RNA-MATE user manual

(preliminary documentation)

Version 1.01
March 2009

Contact:
Nicole Cloonan
n.cloonan@expressiongenomics.org
Institute for Molecular Bioscience
The University of Queensland
St Lucia, QLD, 4072

License:

Table of Contents

The RNA-MATE pipeline

General Description

For mammalian genomes, there are technical challenges associated with mapping and counting short-tag sequences derived from high-throughput sequencing data. Firstly, mammalian transcripts are non-contiguous due to the splicing of introns from the pre-mRNA. This means that there will be a portion of tags that cross exon-exon boundaries that will not map directly to the genome. The ability to use short tag information relies directly upon being able to place short tags uniquely within the genome. The presence of genome wide repeats and other repetitive sequence in the mouse and human genomes mean that a sizeable proportion of short tags can not be placed uniquely. Finally, the random fragmentation of mRNA creates a distribution of sizes, of which a significant proportion will be less than the full length of the tag, and these will contain adaptor sequence that will not map to the genome. Here we present a computational pipeline to map RNAseq data, which generates both tag counting and genome-browser visualization of genomic and exon-junction matching results. RNA-MATE (Mapping and Alignment Tool for Expression) is designed for the rapid mapping of data from the Applied Biosystems SOLiD system (Figure 1).

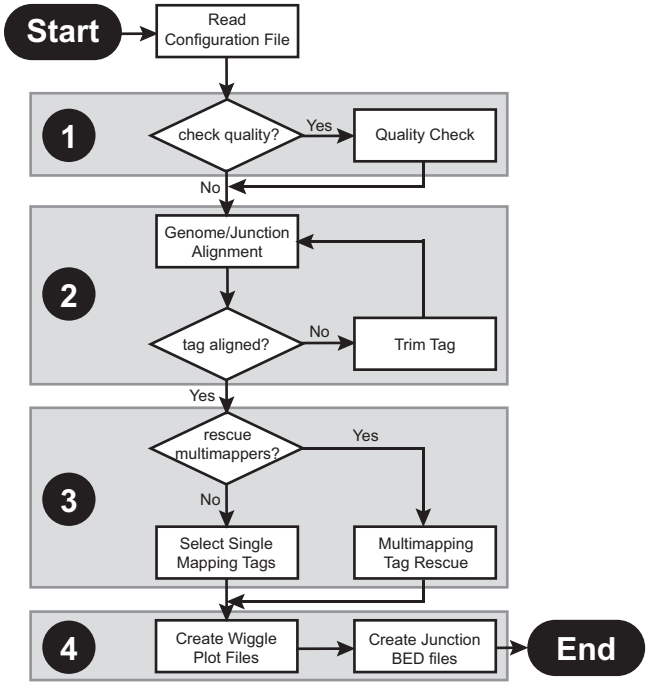

Figure 1. The RNA-MATE recursive mapping pipeline. The pipeline consists of 4 major components. (1) The optional tag quality module filters tags based on the quality values for each basecall. (2) The alignment module attempts to align tags first to the genome, and then to a library of known exon-junction sequences. If a tag fails to align, then the tag is truncated, and the process is repeated. (3) The optional tag rescue module is uses information derived from both single-mapping and multi-mapping tags to uniquely place multi-mapping tags. (4) Finally, UCSC genome browser compatible wiggle plots and BED files are generated.

Part 1: Quality checking of the tag (optional)

Depending on the downstream applications of the matched data the quality of individual tags may need to be assessed before their inclusion in the mapping pipeline. To accommodate this, we have provided an optional tag quality module which assesses the tags by the number of basecalls with PHRED scores of less than 10. Tags that pass the QC are fed into the recursive alignment module. If this option is disabled, all tags are passed to the alignment module.

