MAVIS

creisle

Jan 26, 2020

CONTENTS

1 About **3**
- 1.1 Getting Help . . . 3
- 1.2 Install Instructions . . . 3
- 1.3 Citation . . . 5

2 Inputs **7**
- 2.1 Reference Input Files . . . 7
- 2.2 MAVIS standard input file format . . . 10

3 Supported Dependencies **13**
- 3.1 Job Schedulers . . . 13
- 3.2 Aligners . . . 13
- 3.3 SV Callers . . . 14

4 Running the Pipeline **17**
- 4.1 Running MAVIS using a Job Scheduler . . . 17
- 4.2 MAVIS (Mini) Tutorial . . . 19
- 4.3 MAVIS (Full) Tutorial . . . 21

5 Configuration and Settings **27**
- 5.1 Pipeline Configuration File . . . 27
- 5.2 Environment Variables . . . 27
- 5.3 Adjusting the Resource Requirements . . . 27

6 Resource Requirements **29**
- 6.1 Validation Resources . . . 29
- 6.2 Annotation Resources . . . 30

7 Theory and Models **31**
- 7.1 Introduction . . . 31
- 7.2 Evidence . . . 31
- 7.3 Classifying Events . . . 35
- 7.4 Assembling Contigs . . . 36
- 7.5 Annotating Events . . . 36
- 7.6 Predicting Splicing Patterns . . . 37
- 7.7 Pairing Similar Events . . . 38

8 Illustrations **39**
- 8.1 Fusion Diagrams . . . 39
- 8.2 Transcript Overlays . . . 39

9 Guidelines for Contributors **41**
9.1 Getting Started . . . 41
9.2 Install (for Development) . . . 41
9.3 Build the Sphinx Documentation . . . 41
9.4 Deploy to PyPi . . . 42
9.5 Major Assumptions . . . 43
9.6 Current Limitations . . . 43
9.7 Computing Code coverage . . . 43

10 Package Documentation **45**
10.1 align module . . . 45
10.2 annotate subpackage . . . 45
10.3 assemble module . . . 46
10.4 bam subpackage . . . 46
10.5 blat module . . . 47
10.6 breakpoint module . . . 47
10.7 cluster subpackage . . . 47
10.8 config module . . . 48
10.9 constants module . . . 48
10.10 error module . . . 48
10.11 illustrate subpackage . . . 48
10.12 interval module . . . 48
10.13 pairing subpackage . . . 48
10.14 validate subpackage . . . 49
10.15 schedule subpackage . . . 50
10.16 summary subpackage . . . 51
10.17 tools module . . . 51
10.18 util module . . . 51

11 Glossary **53**
11.1 General Terms . . . 53
11.2 Configurable Settings . . . 54
11.3 Column Names . . . 61

12 Indices and tables **65**

Bibliography **67**

Python Module Index **69**

Index **71**

CHAPTER

ONE

ABOUT

MAVIS is python command-line tool for the post-processing of structural variant calls. The general MAVIS pipeline consists of six main stages

- convert
- cluster
- validate
- annotate
- pairing
- summary

1.1 Getting Help

All steps in the MAVIS pipeline are called following the main mavis entry point. The usage menu can be viewed by running without any arguments, or by giving the -h/–help option

```
mavis -h
```

Help sub-menus can be found by giving the pipeline step followed by no arguments or the -h options

```
mavis cluster -h
```

Common problems and questions are addressed on the wiki. If you have a question or issue that is not answered there (or already a github issue) please submit a github issue to our github page or contact us by email at mavis@bcgsc.ca

1.2 Install Instructions

There are 3 major steps to setting up and installing MAVIS. If you are a developer contributing to mavis, please see the instructions for developers page instead

1.2.1 1. Install Aligner

In addition to the python package dependencies, MAVIS also requires an aligner to be installed. Currently the only aligners supported are blat and bwa mem. For MAVIS to run successfully the aligner must be installed and accessible on the path. If you have a non-standard install you may find it useful to edit the PATH environment variable. For example

```
export PATH=/path/to/directory/containing/blat/binary:$PATH
```

blat is the default aligner. To configure MAVIS to use bwa mem as a default instead, use the MAVIS environment variables. Make sure to specify BOTH of the variables below to change the default aligner.

```
export MAVIS_ALIGNER='bwa mem'
export MAVIS_ALIGNER_REFERENCE=/path/to/mem/fasta/ref/file
```

After this has been installed MAVIS itself can be installed through pip

1.2.2 2. Install MAVIS

Install using pip

The easiest way to install MAVIS is through the python package manager, pip. If you do not have python3 installed it can be found here

Ensuring you have a recent version of pip and setuptools will improve the install experience. Older versions of pip and setuptools may have issues with obtaining some of the mavis python dependencies

```
pip install --upgrade pip setuptools
```

or (for Anaconda users)

```
conda update pip setuptools
```

If this is not a clean/new python install it may be useful to set up mavis in a virtual python environment

Then install mavis itself

```
pip install mavis
```

This will install mavis and its python dependencies.

Install using Buildout

Alternatively you can use the bootstrap/buildout to install mavis into bin/mavis

```
git clone https://github.com/bcgsc/mavis.git
cd mavis
pip install zc.buildout
python bootstrap.py
bin/buildout
```

This will install mavis and its python dependencies into eggs inside the cloned mavis directory which can be used by simply running bin/mavis

1.2.3 3. Build or Download Reference Files

After MAVIS is installed the reference files must be generated (or downloaded) before it can be run. A simple bash script to download the hg19 reference files and generate a MAVIS environment file is provided under mavis/tools for convenience.

```
cd /path/to/where/you/want/to/put/the/files
wget https://raw.githubusercontent.com/bcgsc/mavis/master/tools/get_hg19_reference_
↪files.sh
bash get_hg19_reference_files.sh
source reference_inputs/hg19_env.sh
```

Once the above 3 steps are complete MAVIS is ready to be run. See the MAVIS tutorial to learn about running MAVIS.

1.3 Citation

If you use MAVIS as a part of your project please cite

Reisle,C. et al. (2018) MAVIS: Merging, Annotation, Validation, and Illustration of Structural variants. Bioinformatics.

CHAPTER

TWO

INPUTS

2.1 Reference Input Files

There are several reference files that are required for full functionality of the MAVIS pipeline. If the same reference file will be reused often then the user may find it helpful to set reasonable defaults. Default values for any of the reference file arguments can be *configured through environment variables*.

To improve the install experience for the users, different configurations of the MAVIS annotations file have been made available. These files can be downloaded below, or if the required configuration is not available, instructions on generating the annotations file can be found below.

File Name (Type/Format)	Environment Variable	Download
reference genome (*fasta*)	`MAVIS_REFERENCE_GENOME`	
annotations (*JSON*)	`MAVIS_ANNOTATIONS`	
masking (text/tabbed)	`MAVIS_MASKING`	
template metadata (text/tabbed)	`MAVIS_TEMPLATE_METADATA`	
DGV annotations (text/tabbed)	`MAVIS_DGV_ANNOTATION`	
aligner reference	`MAVIS_ALIGNER_REFERENCE`	

If the environment variables above are set they will be used as the default values when any step of the pipeline script is called (including generating the template config file)

2.1.1 Reference Genome

These are the sequence files in fasta format that are used in aligning and generating the fusion sequences.

2.1.2 Template Metadata

This is the file which contains the band information for the chromosomes. This is only used during visualization.

The structure of the file should look something like this

```
chr1    0       2300000 p36.33  gneg
chr1    2300000 5400000 p36.32  gpos25
chr1    5400000 7200000 p36.31  gneg
chr1    7200000 9200000 p36.23  gpos25
chr1    9200000 12700000        p36.22  gneg
```

2.1.3 Masking File

The masking file is a tab delimited file which contains regions that we should ignore calls in. This can be used to filter out regions with known false positives, bad mapping, centromeres, telomeres etc. An example of the expected format is shown below. The file should have four columns: chr, start, end and name.

```
#chr    start   end     name
chr1    0       2300000 centromere
chr1    9200000 12700000        telomere
```

The pre-built masking files in the downloads table above are telomere regions, centromere regions (based on the cytoband file), and nspan regions (computed with tools/find_repeats.py).

Masking is not required (can provide a header-only file), but is recommended as it will improve performance and specificity.

2.1.4 Annotations

This is a custom file format. It is a *JSON* file which contains the gene, transcript, exon, translation and protein domain positional information

Pre-built annotation files can be downloaded above. The 'best transcript' flag is based on an in-house model. We have also pre-built the ensembl annotations file including non-coding transcripts below.

Warning: It is worth noting that using the reference annotation file including the non-coding genes will require an increase in the default amount of memory for the annotation step due to the increased size of the annotations file. On our standard COLO829 we increased the default memory for the annotation step from 12G to 18G.

Warning: the `load_reference_genes()` will only load valid translations. If the cds sequence in the annotation is not a multiple of `CODON_SIZE` or if a reference genome (sequences) is given and the cds start and end are not M and * amino acids as expected the translation is not loaded

Example of the *JSON* file structure can be seen below

```
[
    {
        "name": string,
        "start": int,
        "end": int
        "aliases": [string, string, ...],
        "transcripts": [
            {
                "name": string,
                "start": int,
                "end": int,
                "exons": [
                    {"start": int, "end": int, "name": string},
                    ...
                ],
                "cdna_coding_start": int,
                "cdna_coding_end": int,
                "domains": [
```

(continues on next page)

(continued from previous page)

```
                    {
                        "name": string,
                        "regions": [
                            {"start" aa_start, "end": aa_end}
                        ],
                        "desc": string
                    },
                    ...
                ]
            },
            ...
        ]
    },
    ...
}
```

The provided files were generated with *Ensembl*, however it can be generated from any database with the necessary information so long as the above *JSON* structure is respected.

Generating the Annotations from Ensembl

There is a helper script included with mavis to facilitate generating the custom annotations file from an instance of the *Ensembl* database. This uses the *Ensembl* perl api to connect and pull information from the database. This has been tested with both Ensembl69 and Ensembl79.

Instructions for downloading and installing the perl api can be found on the ensembl site

1. **Make sure the ensembl perl api modules are added to the PERL5LIB environment variable**

Also ensure that the tools directory is on the PERL5LIB path so that the TSV.pm module can be found

```
INSTALL_PATH=$(pwd)
PERL5LIB=${PERL5LIB}:$HOME/ensembl_79/bioperl-live
PERL5LIB=${PERL5LIB}:$HOME/ensembl_79/ensembl/modules
PERL5LIB=${PERL5LIB}:$HOME/ensembl_79/ensembl-compara/modules
PERL5LIB=${PERL5LIB}:$HOME/ensembl_79/ensembl-variation/modules
PERL5LIB=${PERL5LIB}:$HOME/ensembl_79/ensembl-funcgen/modules
# include tools/TSV.pm module
PERL5LIB=${PERL5LIB}:$INSTALL_PATH/tools
export PERL5LIB
```

2. **Run the perl script**

The below instructions are shown running from inside the tools directory to avoid prefixing the script name, but it is not required to be run from here provided the above step has been executed correctly.

you can view the help menu by running

```
perl generate_ensembl_json.pl
```

you can override the default parameters (based on hard-coded defaults or environment variable content) by providing arguments to the script itself

```
perl generate_ensembl_json.pl --best_transcript_file /path/to/best/transcripts/file --
↪output /path/to/output/json/file.json
```

or if you have configured the environment variables as given in step 2, then simply provide the output path

```
perl generate_ensembl_json.pl --output /path/to/output/json/file.json
```

2.1.5 DGV (Database of Genomic Variants) Annotations

The DGV annotations file contains regions corresponding to what is found in the database of genomic variants. This is used to annotate events that are found in healthy control samples and therefore may not be of interest if looking for somatic events.

The above (downloads table) files were generated from from DGV and reformatted to have 4 columns after download. We used awk to convert the raw file

```
awk '{print $2"\t"$3"\t"$4"\t"$1} GRCh37_hg19_variants_2016-05-15.txt > dgv_hg19_
↪variants.tab
```

Note in hg19 the column is called "name" and in hg38 the column is called "variantaccession". An example is shown below

```
#chr     start    end      name
1        1        2300000 nsv482937
1        10001    22118    dgv1n82
1        10001    127330   nsv7879
```

2.1.6 Aligner Reference

The aligner reference file is the reference genome file used by the aligner during the validate stage. For example, if *blat* is the aligner then this will be a *2bit* file.

2.2 MAVIS standard input file format

These requirements pertain to the columns of input files from the various tools you want to merge. The input files should be tab-delimited text files. Comments at the top of may be included. Comments should begin with two hash marks. They will be ignored when the file is read

```
## This is a comment
```

The header row contains the column names and is the first row following the comments (or the first row if no comments are included). Optionally the header row may (or may not) begin with a hash which will be stripped out on read

```
## This is a comment
## this is another comment
# this is the header row
```

A simple input file might look as follows

```
## File created at: 2018-01-02
## Generated by: MAVIS v1.0.0
#break1_chromosome  break1_position_start   break1_position_end break2_chromosome␣
↪break2_position_start break2_position_end
X   1234    1234    X   77965   77965
```

2.2.1 Required Columns

- *break1_chromosome*
- *break1_position_start*
- *break1_position_end* (can be the same as break1_position_start)
- *break2_chromosome*
- *break2_position_start*
- *break2_position_end* (can be the same as break2_position_start)

2.2.2 Optional Columns

Optional Columns that are not given as input will be added with default (or command line parameter options) during the clustering stage of MAVIS as some are required for subsequent pipeline steps

- *break1_strand* (defaults to not-specified during clustering)
- *break1_orientation* (expanded to all possible values during clustering)
- *break2_strand* (defaults to not-specified during clustering)
- *break2_orientation* (expanded to all possible values during clustering)
- *opposing_strands* (expanded to all possible values during clustering)
- *stranded* (defaults to False during clustering)
- *library* (defaults to command line library parameter during clustering)
- *protocol* (defaults to command line protocol parameter during clustering)
- *tools* (defaults to an empty string during clustering)

2.2.3 Summary by Pipeline Step

The different pipeline steps of MAVIS have different input column requirements. These are summarized below (for the pipeline steps which can act as the pipeline start)

column name	cluster	annotate	validate
break1_chromosome			
break1_position_start			
break1_position_end			
break2_chromosome			
break2_position_start			
break2_position_end			
break1_strand			
break1_orientation			
break2_strand			
break2_orientation			
opposing_strands			
stranded			
library			
protocol			
tools			
event_type			

Some native tool outputs are *supported* and have built in methods to convert to the above format. Any unsupported tools can be used as long as the user converts the tools native output to match the above format.

CHAPTER

THREE

SUPPORTED DEPENDENCIES

MAVIS integrates with *SV callers*, *job schedulers*, and *aligners*. While some of these dependencies are optional, all currently supported options are detailed below. The versions column in the tables below list all the versions which were tested for each tool. Each version listed is known to be compatible with MAVIS.

