TileSeqMave

TileSeqMave constitutes the second half of the TileSeq Pipeline. The first half is implemented in `tilseq_mutcount` and provides alignment and variant calling of TileSeq sequencing reads. TileSeqMave allows for the calculation of fitness scores from these reads and provides additional analysis functions.

Table of contents

Requirements and Installation

`tileseqMave` is compatible with Unix-based operating systems, requires R 3.4.4 or greater and has a number of R package dependencies, which will be installed automatically. It also uses the output of `tilseq_mutcount`, which should be installed separately on an HPC platform.

To install the `tileseqMave` package, use the `R` package `devtools`:

```r
install.packages("devtools")
devtools::install_github("jweile/tileseqMave")
```

Important: To ensure that the scripts included in this software suite can be executed easily on the command line, they need to be available via your `$PATH` variable. The easiest way to achieve this is to symlink them to your preferred `bin/` directory after having installed the package as shown above. For `UNIX` systems, a bootstrap script is provided for this purpose. For example, to create symlinks in `~/.local/bin/` you can use:

```bash
bash `Rscript -e 'cat(system.file("scripts/linkBinaries.R",package="tileseqMave"))'` ~/.local/bin/
```

Overview of the pipeline

The TileSEQ pipeline breaks down into multiple modules:

1. csv2json: This component reads and validates the parameter sheet which contains all relevant options and parameters of the experiment to be analyzed.
2. fastq2count: This component processes the FASTQ files from a sequencing run and produces variant count files. It is implemented in the `tilseq_mutcount` python module.
3. joinCounts: Combines the individual count files from each sequencing sample into a single table, tabulating the read counts and relative frequencies for each unique variant in each condition and replicate. It also calculates the protein-level consequences of each variant and produces marginal counts and frequencies.
4. libraryQC: Produces a number of diagnostic graphs for quality control purposes at the variant library level.
5. scoring: Calculates enrichment ratios and fitness scores for each variant in each selection condition. Also performs error regularization and filtering.

6. selectionQC: Produces additional diagnostic graphs at the selection assay level.

The parameter sheet

tileseqMave uses a central input document called the parameter sheet, which attempts to capture all relevant aspects of the TileSEQ experiment from which the sequencing data was generated. The parameter sheet is provided as a comma-separated-values (CSV) file, following a strict template. The easiest way to create such a file is to use a spreadsheet editor, such as Google Sheets, LibreOffice or Excel. An example spreadsheet can be found here.

The parameter sheet has the following sections:

1. **Project name**: This can be any name you like.
2. **Template construct**: Here we define the sequencing template, its sequence and what it represents.
   - Gene name: The official HGNC gene name
   - Sequence: The full nucleotide sequence of the template including the coding sequence (CDS) and its flanking priming sequences.
   - CDS start: The position in the above sequence at which the coding sequence (CDS) begins. This must be an ATG codon! (Numbering starts at 1, not at 0).
   - CDS end: The equivalent sequence position at which the CDS ends. This must be in-frame with the start position, i.e. end-start+1 must be divisible by 3.
   - Uniprot Accession: The UniprotKB accession of the protein encoded by the template gene.
3. **Assay**: A summary of the underlying selection assay
   - Assay Type: This can be any free-text label you like. But we would encourage you to pick a simple, recognizable name such as “Y2H”, “Yeast complementation” or “LDL uptake via FACS”
   - Description: This can be any free-text description you like.
   - Negative selection: This field indicates whether the assay performs a positive or negative selection. So only the values “Yes” and “No” are allowed. Positive selection indicates that the assay causes damaging variants to be depleted in the pool, where as negative selection indicates that the assay causes neutral variants to be depleted. (Most assays will be positive, so for most cases the value would be “No”).
4. **Conditions and replicates**: This is a custom table that lists the different experimental conditions and their intended number of replicates and time points.
   - List of conditions: In this table row, provide a list of condition identifiers. These should be short and must not contain any special characters.
   - Number of replicates: For each of the defined conditions in the previous row, provide the number of technical replicates here.
   - Number of time points: For each of the defined conditions in the first row, provide the number of time points. (At the time of writing, this feature has not been implemented yet).
5. **Mutagenesis regions**: This is a fixed table defining the regions which underwent separate PopCode mutagenesis. It has the following columns:
   - Region number: A simple numbering of the regions to serve as identifiers thereof.
   - Start AA: The first amino acid position that counts as within the region.
   - End AA: The last amino acid position that counts as within the region.
6. **Sequencing tiles**: This also a fixed table, defining the eponymous TileSeq sequencing tiles in the experiment. It has the following columns:
   - Tile number: A simple numbering of the tiles to serve as identifiers thereof.
   - Start AA: The first amino acid position that counts as within the tile.
   - End AA: The last amino acid position that counts as within the tile.
7. **Condition definitions**: This section is used to define the meanings of the different conditions as to how they relate to each other. Currently, two kinds of relationships are supported: `is_selection_for`, and `is_wt_control_for`. For example, if we have defined three conditions in section 4; sel, non and ctrl, then sel could be the selection condition for non and ctrl could be the wt control for either of them. These relationships are defined using a fixed table with three columns:

    - Condition 1: The ID of the first condition in the relationship. Must have been previously declared in the list of conditions.
    - Relationship: The name of the relationship, either `is_selection_for` or `is_wt_control_for`
    - Condition 2: The ID of the section condition in the relationship. Must have been previously declared in the list of conditions.

8. **Time point definitions**: This fixed table is used to define time points in the experiment. At least one time point must be defined. The table has the following columns:
    - Time point name: The name of the time point. This should be a short identifier without spaces or special characters
    - Time: The numerical part of time point definition.
    - Unit: The time unit, such as "s" for seconds, "m" for minutes, "h" for hours, "d" for days, etc.
9. **Sequencing samples**: This final table contains the information of what the sequencing samples represent. It has the following columns:
    - Sample ID: This is the ID of the sample used in the name of the corresponding pair of FASTQ files. This can be a number or an alphanumerical label. No special characters (except "-" signs) are allowed.
    - Tile ID: The sequencing tile to which this sample belongs. This is a cross-reference to the "Tile Number" in the "Sequencing tiles" table and must have a matching entry there.
    - Condition: The condition to which this sample belongs. This is a cross-reference to the list of condition names and must have a matching entry there.
    - Time point: The time point to which this sample belongs. This is a cross-reference to the Time point definitions and must have a matching entry there.
    - Replicate: The replicate to which this sample belongs. This must be an integer number and must be within the range of replicates defined for the appropriate condition.
10. **Variant calling parameters**: This section contains meta-parameters for the variant caller.
    - Posterior threshold: The posterior probability of a variant call that must be surpassed for the call to be accepted (as opposed to the WT base). Must be between 0.5 and 1.
    - Minimum coverage: The fraction of a read's length that must fall within the correct tile for it to be counted. Must be between 0 and 1.
    - Per-base mutation rate: The assumed per-base mutation rate in the library as a result of PopCode mutagenesis. This is used to inform the prior for variant calls.
11. **Scoring function parameters**: This section contains meta-parameters for the scoring function.
    - Minimum read count: The minimum marginal read count a variant must have in the non-selective condition to pass filter. Must be at least 1.
    - Pseudo-replicates: The number of pseudo-replicates assigned to the prior estimate of error during Baldi & Long error regularization. Must be at least 1.
    - SD threshold: The standard deviation threshold to determine the quality of variants that are included in calculating synonymous and nonsense log-ratio medians, as part of score scaling.
12. **Score normalization overrides**: This table is optional and can be used to provide manual overrides for the synonymous and nonsense log ratio modes used during score scaling. This table has the following columns:
    - Condition: The ID of the selective condition to which this override applies.
    - Time point: The ID of the time point to which this override applies.
    - Region: The mutagenesis region number to which this override applies.
    - Type: Whether this is an override for the "synonymous" or "nonsense" mode.
    - Value: The numerical value of the log ratio mode to use.

Running tileseqMave

Converting the parameter sheet

To convert the parameter sheet to JSON format you can use the csv2json.R script. This will also perform a validation of the parameter sheet, ensuring that all requirements have been met.

The resulting JSON file is used by all subsequent components of the pipeline. However, if you plan on running the entire pipeline including the tilseq_mutcount component, you can skip this step, as tilseq_mutcount will run this conversion automatically.

```
usage: csv2json.R [--] [--help] [--srOverride] [--opts OPTS] [--outfile
       OUTFILE] [--logfile LOGFILE] infile

Checks TileSeq parameter CSV file for validity and converts it to JSON
format.

positional arguments:
  infile            input file

flags:
  -h, --help        show this help message and exit
  -s, --srOverride  Manual override to allow singleton replicates. USE
                    WITH EXTREME CAUTION!

optional arguments:
  -x, --opts        RDS file containing argument values
  -o, --outfile     output file. Defaults to parameters.json in the
                    same directory.
  -l, --logfile     log file. Defaults to csv2json.log in the same
                    directory.
```

Counting variants from FASTQ data

This step is performed by tilseq_mutcount. Check out the respective github page for full instructions. This requires a HPC environment. In the Roth Lab, this will be either the Guru or Galen system at LTRI, or the BC2 or DC system at the Donnelly Centre.