Part 2: Recursive alignment to the human or mouse genome

Alignment of the short tags to a reference genome is done using mapreads (http://solidsoftwaretools.com/gf/project/mapreads/), an algorithm specifically designed for the rapid mapping of data from the Applied Biosystems SOLiD system (ie. color-space data). Tags are first matched against all chromosomes of the reference genome, and then against a library of known exon-junctions (hg18 and mm9 are currently supported). Tags that fail to map to the genome or junctions are chopped to user defined lengths, and the genomic mapping is restarted. In this way, tags that have adaptor sequence, or poor quality ends are recovered at their longest length. The number of mismatches between the reference and tag is user defined, and when mappings from all tags are collated into a single file, only the mappings at the highest level of stringency are retained.

Part 3: Multi-mapping tag rescue (optional)

For most downstream applications, tags are only informative if they can be placed uniquely within a genome. Tags that align to multiple places within a genome make up a sizeable proportion of transcriptome derived tags, primarily from the inherent redundancy of the genome, but also from CpG islands and genome wide repeat elements. Strategies to rescue ambiguous sequences have recently been applied to high-throughput sequencing data, and we have refined our previously published algorithm to work efficiently with large data sets. For every multi-mapping tag, the algorithm considers all tags that map near to each of the possible locations of the tag (within a user-specified window) to determine the most likely mapping position of the tag. Where a tag can not be unambiguously assigned, a fractional weighting to the relevant positions is assigned. In practice, between 40-60% of multi-mapping tags can be assigned a single position with $\geq$60% likelihood, depending on the relative sequence coverage. The recommended window size for shotgun sequencing is 10 (Cloonan, et al., 2008), whereas the window recommended for CAGE data and other disparate data sets is X (ref?).

Part 4: Creation of visualization files

Finally, UCSC genome browser compatible wiggle plots for genome mapped data, and BED files for exon-junction mapped data are generated automatically from the collated results. The wiggle plots are strand-specific, single-nucleotide resolution coverage plots, and directly represent the number of times an individual nucleotide has been seen in the sequencing data. BED files depict hits to junction sequences, and graphically display exon combinatorics. In addition, plots containing only start sites of tags are included to facilitate tag-counting applications.

Availability

All source code, documentation, and associated files described in this manual are freely available for download from:

http://grimmond.imb.uq.edu.au/RNA-MATE/

or

http://solidsoftwaretools.com/gf/project/rnamate/

Requirements

This pipeline is written predominantly in perl (with some python thrown in for good measure), and requires that you have version 5.8.8 of perl or later, and python version 2.4 or later. It is designed to run in a unix environment, with a PBS queue manager. The scripts can be modified to work with an LSF or SGE manager. It is not recommended to run this pipeline on a system without access to a cluster due to the large computational requirements of mapping to mammalian genomes – however, the scripts could potentially be modified to do this.

You will need to install the ForkManager.pm perl module if you do not already have it, as well as Path-Class-0.16. Both are available from CPAN.

The alignment section of this pipeline is dependant upon the mapreads tool, available from:

http://solidsoftwaretools.com/gf/project/mapreads/.

Installation

Simply unzip the tarball and add the path of the installation directory to @INC using the command:

```
export PERL5LIB=${PERL5LIB}:/[full path]/RNA-MATEv1.0/perl/
```

This can be added to the ~/.bash_profile or ~/.profile files for automatic loading, or it can be added to the default profile for all users. The script **mask_schemas_mapreads.pl** should be placed in the same folder as the mapreads program.

Scripts

Master script: rna-mate-v1.01.pl

This script will call the required modules in order. There is only one user-defined parameter for this script, which allows you to specify a configuration file containing all the required parameters for the entire mapping pipeline.

