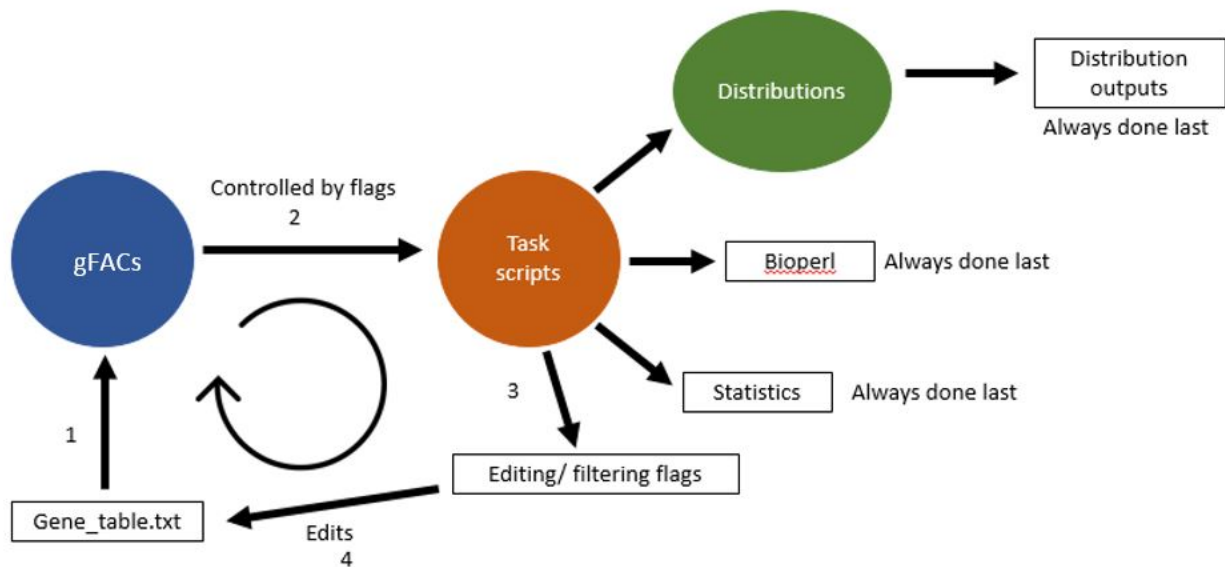
3.4 The log file

In addition to the gene table, the gFACs_log.txt file is created every time the script is run, no exceptions. If a prefix is included, the log will have this prefix. The log file is reported by the master script and is appended with information regarding filtering at each step. It will also report what flags are being activated and the very specific corresponding system commands.

The log may be helpful for the user to see what is happening and the results of a particular filter. It is also helpful for noticing bugs and verifying script efficacy. gFACs supports readable and understandable log files!

Filter flags

As the title suggests, these flags edit or remove content from the input file. Flag use depends on whether additional resources are provided such as fasta file or EnTAP annotation file.

4.1 Basic flags

These are flags that can be run on any input without the addition of a fasta or EnTAP file. Given that, these flags are not capable of performing sequence level analysis. You can include as many flags as you want in any order. However, the order in which the flags are run is **predetermined** by the gFACs.pl script. This section is designed to tell you what the flags do conceptually. This is not the true order. See the log file for the order as the log is printed in sequence.

4.1.1 -p [prefix]

All files created will have your designated prefix. For example, if you provide the prefix "test", your gene table will be called test_gene_table.txt. Every file (even temporary files) will have this prefix. However, it is not a mandatory argument.

4.1.2 --false-run

This flag allows for the user to understand the effect of filters on the original input without an additive effect. Before committing to a gFACs run, it may be important to understand what gFACs is planning on removing from the original set of models. When gFACs filters, it removes sequentially from a shrinking pool of models. However, it may be important to know how many gene models would be removed from the original set. How many overall do not have a stop codon? How many overall are of a certain CDS length?

This flag allows gFACs to run normally but always resets to the original set of genes in the gene table. Therefore, the log will print what is removed and retained but never follows through with filtering. Resulting files will be the gene table completely unedited and the log file will reflect resetting numbers.

NOTE: This flag does NOT keep the results of splice rescue nor unique genes only. This is problematic with overlap when isoforms or multiple RNA evidence models are within the same called gene. Sequence pull filters like in-frame

stop codon, fasta creation, or analysis can then be wrong. For example, when setting in-frame stops to 0, all genes with isoforms will print one after another in a long sequence string that will have as many stop codons as models presented. This may create the appearance of increased removal of models. To work around this problem, run classic gFACs on your set using --splice-rescue and --unique-genes-only. Then use --false-run on that parsed gene table output. Feel free to contact me if you have problems!

4.1.3 –no-processing

The following occurs by DEFAULT. Use –no-processing to opt out.

Processing involves two scripts: task_scripts/overlapping_exons.pl and task_scripts/splice_variants.pl. The first analyzes the gene table for overlapping exon space and then the second analyzes genes with overlapping exon space for evidence that these are separate transcripts or models.

Take this particular gene for example:

Although three exons are called, they overlap as the third is supported by a different parent transcript. The first two claim an intron while the third spans over that particular intron like this:

Intron prediction, when not taking into account that these are separate models of the gene space, would be wrong. To solve this, genes that show exon overlap are separated out into their own file. Then they are evaluated.
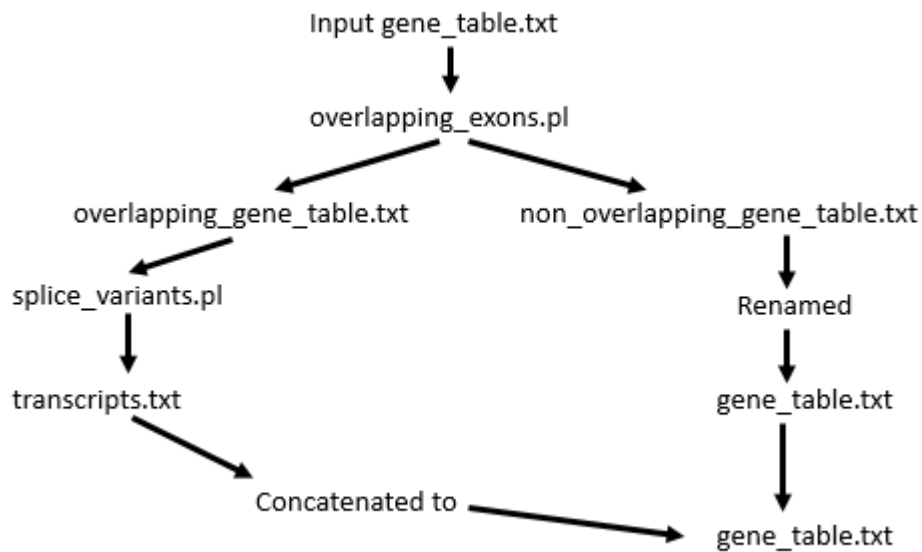
The way splice variants are confirmed is due to the labeling on the 6th column in the gene table. If an exon has a different transcript ID (often labeled t1, t2, t3...) then the exon is separated into its own "batch". What was originally "gene22" with three separate transcripts then becomes gene22.1, gene22.2, and gene22.3.
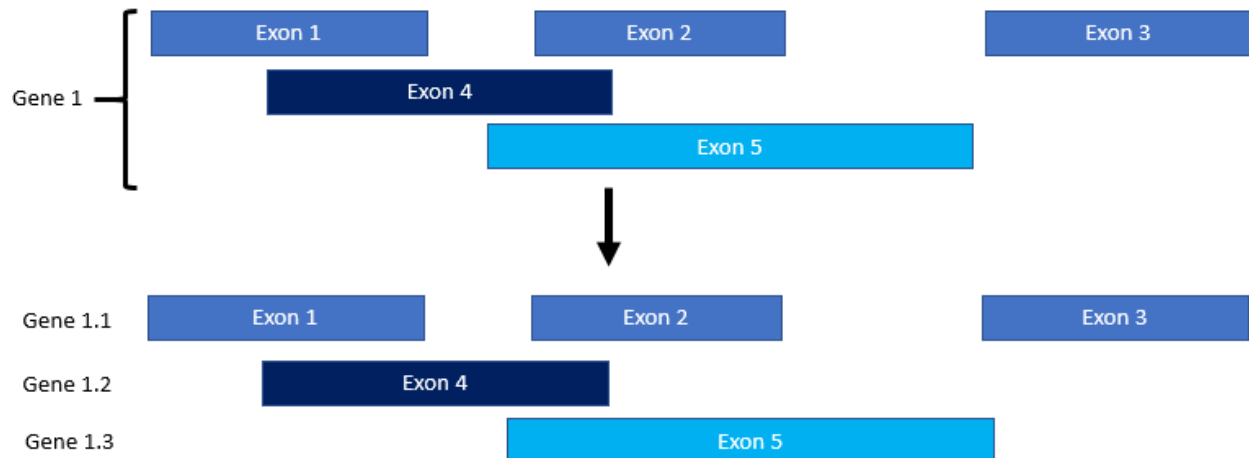
Introns are then recalculated for each of the separate isoforms. If some of the models are incomplete, they can be filtered out with other flags since they are now treated as separate "genes". (This is because they are separated by a ### partition).

Passing genes will remain in the gene table. Results of this filter are printed in the log.

To resolve transcripts back to unique genes (selecting largest, if available, or first transcript) can be done with the --unique-genes-only flag.

Following this step, another script called task_scripts/gene_table_fixer.pl is implemented. This is done to check that exons and introns are in positive strand ascending order. Without it, printing the fastas will be problematic. It is done regardless of whether or not you use splice rescue.

4.1.4 –no-gene-redefining

The following occurs by DEFAULT. Use –no-gene-redefining to opt out. Genes that are incomplete in the annotation such that it appears they start or end with an intron are labeled as:

COMP : Complete. Note this does NOT mean there is a start and in-frame stop codon. 3*_*INC : 3' incomplete 5*_*INC : 5' incomplete 3*_*INC+5*_*INC : Both gene ends are incomplete

Incompleteness checks do NOT require a fasta and therefore do not judge incompleteness based on codon content. For examples of this, see the filters directly below for removing incompletes.

Genes will be trimmed to reflect available CDS in the first step following format conversion but incomplete labels will remain to identify the modification made. This is opt-out using --no-gene-redefining. All filters will be applied normally regardless of choice to trim. Note that a gene labeled as a 3' incomplete will always fail a start codon check even if the first three nucleotides of the trimmed gene is an ATG (or alternate). Same for 5' incompletes and stop codon checks.

4.1.5 –rem-3prime-incompletes

This option is controlled by the script task_scripts/remove_starting_introns.pl. It is designed to pull out genes that start with "intronic" space. This is almost always because of missing evidence due to missing scaffold sequence. Genes are automatically trimmed unless –no-gene-redefining is used.

Take this for example:

```
###
gene     22546   1       22546   +       g193    super417
exon     73      86      158     +       transcript_id "g193.t1"; gene_id "g193";    super417
exon     325     925     1249    +       transcript_id "g193.t1"; gene_id "g193";    super417
```

The very first exon begins 86 nucleotides into the proposed gene space. You can also see that the gene "begins" at position 1 in super417. This particular model came from BRAKER 2.0.

This script finds genes where gene start and first exon start do not agree. Passing genes will remain in the gene table. Results of this filter are printed in the log.

NOTE: This script takes into account directionality. Meaning a positive strand gene without a start intron would occur at the start of the gene (all gene coordinates are positive stranded). In a negative strand gene, missing the start intron would occur at the end of the gene.

NOTE: This filter does not take into acount sequence. To remove all incompletes with codons in mind, combine with –rem-genes-without-start-codon.