3.1 Job Schedulers

MAVIS can be run locally without a job scheduler (`MAVIS_SCHEDULER=LOCAL`) however, due to the computational resources generally required, it is recommended that you use one of the supported schedulers listed below.

Name	Version(s)	Environment Setting
TORQUE	6.1.2	MAVIS_SCHEDULER=TORQUE
SGE	8.1.8	MAVIS_SCHEDULER=SGE
SLURM	17.02.1-2	MAVIS_SCHEDULER=SLURM

Users requiring support for other schedulers may make a request by submitting an issue to our github page. Additionally, developers looking to extend the functionality may submit a pull request (Please see the guidelines for contributors).

MAVIS running locally uses the python `concurrent.futures` library to manage jobs.

3.2 Aligners

Two aligners are supported *bwa* and *blat* (default).

Name	Version(s)	Environment Setting
blat	36x2 36	MAVIS_ALIGNER=blat
bwa mem	0.7.15-r1140 0.7.12	MAVIS_ALIGNER='bwa mem'

Note: When setting the aligner you will also need to set the *aligner_reference* to match

---

3.3 SV Callers

MAVIS supports output from a wide-variety of *SV* callers. Assumptions are made for each tool based on interpretation of the output and the publications for each tool. The tools and versions currently supported are given below. Versions listed indicate the version of the tool for which output files have been tested as input into MAVIS

MAVIS also supports a *general VCF input*. It should be noted however that the tool tracked will only be listed as 'vcf' then.

Name	Version(s)	MAVIS input	Publication
BreakDancer	`1.4.5`	`Tools main output file(s)`	[Chen-2009]
BreakSeq	`2.2`	`work/breakseq.vcf.gz`	[Abyzov-2015]
Chimeras-can	`0.4.5`	`*.bedpe`	[Iyer-2011]
CNVnator	`0.3.3`	`Tools main output file(s)`	[Abyzov-2011]
DeFuse	`0.6.2`	`results/results.classify.tsv`	[McPherson-2011]
DELLY	`0.6.1 0.7.3`	`combined.vcf` (converted from bcf)	[Rausch-2012]
Manta	`1.0.0`	`{diploidSV,somaticSV}.vcf`	[Chen-2016]
Pindel	`0.2.5b9`	`Tools main output file(s)`	[Ye-2009]
Trans-ABySS	`1.4.8 (custom)`	`{indels/events_novel_exons,fusions/*}.tsv`	[Robertson-2010]
Strelka	`1.0.6`	`passed.somatic.indels.vcf`	[Saunders-2012]
STAR-Fusion	`1.4.0`	`star-fusion.fusion_predictions.abridged.tsv`	[Haas-2017]

Note: *Trans-ABySS*: The trans-abyss version used was an in-house dev version. However the output columns are compatible with 1.4.8 as that was the version branched from. Additionally, although indels can be used from both genome and transcriptome outputs of Trans-ABySS, it is reccommended to only use the genome indel calls as the transcriptome indels calls (for versions tested) introduce a very high number of false positives. This will slow down validation. It is much faster to simply use the genome indels for both genome and transcriptome.

3.3.1 DELLY Post-processing

Some post-processing on the delly output files is generally done prior to input. The output BCF files are converted to a VCF file

```
bcftools concat -f /path/to/file/with/vcf/list --allow-overlaps --output-type v --
↪output combined.vcf
```

3.3.2 Writing A Custom Conversion Script

Logic Example - Chimerascan

The following is a description of how the conversion script for *Chimerascan* was generated. While this is a built-in conversion command now, the logic could also have been put in an external script. As mentioned above, there are a number of assumptions that had to be made about the tools output to convert it to the *standard mavis format*. Assumptions were then verified by reviewing at a series of called events in *IGV*. In the current example, *Chimerascan* output has six columns of interest that were used in the conversion

- start3p
- end3p
- strand3p
- start5p
- end5p
- strand5p

The above columns describe two segments which are joined. MAVIS requires the position of the join. It was assumed that the segments are always joined as a sense fusion. Using this assumption there are four logical cases to determine the position of the breakpoints.

i.e. the first case would be: If both strands are positive, then the end of the five-prime segment (end5p) is the first breakpoint and the start of the three-prime segment is the second breakpoint

The logic for all cases is shown in the code below

```
def _parse_chimerascan(row):
    """
    transforms the chimerscan output into the common format for expansion. Maps the␣
↪input column
    names to column names which MAVIS can read
    """
    std_row = {}
    for retained_column in ['genes5p', 'genes3p']:
        if retained_column in row:
            std_row['{}_{}'.format(SUPPORTED_TOOL.CHIMERASCAN, retained_column)] =␣
↪row[retained_column]
    if TRACKING_COLUMN not in row:
        std_row[TRACKING_COLUMN] = '{}-{}'.format(SUPPORTED_TOOL.CHIMERASCAN, row[
↪'chimera_cluster_id'])

    std_row.update({COLUMNS.break1_chromosome: row['chrom5p'], COLUMNS.break2_
↪chromosome: row['chrom3p']})
    if row['strand5p'] == '+':
        std_row[COLUMNS.break1_position_start] = row['end5p']
        std_row[COLUMNS.break1_orientation] = ORIENT.LEFT
    else:
        std_row[COLUMNS.break1_position_start] = row['start5p']
        std_row[COLUMNS.break1_orientation] = ORIENT.RIGHT
    if row['strand3p'] == '+':
        std_row[COLUMNS.break2_position_start] = row['start3p']
        std_row[COLUMNS.break2_orientation] = ORIENT.RIGHT
    else:
        std_row[COLUMNS.break2_position_start] = row['end3p']
        std_row[COLUMNS.break2_orientation] = ORIENT.LEFT
```

(continues on next page)

(continued from previous page)

```
    std_row[COLUMNS.opposing_strands] = row['strand5p'] != row['strand3p']
    return std_row
```

3.3.3 Calling A Custom Conversion Script

Custom conversion scripts can be specified during *automatic config generation* using the `--external_conversion` option.

Note: Any external conversion scripts must take a `-o` option which requires a single outputfile argument. This outputfile must be the converted file output by the script. Additionally, the conversion script must be specified by its full path name and have executeable permissions.

In the following example the user has created a custom conversion script `my_convert_script.py` which they are passing an input file named `my_input1.txt`.

```
mavis config --external_conversion my_converted_input1 "my_convert_script.py my_
↪input1.txt ... "
```

This will then be called during the pipeline step as

```
my_convert_script.py my_input1.txt ... -o /path/to/output/dir/converted_inputs/my_
↪converted_input1.tab
```

You can also re-use the same conversion script if you have multiple inputs to convert simply by specifying a different alias

```
mavis config \
    --external_conversion my_converted_input1 "my_convert_script.py my_input1.txt" \
    --external_conversion my_converted_input2 "my_convert_script.py my_input2.txt"
```

3.3.4 General VCF inputs

Assuming that the tool outputting the VCF file follows standard conventions, then it is possible to use a general VCF conversion that is not tool-specific. Given the wide variety in content for VCF files, MAVIS makes a number of assumptions and the VCF conversion may not work for all VCFs. In general MAVIS follows the VCF 4.2 specification. If the input tool you are using differs, it would be better to use a *custom conversion script*.

Assumptions on non-standard INFO fields

- `PRECISE` if given, Confidence intervals are ignored if given in favour of exact breakpoint calls using pos and END as the breakpoint positions
- `CT` values if given are representative of the breakpoint orientations.
- `CHR2` is given for all interchromosomal events

CHAPTER

FOUR

RUNNING THE PIPELINE

4.1 Running MAVIS using a Job Scheduler

The setup step of MAVIS is set up to use a *Job Schedulers* job scheduler on a compute cluster. will generate submission scripts and a wrapper bash script for the user to execute on their cluster head node.

Figure 4.1.: The MAVIS pipeline is highly configurable. Some pipeline steps (cluster, validate) are optional and can be automatically skipped. The standard pipeline is far-left.

The most common use case is *auto-generating a configuration file* and then running the pipeline setup step. The pipeline setup step will run clustering and create scripts for running the other steps.

```
mavis config .... -w config.cfg
mavis setup config.cfg -o /path/to/top/output_dir
```

This will create the build.cfg configuration file, which is used by the scheduler to submit jobs. To use a particular scheduler you will need to set the *MAVIS_SCHEDULER* environment variable. After the build configuration file has been created you can run the mavis schedule option to submit your jobs

```
ssh cluster_head_node
mavis schedule -o /path/to/output_dir --submit
```

This will submit a series of jobs with dependencies.

Figure 4.2.: Dependency graph of MAVIS jobs for the standard pipeline setup. The notation on the arrows indicates the SLURM setting on the job to add the dependency on the previous job.

4.1.1 Configuring Scheduler Settings

There are multiple ways to configure the scheduler settings. Some of the configurable options are listed below

- *queue* `MAVIS_QUEUE`
- *memory_limit* `MAVIS_MEMORY_LIMIT`
- *time_limit* `MAVIS_TIME_LIMIT`
- *import_env* `MAVIS_IMPORT_ENV`
- *scheduler* `MAVIS_SCHEDULER`

For example to set the job queue default using an *environment variable*

```
export MAVIS_QUEUE=QUEUENAME
```

Or it can also be added to the config file manually

```
[schedule]
queue = QUEUENAME
```

4.1.2 Troubleshooting Dependency Failures

The most common error to occur when running MAVIS on the cluster is a memory or time limit exception. These can be detected by running the schedule step or looking for dependency failures reported on the cluster. The suffix of the job name will be a number and will correspond to the suffix of the job directory.

```
mavis schedule -o /path/to/output/dir
```

This will report any failed jobs. For example if this were a crash report for one of the validation jobs we might expect to see something like below in the schedule output

```
[2018-05-31 13:02:06] validate
                        MV_<library>_<batch id>-<task id> (<job id>) is FAILED
                          CRASH: <error from log file>
```

Any jobs in an error, failed, etc. state can be resubmitted by running mavis schedule with the resubmit flag

```
mavis schedule -o /path/to/output/dir --resubmit
```

If a job has failed due to memory or time limits, editing the `/path/to/output/dir/build.cfg` file can allow the user to change a job without resetting up and rerunning the other jobs. For example, below is the configuration for a validation job

```
[MV_mock-A47933_batch-D2nTiy9AhGye4UZNapAik6]
stage = validate
job_ident = 1691742
name = MV_mock-A47933_batch-D2nTiy9AhGye4UZNapAik6
dependencies =
script = /path/to/output/dir/mock-A47933_diseased_transcriptome/validate/submit.sh
status = FAILED
output_dir = /path/to/output/dir/mock-A47933_diseased_transcriptome/validate/batch-
↪D2nTiy9AhGye4UZNapAik6-{task_ident}
stdout = /path/to/output/dir/mock-A47933_diseased_transcriptome/validate/batch-
↪D2nTiy9AhGye4UZNapAik6-{task_ident}/job-{name}-{job_ident}-{task_ident}.log
created_at = 1527641526
status_comment =
memory_limit = 18000
queue = short
time_limit = 57600
import_env = True
mail_user =
mail_type = NONE
concurrency_limit = None
task_list = 1
    2
    3
```

The memory_limit is in Mb and the time_limit is in seconds. Editing the values here will cause the job to be resubmitted with the new values.

Warning: Incorrectly editing the build.cfg file may have unanticipated results and require re-setting up MAVIS to fix. Generally the user should ONLY edit `memory_limit` and `time_limit` values.

If memory errors are frequent then it would be better to adjust the default values (*trans_validation_memory*, *validation_memory*, *time_limit*)

4.2 MAVIS (Mini) Tutorial

This tutorial is based on the data included in the tests folder of MAVIS. The data files are very small and this tutorial is really only intended for testing a MAVIS install. The data here is simulated and results are not representitive of the typical events you would see reported from MAVIS. For a more complete tutorial with actual fusion gene examples, please see the full-tutorial below.

The first step is to clone or download a zip of the MAVIS repository (https://github.com/bcgsc/mavis). You will need the tests directory. The tag you check out should correspond to the MAVIS version you have installed

```
git clone https://github.com/bcgsc/mavis.git
git checkout v2.0.0
mv mavis/tests .
rm -r mavis
```

Now you should have a folder called `tests` in your current directory. You will need to specify the scheduler if you want to test one that is not the default. For example

```
export MAVIS_SCHEDULER=LOCAL
```

Since this is a trivial example, it can easily be run locally. By default MAVIS in local mode will run a maximum of 1 less than the current cpu count processes. If you are running other things on the same machine you may find it useful to set this directly.

```
export MAVIS_CONCURRENCY_LIMIT=2
```

The above will limit mavis to running 2 processes concurrently.

Now you are ready to run MAVIS itself. This can be done in two commands (since the config file we are going to use is already built). First set up the pipeline

```
mavis setup tests/data/pipeline_config.cfg -o output_dir
```

Now if you run the schedule step (without the submit flag, schedule acts as a checker) you should see something like

```
mavis schedule -o output_dir/
```

```
                      MAVIS: 1.8.4
                      hostname: gphost08.bcgsc.ca
[2018-06-01 12:19:31] arguments
                        command = 'schedule'
                        log = None
                        log_level = 'INFO'
                        output = 'output_dir/'
                        resubmit = False
                        submit = False
```

(continues on next page)

(continued from previous page)

```
[2018-06-01 12:19:31] validate
                        MV_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-1 is NOT SUBMITTED
                        MV_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-2 is NOT SUBMITTED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-1 is NOT SUBMITTED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-2 is NOT SUBMITTED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-3 is NOT SUBMITTED
[2018-06-01 12:19:31] annotate
                        MA_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-1 is NOT SUBMITTED
                        MA_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-2 is NOT SUBMITTED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-1 is NOT SUBMITTED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-2 is NOT SUBMITTED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-3 is NOT SUBMITTED
[2018-06-01 12:19:31] pairing
                        MP_batch-s4W2Go4tinn49nkhSuusrE is NOT SUBMITTED
[2018-06-01 12:19:31] summary
                        MS_batch-s4W2Go4tinn49nkhSuusrE is NOT SUBMITTED
                      rewriting: output_dir/build.cfg
```

Adding the submit argument will start the pipeline

```
mavis schedule -o output_dir/ --submit
```

After this completes, run schedule without the submit flag again and you should see something like

```
                      MAVIS: 1.8.4
                      hostname: gphost08.bcgsc.ca
[2018-06-01 13:15:28] arguments
                        command = 'schedule'
                        log = None
                        log_level = 'INFO'
                        output = 'output_dir/'
                        resubmit = False
                        submit = False
[2018-06-01 13:15:28] validate
                        MV_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-1␣
↪(zQJYndSMimaoALwcSSiYwi) is COMPLETED
                        MV_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-2␣
↪(BHFVf3BmXVrDUA5X4GGSki) is COMPLETED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-1␣
↪(tUpx3iabCrpR9iKu9rJtES) is COMPLETED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-2␣
↪(hgmH7nqPXZ49a8yTsxSUWZ) is COMPLETED
                        MV_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-3␣
↪(cEoRN582An3eAGALaSKmpJ) is COMPLETED
[2018-06-01 13:15:28] annotate
                        MA_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-1␣
↪(tMHiVR8ueNokhBDnghXYo6) is COMPLETED
                        MA_mock-A36971_batch-s4W2Go4tinn49nkhSuusrE-2␣
↪(AsNpNdvUyhNtKmRZqRSPpR) is COMPLETED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-1␣
↪(k7qQiAzxfC2dnZwsGH7BzD) is COMPLETED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-2␣
↪(dqAuhhcVKejDvHGBXn22xb) is COMPLETED
                        MA_mock-A47933_batch-s4W2Go4tinn49nkhSuusrE-3␣
↪(eB69Ghed2xAdp2VRdaCJBf) is COMPLETED
[2018-06-01 13:15:28] pairing
                        MP_batch-s4W2Go4tinn49nkhSuusrE (6LfEgBtBsmGhQpLQp9rXmi) is␣
↪COMPLETED
```

(continues on next page)

(continued from previous page)

```
[2018-06-01 13:15:28] summary
                        MS_batch-s4W2Go4tinn49nkhSuusrE (HDJhXgKjRmseahcQ7mgNoD) is␣
↪COMPLETED
                      rewriting: output_dir/build.cfg
                      run time (hh/mm/ss): 0:00:00
                      run time (s): 0
```

If you see the above, then MAVIS has completed correctly!