To run this script, use the following command:

```
[path]/rna-mate-v1.01.pl –c [configuration file]
```

Configuration file

The configuration file is a text file containing all the required parameters to run RNA-MATE. In this file, directory listings must end with a "/", there must be no other punctuation at the end of the lines, and there should be no empty lines in this file. An example of the configuration file is given below:

```
max_length_tag=35
tag_length=35,30
num_mismatch=3
mask=111111111111111111111111111111111
max_multimatch=10
expect_strand=+
rescue_window=10
exp_name=tag_20000_F3
chromosomes=chrM,chr2
chr_path=/data/matching/hg18_fasta/
junction=/data/matching/libraries/hg18_junctions_best_quality.fas
ta.cat
junction_index=/data/matching/libraries/hg18_junctions_best_quali
ty.fasta.index
output_root=/data/cxu/
output_dir=/data/cxu/tag_20000_F3/
raw_qual=/data/raw/tag20000.qual
raw_csfasta=/data/raw/tag20000.csfasta
status_out=/data/cxu/tag_20000_F3/total_rep2/map_status.out
raw_csfasta=/data/cxu/tag_20000_F3/total_rep2.csfasta
email=bob@bobstown.com
run_rescue=false
num_parallel_rescue=4
quality_check=true
script_chr_start=/data/matching/chr_start.pl
script_chr_wig=/data/matching/chr_wig.pl
f2m=/data/matching/f2m.pl
mapreads=/data/matching/mapreads
rescue=/data/matching/chr_rescueSOLiD.py
master_script=/data/matching/rna-mate-v1.01.pl
```

Configuration options

`max_length_tag=35`

This parameter defines the longest length of the tags contained in the csfasta file.

`tag_length=35,30`

This parameter defines the lengths at which matching will occur recursively. Lengths should be in multiples of 5nt. The minimum recommended length for transcriptome data is 30nt although this will depend on how many mismatches you allow. For up to 3 mismatches to a mammalian genome, the minimum length should be 30nt because even though you still get a large proportion of single mapping tags at this length, the specificity is very poor. If you allow up to 1 mismatch, then you can still achieve good sensitivity at 25nt. For other smaller genomes, the minimum acceptable matches should be determined on a case by case basis.

`num_mismatch=3`

The number of mismatches permissible between the tag and the reference sequence. Currently this pipeline does not support the "valid adjacent error" feature of mapreads. This upgrade is planned for a future release.

NOTE: Mapping schemas must be available to do the mapping at the specified length and number of mistmatches, or else the pipeline will fail. ie. in this example, the schemas required are:

```
schema_35_3
schema_30_3
```

Mapping schemas are available from http://solidsoftwaretools.com/.

`mask=1111111111111111111111111111111111`

This setting allows you to ignore particular bases in the tag when computing the number of mismatches. 1 = consider this base, 0 = do not consider this base. The length of the mask should equal the length of the longest tags.

`max_multimatch=10`

Defines the maximum number of positions to be reported for multi-mapping tags. The higher this number, the more disk space is required to store the data, and the slower the program will run. Recommended size for most applications is 10.

expect_strand=+

This defines the strandedness of the data. For example, libraries made with the SREK protocol or other direct ligation protocols will have tags that are sequenced in the sense (+) strand relative to the expressed gene. Libraries made with the SQRL protocol will have tags that are sequenced in the antisense (-) relative to the expressed gene.

rescue_window=10

This parameter defines the window size used for multi-map tag rescue. The recommended setting for shotgun sequencing data is 10, whereas the recommended setting for CAGE and other disparate data sets is 100.

exp_name=tag_20000_F3

Set the experiment name with this parameter.

chromosomes=chrM,chr2

Defines the names of the chromosomes to map against. The filenames are expected to be:

```
[chromosome_name].fa
```

chr_path=/data/matching/hg18_fasta/

The full path of the chromosome fasta files.

junction=/data/matching/libraries/hg18_junction.fasta.cat

The full path of the junction library against which you can map.

junction_index=/data/matching/libraries/hg18_junction.fasta.index

The full path of the junction index file.

output_root=/data/cxu/
output_dir=/data/cxu/tag_20000_F3/

The full paths of the output root and output directories.

raw_qual=/data/raw/tag20000.qual

The full path of the QV file.

raw_csfasta=/data/raw/tag20000.csfasta

The full path of the csfasta file to be matched.

status_out=/data/cxu/tag_20000_F3/total_rep2/map_status.out

Not used in this implementation of RNA-MATE.

email=bob@bobstown.com

Not used in this implementation of RNA-MATE.

run_rescue=true

This parameter allows you to turn on or off the rescue of multi-mapping tags module. Acceptable values are "true" or "false". True = run multi-map rescue, false = do not run multi-map rescue.