4.1.6 –rem-5prime-incompletes

This option is controlled by the script task_scripts/remove_ending_introns.pl. It is nearly identical to removing start and also takes into account strandedness and directionality. For another example in the exact same BRAKER 2.0 gene table:

```
gene      3088   1231   4318   +        g59947   jcf7180001216318
exon      117    1231   1347   +        transcript_id "g59947.t1"; gene_id "g59947";   jcf7180001216318
exon      105    1547   1651   +        transcript_id "g59947.t1"; gene_id "g59947";   jcf7180001216318
exon      78     1833   1910   +        transcript_id "g59947.t1"; gene_id "g59947";   jcf7180001216318
exon      117    1231   1347   +        transcript_id "g59947.t2"; gene_id "g59947";   jcf7180001216318
exon      105    1547   1651   +        transcript_id "g59947.t2"; gene_id "g59947";   jcf7180001216318
exon      41     1847   1887   +        transcript_id "g59947.t2"; gene_id "g59947";   jcf7180001216318
intron    115    1232   1346   +        .        jcf7180001216318
intron    199    1348   1546   +        .        jcf7180001216318
intron    103    1548   1650   +        .        jcf7180001216318
intron    181    1652   1832   +        .        jcf7180001216318
intron    39     1848   1886   +        .        jcf7180001216318
###
```

Here you can see that the last exon ends at 1887 but the gene claims to end at 4318.

Sometimes this occurs because of the scaffold ending as before, but further fasta involvement can find incomplete genes due to codon evidence.

Passing genes will remain in the gene table. Results of this filter are printed in the log.

NOTE: This filter does not take into acount sequence. To remove all incompletes with codons in mind, combine with –rem-genes-without-stop-codon.

4.1.7 –rem-all-incompletes

Performs the tasks of --rem-3prime-incompletes and --rem-5prime-incompletes. See above for what each task does individually. Passing genes will remain in the gene table. Results and commands of this filter are printed in the log as if each command was run separately.

4.1.8 –rem-monoexonics

Removes monoexonics based off the presence of introns. All multiexonic genes then remain in the gene table. The script that does this command is task_scripts/remove_monoexonics.pl.

4.1.9 –rem-multiexonics

Remove multiexonics based off the presence of introns. All monoexonics genes then remain in the gene table. The script that does this command is task_scripts/remove_multiexonics.pl.

4.1.10 –min-exon-size [number]

Default: 20

Creates a filter to remove genes with exons below a certain size. The script that performs this command is task_scripts/minimum_exon.pl. If you do not provide a following number, 20 is used as a benchmark for an exon that is suspiciously too small.

Passing genes will remain in the gene table. Results of this filter are printed in the log.

4.1.11 –min-intron-size [number]

Default: 20

Creates a filter to remove genes with introns below a certain size. The script that performs this command is task_scripts/minimum_intron.pl. If you do not provide a following number, 20 is used as a benchmark for an intron that is suspiciously too small. However, it might be technically possible to have introns that are less than 20 nucleotides.

Passing genes will remain in the gene table. Results of this filter are printed in the log.

4.1.12 –min-CDS-size [number]

Default: 74

Creates a filter to remove genes with a coding sequence (CDS) below a certain nucleotide length. Introns do not count, only exon sequence size. The default is based off the smallest known gene and will be used if no input is provided.

Passing genes will remain in the gene table. Results of this filter are printed in the log.

4.1.13 –unique-genes-only

This option will collapse directly overlapping genes and resolve transcripts created using -–splice-rescue.

When using --splice-rescue, multiple transcripts are created that represent the same gene. They may be isoforms of one gene or the exact same gene model repeated due to multiple pieces of evidence. Since separation treats each transcript as if it was its own gene for statistics and file-creation steps, this step will return only unique genes.

This is done by the script task_scripts/unique_genes.pl. It separates out transcripts denoted by their .1, .2, etc… modification. For those representing the same gene, the largest transcript is selected if available. Otherwise it will just take the first one.

When not dealing with transcripts, if two separate genes with different IDs share the exact same space, the first one numerically will be chosen. This only affects genes with 100% overlap where each gene is the same size and starts and ends at the same coordinates.

This step is done before any outputs are created such as statistics, fastas, output tables, or gtf files. Unique genes will remain in the gene table. Results of this filter including genes in, transcripts present, unique transcripts, non-transcript duplicates, lost, and final returned genes are printed in the log.

4.2 EnTAP required flags

4.2.1 –entap-annotation /path/to/your/final_annotation.tsv

Provide the path to the output of the protein annotation. The first column should be the name of a gene in the "g###.t1" format. As of this version, only braker 2.05 outputs annotated with ENTAP are recognized due to building resources. It will refuse to run with other formats.

The trailing t1 is disregarded and there should be no recognition issues if everything is run with the latest version of gFACs. Additionally, all ENTAP steps are done before fasta steps so all outputs will reflect filters that require the annotation.

The annotation should look something like this:

```
g60348.t1

g60343.t1

g60341.t1        sp|F4JLP5|PLPD2_ARATH   80.6    572     92      5       1       568     1       557     2.8e-252        92.400000       sp|F4JLP5|PLPD2_ARATH Dihydrolip
oyl dehydrogenase 2, chloroplastic OS=Arabidopsis thaliana GN=LPD2 PE=2 SV=2    Arabidopsis thaliana    /UCHC/LABS/Wegrzyn/WalnutGenomes/Regia/outfiles/similarity_searc
h/blastp_RegiaCompleteMultiExonics_final_uniprot_sprot.out      No      Yes     29760.VIT_05s0077g01210.t01     2.8e-277        959.5           Viridiplantae
1BH5S@strNOG,1DUCK@virNOG,COG1249@NOG,KOG1335@euNOG     dihydrolipoyl dehydrogenase     Biosynthesis of secondary metabolites (01110), Pyruvate metabolism (00620), Meta
bolic pathways (01100), Citrate cycle (TCA cycle) (00020), Glycine, serine and threonine metabolism (00260), Glycolysis / Gluconeogenesis (00010), Microbial metabolism
in diverse environments (01120), Valine, leucine and isoleucine degradation (00280)     PFAM (Pyr_redox_2, Pyr_redox_dim, Pyr_redox, GIDA, FAD_binding_2)     GO:00090
58-biosynthetic process(L=2),GO:0044237-cellular metabolic process(L=2),GO:0044710-single-organism metabolic process(L=2),GO:0071704-organic substance metabolic process
(L=2),  GO:0043227-membrane-bounded organelle(L=2),GO:0043228-non-membrane-bounded organelle(L=2),GO:0043233-organelle lumen(L=2),GO:0044422-organelle part(L=2),GO:0044
464-cell part(L=2),     GO:0016491-oxidoreductase activity(L=2),
g60340.t1
```

4.2.2 –annotated-all-genes-only

Only genes that have an associated similarity search OR EggNOG annotation are kept. Done by the script task_scripts/annotated_all_genes_only.pl. Passing genes will remain in the gene table. Results of this filter are printed in the log.