4.3 MAVIS (Full) Tutorial

The following tutorial is an introduction to running MAVIS. You will need to download the tutorial data. Additionally the instructions pertain to running MAVIS on a *SLURM* cluster. This tutorial will require more resources than the mini-tutorial above.

4.3.1 Getting the Tutorial Data

The tutorial data can be downloaded from the link below. Note that it may take a while as the download is ~29GB

```
wget http://www.bcgsc.ca/downloads/mavis/tutorial_data.tar.gz
tar -xvzf tutorial_data.tar.gz
```

The expected contents are

Path	Description
README	Information regarding the other files in the directory
L1522785992_expected_events.tab	The events that we expect to find, either experimentally validated or 'spiked' in
L1522785992_normal.sorted.bam	Paired normal library BAM file
L1522785992_normal.sorted.bam.bai	BAM index
L1522785992_trans.sorted.bam	Tumour transcriptome BAM file
L1522785992_trans.sorted.bam.bai	BAM index file
L1522785992_tumour.sorted.bam	Tumour genome BAM file
L1522785992_tumour.sorted.bam.bai	BAM index file
breakdancer-1.4.5/	Contains the *BreakDancer* output which was run on the tumour genome BAM file
breakseq-2.2/	Contains the *BreakSeq* output which was run on the tumour genome BAM file
chimerascan-0.4.5/	Contains the *ChimeraScan* output which was run on the tumour transcriptome BAM file
defuse-0.6.2/	Contains the *deFuse* output which was run on the tumour transcriptome BAM file
manta-1.0.0/	Contains the *Manta* output which was run on the tumour genome and paired normal genome BAM files

4.3.2 Downloading the Reference Inputs

Run the following to download the hg19 reference files and set up the environment variables for configuring MAVIS

```
wget https://raw.githubusercontent.com/bcgsc/mavis/master/tools/get_hg19_reference_
↪files.sh
bash get_hg19_reference_files.sh
source reference_inputs/hg19_env.sh
```

4.3.3 Generating the Config File

The *config* command does most of the work of creating the config for you but there are a few things you need to tell it

1. **Where your bams are and what library they belong to**

```
--library L1522785992-normal genome normal False tutorial_data/L1522785992_normal.
↪sorted.bam
--library L1522785992-tumour genome diseased False tutorial_data/L1522785992_tumour.
↪sorted.bam
--library L1522785992-trans transcriptome diseased True tutorial_data/L1522785992_
↪trans.sorted.bam
```

2. **Where your SV caller output files (events) are**

If they are raw tool output as in the current example you will need to use the convert argument to tell MAVIS the file type

```
--convert breakdancer tutorial_data/breakdancer-1.4.5/*txt breakdancer
--convert breakseq tutorial_data/breakseq-2.2/breakseq.vcf.gz breakseq
--convert chimerascan tutorial_data/chimerascan-0.4.5/chimeras.bedpe chimerascan
--convert defuse tutorial_data/defuse-0.6.2/results.classify.tsv defuse
--convert manta tutorial_data/manta-1.0.0/diploidSV.vcf.gz tutorial_data/manta-1.0.0/
↪somaticSV.vcf manta
```

Note: For older versions of MAVIS the convert command may require the path to the file(s) be quoted and the strandedness be specified (default is False)

---

3. **Which events you should validate in which libraries**

For this example, because we want to determine which events are germline/somatic we are going to pass all genome calls to both genomes. We can use either full file paths (if the input is already in the standard format) or the alias from a conversion (the first argument given to the convert option)

```
--assign L1522785992-trans chimerascan defuse
--assign L1522785992-tumour breakdancer breakseq manta
--assign L1522785992-normal breakdancer breakseq manta
```

Putting this altogether with a name to call the config, we have the command to generate the pipeline config. You should expect this step with these inputs to take about ~5GB memory.

```
mavis config \
    --library L1522785992-normal genome normal False tutorial_data/L1522785992_normal.
↪sorted.bam \
    --library L1522785992-tumour genome diseased False tutorial_data/L1522785992_
↪tumour.sorted.bam \
```

(continues on next page)

(continued from previous page)

```
    --library L1522785992-trans transcriptome diseased True tutorial_data/L1522785992_
↪trans.sorted.bam \
    --convert breakdancer tutorial_data/breakdancer-1.4.5/*txt breakdancer \
    --convert breakseq tutorial_data/breakseq-2.2/breakseq.vcf.gz breakseq \
    --convert chimerascan tutorial_data/chimerascan-0.4.5/chimeras.bedpe chimerascan \
    --convert defuse tutorial_data/defuse-0.6.2/results.classify.tsv defuse \
    --convert manta tutorial_data/manta-1.0.0/diploidSV.vcf.gz tutorial_data/manta-1.
↪0.0/somaticSV.vcf manta \
    --assign L1522785992-trans chimerascan defuse \
    --assign L1522785992-tumour breakdancer breakseq manta  \
    --assign L1522785992-normal breakdancer breakseq manta \
    -w mavis.cfg
```

4.3.4 Setting Up the Pipeline

The next step is running the setup stage. This will perform conversion, clustering, and creating the submission scripts for the other stages.

```
mavis setup mavis.cfg -o output_dir/
```

At this stage you should have something that looks like this. For simplicity not all files/directories have been shown.

```
output_dir/
|-- build.cfg
|-- converted_inputs
|   |-- breakdancer.tab
|   |-- breakseq.tab
|   |-- chimerascan.tab
|   |-- defuse.tab
|   `-- manta.tab
|-- L1522785992-normal_normal_genome
|   |-- annotate
|   |   |-- batch-aUmErftiY7eEWvENfSeJwc-1/
|   |   `-- submit.sh
|   |-- cluster
|   |   |-- batch-aUmErftiY7eEWvENfSeJwc-1.tab
|   |   |-- cluster_assignment.tab
|   |   |-- clusters.bed
|   |   |-- filtered_pairs.tab
|   |   `-- MAVIS-batch-aUmErftiY7eEWvENfSeJwc.COMPLETE
|   `-- validate
|       |-- batch-aUmErftiY7eEWvENfSeJwc-1/
|       `-- submit.sh
|-- pairing
|   `-- submit.sh
`-- summary
    `-- submit.sh
```

4.3.5 Submitting Jobs to the Cluster

The last step is simple, ssh to your head node of your *SLURM* cluster (or run locally if you have configured *remote_head_ssh*) and run the schedule step. This will submit the jobs and create the dependency chain

```
ssh head_node
mavis schedule -o output_dir --submit
```

The schedule step also acts as a built-in checker and can be run to check for errors or if the pipeline has completed.

```
mavis schedule -o output_dir
```

This should give you output something like below (times may vary) after your run completed correctly.

```
                      MAVIS: 2.0.0
                      hostname: gphost08.bcgsc.ca
[2018-06-02 19:47:56] arguments
                        command = 'schedule'
                        log = None
                        log_level = 'INFO'
                        output = 'output_dir/'
                        resubmit = False
                        submit = False
[2018-06-02 19:48:01] validate
                        MV_L1522785992-normal_batch-aUmErftiY7eEWvENfSeJwc (1701000)␣
↪is COMPLETED
                          200 tasks are COMPLETED
                          run time: 609
                        MV_L1522785992-tumour_batch-aUmErftiY7eEWvENfSeJwc (1701001)␣
↪is COMPLETED
                          200 tasks are COMPLETED
                          run time: 669
                        MV_L1522785992-trans_batch-aUmErftiY7eEWvENfSeJwc (1701002)␣
↪is COMPLETED
                          23 tasks are COMPLETED
                          run time: 1307
[2018-06-02 19:48:02] annotate
                        MA_L1522785992-normal_batch-aUmErftiY7eEWvENfSeJwc (1701003)␣
↪is COMPLETED
                          200 tasks are COMPLETED
                          run time: 622
                        MA_L1522785992-tumour_batch-aUmErftiY7eEWvENfSeJwc (1701004)␣
↪is COMPLETED
                          200 tasks are COMPLETED
                          run time: 573
                        MA_L1522785992-trans_batch-aUmErftiY7eEWvENfSeJwc (1701005)␣
↪is COMPLETED
                          23 tasks are COMPLETED
                          run time: 537
[2018-06-02 19:48:07] pairing
                        MP_batch-aUmErftiY7eEWvENfSeJwc (1701006) is COMPLETED
                          run time: 466
[2018-06-02 19:48:07] summary
                        MS_batch-aUmErftiY7eEWvENfSeJwc (1701007) is COMPLETED
                          run time: 465
                      parallel run time: 3545
                      rewriting: output_dir/build.cfg
```

(continues on next page)

(continued from previous page)

```
                    run time (hh/mm/ss): 0:00:11
                    run time (s): 11
```

The parallel run time reported corresponds to the sum of the slowest job for each stage and does not include any queue time etc.

4.3.6 Analyzing the Output

The best place to start with looking at the MAVIS output is the summary folder which contains the final results. For column name definitions see the *glossary*.

```
output_dir/summary/mavis_summary_all_L1522785992-normal_L1522785992-trans_L1522785992-
↪tumour.tab
```

CHAPTER

FIVE

CONFIGURATION AND SETTINGS

5.1 Pipeline Configuration File

The pipeline can be run in steps or it can be configured using a configuration file and setup in a single step. Scripts will be generated to run all steps following clustering. The configuration file can be built from scratch or a template can be output as shown below

```
>>> mavis config --write template.cfg
```

This will create a template config file called template.cfg which can then be edited by the user. However this will be a simple config with no library information. To generate a configuration file with the library information as well as estimates for the fragment size parameters more inputs are required (see generating the config file for more information).

5.2 Environment Variables

Most of the default settings can be changed by using environment variables. The value given by the environment variables will be used as the new default. Config or command-line parameters will still override these settings.

All environment variables are prefixed with MAVIS and an underscore. Otherwise the variable name is the same as that used for the command line parameter or config setting (uppercased). For example to change the default minimum mapping quality used during the validate stage

```
>>> export MAVIS_MIN_MAPPING_QUALITY=10
```

5.3 Adjusting the Resource Requirements

5.3.1 Choosing the Number of Validation/Annotation Jobs

MAVIS chooses the number of jobs to split validate/annotate stages into based on two settings: *max_files* and *min_clusters_per_file*.

For example, in the following situation say you have: 1000 clusters, `max_files=10`, and `min_clusters_per_file=10`. Then MAVIS will set up 10 validation jobs each with 100 events.

However, if `min_clusters_per_file=500`, then MAVIS would only set up 2 jobs each with 500 events. This is because *min_clusters_per_file* takes precedence over *max_files*.

Splitting into more jobs will lower the resource requirements per job (see *resource requirements*). The memory and time requirements for validation are linear with respect to the number of events to be validated.

5.3.2 Uninformative Filter

For example, if the user is only interested in events in genes, then the *uninformative_filter* can be used. This will drop all events that are not within a certain distance (*max_proximity*) to any annotation in the annotations reference file. These events will be dropped prior to the validation stage which results in significant speed up.

This can be set using the environment variable

```
export MAVIS_UNINFORMATIVE_FILTER=True
```

or in the pipeline config file

```
[cluster]
uninformative_filter = True
```

or as a command line argument to the cluster stage

```
mavis cluster --uninformative_filter True ....
```

CHAPTER

SIX

RESOURCE REQUIREMENTS

MAVIS has been tested on both unix and linux systems. For the standard pipeline, the validation stage is the most computationally expensive. The memory and cpu requirements will vary with two main factors: the number of structural variants you are validating per job, and the size of the bam file you are validating against.

There are a number of settings that can be adjusted to reduce memory and cpu requirements depending on what the user is trying to analyze. See *configuration and settings* for more details.

6.1 Validation Resources

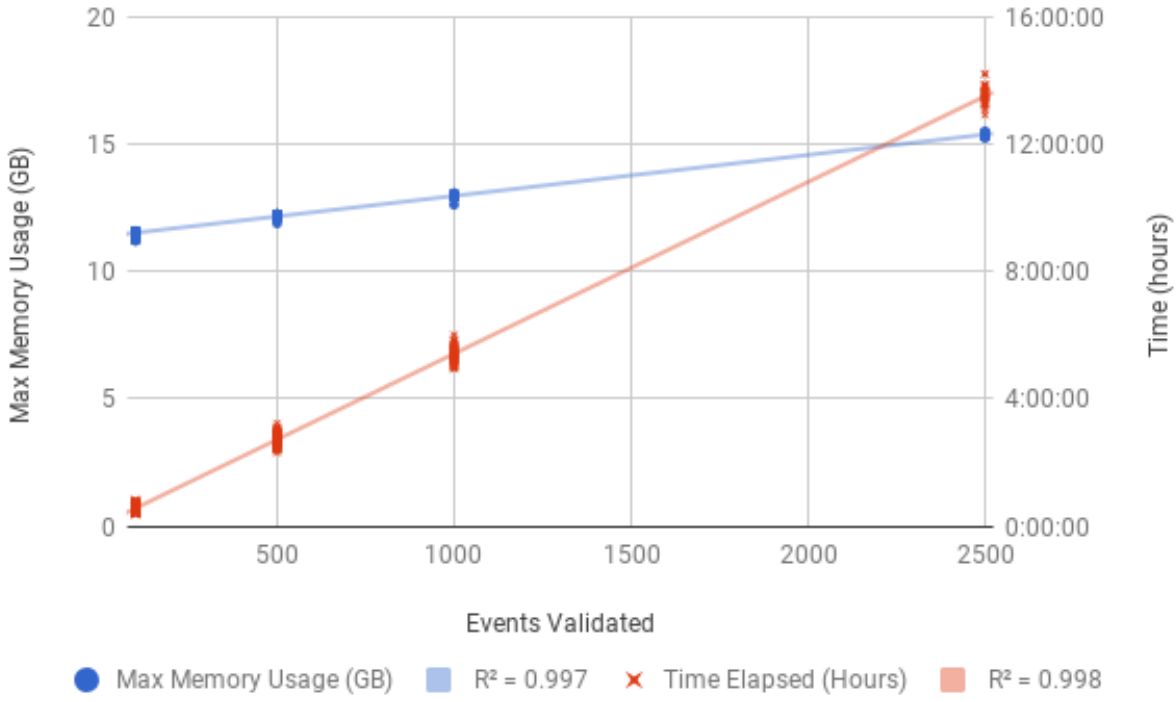

Figure 6.1.: Resource Requirements (MAVIS 1.8.0) for each validation job of the COLO829 tumour genome. The BAM file for the tumour genome is 127GB. Validation jobs were tested splitting into: 100, 500, 1000, and 2500 structural variant validations per job. The effect of number of events validated on both memory and time is plotted above.