NOTE: multi-map rescue can be a very memory intensive process. Rescue for a single chromosome of a transcriptome dataset with > 100 million mappable tags can consume more than 20 Gb of resident memory. The amount of memory used will depend on the size of the data set, the number of multi-mapping tags versus single mapping tags, the underlying complexity of the data set, and the number of positions of each tag to be rescued.

num_parallel_rescue=4

This parameter allows you to adjust the number of rescue jobs that are run in parallel. The settings chosen here will depend on the amount of memory available on your system, the number of CPUs available, and the amount of memory consumed by the rescue (see the note above regarding multi-mapping tag rescue and memory usage).

quality_check=true

This parameter allows you to turn on or off the quality checking of tags module. Acceptable values are "true" or "false". True = run quality check, False = do not run quality check.

script_chr_start=/data/matching/chr_start.pl
script_chr_wig=/data/matching/chr_wig.pl
f2m=/data/matching/f2m.pl
mapreads=/data/matching/mapreads
rescue=/data/matching/chr_rescueSOLiD.py
master_script=/data/matching/ rna-mate-v1.01.pl

These parameters define the full path showing the location of the various scripts required to run RNA-MATE.

Modules

`tools_mapping.pm`

This module includes four functions: creating log files; checking whether the jobs on the queue are finished; creating new csfasta files"; and chopping tags for recursive mapping.

`tag_quality.pm`

This module checks tag quality, making sure that each tag contains less then five nucleotides where the QV value for that basecall is less than 10 (PHRED scale). Currently this threshold is hardcoded. Future implementations will allow user defined values at this point.

`mapping.pm`

This module automatically arranges genome and junction mapping for different tag lengths.

`single_selection.pm`

This module attempts to select a single mapping position for each tag based on the mapping results at the highest stringency. For example, if a tag maps once with zero mismatches, and 3 times with one mismatch, then the tag is recorded as a single mapping tag at a stringency of zero mismatches.

`new_rescue.pm`

This module use new version rescue program which can parallel rescue for each chromosome and use less memory.

`wiggle_plot.pm:`

This module creates strand specific wiggle plot (or bedGraph) files for visualization in the UCSC genome browser. This module also creates "start site plots" which facilitates tag counting applications.

`UCSC_junction.pm.`

This module creates BED files for displaying exon-junction usage in the UCSC genome browser.