4.2.3 –annotated-ss-genes-only

Only genes that have an associated similarity search annotation are kept. Done by the script task_scripts/annotated_ss_genes_only.pl. Passing genes will remain in the gene table. Results of this filter are printed in the log.

4.3 Fasta required flags

These scripts require that there is a fasta, because sequence is being evaluated. The gFACs.pl script will index your fasta, and then task scripts that require sequence will find and use that index. If there is already an index, the indexing step will be skipped.

To specify a fasta:

4.3.1 –fasta /path/to/your/nucleotide/fasta.fasta

Bioperl will create an index with the ending ".fasta.idx". It is a fairly fast process. The file may end with .fa or .fasta, but no other naming formats can be recognized.

NOTE: This fasta **MUST MUST MUST** be the same fasta used when making your particular gff3/gtf/gff. Bioperl needs to recognize the name on the fasta info line to the sixth column in the gene table.

4.3.2 –canonical-only

Analyzes introns for a canonical splice sites (GT-AG on the positive strand). The script that performs this task is task_scripts/canonical_only.pl.

To pass, all introns in a gene must have canonical splice sites. Monoexonics will also pass this filter because they do not have the evidence to be pulled out. Of course, monoexonic genes can be removed by –rem-monoexonics filter.

Genes that pass this filter are kept in the gene table and results are printed in the log.

NOTE: Splice sites take into account directionality and reverse compliment.

4.3.3 –rem-genes-without-start-codon

The first three nucleotides of the sequence are analyzed to match ATG. With this flag alone, no alternate start codons are taken into account (use --allow-alternate-starts to include alternate starts). This task is performed by task_scripts/rem_genes_without_start.pl. Again, gene directionality is considered.

Genes that pass this filter are kept in the gene table and results are printed in the log.

4.3.4 –allow-alternate-starts

If --rem-genes-without-start-codon is used, the start codons of GTG and TTG will also be included alongside ATG. This may be useful in Prokaryotic annotations. This task is performed by task_scripts/rem_genes_without_start_alternate.pl.

Outputs are identical to --rem-genes-without-start-codon.

4.3.5 –rem-genes-without-stop-codon

The last three nucleotides of the sequence are analyzed to match TAA, TAG, and TGA. Currently, all end codons are assumed to be within the reported gene. This task is performed by task_scripts/rem_genes_without_stop.pl. Again, gene directionality is considered.

Following this step, a script called task_scripts/frame_detection.pl is run. It is designed to pick out any genes that technically have a stop codon as the last three nucleotides, but it is not real because the codon is actually out of frame. These are rare occurrences, often happening on negative strand genes that run into the beginning of a scaffold where the first three nucleotides of the scaffold are a reverse complement stop codon. To solve this, any gene whose CDS is not divisible by 3, is removed.

Genes that pass this filter are kept in the gene table and results are printed in the log.

4.3.6 –allowed-inframe-stop-codons [number]

Default: 0

Creates a filter that removes genes based on the presence of a stop codon that is not the last codon in the gene. For example, setting this parameter as 1 will allow one other stop codon between the methionine and the terminating stop codon.

If you are not filtering for start and stop codons, this will still work so long as there are stop codons within the amino acid sequence but not necessarily at the end.

Genes that pass this filter are kept in the gene table and results are printed in the log.

4.3.7 –splice-table

To understand splice usage, a splice-site table is printed to the log that tells the frequency of every type of splice site used. This command is performed by task_scripts/splice_table.pl.

The splice table will look something like this:

```
gt_ag    113699    99.286%
at_ac    39        0.034%
gc_ag    779       0.680%
```

This splice table will show you everything present in the file adjusted to lower case letters including N-bases. If you specify canonical genes only, the table will only show you gt_ag counts.

4.3.8 –nt-content

The CDS (all exon sequences) is analyzes for GC, AT, and N content by percent composition. This information is printed to the log. Here is an example of the output:

```
GC content:      46.708%
AT content:      53.271%
N content:       0.021%
```

CHAPTER 5

Output flags

5.1 Basic output flags

These are output flags that do not require the input of any fasta or EnTAP files.

5.1.1 --statistics

Statistics will be run on the gene table and printed to statistics.txt. This command is performed by task_scripts/classic_stats.pl. If a prefix is used, the statistics file will be named accordingly.

These are all the potential statistics in the reported format:

Number of genes: Number of monoexonic genes: Number of multiexonic genes:

Number of positive strand genes: Monoexonic: Multiexonic:

Number of negative strand genes: Monoexonic: Multiexonic:

Average overall gene size: Median overall gene size: Average overall CDS size: Median overall CDS size: Average overall exon size: Median overall exon size:

Average size of monoexonic genes: Median size of monoexonic genes: Largest monoexonic gene: Smallest monoexonic gene:

Average size of multiexonic genes: Median size of multiexonic genes: Largest multiexonic gene: Smallest multiexonic gene:

Average size of multiexonic CDS: Median size of multiexonic CDS: Largest multiexonic CDS: Smallest multiexonic CDS:

Average size of multiexonic exons: Median size of multiexonic exons: Average size of multiexonic introns: Median size of multiexonic introns:

Average number of exons per multiexonic gene: Median number of exons per multiexonic gene: Largest multiexonic exon: Smallest multiexonic exon: Most exons in one gene:

Average number of introns per multiexonic gene: Median number of introns per multiexonic gene: Largest intron: Smallest intron:

If your set is only monoexonics, a smaller version of the statistics will be printed that only contain the categories where monoexonic genes are evaluated.