6.2 Annotation Resources

Similar trends were observed for the annotation step (see below) with regards to time elapsed. However the memory requirements remained more constant which is expected since, unlike validation, anntotation does not read more data in for more events.

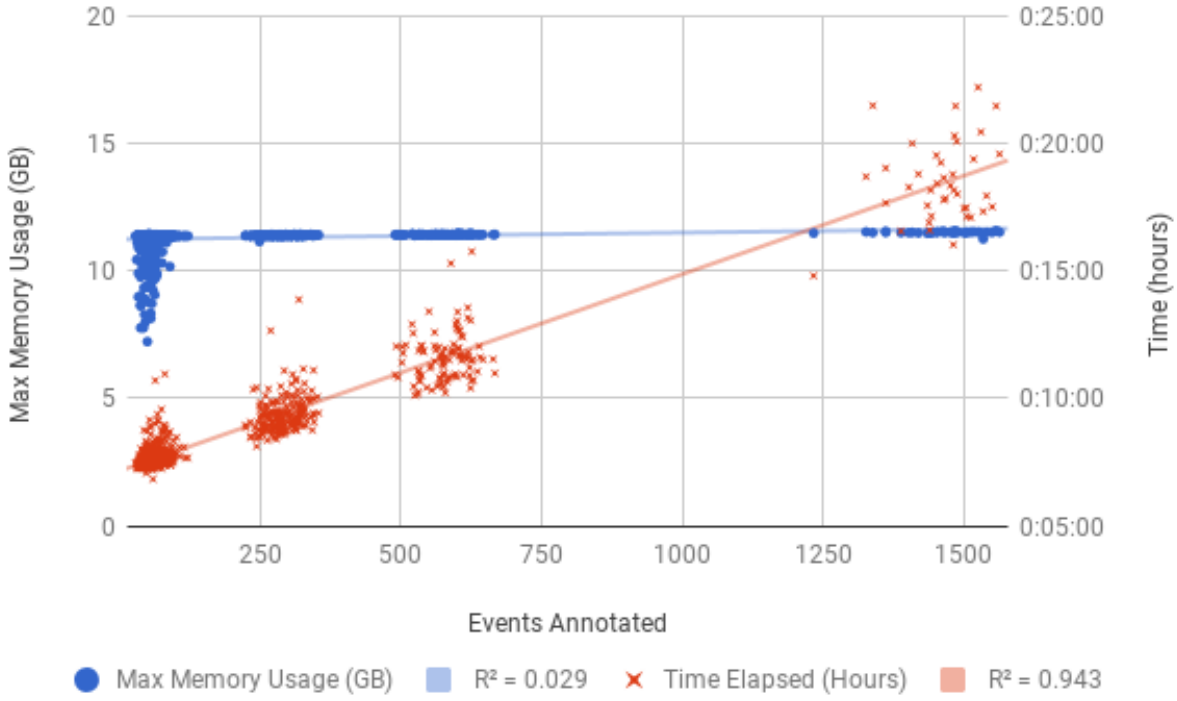

Figure 6.2.: Resource Requirements (MAVIS 1.8.0) for each annotation job of the COLO829 tumour genome. The events which passed validation (see above) represent the number of events input to the annotation step.

CHAPTER

SEVEN

THEORY AND MODELS

7.1 Introduction

In MAVIS structural variants (SVs) are defined by a pair of breakpoints

And a breakpoint is defined by

1. chromosome
2. base-pair range (start, end). This has a length of 1 for exact calls and more for uncertain/non-specific calls
3. orientation. This is Left or Right with respect to the positive/forward strand. This defines which portion of the genome is 'retained'
4. strand. (only applicable to stranded transcriptome libraries)

So then a breakpoint pair is any two intervals on the reference genome which are adjacent in the mutant genome

7.2 Evidence

There are many ways that single reads or paired-end reads can act as support for an SV call.

Figure 7.1.: In the figure above the red rectangle represents a deletion structural variant. The arrows are types of single or paired-end reads supporting the event: *flanking read pair*s (F), *split read*s (S), *half-mapped read*s (H), and *spanning read*s (N).

7.2.1 Types of Flanking evidence

One of the most confusing parts about working with contig and paired-end reads is relating them to the breakpoint so that you can determine which types will support an event. The flanking read types we outline here are similarly described by IGV. We have used similar coloring for the read pairs in the following diagrams to facilitate ease of use for those already familiar with viewing bam files in IGV.

Note: The major assumptions here are that the 'normal' read-pair is a read pair which has one read on the positive/forward strand and its partner on the negative/reverse strand. It is assumed that partners share a read name, as is the case for Illumina reads.

Deletion

For a deletion, we expect the flanking reads to be in the normal orientation but that the fragment size should be abnormal (for large deletions).

Figure 7.2.: Flanking read pair evidence for a deletion event. the read pairs will have a larger than expected fragment size when mapped to the reference genome because in the mutant genome they are closer together, owing to the deletion event. (B1) The first breakpoint which has a left orientation. (B2) The second breakpoint which has a right orientation. Both breakpoints would be on the positive strand (assuming that the input is stranded) which means that the first read in the pair would be on the positive strand and the second read in the pair would be on the negative/reverse strand.

Insertion

Figure 7.3.: Flanking read pair evidence for an insertion event. The read pairs will have a smaller than expected fragment size when mapped to the reference genome because in the mutant genome they are father apart, owing to the insertion event. (B1) The first breakpoint which has a left orientation. (B2) The second breakpoint which has a right orientation. Both breakpoints would be on the positive strand (assuming that the input is stranded) which means that the first read in the pair would be on the positive strand and the second read in the pair would be on the negative/reverse strand.

Duplication

Figure 7.4.: Flanking read pair evidence for a tandem duplication event. The read pairs will have an abnormal orientation but still the same strands as the normal read pair. (B1) The first breakpoint will be on the positive strand and have a right orientation. (B2) The second breakpoint will be on the positive strand and have a left orientation.

Inversion

Figure 7.5.: Flanking read pair evidence for an inversion. Both breakpoints have left orientation.

Figure 7.6.: Flanking read pair evidence for an inversion. Both breakpoints have right orientation.

Translocation

Figure 7.7.: Flanking read pair evidence for a translocation. (B1) the first breakpoint with a left orientation. (B2) the second breakpoint with a right orientation.

Inverted Translocation

7.2.2 Compatible Flanking Pairs

For insertion and duplication events compatible flanking pairs are collected. This means that flanking pairs that support a duplication may be used as compatible flanking evidence for an insertion (in the same region) and similarly flanking pairs which support an insertion may be compatible flanking evidence for a duplication

7.2.3 Calculating the Evidence Window

We make some base assumptions with regards to paired-end read data:

Note: the distribution of fragment sizes approximately follows a normal distribution

Note: the most common fragment size is the unmutated 'normal' fragment

Figure 7.8.: Flanking read pair evidence for a translocation. (B1) the first breakpoint with a right orientation. (B2) the second breakpoint with a left orientation.

Figure 7.9.: Flanking read pair evidence for an inverted translocation. Both breakpoints have left orientation.

With the above assumptions we take the median fragment size to be the expected normal.

Given that we expect mutations and therefore abnormal fragment sizes, we use a modified method to calculate the **median standard deviation** (see code below). We calculate the squared distance away from the median for each fragment and then take a fraction of this to be 'normal' variation. So the most abnormal portion is ignored, assuming it is supposed to be abnormal. This results in a calculation as follows.

```
from statistics import median
import math

fragments = [abs(read.template_length) for read in reads]  # the fragment sizes of␣
↪the reads
f = 0.95 # fraction
m = median(fragments) # get the median fragment size value
X = [math.pow(i - m, 2) for i in fragments]  # take the square error for each point
end = int(round(len(X) * f))
X = sorted(X)[0:end]
stdev = math.sqrt(sum(X) / len(X))
```

This gives us an idea of when to judge an fragment size as abnormal and where we expect our normal read pairs fragment sizes to fall.

As we can see from the diagram above, removing the outliers reproduces the observed distribution better than using all data points

We use this in two ways

1. to find flanking evidence supporting deletions and insertions
2. to estimate the window size for where we will need to read from the bam when looking for evidence for a given event

The `_generate_window()` function uses the above concepts. The user will define the `median_fragment_size` the `stdev_fragment_size` , and the `stdev_count_abnormal` parameters defined in the `VALIDATION_DEFAULTS` class.

If the library has a transcriptome protocol this becomes a bit more complicated and we must take into account the possible annotations when calculating the evidence window. see `_generate_window()` for more

Figure 7.10.: Flanking read pair evidence for an inverted translocation. Both breakpoints have right orientation.

Figure 7.11.: The event depicted above may be called as either a duplication or an insertion (depending on the input call). If the even were called as a duplication the reads in green would be the flanking supoprt and the reads in blue would be given as compatible flanking support. If the event were called as an insertion the reverse would apply.

Figure 7.12.: Basic Terms used in describing read pairs are shown above: fragment size: the distance between the pair; read length: the length of the read; fragment size: the combined length of both reads and the fragment size

7.2.4 Calling Breakpoints by Flanking Evidence

Breakpoints are called by contig, split-read, or flanking pairs evidence. Contigs and split reads are used to call exact breakpoints, where breakpoints called by flanking reads are generally assigned a probabilistic range.

The metrics used here are similar to those used in calculating the evidence window. We use the max_expected_fragment_size as the outer limit of how large the range can be. This is further refined taking into account the range spanned by the *flanking read pair* evidence and the position of the opposing breakpoint.

7.2.5 Determining Flanking support

7.3 Classifying Events

The following decision tree is used in classifying events based on their breakpoints. Only valid combinations have been shown. see `classify()`

Figure 7.13.: Distribution of fragment sizes (absolute values) of proper read pairs. The black curve representings the fit for a normal distribution using the standard deviation calculated with all data points. The blue curve is the expected distribution using a 0.95 fraction of the data. The thick vertical black line is the median and the thin black lines are standard deviations away from the median.

Figure 7.14.: Calculation of the left-oriented breakpoint by flanking reads. Reads mapped to the breakpoint are shown in grey. The read on the right (black outline, no fill) demonstrates the read length used to narrow the right side bound of the estimated breakpoint interval.

Figure 7.15.: After a breakpoint has been called we can narrow the interval of expected fragment sizes using the size of the event. (Left) The colored portion of the graph represents the range in fragment sizes we expect for a normal/unmutated genome. (Right) For a deletion event we expect the fragment size to be bigger, outside the normal range. Reads that would flank the breakpoint should follow a similar distribution to the normal genome but the median will be shifted by the size of the event. The shaded portion of the graph represents the range in fragment sizes we expect for flanking pairs supporting the deletion event.

7.4 Assembling Contigs

During validation, for each breakpoint pair, we attempt to assemble a contig to represent the sequence across the breakpoints. This is assembled from the supporting reads (*split read*, *half-mapped read*, *flanking read pair*, and *spanning read*) which have already been collected for the given event. The sequence from each read and its reverse complement are assembled into contigs using a DeBruijn graph. For strand specific events, we then attempt to resolve the sequence strand of the contig.

7.5 Annotating Events

We make the following assumptions when determining the annotations for each event

Note: If both breakpoints are in the same gene, they must also be in the same transcript

Note: If the breakpoint intervals overlap we do not annotate encompassed genes

Figure 7.16.: Classification Decision Tree. The above diagram details the decsion logic for classifying events based on the orientation, strand and chromosomes or their respective breakpoints

Note: Encompassed and 'nearest' genes are reported without respect to strand

There are specific questions we want annotation to answer. We collect gene level annotations which describes things like what gene is near the breakpoint (useful in the case of a potential promoter swap); what genes (besides the one selected) also overlap the breakpoint; what genes are encompassed between the breakpoints (for example in a deletion event the genes that would be deleted).

Figure 7.17.: Gene level annotations at each breakpoint. Note: genes which fall between a breakpoint pair, encompassed genes, will not be present for interchromosomal events (translocations)

Next there are the fusion-product level annotations. If the event result in a fusion transcript, the sequence of the fusion transcript is computed. This is translated to a putative amino acid sequence from which protein metrics such as the possible ORFs and domain sequences can be computed.

7.6 Predicting Splicing Patterns

After the events have been called and an annotation has been attached, we often want to predict information about the putative fusion protein, which may be a product. In some cases, when a fusion transcript disrupts a splice-site, it is not clear what the processed fusion transcript may be. MAVIS will calculate all possibilities according to the following model.

Figure 7.18.: The default splicing pattern is a list of pairs of donor and acceptor splice sites

For a given list of non-abrogated splice sites (listed 5' to 3' on the strand of the transcript) donor splice sites are paired with all following as seen below

Figure 7.19.: Multiple abrogated acceptors sites. As one can see above this situation will result in 3 different splicing patterns depending on which donor is paired with the 2nd acceptor site

More complex examples are drawn below. There are five classifications (`SPLICE_TYPE`) for the different splicing patterns:

1. Retained intron (`RETAIN`)
2. Skipped exon (`SKIP`)
3. Multiple retained introns (`MULTI_RETAIN`)
4. Multiple skipped exons (`MULTI_SKIP`)
5. Some combination of retained introns and skipped exons (`COMPLEX`)

Figure 7.20.: Multiple abrogated donor sites. As one can see above this situation will result in 3 different splicing patterns depending on which acceptor is paired with the 2nd donor site

Figure 7.21.: Splicing scenarios

7.7 Pairing Similar Events

After breakpoints have been called and annotated we often need to see if the same event was found in different samples. To do this we will need to compare events between genome and transcriptome libraries. For this, the following model is proposed. To compare events between different protocol (genome vs transcriptome) we use the annotation overlying the genome breakpoint and the splicing model we defined above to predict where we would expect to find the transcriptomic breakpoints. This gives rise to the following basic cases.