You will first want to create a workspace folder in which the output will be collected. For example "MTHFR_Complementation/". Place your parameter sheet into this folder to ensure all input and output is organized together. Then, run the script using this directory as the output folder.

The result of this tool will include a subdirectory with a name like: `MTHFR_Complementation/2020-02-20-11-15-49_mut_call` containing a timestamp of the script execution. This "mut_call" subdirectory will serve as the input for the next step. It contains individual count files for each sequencing sample.

Joining variant counts and computing marginal frequencies

This step is performed by `joinCounts.R`. It will collate the counts from the individual sequencing samples and organize them by variant and condition/replicate. This step will also translate the called variants to protein level. As a result, two new files will be added to the `mut_call` directory: all_counts.csv and marginal_counts.csv

```
usage: joinCounts.R [--] [--help] [--srOverride] [--opts OPTS]
       [--parameters PARAMETERS] [--logfile LOGFILE] [--cores CORES]
       dataDir

Reads the output of fastq2Count to construct allCounts.csv and
marginalCounts.csv

positional arguments:
  dataDir           workspace data directory
```

```
flags:
  -h, --help        show this help message and exit
  -s, --srOverride  Manual override to allow singleton replicates. USE
                    WITH EXTREME CAUTION!

optional arguments:
  -x, --opts        RDS file containing argument values
  -p, --parameters  parameter file. Defaults to parameters.json in the
                    data directory.
  -l, --logfile     log file. Defaults to joinCounts.log in the same
                    directory
  -c, --cores       number of CPU cores to use in parallel for
                    multi-threading [default: 6]
```

Running a library QC analysis

After `joinCounts.R` has completed, a library QC analysis can be performed using `runLibraryQC.R`. This will generate a number of diagnostic plots for each non-select condition in the dataset, including:

- Nucleotide bias analysis
- Variant census
- Coverage map
- Complexity analysis
- Well-measuredness analysis
- Variant-type breakdown

The plots will be saved as PDF files in a new folder called `<timestamp>_QC` where `timestamp` is the time of the original variant caller execution.

```
usage: runLibraryQC.R [--] [--help] [--srOverride] [--opts OPTS]
       [--parameters PARAMETERS] [--logfile LOGFILE] [--cores CORES]
       dataDir

Performs a library QC analysis on the output of joinCounts.R, yielding
informative plots.

positional arguments:
  dataDir           workspace data directory

flags:
  -h, --help        show this help message and exit
  -s, --srOverride  Manual override to allow singleton replicates. USE
                    WITH EXTREME CAUTION!

optional arguments:
  -x, --opts        RDS file containing argument values
  -p, --parameters  parameter file. Defaults to parameters.json in the
                    data directory.
  -l, --logfile     log file. Defaults to libraryQC.log in the same
                    directory
  -c, --cores       number of CPU cores to use in parallel for
                    multi-threading [default: 6]
```

Running the scoring function

After `joinCounts.R` has completed, the scoring function `runScoring.R` can be called. This will require that at least one selection condition was declared in the parameter sheet. The scoring function will then:

- Create error models for each tile and condition and use these models for error regularization
- Perform quality filtering (based on non-select counts, bottlenecking and quality)
- Calculate log ratios and final fitness scores.

The results of the scoring function will be produced in a new directory called `<timestamp>_scores` where `timestamp` is the time of the original variant caller execution. It will contain the following files for each selection condition:

```
* complete.csv: A table containing all intermediate results for all variants, regardless of filter statu
* simple.csv: A simplified result containing only the variants that passed all filters with their final
* simple_aa.csv: The same results collapsed for equivalent amino acid changes via confidence-weighted av
```

```
usage: runScoring.R [--] [--help] [--srOverride] [--opts OPTS]
       [--parameters PARAMETERS] [--countThreshold COUNTTHRESHOLD]
       [--pseudoReplicates PSEUDOREPLICATES] [--sdThreshold
       SDTHRESHOLD] [--logfile LOGFILE] [--cores CORES] dataDir

Runs the main scoring function on the output of joinCounts.R

positional arguments:
  dataDir               workspace data directory

flags:
  -h, --help            show this help message and exit
  --srOverride          Manual override to allow singleton replicates.
                        USE WITH EXTREME CAUTION!

optional arguments:
  -x, --opts            RDS file containing argument values
  -p, --parameters      parameter file. Defaults to parameters.json in
                        the data directory.
  -l, --logfile         log file. Defaults to scoring.log in the same
                        directory
  --cores               number of CPU cores to use in parallel for
                        multi-threading [default: 6]
```