5.1.2 –statistics-at-every-step

A statistical analysis of the gene table is run following every filtering step. This information is in the same format as regular --statistics but prints to the **log** following the information line for each flag. To ensure statistics.txt is created at the end, make sure to include -–statistics in your command.

5.1.3 –create-simple-gtf

Identical to --create-gtf, but lacks start and stop codon information. This option is significantly faster.

5.1.4 –create-gff3

An Ensembl v3 gff3 gff3 will be created that contains mRNA, exon, and intron information. ID, Name, and Parent information will be shown.

5.2 Fasta required output flags

In order to create fasta required outputs, you will need to provide a fasta input. See how to here. If a proper fasta is provided, you unlock all these flags:

5.2.1 –get-fasta-with-introns

The nucleotide fasta sequence is printed to genes_with_introns.fasta. The genes are always printed on the positive strand. This fasta will contain the intron sequences so number of genes printed will be the same for both with and without introns. The header for each sequence is the fifth column of the gene line in the gene table.

This command is performed by task_scripts/get_fasta_with_introns.pl. If a prefix is specified, the output fasta will be named accordingly.

5.2.2 –get-fasta-without-introns

The nucleotide fasta sequence is printed to genes_without_introns.fasta. The genes are always printed on the positive strand. This fasta will not contain the intron sequences. The header for each sequence is the fifth column of the gene line in the gene table.

This command is performed by task_scripts/get_fasta_with_introns.pl. If a prefix is specified, the output fasta will be named accordingly.

5.2.3 –get-protein-fasta

A protein fasta of the genes is created called genes_without_introns.fasta.faa. Genes never include the introns (because of course not). All genes are printed in the N-terminus to C-terminus orientation (so M would be first) but reverse complementation of the negative strand is considered to choose the correct amino acids. Stop codons are depicted as *. The header for each sequence is the fifth column of the gene line in the gene table.

This command is performed by task_scripts/get_protein_fasta.pl. If a prefix is specified, the output fasta will be named accordingly.

5.2.4 –create-gtf

A gtf file called out.gtf is created. If a prefix is specified, the gtf file will have it. This step is done with two scripts, task_scripts/add_start_stop_to_gene_table.pl and task_scripts/gtf_creator.pl.

Since GTF files (as a general rule) require start and stop codon information, the locations of the start and stop codon (if found) are added to the gene table and the final gtf. CDS scores that correspond to an exon are retrieved from the original input file if found and the "exon" attribute is returned to "CDS". Introns currently remain.

The source line does say gFACs. Not to steal the credit, it just might be helpful to know where the information is coming from particularly after filtering and rearranging.

5.3 Distributions flags

gFACs is capable of reformatting annotations into formats for distributions. It can provide distribution summaries or raw data. To signify distributions, a single flag followed by options may be used:

5.3.1 –distributions [option] [option] . . .

Activates the ability to create distributions. This task is always done last on the final version of the gene table. If a prefix is specified, all output files will reflect that.

All outputs are printed in a .tsv file that can be opened for viewing on excel or R. The options available for distributions are as follow:

5.3.2 exon_lengths

Creates the file exon_lengths_distributions.tsv. In it, a range of exon lengths and the corresponding representation is printed. In this example, --min-exon size was set to 40, which is reflected in the numbers:

```
exon_lengths    N
0-9      0
10-19    0
20-29    0
30-39    0
40-49    350
50-59    404
60-69    580
70-79    630
80-89    614
90-99    760
```

The above data, when rendered into a histogram using R, looks like this:

Exon distribution : Protea
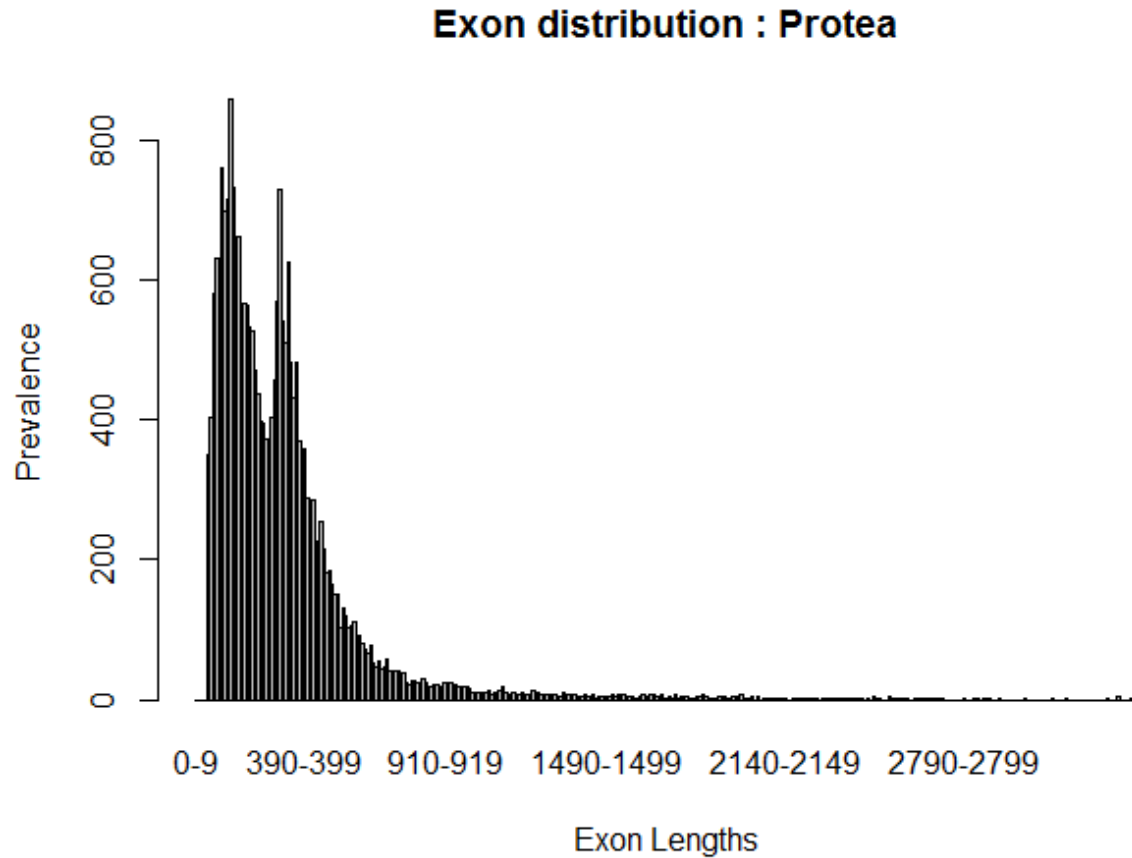
Notice that the curve is bimodal, which is indicative of the mono and multiexonic genes. Utilizing two runs one with --rem-monoexonics (red) and one with --rem-multiexonics (yellow) you can see the curves are indeed the difference in gene type where smaller exon lengths are in multiexonic genes:

Exon distribution : Protea
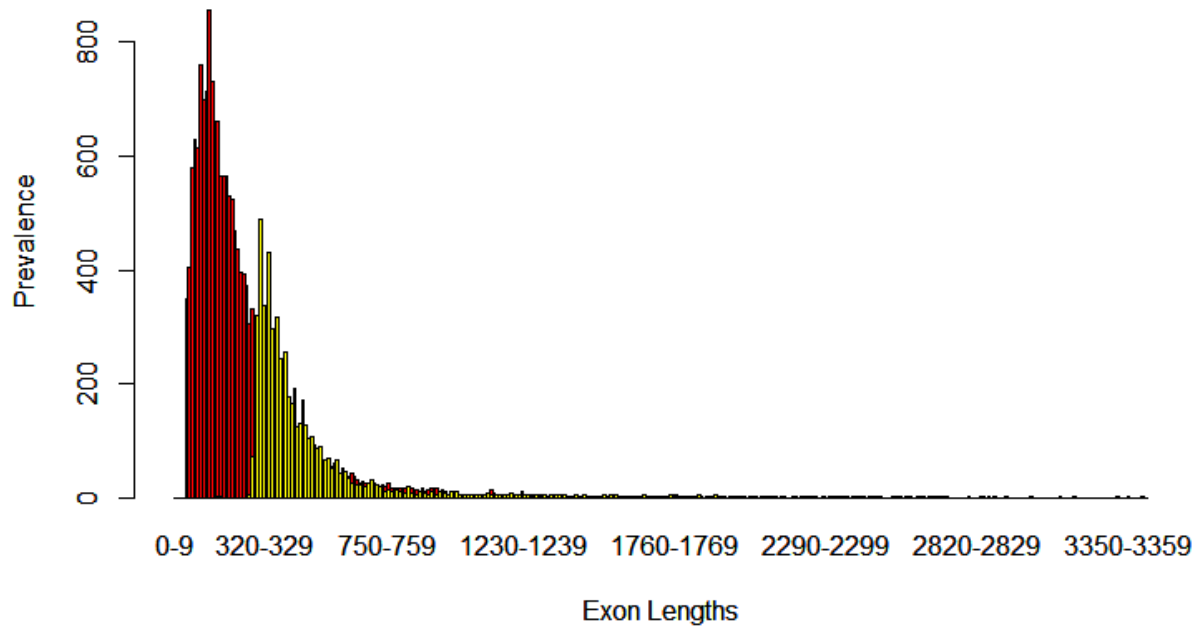
Advanced: Zoom of exon lengths can be controlled with a trailing number. This changes the size of the step. In the example above, the range of values as the cluster for the distribution is 10, but it can be controlled like this:

exon_lengths 5

This would change the above table to:

The default, if no number is chosen, is decided by the maximum exon length of the provided data. For a maximum length that is less than 100 nucleotides, the step is 1. For a maximum value of exon length that is more than 100 but less than 1,000, the step is 10 and so on.

Changing this step number should not drastically change the time it takes to run. However, the file will be larger and have more lines when a smaller number is used!

5.3.3 intron_lengths

Creates the file intron_lengths_distributions.tsv. In it, a range of intron lengths and the corresponding representation is printed. The outputs and applications are identical to exon_lengths.

Advanced: Zoom of intron lengths can be controlled with a trailing number. This changes the size of the step. It can be controlled like this:

intron_lengths 20

The default, if no number is chosen, is decided by the maximum intron length of the provided data. For a maximum length that is less than 100 nucleotides, the step is 1. For a maximum value of intron length that is more than 100 but less than 1,000, the step is 10 and so on.

Changing this step number should not drastically change the time it takes to run. However, the file will be larger and have more lines when a smaller number is used!

5.3.4 CDS_lengths

Creates the file CDS_lengths_distributions.tsv. In it, a range of CDS lengths and the corresponding representation is printed. The outputs and applications are identical to exon_lengths.

Advanced: Zoom of CDS lengths can be controlled with a trailing number. This changes the size of the step. It can be controlled like this:

CDS_lengths 25

The default, if no number is chosen, is decided by the maximum CDS length of the provided data. For a maximum length that is less than 100 nucleotides, the step is 1. For a maximum value of CDS length that is more than 100 but less than 1,000, the step is 10 and so on.

Changing this step number should not drastically change the time it takes to run. However, the file will be larger and have more lines when a smaller number is used!

5.3.5 gene_lengths

Creates the file gene_lengths_distributions.tsv. In it, a range of gene lengths and the corresponding representation is printed. These sequence lengths do include all introns. The outputs and applications are identical to exon_lengths.

Advanced: Zoom of gene lengths can be controlled with a trailing number. This changes the size of the step. It can be controlled like this:

gene_lengths 1000

The default, if no number is chosen, is decided by the maximum gene length of the provided data. For a maximum length that is less than 100 nucleotides, the step is 1. For a maximum value of gene length that is more than 100 but less than 1,000, the step is 10 and so on.

Changing this step number should not drastically change the time it takes to run. However, the file will be larger and have more lines when a smaller number is used!