Note: In all cases the predicted breakpoint is either the same as the genomic breakpoint, or it is the same as the nearest retained donor/acceptor to the breakpoint.

Figure 7.22.: (A-D) The breakpoint lands in an exon and the five prime portion of the transcript is retained. (A) The original splicing pattern showing the placement of the genomic breakpoint and the retained five prime portion. (B) The first splice site following the breakpoint is a donor and the second donor is used. (C) The first splice site following the breakpoint is a donor and the first donor is used. (D) The first splice site following the breakpoint is an acceptor. (E-H) The breakpoint lands in an exon and the three prime portion of the transcript is retained. (E) The original splicing pattern showing the placement of the genomic breakpoint and the retained three prime portion. (F) The first splice site prior to the breakpoint is an acceptor and the first acceptor is used. (G) The first splice site prior to the breakpoint is an acceptor and the second acceptor is used. (H) The first splice site prior to the breakpoint is a donor

7.7.1 Literature

CHAPTER

EIGHT

ILLUSTRATIONS

8.1 Fusion Diagrams

These are diagrams produced during the annotate step. These represent the putative fusion events of a single breakpoint pair.

Figure 8.1.: Fusion from transcriptome data. Intronic breakpoints here indicate retained intron sequence and a novel exon is predicted.

If the *draw_fusions_only* flag is set to False then all events will produce a diagram, even anti-sense fusions

Figure 8.2.: Disruptive Anti-sense Fusion

8.2 Transcript Overlays

MAVIS supports generating diagrams of all transcripts for a given gene. These can be overlaid with markers and bam_file pileup data. This is particularly useful for visualizing splice site mutations.

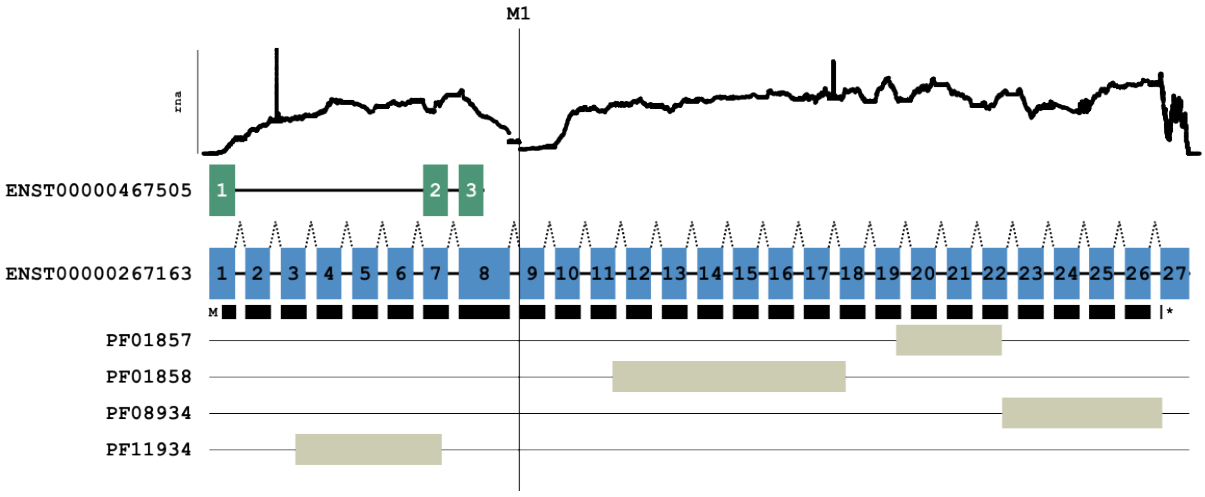

Figure 8.3.: RB1 splice site mutation results in skipping of exon 9

The above diagram was generated using the overlay command

```
mavis overlay RB1 \
    -o /path/to/output/dir \
    --read_depth_plot rna /path/to/bam/file \
    --marker M1 48939029 \
    --annotations /path/to/mavis/annotations/reference/file
```

CHAPTER

NINE

GUIDELINES FOR CONTRIBUTORS

9.1 Getting Started

If you are new to the project a good way to get started is by adding to the documentation, or adding unit tests where there is a lack of code coverage.

9.2 Install (for Development)

Clone the repository and switch to the development branch

```
git clone https://github.com/bcgsc/mavis.git
cd mavis
git checkout develop
```

Set up a python virtual environment. If you are developing in python setting up with a virtual environment can be incredibly helpful as it allows for a clean install to test. Instructions for setting up the environment are below

```
python3 -m venv venv
source venv/bin/activate
```

Install the MAVIS python package. Running the setup in develop mode will ensure that your code changes are run when you run MAVIS from within that virtual environment

```
pip install -e .[dev]
```

Run the tests and compute code coverage

```
pytest tests
```

9.3 Build the Sphinx Documentation

```
pip install .[docs]
sphinx-build docs/source/ html
```

The contents of the user manual can then be viewed by opening the build/html/index.html in any available web browser (i.e. google-chrome, firefox, etc.)

9.4 Deploy to PyPi

Install deployment dependencies

```
pip install .[deploy]
```

Build the distribution files

```
python setup.py install sdist bdist_wheel
```

Use twine to upload

```
twine upload -r pypi dist/*
```

9.4.1 Reporting a Bug

Please make sure to search through the issues before reporting a bug to ensure there isn't already an open issue.

9.4.2 Coding Conventions

Formatting/Style

- In general, follow pep8 style guides (except maximum line width)
- docstrings should follow sphinx google code style
- any column name which may appear in any of the intermediate or final output files must be defined in `mavis.constants.COLUMNS`

Types in docstrings

if you want to be more explicit with nested types, the following conventions are used throughout the code

- dictionary: `d = {<key>: <value>}` becomes `dict of <value> by <key>`
- list: `l = [1, 2, 3]` becomes `list of int`
- mixed: `d = {'a': [1, 2, 3], 'b': [4, 5, 6]}` becomes `dict of list of int by str`
- tuples: `('a', 1)` becomes `tuple of str and int`

Tests

- all new code must have unit tests in the tests subdirectory
- in general for assertEqual statements, the expected value is given first

Tests can be run as follows

```
pytest tests
```

To run the tests with tox (multiple python installs tested). Note that you will need to have multiple python installs on your path

```
tox
```

9.5 Major Assumptions

Some assumptions have been made when developing this project. The major ones have been listed here to facilitate debugging/development if any of these are violated in the future.

- The input bam reads have stored the sequence wrt to the positive/forward strand and have not stored the reverse complement.
- The distribution of the fragment sizes in the bam file approximately follows a normal distribution.

9.6 Current Limitations

- Assembling contigs will always fail for repeat sequences as we do not resolve this. Unlike traditional assemblies we cannot assume even input coverage as we are taking a select portion of the reads to assemble.
- Currently no attempt is made to group/pair single events into complex events.
- Transcriptome validation uses a collapsed model of all overlapping transcripts and is not isoform specific. Allowing for isoform specific validation would be computationally expensive but may be considered as an optional setting for future releases.

9.7 Computing Code coverage

Since MAVIS uses multiple processes, it adds complexity to computing the code coverage. Running coverage normally will undereport. To ensure that the coverage module captures the information from the subprocesses we need to do the following

In our development python virtual environment put a coverage.pth file (ex. `venv/lib/python3.6/site-packages/coverage.pth`) containing the following

```
import coverage; coverage.process_startup()
```

Additionally you will need to set the environment variable

```
export COVERAGE_PROCESS_START=/path/to/mavis/repo/mavis/.coveragerc
```

CHAPTER

TEN

PACKAGE DOCUMENTATION

holds submodules related to structural variants

10.1 align module

10.2 annotate subpackage

10.2.1 Sub-package Documentation

Types of Output Files

expected name/suffix	file type/format	content
`annotations.tab`	text/tabbed	annotated events
`annotations.fusion-cdna.fa`	*fasta*	putative fusion unspliced cDNA sequences
`drawings/*.svg`	*SVG*	diagrams
`drawings/*.legend.json`	*JSON*	diagram legend/metadata

Algorithm Overview

see *theory - annotating events*

- read in breakpoint pairs
- generate strand-specific annotations (one annotation per strand, multiple if multiple genes/transcripts in the region)
- try building fusion transcripts for bp-specific calls
- generate *SVG* diagrams

Levels of Annotations

Overview of Class Relationships

Figure 10.1.: The Annotation sub-package has objects for genetic annotations and related calculations. The basic layout of the package is shown above. IS-A relationships are given by the blue arrows. HAS-A relationships are shown in black. And reference_object/parent type relationships are shown in red. `Gene` is a gene. Start and end are genomic positions wrt to the template/chr. `PreTranscript` is the unspliced transcript. Start and end are genomic positions wrt to the template/chr. `Transcript`: is the spliced transcript. Start and end coordinates are 1 to the length of the spliced product in base pairs. `Translation`: is the translation of the spliced transcript. Start and end are cdna positions wrt the 5' end of the spliced transcript. The start and end here describe the start and end of the coding sequence

---

modules

10.2.2 base module

10.2.3 constants module

10.2.4 file_io module

10.2.5 fusion module

10.2.6 genomic module

10.2.7 main module

10.2.8 protein module

10.2.9 splicing module

10.2.10 variant module

10.3 assemble module

10.4 bam subpackage

—

10.4.1 cache module

10.4.2 cigar module

10.4.3 read module

10.4.4 stats module

10.5 blat module

10.6 breakpoint module

10.7 cluster subpackage

10.7.1 Sub-package Documentation

The cluster sub-package is responsible for merging variants coming from different inputs (i.e. different tools).

Types of Output Files

expected name/suffix	file type/format	content
`cluster_assignment.tab`	text/tabbed	
`uninformative_clusters.txt`	text	list of cluster ids that were dropped by annotation proximity filter
`clusters.bed`	*bed*	cluster positions
`cluster-*.tab`	text/tabbed	computed clusters

Algorithm Overview

- Collapse any duplicate breakpoint pairs
- Split breakpoint pairs by type
- Cluster breakpoint pairs by distance (within a type)
  - Create a graph representation of the distances between pairs
  - Find cliques up to a given input size (cluster_clique_size)
  - Hierarchically cluster the cliques (allows redundant participation)
  - For each input node/pair pick the best cluster(s)
- Output the clusters and the mapping to the input pairs

---

modules

10.7.2 cluster module

10.7.3 constants module

10.7.4 main module

10.8 config module

10.9 constants module

10.10 error module

10.11 illustrate subpackage

10.11.1 constants module

10.11.2 diagram module

10.11.3 elements module

10.11.4 scatter module

10.11.5 util module

10.12 interval module

10.13 pairing subpackage

10.13.1 Sub-package Documentation

This is the package responsible for pairing/grouping calls between libraries. In many cases this will be where somatic vs germline is determined or genomic only vs expressed.

Output Files

expected name/suffix	file type/format	content
`mavis_paired_*.tab`	text/tabbed	call information and pairing information using product id

Algorithm Overview

- pairwise comparison of breakpoint pairs between libraries
  - fail if orientations do not match
  - fail if template/chromosomes do not match
  - if the protocols are mixed
    - pass if the fusion products match at the sequence level
    - pass if the breakpoint predicted from the genome matches the transcriptome breakpoint
  - if the protocols are the same
    - pass if the breakpoints are co-located
- filter matches based on annotations
  - if both breakpoints have the same gene annotation, they must also both have the same transcript annotation

---

modules

10.13.2 constants module

10.13.3 main module

10.13.4 pairing module

10.14 validate subpackage

10.14.1 Sub-package Documentation

The validation sub-package is responsible for pulling supporting reads from the bam file and re-calling events based on the evidence in a standard notation.

Types of Output Files

A variety of intermediate output files are given for the user. These can be used to "drill down" further into events and also for developers debugging when adding new features, etc.

expected name/suffix	file type/format	content
`*.raw_evidence.bam`	*bam*	raw evidence
`*.contigs.bam`	*bam*	aligned contigs
`*.evidence.bed`	*bed*	evidence collection window regions
`*.validation-passed.bed`	*bed*	validated event positions
`*.validation-failed.tab`	text/tabbed	failed events
`*.validation-passed.tab`	text/tabbed	validated events
`*.contigs.fa`	*fasta*	assembled contigs
`*.contigs.blat_out.pslx`	*pslx*	results from blatting contigs
`*.igv.batch`	*IGV batch file*	igv batch file

Algorithm Overview

- (For each breakpoint pair)
  - *Calculate the window/region* to read from the bam and collect evidence
  - Store evidence (*flanking read pair*, *half-mapped read*, *spanning read*, *split read*, compatible flanking pairs) which match the expected event type and position
  - Assemble a contig from the collected reads. see *theory - assembling contigs*
- Generate a *fasta* file containing all the contig sequences
- Align contigs to the reference genome (currently *blat* is used to perform this step)
- Make the final event calls. Each level of calls consumes all supporting reads so they are not re-used in subsequent levels of calls.
- (For each breakpoint pair)
  - call by contig
  - call by *spanning read*
  - call by *split read*
  - call by *flanking read pair*. see *theory - calling breakpoints by flanking evidence*
- Output new calls, evidence, contigs, etc

---

modules

10.14.2 base module

10.14.3 call module

10.14.4 constants module

10.14.5 evidence module

10.14.6 main module

10.15 schedule subpackage

---

modules

10.15.1 constants module

10.15.2 job module

10.15.3 local module

10.15.4 pipeline module

10.15.5 scheduler module

10.16 summary subpackage

10.16.1 Sub-package Documentation

This is the package responsible for summarizing the calls between libraries. In many cases this will be where somatic vs germline is determined or genomic only vs expressed.

Output Files

expected name/suffix	file type/format	content
`mavis_summary_*.tab`	text/tabbed	?

Algorithm Overview

TODO

---

modules

10.16.2 constants module

10.16.3 main module

10.16.4 summary module

10.17 tools module

10.18 util module

CHAPTER

ELEVEN

GLOSSARY

11.1 General Terms

2bit File format specification. See https://genome.ucsc.edu/FAQ/FAQformat#format7.

BAM File format specification. See https://genome.ucsc.edu/FAQ/FAQformat#format5.1.

bed File format specification. See https://genome.ucsc.edu/FAQ/FAQformat#format1.

blat Alignment tool. see https://genome.ucsc.edu/FAQ/FAQblat.html#blat3 for instructions on download and install.

BreakDancer BreakDancer is an SV caller. Source for BreakDancer can be found here [Chen-2009]

breakpoint A breakpoint is a genomic position (interval) on some reference/template/chromosome which has a strand and orientation. The orientation describes the portion of the reference that is retained.

breakpoint pair Basic definition of a *structural variant*. Does not automatically imply a classification/type.