Running a Selection QC analysis

After the scoring function has been called, a selection QC can be performed using `runSelectionQC.R`. Similarly to the library QC function, this will generate a number of diagnostic plots as PDF files. In particular:

- errorModel: Plots for each tile that illustrate how well the error corresponds to the expected Poisson model.
- errorProfile: Plot the score of each variant against its associated standard deviation.
- logPhiDistribution: Shows the distribution of enrichment log ratios for synonymous, nonsense and missense variants.
- replicates: Shows the correlation of technical replicates as scatterplots.

```
usage: runSelectionQC.R [--] [--help] [--srOverride] [--opts OPTS]
       [--parameters PARAMETERS] [--sdThreshold SDTHRESHOLD] [--logfile
       LOGFILE] dataDir

Performs a selection QC analysis on the output of runScoring.R,
yielding informative plots.

positional arguments:
  dataDir            workspace data directory

flags:
  -h, --help         show this help message and exit
  --srOverride       Manual override to allow singleton replicates. USE
                     WITH EXTREME CAUTION!

optional arguments:
  -x, --opts         RDS file containing argument values
  -p, --parameters   parameter file. Defaults to parameters.json in the
                     data directory.
  -l, --logfile      log file. Defaults to selectionQC.log in the same
                     directory
```

Example execution

Let us assume we have already used tilseq_mutcount to convert your FASTQ files into counts. As a result, we have a directory "CHEK2_tileseq/" with a parameters JSON file and a `mut_call` subdirectory:

```
$ ll

total 20136
drwxrwxr-x  6 jweile jweile     4096 Mar 13 11:48 ./
drwxr-xr-x 14 jweile jweile     4096 Mar 13 14:33 ../
drwxrwxr-x  4 jweile jweile     4096 Mar 13 15:41 2020-03-11-00-16-59_mut_call/
-rw-r--r--  1 jweile jweile     7242 Mar 12 17:12 parameters.json
```

The latter contains all individual samples' variant counts:

```
$ ll 2020-03-11-00-16-59_mut_call

total 70812
drwxrwxr-x 4 jweile jweile     4096 Mar 13 15:41 ./
drwxrwxr-x 6 jweile jweile     4096 Mar 13 11:48 ../
-rw-r--r-- 1 jweile jweile    39783 Mar 12 16:08 counts_sample_103.csv
-rw-r--r-- 1 jweile jweile    41449 Mar 12 16:08 counts_sample_104.csv
-rw-r--r-- 1 jweile jweile    31431 Mar 12 16:08 counts_sample_105.csv
-rw-r--r-- 1 jweile jweile    35578 Mar 12 16:08 counts_sample_106.csv
-rw-r--r-- 1 jweile jweile     8831 Mar 12 16:08 counts_sample_107.csv
-rw-r--r-- 1 jweile jweile    10656 Mar 12 16:08 counts_sample_108.csv
-rw-r--r-- 1 jweile jweile   145330 Mar 12 16:08 counts_sample_10.csv
-rw-r--r-- 1 jweile jweile   147215 Mar 12 16:08 counts_sample_12.csv
-rw-r--r-- 1 jweile jweile   329760 Mar 12 16:08 counts_sample_13.csv
...
```

The first step will then be to use `joinCounts.R`. It will automatically detect both the parameter file and the mutation call directory.

```
$ joinCounts.R ./
```

```
2020-03-12_18:40:24 INFO: Reading parameters
2020-03-12_18:40:24 INFO: Accounting for all input count files
2020-03-12_18:40:24 INFO: Reading count data
2020-03-12_18:40:25 INFO: Exporting sequencing depth information.
2020-03-12_18:40:28 INFO: Translating 333914 unique variant calls to protein level. This may take some
2020-03-12_18:54:30 INFO: Merging tables...
2020-03-12_18:54:30 INFO: Processing condition nonselect timepoint 1
2020-03-12_18:54:32 INFO: Processing condition select timepoint 1
2020-03-12_18:54:34 INFO: Processing condition wtCtrl timepoint 1
2020-03-12_18:54:37 INFO: Writing results to file.
2020-03-12_18:54:40 INFO: Calculating marginal frequencies
2020-03-12_18:55:14 INFO: Writing results to file.
2020-03-12_18:55:14 INFO: Done.
```