5.3.6 exon_position

Analyzes and creates an output that evaluates exon position in a gene to its size. Position meaning which exon comes first. In positive strand genes, these are in the order they appear in the gene table. For reverse strand genes, the first exon is the last one to appear in the gene table. Creates the output file exon_position_distributions.tsv. The output looks like this:

```
exon_position   n(Individual exons)     n(Max exons in gene)    average_size    median_size     Min     Max
1       70923   28762   335.161 294     20      4955
2       42161   19901   229.762 170     20      4975
3       22260   8596    212.085 146     20      3210
4       13664   4376    195.893 131     20      8028
5       9288    2557    177.604 119     20      4056
6       6731    1656    172.398 114     20      6544
```

Exon position goes from 1 to whatever the maximum number of exons in one gene is. It will match what a statistics output would say. The second column is how many exons are representative of that position. The first exon support (70,923 above) will always be equal to the overall number of genes because even monoexonics have a first exon. (You can remove those, of course). You can also say there are 70,923 first exons, 42,161 second exons, etc. . .

The third column is how many genes have the first column number as their maximum number of exons. So, in the last row shown, there are 1,656 genes that have 6 total exons. There are 28,762 monoexonics then as well by this same logic.

The third and fourth columns are average and median size of an exon at that positon. The last two are minimum and maximum. If you use a minimum exon parameter (as I did above) it will be reflected!

5.3.7 exon_position_data

Provides the raw data in data_intron_position_distributions.tsv on exon positions alongside exon_position_distributions.tsv produced from the command above. This set of data can be used to make boxplots.

The data appears like this:

```
1    4379   694    117    806    993    605    492    102    106    93     5699   4578   593    1578   378    886    922    944    83     709
2    664    115    82     229    2882   444    115    240    4030   745    80     197    211    771    98     4333   8060   9910   189    531
3    383    2926   351    6307   96     202    1133   138    456    102    1067   140    276    743    191    2031   1309   1299   131    1567
4    98     455    78     1150   1413   1071   104    93     526    176    199    197    5313   2038   1641   179    1525   3193   126    4120
5    570    137    99     115    93     125    123    108    115    312    1904   622    104    613    198    83     490    6888   483    1712
6    430    97     120    104    1760   81     157    96     83     695    76     606    409    83     126    1329   96     190    1807   148
7    228    125    547    325    553    2870   87     167    2959   1358   90     146    167    332    1133   89     84     6996   149    524
8    136    533    356    3615   454    15203  923    148    211    88     115    6295   701    83     94     380    2746   133    4643   440
9    369    707    801    93     112    119    178    160    566    87     136    8994   189    167    1147   5560   82     78     2175   124
10   79     328    1341   2514   87     313    412    84     222    162    121    81     934    98     86     779    314    217    135    218
11   152    127    423    111    196    7459   242    383    3965   214    95     482    120    1338   1624   158    1657   108    184    88
12   96     252    923    122    476    89     101    117    530    103    889    1796   701    879    2862   288    135    3676   95     1508
13   1883   205    15194  881    782    6060   2702   94     269    82     140    113    3335   90     672    358    160    2220   116    2187
14   76     88     1756   94     275    153    59     91     92     417    82     413    4907   80     2328   104    93     829    103    395
15   678    653    100    107    101    740    86     1058   103    101    112    596    257    200    119    393    278    198    11154  1714
16   1420   116    96     106    416    407    172    4972   79     82     113    1229   1103   128    88     93     136    1325   1007   153
17   71     105    767    234    986    102    89     153    425    93     287    116    470    913    920    91     103    89     105    83
18   93     135    235    129    670    85     136    114    141    517    162    239    428    70     235    624    893    91     191    431
19   928    165    553    344    185    84     1152   100    768    369    94     88     322    75     4456   915    1771   310    1654   1027
20   106    5654   90     90     126    436    7128   620    1961   839    275    775    1039   198    105    145    996    90     100    90
21   4446   2071   94     1636   775    2233   2396   43     4235   132    162    149    433    116    765    85     2750   102    586    79
22   192    80     134    399    100    127    122    91     127    138    1718   646    3001   176    99     1744   134    73     5776   128
23   286    172    654    997    84     91     86     89     664    117    70     114    105    80     559    1526   79     154    102    3086
24   97     298    117    416    1032   77     271    1423   437    432    1681   553    108    3784   298    149    97     96     84     120
25   998    210    1651   130    142    85     1279   81     410    110    90     2690   3853   106    84     188    87     108
26   888    788    253    95     1946   121    921    2258   115    86     89     1032   178    1806   569    299
27   181    80     3183   603    1046   784    227    115    79     173    133    1440   5005
28   154    439    125    1410   131    92     78     240    288    110    9226
29   236    152    86     100    98     104    103    80     6718
30   87     91     93     1894   93     485    110    137    3452
31   689    122    84     90     1908   96     4998
32   104    94     107    16344  2053
```

The first column is the exon position and the following values in the row are the sizes of exons (non-sorted). The row will have as many columns as exon position data points. Notice how 32 (the last visible row) has only 5 numbers, showing there are only 5 genes that have a 32nd exon where the values are the sizes.

5.3.8 intron_position

Intron positioning works identically to exon positions. However, this will only include multiexonic genes! All header names have the same meaning as exon position.

Creates the output file intron_position_distributions.tsv.

5.3.9 intron_position_data

Intron position raw data works identically to exon positions and will also only include multiexonic genes. Creates the output file data_intron_position_distributions.tsv and the default intron_position_distributions.tsv.