BreakSeq BreakSeq is an SV caller. Source for BreakSeq can be found here [Abyzov-2015]

BWA BWA is an alignement tool. See https://github.com/lh3/bwa for install instructions.

Chimerascan Chimerascan is an SV caller. Source for Chimerascan can be found here [Iyer-2011]

CNVnator CNVnator is an SV caller. Source for CNVnator can be found here [Abyzov-2011]

DeFuse DeFuse is an SV caller. Source for DeFuse can be found here [McPherson-2011]

DELLY DELLY is an SV caller. Source for DELLY can be found here [Rausch-2012]

event Used interchangeably with *structural variant*.

event type Classification for a structural variant. see *event_type*.

fasta File format specification. See https://genome.ucsc.edu/FAQ/FAQformat#format18.

flanking read pair A pair of reads where one read maps to one side of a set of breakpoints and its mate maps to the other.

half-mapped read A read whose mate is unaligned. Generally this refers to reads in the evidence stage that are mapped next to a breakpoint.

HGVS Community based standard of reccommendations for variant notation. See http://varnomen.hgvs.org/

IGV Integrative Genomics Viewer is a visualization tool. see http://software.broadinstitute.org/software/igv.

IGV batch file This is a file format type defined by *IGV* see running IGV with a batch file.

JSON JSON (JavaScript Object Notation) is a data file format. see https://www.w3schools.com/js/js_json_intro.asp.

Manta Manta is an SV caller. Source for Manta can be found here [Chen-2016]

Pindel Pindel is an SV caller. Source for Pindel can be found here [Ye-2009]

psl File format specification. See https://genome.ucsc.edu/FAQ/FAQformat#format2.

pslx Extended format of a *psl*.

SGE Sun Grid Engine (SGE) is a job scheduling system for cluster management see http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html.

SLURM SLURM is a job scheduling system for cluster management see https://slurm.schedmd.com/archive/slurm-17.02.1.

spanning read Applies primarily to small structural variants. Reads which span both breakpoints.

split read A read which aligns next to a breakpoint and is softclipped at one or more sides.

STAR-Fusion STAR-Fusion is an SV caller. Source for STAR-Fusion can be found here [Haas-2017]

Strelka Strelka is an SNV and small indel caller. Only the small indels can be processed, since SNVs are not currently suported. Source for Strelka can be found here [Saunders-2012]

structural variant A genomic alteration that can be described by a pair of breakpoints and an *event type*. The two breakpoints represent regions in the genome that are broken apart and reattached together.

SV Structural Variant

SVG SVG (Scalable vector graph) is an image format. see https://www.w3schools.com/graphics/svg_intro.asp.

TORQUE TORQUE is a job scheduling system for cluster management see http://www.adaptivecomputing.com/products/open-source/torque/.

Trans-ABySS Trans-ABySS is an SV caller. Source for Trans-ABySS can be found here [Robertson-2010]

11.2 Configurable Settings

aligner `SUPPORTED_ALIGNER` - The aligner to use to map the contigs/reads back to the reference e.g blat or bwa. The corresponding environment variable is `MAVIS_ALIGNER` and the default value is `'blat'`. Accepted values include: `'bwa mem'`, `'blat'`

aligner_reference `filepath` - Path to the aligner reference file used for aligning the contig sequences. The corresponding environment variable is `MAVIS_ALIGNER_REFERENCE` and the default value is `None`

annotation_filters `str` - A comma separated list of filters to apply to putative annotations. The corresponding environment variable is `MAVIS_ANNOTATION_FILTERS` and the default value is `'choose_more_annotated,choose_transcripts_by_priority'`

annotation_memory `int` - Default memory limit (mb) for the annotation stage. The corresponding environment variable is `MAVIS_ANNOTATION_MEMORY` and the default value is `12000`

annotations `filepath` - Path to the reference annotations of genes, transcript, exons, domains, etc. The corresponding environment variable is `MAVIS_ANNOTATIONS` and the default value is `[]`

assembly_kmer_size `float_fraction` - The percent of the read length to make kmers for assembly. The corresponding environment variable is `MAVIS_ASSEMBLY_KMER_SIZE` and the default value is `0.74`

assembly_max_paths `int` - The maximum number of paths to resolve. this is used to limit when there is a messy assembly graph to resolve. the assembly will pre-calculate the number of paths (or putative assemblies) and stop if it is greater than the given setting. The corresponding environment variable is `MAVIS_ASSEMBLY_MAX_PATHS` and the default value is `8`

assembly_min_edge_trim_weight `int` - This is used to simplify the debruijn graph before path finding. edges with less than this frequency will be discarded if they are non-cutting, at a fork, or the end of a path. The corresponding environment variable is `MAVIS_ASSEMBLY_MIN_EDGE_TRIM_WEIGHT` and the default value is `3`

assembly_min_exact_match_to_remap `int` - The minimum length of exact matches to initiate remapping a read to a contig. The corresponding environment variable is `MAVIS_ASSEMBLY_MIN_EXACT_MATCH_TO_REMAP` and the default value is `15`

assembly_min_remap_coverage `float_fraction` - Minimum fraction of the contig sequence which the remapped sequences must align over. The corresponding environment variable is `MAVIS_ASSEMBLY_MIN_REMAP_COVERAGE` and the default value is `0.9`

assembly_min_remapped_seq `int` - The minimum input sequences that must remap for an assembled contig to be used. The corresponding environment variable is `MAVIS_ASSEMBLY_MIN_REMAPPED_SEQ` and the default value is `3`

assembly_min_uniq `float_fraction` - Minimum percent uniq required to keep separate assembled contigs. if contigs are more similar then the lower scoring, then shorter, contig is dropped. The corresponding environment variable is `MAVIS_ASSEMBLY_MIN_UNIQ` and the default value is `0.1`

assembly_strand_concordance `float_fraction` - When the number of remapped reads from each strand are compared, the ratio must be above this number to decide on the strand. The corresponding environment variable is `MAVIS_ASSEMBLY_STRAND_CONCORDANCE` and the default value is `0.51`

blat_limit_top_aln `int` - Number of results to return from blat (ranking based on score). The corresponding environment variable is `MAVIS_BLAT_LIMIT_TOP_ALN` and the default value is `10`

blat_min_identity `float_fraction` - The minimum percent identity match required for blat results when aligning contigs. The corresponding environment variable is `MAVIS_BLAT_MIN_IDENTITY` and the default value is `0.9`

breakpoint_color `str` - Breakpoint outline color. The corresponding environment variable is `MAVIS_BREAKPOINT_COLOR` and the default value is `'#000000'`

call_error `int` - Buffer zone for the evidence window. The corresponding environment variable is `MAVIS_CALL_ERROR` and the default value is `10`

clean_aligner_files `bool` - Remove the aligner output files after the validation stage is complete. not required for subsequent steps but can be useful in debugging and deep investigation of events. The corresponding environment variable is `MAVIS_CLEAN_ALIGNER_FILES` and the default value is `False`

cluster_initial_size_limit `int` - The maximum cumulative size of both breakpoints for breakpoint pairs to be used in the initial clustering phase (combining based on overlap). The corresponding environment variable is `MAVIS_CLUSTER_INITIAL_SIZE_LIMIT` and the default value is `25`

cluster_radius `int` - Maximum distance allowed between paired breakpoint pairs. The corresponding environment variable is `MAVIS_CLUSTER_RADIUS` and the default value is `100`

concurrency_limit `int` - The concurrency limit for tasks in any given job array or the number of concurrent processes allowed for a local run. The corresponding environment variable is `MAVIS_CONCURRENCY_LIMIT` and the default value is `None`

contig_aln_max_event_size `int` - Relates to determining breakpoints when pairing contig alignments. for any given read in a putative pair the soft clipping is extended to include any events of greater than this size. the softclipping is added to the side of the alignment as indicated by the breakpoint we are assigning pairs to. The corresponding environment variable is `MAVIS_CONTIG_ALN_MAX_EVENT_SIZE` and the default value is `50`

contig_aln_merge_inner_anchor `int` - The minimum number of consecutive exact match base pairs to not merge events within a contig alignment. The corresponding environment variable is `MAVIS_CONTIG_ALN_MERGE_INNER_ANCHOR` and the default value is `20`

contig_aln_merge_outer_anchor `int` - Minimum consecutively aligned exact matches to anchor an end for merging internal events. The corresponding environment variable is `MAVIS_CONTIG_ALN_MERGE_OUTER_ANCHOR` and the default value is `15`

contig_aln_min_anchor_size int - The minimum number of aligned bases for a contig (m or =) in order to simplify. do not have to be consecutive. The corresponding environment variable is MAVIS_CONTIG_ALN_MIN_ANCHOR_SIZE and the default value is 50

contig_aln_min_extend_overlap int - Minimum number of bases the query coverage interval must be extended by in order to pair alignments as a single split alignment. The corresponding environment variable is MAVIS_CONTIG_ALN_MIN_EXTEND_OVERLAP and the default value is 10

contig_aln_min_query_consumption float_fraction - Minimum fraction of the original query sequence that must be used by the read(s) of the alignment. The corresponding environment variable is MAVIS_CONTIG_ALN_MIN_QUERY_CONSUMPTION and the default value is 0.9

contig_aln_min_score float_fraction - Minimum score for a contig to be used as evidence in a call by contig. The corresponding environment variable is MAVIS_CONTIG_ALN_MIN_SCORE and the default value is 0.9

contig_call_distance int - The maximum distance allowed between breakpoint pairs (called by contig) in order for them to pair. The corresponding environment variable is MAVIS_CONTIG_CALL_DISTANCE and the default value is 10

dgv_annotation filepath - Path to the dgv reference processed to look like the cytoband file. The corresponding environment variable is MAVIS_DGV_ANNOTATION and the default value is []

domain_color str - Domain fill color. The corresponding environment variable is MAVIS_DOMAIN_COLOR and the default value is '#ccccb3'

domain_mismatch_color str - Domain fill color on 0%% match. The corresponding environment variable is MAVIS_DOMAIN_MISMATCH_COLOR and the default value is '#b2182b'

domain_name_regex_filter str - The regular expression used to select domains to be displayed (filtered by name). The corresponding environment variable is MAVIS_DOMAIN_NAME_REGEX_FILTER and the default value is '^PF\\d+$'

domain_scaffold_color str - The color of the domain scaffold. The corresponding environment variable is MAVIS_DOMAIN_SCAFFOLD_COLOR and the default value is '#000000'

draw_fusions_only bool - Flag to indicate if events which do not produce a fusion transcript should produce illustrations. The corresponding environment variable is MAVIS_DRAW_FUSIONS_ONLY and the default value is True

draw_non_synonymous_cdna_only bool - Flag to indicate if events which are synonymous at the cdna level should produce illustrations. The corresponding environment variable is MAVIS_DRAW_NON_SYNONYMOUS_CDNA_ONLY and the default value is True

drawing_width_iter_increase int - The amount (in pixels) by which to increase the drawing width upon failure to fit. The corresponding environment variable is MAVIS_DRAWING_WIDTH_ITER_INCREASE and the default value is 500

exon_min_focus_size int - Minimum size of an exon for it to be granted a label or min exon width. The corresponding environment variable is MAVIS_EXON_MIN_FOCUS_SIZE and the default value is 10

fetch_min_bin_size int - The minimum size of any bin for reading from a bam file. increasing this number will result in smaller bins being merged or less bins being created (depending on the fetch method). The corresponding environment variable is MAVIS_FETCH_MIN_BIN_SIZE and the default value is 50

fetch_reads_bins int - Number of bins to split an evidence window into to ensure more even sampling of high coverage regions. The corresponding environment variable is MAVIS_FETCH_READS_BINS and the default value is 5

fetch_reads_limit int - Maximum number of reads, cap, to loop over for any given evidence window. The corresponding environment variable is MAVIS_FETCH_READS_LIMIT and the default value is 3000

filter_cdna_synon `bool` - Filter all annotations synonymous at the cdna level. The corresponding environment variable is `MAVIS_FILTER_CDNA_SYNON` and the default value is `True`

filter_min_complexity `float_fraction` - Filter event calls based on call sequence complexity. The corresponding environment variable is `MAVIS_FILTER_MIN_COMPLEXITY` and the default value is `0.2`

filter_min_flanking_reads `int` - Minimum number of flanking pairs for a call by flanking pairs. The corresponding environment variable is `MAVIS_FILTER_MIN_FLANKING_READS` and the default value is `10`

filter_min_linking_split_reads `int` - Minimum number of linking split reads for a call by split reads. The corresponding environment variable is `MAVIS_FILTER_MIN_LINKING_SPLIT_READS` and the default value is `1`

filter_min_remapped_reads `int` - Minimum number of remapped reads for a call by contig. The corresponding environment variable is `MAVIS_FILTER_MIN_REMAPPED_READS` and the default value is `5`

filter_min_spanning_reads `int` - Minimum number of spanning reads for a call by spanning reads. The corresponding environment variable is `MAVIS_FILTER_MIN_SPANNING_READS` and the default value is `5`

filter_min_split_reads `int` - Minimum number of split reads for a call by split reads. The corresponding environment variable is `MAVIS_FILTER_MIN_SPLIT_READS` and the default value is `5`

filter_protein_synon `bool` - Filter all annotations synonymous at the protein level. The corresponding environment variable is `MAVIS_FILTER_PROTEIN_SYNON` and the default value is `False`

filter_secondary_alignments `bool` - Filter secondary alignments when gathering read evidence. The corresponding environment variable is `MAVIS_FILTER_SECONDARY_ALIGNMENTS` and the default value is `True`

filter_trans_homopolymers `bool` - Filter all single bp ins/del/dup events that are in a homopolymer region of at least 3 bps and are not paired to a genomic event. The corresponding environment variable is `MAVIS_FILTER_TRANS_HOMOPOLYMERS` and the default value is `True`

flanking_call_distance `int` - The maximum distance allowed between breakpoint pairs (called by flanking pairs) in order for them to pair. The corresponding environment variable is `MAVIS_FLANKING_CALL_DISTANCE` and the default value is `50`

fuzzy_mismatch_number `int` - The number of events/mismatches allowed to be considered a fuzzy match. The corresponding environment variable is `MAVIS_FUZZY_MISMATCH_NUMBER` and the default value is `1`

gene1_color `str` - The color of genes near the first gene. The corresponding environment variable is `MAVIS_GENE1_COLOR` and the default value is `'#657e91'`

gene1_color_selected `str` - The color of the first gene. The corresponding environment variable is `MAVIS_GENE1_COLOR_SELECTED` and the default value is `'#518dc5'`

gene2_color `str` - The color of genes near the second gene. The corresponding environment variable is `MAVIS_GENE2_COLOR` and the default value is `'#325556'`