As a result, we now have three new files CSV in the `mut_call` folder, containing the full counts, marginal counts and sequencing depth summary.

```
$ ll 2020-03-11-00-16-59_mut_call

total 70812
drwxrwxr-x 4 jweile jweile     4096 Mar 13 15:41 ./
drwxrwxr-x 6 jweile jweile     4096 Mar 13 11:48 ../
-rw-r--r-- 1 jweile jweile 58078880 Mar 12 18:54 allCounts.csv
-rw-r--r-- 1 jweile jweile  3739904 Mar 12 18:55 marginalCounts.csv
-rw-r--r-- 1 jweile jweile     9158 Mar 12 18:40 sampleDepths.csv
...
```

Now, we can run the scoring function:

```
$ runScoring.R ./

2020-03-12_19:32:13 INFO: Reading parameters
2020-03-12_19:32:13 INFO: Scoring function uses the following parameters:
2020-03-12_19:32:13 INFO: countThreshold =  10
2020-03-12_19:32:13 INFO: pseudoReplicates (pseudo.n) =  8
2020-03-12_19:32:13 INFO: sdThreshold =  0.3
2020-03-12_19:32:13 INFO: Selecting latest data directory:  .//2020-03-11-00-16-59_mut_call
2020-03-12_19:32:14 INFO: Processing selection select ; time point 1
2020-03-12_19:32:14 INFO: Processing region 1
2020-03-12_19:32:14 INFO: Fitting error models for each tile
2020-03-12_19:32:16 INFO: Performing error regularization
2020-03-12_19:32:17 INFO: Filtering...
2020-03-12_19:32:17 INFO: Scoring...
2020-03-12_19:32:17 INFO: Normalizing...
2020-03-12_19:32:17 INFO: Processing region 2
2020-03-12_19:32:17 INFO: Fitting error models for each tile
2020-03-12_19:32:18 INFO: Performing error regularization
2020-03-12_19:32:19 INFO: Filtering...
2020-03-12_19:32:19 INFO: Scoring...
2020-03-12_19:32:19 INFO: Normalizing...
2020-03-12_19:32:19 INFO: Processing region 3
2020-03-12_19:32:19 INFO: Fitting error models for each tile
2020-03-12_19:32:21 INFO: Performing error regularization
2020-03-12_19:32:21 INFO: Filtering...
2020-03-12_19:32:21 INFO: Scoring...
2020-03-12_19:32:21 INFO: Normalizing...
```

```
2020-03-12_19:32:21 INFO: Processing region 4
2020-03-12_19:32:22 INFO: Fitting error models for each tile
2020-03-12_19:32:23 INFO: Performing error regularization
2020-03-12_19:32:24 INFO: Filtering...
2020-03-12_19:32:24 INFO: Scoring...
2020-03-12_19:32:24 INFO: Normalizing...
2020-03-12_19:32:24 INFO: Apply formatting...
2020-03-12_19:32:24 INFO: Writing full table to file.
2020-03-12_19:32:24 INFO: Writing simplified table to file.
2020-03-12_19:32:24 INFO: Collapsing amino acid changes...
2020-03-12_19:32:24 INFO: Writing AA-centric table to file.
2020-03-12_19:32:24 INFO: Scoring complete.
```

As a result we now see a new subdirectory with the score tables:

```
$ ll
total 20136
drwxrwxr-x  6 jweile jweile     4096 Mar 13 11:48 ./
drwxr-xr-x 14 jweile jweile     4096 Mar 13 14:33 ../
drwxrwxr-x  4 jweile jweile     4096 Mar 13 15:41 2020-03-11-00-16-59_mut_call/
drwxr-xr-x  2 jweile jweile     4096 Mar 13 11:44 2020-03-11-00-16-59_scores/
-rw-r--r--  1 jweile jweile     2218 Mar 12 18:55 joinCounts.log
-rw-r--r--  1 jweile jweile     7242 Mar 12 17:12 parameters.json
-rw-r--r--  1 jweile jweile     3685 Mar 12 19:32 scoring.log

$ ll 2020-03-11-00-16-59_scores/
total 9520
drwxr-xr-x 2 jweile jweile    4096 Mar 13 11:44 ./
drwxrwxr-x 6 jweile jweile    4096 Mar 13 11:48 ../
-rw-r--r-- 1 jweile jweile 8479512 Mar 12 19:32 select_t1_complete.csv
-rw-r--r-- 1 jweile jweile    3781 Mar 12 19:32 select_t1_errorModel.csv
-rw-r--r-- 1 jweile jweile  391915 Mar 12 19:32 select_t1_simple_aa.csv
-rw-r--r-- 1 jweile jweile  674856 Mar 12 19:32 select_t1_simple.csv
```