Log File

tag_20000_F3.log

This is an example of the output log file for the tag_20000_F3 experiment. Each status output includes two lines, the first line is system time and the second is what the system doing at that time.

```
Mon Nov 17 13:45:41 2008
[PROCESS]: Welcome to our mapping strategy system!
Mon Nov 17 13:45:42 2008
[SUCCESS]: Created csfasta file for different tag length, in which
tag quality is checked!
Mon Nov 17 13:45:42 2008
[PROCESS]: mapping to all chromosomes
Mon Nov 17 13:45:42 2008
waiting for queue
Mon Nov 17 14:04:42 2008
[SUCCESS]: mapped to all chromosomes
Mon Nov 17 14:04:42 2008
[PROCESS]: collating genome mers:35
Mon Nov 17 14:04:42 2008
[SUCCESS]: collated genome mers:35
Mon Nov 17 14:04:42 2008
[PROCESS]: mapping to junction
Mon Nov 17 14:10:42 2008
[SUCCESS]: mapped to junction
Mon Nov 17 14:10:42 2008
[PROCESS]: collating junction mers:35
Mon Nov 17 14:10:42 2008
[SUCCESS]: collated junction mers:35
Mon Nov 17 14:10:42 2008
[PRPCESS]: chopping tag
Mon Nov 17 14:10:42 2008
[PROCESS]: mapping to all chromosomes
Mon Nov 17 14:10:42 2008
waiting for queue
Mon Nov 17 14:28:42 2008
[SUCCESS]: mapped to all chromosomes
Mon Nov 17 14:28:42 2008
[PROCESS]: collating genome mers:30
Mon Nov 17 14:28:42 2008
[SUCCESS]: collated genome mers:30
Mon Nov 17 14:28:42 2008
[PROCESS]: mapping to junction
Mon Nov 17 14:33:42 2008
[SUCCESS]: mapped to junction
Mon Nov 17 14:33:42 2008
[PROCESS]: collating junction mers:30
Mon Nov 17 14:33:42 2008
[SUCCESS]: collated junction mers:30
Mon Nov 17 14:33:42 2008
[PROCESS]: rescue multi mapped tags
Mon Nov 17 14:33:42 2008
```

```
[SUCCESS]: rescue tags are done!
Mon Nov 17 14:33:42 2008
[PROCESS]: prepare data for wiggle plot...
Mon Nov 17 14:33:42 2008
[SUCCESS]: prepared data file for parallel wig plot.
Mon Nov 17 14:33:42 2008
[SUCCEED]: all done! enjoy the data!
```

Module inputs and outputs

This section details the input and output files generated from each of the modules in this pipeline for the tag_20000_F3 experiment with the configuration file as above.

tools_mapping.pm

sub create_csfata

input:
tag20000_F3.csfasta
output:
tag_20000_F3.mers35.unique.csfasta
tag_20000_F3.mers30.unique.csfasta

mapping.pm (35mers)

sub genomic_mapping (35)

input:
tag_20000_F3.mers35.unique.csfasta
output:
chr2.tag_20000_F3.mers35.unique.csfasta.ma.35.3
chrM.tag_20000_F3.mers35.unique.csfasta.ma.35.3

sub collate_genomic_matches (35)

input:
chr*.mers35.*.3
output:
tag_20000_F3.mers35.genomic.collated
tag_20000_F3.mers35.genomic.non_matched

sub junction_mapping (35)

input:
tag_20000_F3.mers35.genomic.non_matched
output:
hg18_junctions_best_quality.tag_20000_F3.mers35.genomic.non_matched.ma.35.3

sub collate_junction_matches (35)

input:
hg18_junctions_best_quality.tag_20000_F3.mers35.genomic.non_matched.ma.35.3
output:
tag_20000_F3.mers35.junction.non_matched

tools_mapping.pm

sub chop_tag

input:
tag_20000_F3.mers35.junction.non_matched
tag_20000_F3.mers30.unique.csfasta
output:
tag_20000_F3.mers30.unique.csfasta

mapping.pm (30mers)

sub genomic_mapping (30)

input:
tag_20000_F3.mers30.unique.csfasta
output:
chr2.tag_20000_F3.mers30.unique.csfasta.ma.30.3
chrM.tag_20000_F3.mers30.unique.csfasta.ma.30.3

sub collate_genomic_matches (30)

input:
chr*30.*.3
output:
tag_20000_F3.mers30.genomic.collated
tag_20000_F3.mers30.genomic.non_matched

sub junction_mapping (30)

input:
tag_20000_F3.mers30.genomic.non_matched
output:
hg18_junctions_best_quality.tag_20000_F3.mers30.genom
ic.non_matched.ma.30.3

sub collate_junction_matches (30)

input:
hg18_junctions_best_quality.tag_20000_F3.mers30.genom
ic.non_matched.ma.30.3
output:
tag_20000_F3.mers30.junction.non_matched

single_select.pm

input:
tag_20000_F3.mers35.genomic.collated
tag_20000_F3.mers30.genomic.collated
output:
tag_20000_F3.mers30.genomic.stats
tag_20000_F3.mers35.genomic.stats
chr2.tag_20000_F3.for_wig.negative
chr2.tag_20000_F3.for_wig.positive
chrM.tag_20000_F3.for_wig.negative
chrM.tag_20000_F3.for_wig.positive