A sample of a boxplot that can be created:
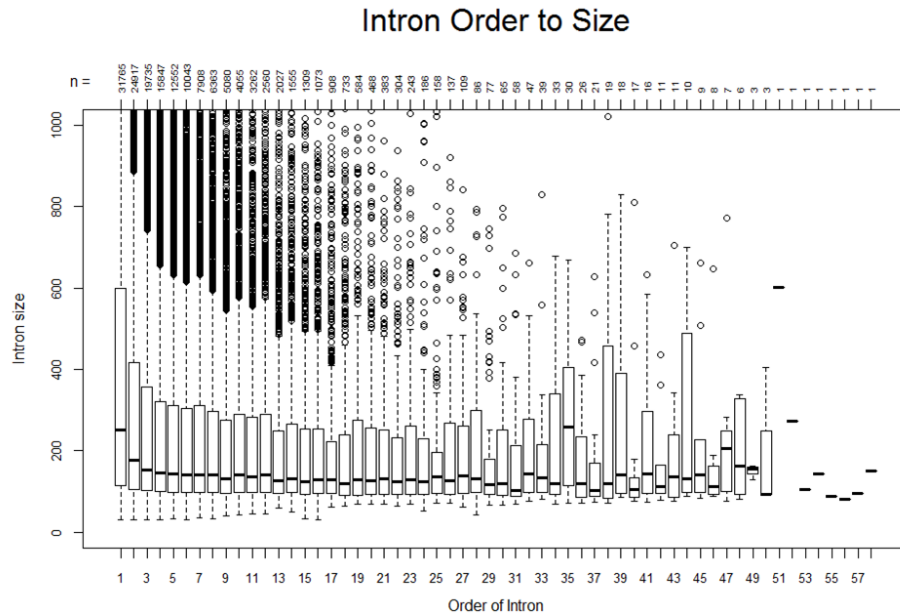
For how I created the boxplot, feel free to contact me!

5.4 Compatibility flags

Just as the gff3/gff/gtf file formats follow their owns rules, so do other software in needing specific input formats. Although the gFACs gtf is fairly standard, a few modifications must be made before it can safely be used within other programs. This includes transitions to other formats such as gff or gff3!

Several formats are alrady compatible by default. –create-gtf output is compatible with Jbrowse and protein/nucleotide FASTAs are compatible with EnTAP!

These options are still being developed and user input is more than welcome! Do you not see a format you would like added? Let me know!

To specify the compatibility arguments, use this flag:

5.4.1 –compatibility [option] [option] etc…

Allows the creation of software compatible files. Available format options are:

5.4.2 SnpEff

A gff file called snpeff_format.gff will be created that can be used for SnpEff build. This format can be used in the default.

format_diagnosis.pl

To determine what format you have, if it is ambiguous, format_diagnosis.pl may be able to help. The script will output information that you can compare with the table below to see if another format may work for you. To use the script:

```
perl format_diagnosis.pl [input_file]
```

The output will look something like this:

```
-bash-4.2$ perl format_diagnosis.pl /UCHC/LABS/Wegrzyn/gstats/format_dropoff/trimmed_i_trans_gstats.gff3
Format  partition       gene    mRNA    exon    CDS     intron  start_codon     stop_codon      Other third column
Your input      no[0]   yes[278]        yes[278]        yes[1583]       yes[1609]       no[0]   no[0]   no[0]   ()
```

The data tells you what information is present followed by the observed quality of the feature. In the above output, the line of the "gene" feature, comes up 278 times and matches that with mRNA. This is not always the case. It also has exon lines and CDS lines but NOT at the same frequency. So CDS will be the more important feature.

Given the comparison, you could choose several formats that might work. braker_2.05_gff3, braker_2.05_gtf, gFACs_gtf, and several more.

FORMAT properties:

NOTE: In this table, know that each format example is **NOT** the same file. The information inside the [brackets] is just an example number and judgement on format should be made from ratios. Your file will not fit the actual numbers in the brackets above, but the ratio between a format's exon to CDS counts may be the same. These were derived from my own collection of sample files across different species and projects.

Format	partition	gene	mRNA	exon	CDS	intron	start_codon	stop_codon	Other third column
gmap_2017_03_17_gff3	Yes [6026]	Yes [6026]	Yes [6026]	Yes [31116]	Yes [30948]	No [0]	No [0]	No [0]	()
braker_2.05_gff3	No [0]	Yes [37757]	Yes [39297]	Yes [137280]	Yes [137280]	Yes [104809]	Yes [32571]	Yes [32936]	(initial single internal terminal)
braker_2.05_gff	No [0]	Yes [37757]	No [0]	No [0]	Yes [137280]	Yes [104809]	Yes [32571]	Yes [32936]	(terminal internal initial transcript single)
braker_2.05_gtf	No [0]	Yes [37757]	No [0]	Yes [137280]	Yes [137280]	Yes [104809]	Yes [32571]	Yes [32936]	(initial single transcript terminal internal)
braker_2.0_gff3	No [0]	Yes [60358]	Yes [63356]	Yes [270267]	Yes [270267]	Yes [209203]	Yes [61583]	Yes [61894]	()
braker_2.0_gff	No [0]	Yes [60358]	No [0]	No [0]	Yes [270267]	Yes [209203]	Yes [61583]	Yes [61894]	(transcript)
braker_2.0_gtf	No [0]	Yes [60358]	No [0]	Yes [270267]	Yes [270267]	Yes [209203]	Yes [61583]	Yes [61894]	(transcript)
maker_2.31.9_gff	Yes [118474]	Yes [34322]	Yes [34322]	Yes [64118]	Yes [64118]	No [0]	No [0]	No [0]	(translated_nucleotide_match contig three_prime_UTR five_prime_UTR expressed_sequence_match match_part protein_match)
genomethreader_1.6.6_gff3	Yes [598452]	Yes [598452]	Yes [816371]	Yes [1182140]	Yes [511162]	No [0]	No [0]	No [0]	(three_prime_cis_splice_site five_prime_cis_splice_site)
gffread_0.9.12_gff3	No [0]	No [0]	Yes [44128]	Yes [90451]	Yes [90451]	No [0]	No [0]	No [0]	()
exonerate_2.4.0_gff	No [0]	Yes [2126]	No [0]	Yes [3843]	Yes [3843]	Yes [1717]	No [0]	No [0]	(similarity splice5 splice3)
EVM_1.1.1_gff3	No [0]	Yes [20678]	Yes [20678]	Yes [66408]	Yes [66408]	No [0]	No [0]	No [0]	()
gFACs_gtf	No [0]	Yes [42798]	No [0]	No [0]	Yes [110386]	Yes [67588]	Yes [42798]	Yes [42798]	()
refseq_gff	Yes [1]	Yes [54981]	Yes [60516]	Yes [413261]	Yes [318259]	No [0]	No [0]	No [0]	(match region sequence_feature cDNA_match lnc_RNA pseudogene primary_transcript rRNA transcript tRNA miRNA)