gene2_color_selected `str` - The color of the second gene. The corresponding environment variable is `MAVIS_GENE2_COLOR_SELECTED` and the default value is `'#4c9677'`

import_env `bool` - Flag to import environment variables. The corresponding environment variable is `MAVIS_IMPORT_ENV` and the default value is `True`

input_call_distance `int` - The maximum distance allowed between breakpoint pairs (called by input tools, not validated) in order for them to pair. The corresponding environment variable is `MAVIS_INPUT_CALL_DISTANCE` and the default value is `20`

label_color `str` - The label color. The corresponding environment variable is `MAVIS_LABEL_COLOR` and the default value is `'#000000'`

limit_to_chr `str` - A list of chromosome names to use. breakpointpairs on other chromosomes will be filteredout. for example '1 2 3 4' would filter out events/breakpoint pairs on any chromosomes

but 1, 2, 3, and 4. The corresponding environment variable is MAVIS_LIMIT_TO_CHR and the default value is ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', 'X', 'Y']

mail_type MAIL_TYPE - When to notify the mail_user (if given). The corresponding environment variable is MAVIS_MAIL_TYPE and the default value is 'NONE'. Accepted values include: 'BEGIN', 'END', 'FAIL', 'ALL', 'NONE'

mail_user str - User(s) to send notifications to. The corresponding environment variable is MAVIS_MAIL_USER and the default value is ''

mask_fill str - Color of mask (for deleted region etc.). The corresponding environment variable is MAVIS_MASK_FILL and the default value is '#ffffff'

mask_opacity float_fraction - Opacity of the mask layer. The corresponding environment variable is MAVIS_MASK_OPACITY and the default value is 0.7

masking filepath - File containing regions for which input events overlapping them are dropped prior to validation. The corresponding environment variable is MAVIS_MASKING and the default value is []

max_drawing_retries int - The maximum number of retries for attempting a drawing. each iteration the width is extended. if it is still insufficient after this number a gene-level only drawing will be output. The corresponding environment variable is MAVIS_MAX_DRAWING_RETRIES and the default value is 5

max_files int - The maximum number of files to output from clustering/splitting. The corresponding environment variable is MAVIS_MAX_FILES and the default value is 200

max_orf_cap int - The maximum number of orfs to return (best putative orfs will be retained). The corresponding environment variable is MAVIS_MAX_ORF_CAP and the default value is 3

max_proximity int - The maximum distance away from an annotation before the region in considered to be uninformative. The corresponding environment variable is MAVIS_MAX_PROXIMITY and the default value is 5000

max_sc_preceeding_anchor int - When remapping a softclipped read this determines the amount of softclipping allowed on the side opposite of where we expect it. for example for a softclipped read on a breakpoint with a left orientation this limits the amount of softclipping that is allowed on the right. if this is set to none then there is no limit on softclipping. The corresponding environment variable is MAVIS_MAX_SC_PRECEEDING_ANCHOR and the default value is 6

memory_limit int - The maximum number of megabytes (mb) any given job is allowed. The corresponding environment variable is MAVIS_MEMORY_LIMIT and the default value is 16000

min_anchor_exact int - Applies to re-aligning softclipped reads to the opposing breakpoint. the minimum number of consecutive exact matches to anchor a read to initiate targeted realignment. The corresponding environment variable is MAVIS_MIN_ANCHOR_EXACT and the default value is 6

min_anchor_fuzzy int - Applies to re-aligning softclipped reads to the opposing breakpoint. the minimum length of a fuzzy match to anchor a read to initiate targeted realignment. The corresponding environment variable is MAVIS_MIN_ANCHOR_FUZZY and the default value is 10

min_anchor_match float_fraction - Minimum percent match for a read to be kept as evidence. The corresponding environment variable is MAVIS_MIN_ANCHOR_MATCH and the default value is 0.9

min_call_complexity float_fraction - The minimum complexity score for a call sequence. is an average for non-contig calls. filters low complexity contigs before alignment. see contig_complexity. The corresponding environment variable is MAVIS_MIN_CALL_COMPLEXITY and the default value is 0.1

min_clusters_per_file int - The minimum number of breakpoint pairs to output to a file. The corresponding environment variable is MAVIS_MIN_CLUSTERS_PER_FILE and the default value is 50

min_domain_mapping_match `float_fraction` - A number between 0 and 1 representing the minimum percent match a domain must map to the fusion transcript to be displayed. The corresponding environment variable is `MAVIS_MIN_DOMAIN_MAPPING_MATCH` and the default value is `0.9`

min_double_aligned_to_estimate_insertion_size `int` - The minimum number of reads which map soft-clipped to both breakpoints to assume the size of the untemplated sequence between the breakpoints is at most the read length - 2 * min_softclipping. The corresponding environment variable is `MAVIS_MIN_DOUBLE_ALIGNED_TO_ESTIMATE_INSERTION_SIZE` and the default value is `2`

min_flanking_pairs_resolution `int` - The minimum number of flanking reads required to call a breakpoint by flanking evidence. The corresponding environment variable is `MAVIS_MIN_FLANKING_PAIRS_RESOLUTION` and the default value is `10`

min_linking_split_reads `int` - The minimum number of split reads which aligned to both breakpoints. The corresponding environment variable is `MAVIS_MIN_LINKING_SPLIT_READS` and the default value is `2`

min_mapping_quality `int` - The minimum mapping quality of reads to be used as evidence. The corresponding environment variable is `MAVIS_MIN_MAPPING_QUALITY` and the default value is `5`

min_non_target_aligned_split_reads `int` - The minimum number of split reads aligned to a breakpoint by the input bam and no forced by local alignment to the target region to call a breakpoint by split read evidence. The corresponding environment variable is `MAVIS_MIN_NON_TARGET_ALIGNED_SPLIT_READS` and the default value is `1`

min_orf_size `int` - The minimum length (in base pairs) to retain a putative open reading frame (orf). The corresponding environment variable is `MAVIS_MIN_ORF_SIZE` and the default value is `300`

min_sample_size_to_apply_percentage `int` - Minimum number of aligned bases to compute a match percent. if there are less than this number of aligned bases (match or mismatch) the percent comparator is not used. The corresponding environment variable is `MAVIS_MIN_SAMPLE_SIZE_TO_APPLY_PERCENTAGE` and the default value is `10`

min_softclipping `int` - Minimum number of soft-clipped bases required for a read to be used as soft-clipped evidence. The corresponding environment variable is `MAVIS_MIN_SOFTCLIPPING` and the default value is `6`

min_spanning_reads_resolution `int` - Minimum number of spanning reads required to call an event by spanning evidence. The corresponding environment variable is `MAVIS_MIN_SPANNING_READS_RESOLUTION` and the default value is `5`

min_splits_reads_resolution `int` - Minimum number of split reads required to call a breakpoint by split reads. The corresponding environment variable is `MAVIS_MIN_SPLITS_READS_RESOLUTION` and the default value is `3`

novel_exon_color `str` - Novel exon fill color. The corresponding environment variable is `MAVIS_NOVEL_EXON_COLOR` and the default value is `'#5D3F6A'`

outer_window_min_event_size `int` - The minimum size of an event in order for flanking read evidence to be collected. The corresponding environment variable is `MAVIS_OUTER_WINDOW_MIN_EVENT_SIZE` and the default value is `125`

queue `str` - The queue jobs are to be submitted to. The corresponding environment variable is `MAVIS_QUEUE` and the default value is `''`

reference_genome `filepath` - Path to the human reference genome fasta file. The corresponding environment variable is `MAVIS_REFERENCE_GENOME` and the default value is `[]`

remote_head_ssh `str` - Ssh target for remote scheduler commands. The corresponding environment variable is `MAVIS_REMOTE_HEAD_SSH` and the default value is `''`

scaffold_color `str` - The color used for the gene/transcripts scaffolds. The corresponding environment variable is `MAVIS_SCAFFOLD_COLOR` and the default value is `'#000000'`

scheduler SCHEDULER - The scheduler being used. The corresponding environment variable is MAVIS_SCHEDULER and the default value is 'SLURM'. Accepted values include: 'SGE', 'SLURM', 'TORQUE', 'LOCAL'

spanning_call_distance int - The maximum distance allowed between breakpoint pairs (called by spanning reads) in order for them to pair. The corresponding environment variable is MAVIS_SPANNING_CALL_DISTANCE and the default value is 20

splice_color str - Splicing lines color. The corresponding environment variable is MAVIS_SPLICE_COLOR and the default value is '#000000'

split_call_distance int - The maximum distance allowed between breakpoint pairs (called by split reads) in order for them to pair. The corresponding environment variable is MAVIS_SPLIT_CALL_DISTANCE and the default value is 20

stdev_count_abnormal float - The number of standard deviations away from the normal considered expected and therefore not qualifying as flanking reads. The corresponding environment variable is MAVIS_STDEV_COUNT_ABNORMAL and the default value is 3.0

strand_determining_read int - 1 or 2. the read in the pair which determines if (assuming a stranded protocol) the first or second read in the pair matches the strand sequenced. The corresponding environment variable is MAVIS_STRAND_DETERMINING_READ and the default value is 2

template_metadata filepath - File containing the cytoband template information. used for illustrations only. The corresponding environment variable is MAVIS_TEMPLATE_METADATA and the default value is []

time_limit int - The time in seconds any given jobs is allowed. The corresponding environment variable is MAVIS_TIME_LIMIT and the default value is 57600

trans_fetch_reads_limit int - Related to *fetch_reads_limit*. overrides fetch_reads_limit for transcriptome libraries when set. if this has a value of none then fetch_reads_limit will be used for transcriptome libraries instead. The corresponding environment variable is MAVIS_TRANS_FETCH_READS_LIMIT and the default value is 12000

trans_min_mapping_quality int - Related to *min_mapping_quality*. overrides the min_mapping_quality if the library is a transcriptome and this is set to any number not none. if this value is none, min_mapping_quality is used for transcriptomes aswell as genomes. The corresponding environment variable is MAVIS_TRANS_MIN_MAPPING_QUALITY and the default value is 0

trans_validation_memory int - Default memory limit (mb) for the validation stage (for transcriptomes). The corresponding environment variable is MAVIS_TRANS_VALIDATION_MEMORY and the default value is 18000

uninformative_filter bool - Flag that determines if breakpoint pairs which are not within max_proximity to any annotations are filtered out prior to clustering. The corresponding environment variable is MAVIS_UNINFORMATIVE_FILTER and the default value is False

validation_memory int - Default memory limit (mb) for the validation stage. The corresponding environment variable is MAVIS_VALIDATION_MEMORY and the default value is 16000

width int - The drawing width in pixels. The corresponding environment variable is MAVIS_WIDTH and the default value is 1000

write_evidence_files bool - Write the intermediate bam and bed files containing the raw evidence collected and contigs aligned. not required for subsequent steps but can be useful in debugging and deep investigation of events. The corresponding environment variable is MAVIS_WRITE_EVIDENCE_FILES and the default value is True

11.3 Column Names

List of column names and their definitions. The types indicated here are the expected types in a row for a given column name.

annotation_figure `FILEPATH` - File path to the svg drawing representing the annotation

annotation_figure_legend `JSON` - *JSON* data for the figure legend

annotation_id Identifier for the annotation step

break1_chromosome `str` - The name of the chromosome on which breakpoint 1 is situated

break1_ewindow `int-int` - Window where evidence was gathered for the first breakpoint

break1_ewindow_count `int` - Number of reads processed/looked-at in the first evidence window

break1_ewindow_practical_coverage `float` - break2_ewindow_practical_coverage, break1_ewindow_count / len(break1_ewindow). Not the actual coverage as bins are sampled within and there is a read limit cutoff

break1_homologous_seq `str` - Sequence in common at the first breakpoint and other side of the second breakpoint

break1_orientation `ORIENT` - The side of the breakpoint wrt the positive/forward strand that is retained.

break1_position_end `int` - End integer inclusive 1-based of the range representing breakpoint 1

break1_position_start `int` - Start integer inclusive 1-based of the range representing breakpoint 1

break1_seq `str` - The sequence up to and including the breakpoint. Always given wrt to the positive/forward strand

break1_split_reads `int` - Number of split reads that call the exact breakpoint given

break1_split_reads_forced `int` - Number of split reads which were aligned to the opposite breakpoint window using a targeted alignment

break1_strand `STRAND` - The strand wrt to the reference positive/forward strand at this breakpoint.

break2_chromosome The name of the chromosome on which breakpoint 2 is situated

break2_ewindow `int-int` - Window where evidence was gathered for the second breakpoint

break2_ewindow_count `int` - Number of reads processed/looked-at in the second evidence window

break2_ewindow_practical_coverage `float` - break2_ewindow_practical_coverage, break2_ewindow_count / len(break2_ewindow). Not the actual coverage as bins are sampled within and there is a read limit cutoff

break2_homologous_seq `str` - Sequence in common at the second breakpoint and other side of the first breakpoint

break2_orientation `ORIENT` - The side of the breakpoint wrt the positive/forward strand that is retained.

break2_position_end `int` - End integer inclusive 1-based of the range representing breakpoint 2

break2_position_start `int` - Start integer inclusive 1-based of the range representing breakpoint 2

break2_seq `str` - The sequence up to and including the breakpoint. Always given wrt to the positive/forward strand

break2_split_reads `int` - Number of split reads that call the exact breakpoint given

break2_split_reads_forced `int` - Number of split reads which were aligned to the opposite breakpoint window using a targeted alignment

break2_strand `STRAND` - The strand wrt to the reference positive/forward strand at this breakpoint.

call_method `CALL_METHOD` - The method used to call the breakpoints

call_sequence_complexity `float` - The minimum amount any two bases account for of the proportion of call sequence. An average for non-contig calls

cdna_synon semi-colon delimited list of transcript ids which have an identical cdna sequence to the cdna sequence of the current fusion product

cluster_id Identifier for the merging/clustering step

cluster_size `int` - The number of breakpoint pair calls that were grouped in creating the cluster

contig_alignment_cigar The cigar string(s) representing the contig alignment. Semi-colon delimited

contig_alignment_query_name The query name for the contig alignment. Should match the 'read' name(s) in the .contigs.bam output file

contig_alignment_reference_start The reference start(s) <chr>:<position> of the contig alignment. Semi-colon delimited

contig_alignment_score `float` - A rank based on the alignment tool blat etc. of the alignment being used. An average if split alignments were used. Lower numbers indicate a better alignment. If it was the best alignment possible then this would be zero.

contig_build_score `int` - Score representing the edge weights of all edges used in building the sequence

contig_remap_coverage `float` - Fraction of the contig sequence which is covered by the remapped reads

contig_remap_score `float` - Score representing the number of sequences from the set of sequences given to the assembly algorithm that were aligned to the resulting contig with an acceptable scoring based on user-set thresholds. For any sequence its contribution to the score is divided by the number of mappings to give less weight to multimaps

contig_remapped_read_names read query names for the reads that were remapped. A -1 or -2 has been appended to the end of the name to indicate if this is the first or second read in the pair

contig_remapped_reads `int` - the number of reads from the input bam that map to the assembled contig

contig_seq `str` - Sequence of the current contig wrt to the positive forward strand if not strand specific

contig_strand_specific `bool` - A flag to indicate if it was possible to resolve the strand for this contig

contigs_aligned `int` - Number of contigs that were able to align

contigs_assembled `int` - Number of contigs that were built from split read sequences

event_type `SVTYPE` - The classification of the event

flanking_median_fragment_size `int` - The median fragment size of the flanking reads being used as evidence

flanking_pairs `int` - Number of read-pairs where one read aligns to the first breakpoint window and the second read aligns to the other. The count here is based on the number of unique query names

flanking_pairs_compatible `int` - Number of flanking pairs of a compatible orientation type. This applies to insertions and duplications. Flanking pairs supporting an insertion will be compatible to a duplication and flanking pairs supporting a duplication will be compatible to an insertion (possibly indicating an internal translocation)

flanking_stdev_fragment_size `float` - The standard deviation in fragment size of the flanking reads being used as evidence

fusion_cdna_coding_end Position wrt the 5' end of the fusion transcript where coding ends last base of the stop codon

fusion_cdna_coding_end `int` - Position wrt the 5' end of the fusion transcript where coding ends last base of the stop codon

fusion_cdna_coding_start `int` - Position wrt the 5' end of the fusion transcript where coding begins first base of the Met amino acid.

fusion_mapped_domains `JSON` - List of domains in *JSON* format where each domain start and end positions are given wrt to the fusion transcript and the mapping quality is the number of matching amino acid positions over the total number of amino acids. The sequence is the amino acid sequence of the domain on the reference/original transcript

fusion_protein_hgvs `str` - Describes the fusion protein in HGVS notation. Will be None if the change is not an indel or is synonymous

fusion_sequence_fasta_file `FILEPATH` - Path to the corresponding fasta output file

fusion_sequence_fasta_id The sequence identifier for the cdna sequence output fasta file

fusion_splicing_pattern `SPLICE_TYPE` - Type of splicing pattern used to create the fusion cDNA.

gene1 Gene for the current annotation at the first breakpoint

gene1_aliases Other gene names associated with the current annotation at the first breakpoint

gene1_direction `PRIME` - The direction/prime of the gene

gene2 Gene for the current annotation at the second breakpoint

gene2_aliases Other gene names associated with the current annotation at the second breakpoint

gene2_direction `PRIME` - The direction/prime of the gene. Has the following possible values

gene_product_type `GENE_PRODUCT_TYPE` - Describes if the putative fusion product will be sense or anti-sense

genes_encompassed Applies to intrachromosomal events only. List of genes which overlap any region that occurs between both breakpoints. For example in a deletion event these would be deleted genes.

genes_overlapping_break1 list of genes which overlap the first breakpoint

genes_overlapping_break2 list of genes which overlap the second breakpoint

genes_proximal_to_break1 list of genes near the breakpoint and the distance away from the breakpoint

genes_proximal_to_break2 list of genes near the breakpoint and the distance away from the breakpoint

inferred_pairing A semi colon delimited of event identifiers i.e. <annotation_id>_<splicing pattern>_<cds start>_<cds end> which were paired to the current event based on predicted products

library Identifier for the library/source

linking_split_reads `int` - Number of split reads that align to both breakpoints

net_size `int-int` - The net size of an event. For translocations and inversion this will always be 0. For indels it will be negative for deletions and positive for insertions. It is a range to accommodate non-specific events.

opposing_strands `bool` - Specifies if breakpoints are on opposite strands wrt to the reference. Expects a boolean

pairing A semi colon delimited of event identifiers i.e. <annotation_id>_<splicing pattern>_<cds start>_<cds end> which were paired to the current event based on breakpoint positions

product_id Unique identifier of the final fusion including splicing and ORF decision from the annotation step

protein_synon semi-colon delimited list of transcript ids which produce a translation with an identical amino-acid sequence to the current fusion product

protocol `PROTOCOL` - Specifies the type of library

raw_break1_split_reads `int` - Number of split reads before calling the breakpoint

raw_break2_split_reads `int` - Number of split reads before calling the breakpoint

raw_flanking_pairs `int` - Number of flanking reads before calling the breakpoint. The count here is based on the number of unique query names

raw_spanning_reads `int` - Number of spanning reads collected during evidence collection before calling the breakpoint

spanning_read_names read query names of the spanning reads which support the current event

spanning_reads `int` - the number of spanning reads which support the event

stranded `bool` - Specifies if the sequencing protocol was strand specific or not. Expects a boolean

supplementary_call `bool` - Flag to indicate if the current event was a supplementary call, meaning a call that was found as a result of validating another event.

tools The tools that called the event originally from the cluster step. Should be a semi-colon delimited list of <tool name>_<tool version>

tracking_id column used to store input identifiers from the original SV calls. Used to track calls from the input files to the final outputs.

transcript1 Transcript for the current annotation at the first breakpoint

transcript2 Transcript for the current annotation at the second breakpoint

untemplated_seq `str` - The untemplated/novel sequence between the breakpoints

validation_id Identifier for the validation step

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Abyzov-2011] Abyzov,A. et al. (2011) CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing. Genome Res., 21, 974–984.

[Abyzov-2015] Abyzov,A. et al. (2015) Analysis of deletion breakpoints from 1,092 humans reveals details of mutation mechanisms. Nat. Commun., 6, 7256.

[Chen-2009] Chen,K. et al. (2009) BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. Nat. Methods, 6, 677–681.

[Chen-2016] Chen,X. et al. (2016) Manta: rapid detection of structural variants and indels for germline and cancer sequencing applications. Bioinformatics, 32, 1220–1222.

[den-Dunnen-2016] den Dunnen,J.T. et al. (2016) HGVS Recommendations for the Description of Sequence Variants: 2016 Update. Hum. Mutat., 37, 564–569.

[Iyer-2011] Iyer,M.K. et al. (2011) ChimeraScan: a tool for identifying chimeric transcription in sequencing data. Bioinformatics, 27, 2903–2904.

[MacDonald-2014] MacDonald,J.R. et al. (2014) The Database of Genomic Variants: a curated collection of structural variation in the human genome. Nucleic Acids Res., 42, D986–92.

[McPherson-2011] McPherson,A. et al. (2011) deFuse: an algorithm for gene fusion discovery in tumor RNA-Seq data. PLoS Comput. Biol., 7, e1001138.

[Rausch-2012] Rausch,T. et al. (2012) DELLY: structural variant discovery by integrated paired-end and split-read analysis. Bioinformatics, 28, i333–i339.

[Robertson-2010] Robertson,G. et al. (2010) De novo assembly and analysis of RNA-seq data. Nat. Methods, 7, 909–912.

[Yates-2016] Yates,A. et al. (2016) Ensembl 2016. Nucleic Acids Res., 44, D710–D716.

[Ye-2009] Ye,K. et al. (2009) Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. Bioinformatics, 25, 2865–2871.

[Saunders-2012] Saunders,C.T. et al. (2012) Strelka: accurate somatic small-variant calling from sequenced tumor–normal sample pairs. Bioinformatics, 28, 1811–1817.

[Haas-2017] Haas,B et al. (2017) STAR-Fusion: Fast and Accurate Fusion Transcript Detection from RNA-Seq. doi: https://doi.org/10.1101/120295

PYTHON MODULE INDEX

m

mavis, 45
mavis.annotate, 45
mavis.bam, 46
mavis.cluster, 47
mavis.illustrate, 48
mavis.pairing, 48
mavis.schedule, 50
mavis.summary, 51
mavis.validate, 49

INDEX

Symbols

2bit, 53

A

aligner, 54
aligner_reference, 54
annotation_figure, 61
annotation_figure_legend, 61
annotation_filters, 54
annotation_id, 61
annotation_memory, 54
annotations, 54
assembly_kmer_size, 54
assembly_max_paths, 54
assembly_min_edge_trim_weight, 54
assembly_min_exact_match_to_remap, 55
assembly_min_remap_coverage, 55
assembly_min_remapped_seq, 55
assembly_min_uniq, 55
assembly_strand_concordance, 55

B

BAM, 53
bed, 53
blat, 53
blat_limit_top_aln, 55
blat_min_identity, 55
break1_chromosome, 61
break1_ewindow, 61
break1_ewindow_count, 61
break1_ewindow_practical_coverage, 61
break1_homologous_seq, 61
break1_orientation, 61
break1_position_end, 61
break1_position_start, 61
break1_seq, 61
break1_split_reads, 61
break1_split_reads_forced, 61
break1_strand, 61
break2_chromosome, 61
break2_ewindow, 61
break2_ewindow_count, 61
break2_ewindow_practical_coverage, 61
break2_homologous_seq, 61
break2_orientation, 61
break2_position_end, 61
break2_position_start, 61
break2_seq, 61
break2_split_reads, 61
break2_split_reads_forced, 61
break2_strand, 61
BreakDancer, 53
breakpoint, 53
breakpoint pair, 53
breakpoint_color, 55
BreakSeq, 53
BWA, 53

C

call_error, 55
call_method, 61
call_sequence_complexity, 61
cdna_synon, 62
Chimerascan, 53
clean_aligner_files, 55
cluster_id, 62
cluster_initial_size_limit, 55
cluster_radius, 55
cluster_size, 62
CNVnator, 53
concurrency_limit, 55
contig_alignment_cigar, 62
contig_alignment_query_name, 62
contig_alignment_reference_start, 62
contig_alignment_score, 62
contig_aln_max_event_size, 55
contig_aln_merge_inner_anchor, 55
contig_aln_merge_outer_anchor, 55
contig_aln_min_anchor_size, 56
contig_aln_min_extend_overlap, 56
contig_aln_min_query_consumption, 56
contig_aln_min_score, 56
contig_build_score, 62
contig_call_distance, 56

contig_remap_coverage, 62
contig_remap_score, 62
contig_remapped_read_names, 62
contig_remapped_reads, 62
contig_seq, 62
contig_strand_specific, 62
contigs_aligned, 62
contigs_assembled, 62

D

DeFuse, 53
DELLY, 53
dgv_annotation, 56
domain_color, 56
domain_mismatch_color, 56
domain_name_regex_filter, 56
domain_scaffold_color, 56
draw_fusions_only, 56
draw_non_synonymous_cdna_only, 56
drawing_width_iter_increase, 56

E

event, 53
event type, 53
event_type, 62
exon_min_focus_size, 56

F

fasta, 53
fetch_min_bin_size, 56
fetch_reads_bins, 56
fetch_reads_limit, 56
filter_cdna_synon, 57
filter_min_complexity, 57
filter_min_flanking_reads, 57
filter_min_linking_split_reads, 57
filter_min_remapped_reads, 57
filter_min_spanning_reads, 57
filter_min_split_reads, 57
filter_protein_synon, 57
filter_secondary_alignments, 57
filter_trans_homopolymers, 57
flanking read pair, 53
flanking_call_distance, 57
flanking_median_fragment_size, 62
flanking_pairs, 62
flanking_pairs_compatible, 62
flanking_stdev_fragment_size, 62
fusion_cdna_coding_end, 62
fusion_cdna_coding_start, 62
fusion_mapped_domains, 63
fusion_protein_hgvs, 63
fusion_sequence_fasta_file, 63
fusion_sequence_fasta_id, 63
fusion_splicing_pattern, 63
fuzzy_mismatch_number, 57

G

gene1, 63
gene1_aliases, 63
gene1_color, 57
gene1_color_selected, 57
gene1_direction, 63
gene2, 63
gene2_aliases, 63
gene2_color, 57
gene2_color_selected, 57
gene2_direction, 63
gene_product_type, 63
genes_encompassed, 63
genes_overlapping_break1, 63
genes_overlapping_break2, 63
genes_proximal_to_break1, 63
genes_proximal_to_break2, 63

H

half-mapped read, 53
HGVS, 53

I

IGV, 53
IGV batch file, 53
import_env, 57
inferred_pairing, 63
input_call_distance, 57

J

JSON, 53

L

label_color, 57
library, 63
limit_to_chr, 57
linking_split_reads, 63

M

mail_type, 58
mail_user, 58
Manta, 53
mask_fill, 58
mask_opacity, 58
masking, 58
mavis (*module*), 45
mavis.annotate (*module*), 45
mavis.bam (*module*), 46
mavis.cluster (*module*), 47
mavis.illustrate (*module*), 48
mavis.pairing (*module*), 48

mavis.schedule (*module*), 50
mavis.summary (*module*), 51
mavis.validate (*module*), 49
max_drawing_retries, **58**
max_files, **58**
max_orf_cap, **58**
max_proximity, **58**
max_sc_preceeding_anchor, **58**
memory_limit, **58**
min_anchor_exact, **58**
min_anchor_fuzzy, **58**
min_anchor_match, **58**
min_call_complexity, **58**
min_clusters_per_file, **58**
min_domain_mapping_match, **59**
min_double_aligned_to_estimate_insertion_size, **59**
min_flanking_pairs_resolution, **59**
min_linking_split_reads, **59**
min_mapping_quality, **59**
min_non_target_aligned_split_reads, **59**
min_orf_size, **59**
min_sample_size_to_apply_percentage, **59**
min_softclipping, **59**
min_spanning_reads_resolution, **59**
min_splits_reads_resolution, **59**

N

net_size, **63**
novel_exon_color, **59**

O

opposing_strands, **63**
outer_window_min_event_size, **59**

P

pairing, **63**
Pindel, **53**
product_id, **63**
protein_synon, **63**
protocol, **63**
psl, **54**
pslx, **54**

Q

queue, **59**

R

raw_break1_split_reads, **63**
raw_break2_split_reads, **63**
raw_flanking_pairs, **63**
raw_spanning_reads, **64**
reference_genome, **59**
remote_head_ssh, **59**

S

scaffold_color, **59**
scheduler, **60**
SGE, **54**
SLURM, **54**
spanning read, **54**
spanning_call_distance, **60**
spanning_read_names, **64**
spanning_reads, **64**
splice_color, **60**
split read, **54**
split_call_distance, **60**
STAR-Fusion, **54**
stdev_count_abnormal, **60**
strand_determining_read, **60**
stranded, **64**
Strelka, **54**
structural variant, **54**
supplementary_call, **64**
SV, **54**
SVG, **54**

T

template_metadata, **60**
time_limit, **60**
tools, **64**
TORQUE, **54**
tracking_id, **64**
trans_fetch_reads_limit, **60**
trans_min_mapping_quality, **60**
trans_validation_memory, **60**
Trans-ABySS, **54**
transcript1, **64**
transcript2, **64**

U

uninformative_filter, **60**
untemplated_seq, **64**

V

validation_id, **64**
validation_memory, **60**

W

width, **60**
write_evidence_files, **60**