wiggle_plot.pm

sub paralle_wig_fork

input:
chr2.tag_20000_F3.for_wig.negative
chr2.tag_20000_F3.for_wig.positive

chrM.tag_20000_F3.for_wig.negative
chrM.tag_20000_F3.for_wig.positive
output:
chr2.tag_20000_F3.for_wig.negative.sorted
chr2.tag_20000_F3.for_wig.negative.wig
chr2.tag_20000_F3.for_wig.negative.wig.success
chr2.tag_20000_F3.for_wig.positive.sorted
chr2.tag_20000_F3.for_wig.positive.wig
chr2.tag_20000_F3.for_wig.positive.wig.success
chrM.tag_20000_F3.for_wig.negative.sorted
chrM.tag_20000_F3.for_wig.negative.wig
chrM.tag_20000_F3.for_wig.negative.wig.success
chrM.tag_20000_F3.for_wig.positive.sorted
chrM.tag_20000_F3.for_wig.positive.wig
chrM.tag_20000_F3.for_wig.positive.wig.success

sub start_plot_fork
input:
chr2.tag_20000_F3.for_wig.negative
chr2.tag_20000_F3.for_wig.positive
chrM.tag_20000_F3.for_wig.negative
chrM.tag_20000_F3.for_wig.positive
output:
chr2.tag_20000_F3.for_wig.negative.starts
chr2.tag_20000_F3.for_wig.negative.starts.success
chr2.tag_20000_F3.for_wig.positive.starts
chr2.tag_20000_F3.for_wig.positive.starts.success
chrM.tag_20000_F3.for_wig.negative.starts
chrM.tag_20000_F3.for_wig.negative.starts.success
chrM.tag_20000_F3.for_wig.positive.starts
chrM.tag_20000_F3.for_wig.positive.starts.success
tag_20000_F3.negative.starts
tag_20000_F3.positive.starts

sub collect_data
input:
chr2.tag_20000_F3.for_wig.negative.starts
chr2.tag_20000_F3.for_wig.positive.starts
chr2.tag_20000_F3.for_wig.negative.wig
chr2.tag_20000_F3.for_wig.positive.wig
chrM.tag_20000_F3.for_wig.negative.wig
chrM.tag_20000_F3.for_wig.positive.wig
chrM.tag_20000_F3.for_wig.negative.starts
chrM.tag_20000_F3.for_wig.positive.starts
output:
tag_20000_F3.negative.wiggle
tag_20000_F3.positive.wiggle

UCSC_junction.pm

sub single_selection (35)
input:
hg18_junctions_best_quality.tag_20000_F3.mers35.genomic.non_matched.ma.35.3

output:
tag_20000_F3.junction35.negative.stats

tag_20000_F3.junction35.positive.stats
tag_20000_F3.junction35.single_map.negative
tag_20000_F3.junction35.single_map.positive

sub search_junctionID (positive file)
input:
tag_20000_F3.junction35.single_map.positive
output:
tag_20000_F3.junction35.single_map.positiveID
sub search_junctionID (negative file)
input:
tag_20000_F3.junction35.single_map.negative
output:
tag_20000_F3.junction35.single_map.negativeID

sub single_selection (30)
input:
hg18_junctions_best_quality.tag_20000_F3.mers30.genom
ic.non_matched.ma.30.3
output:
tag_20000_F3.junction30.negative.stats
tag_20000_F3.junction30.positive.stats
tag_20000_F3.junction30.single_map.negative
tag_20000_F3.junction30.single_map.positive

sub search_junctionID (positive file)
input:
tag_20000_F3.junction30.single_map.positive
output:
tag_20000_F3.junction30.single_map.positiveID

sub search_junctionID (negative file)
input:
tag_20000_F3.junction30.single_map.negative
output:
tag_20000_F3.junction30.single_map.negativeID

sub create_BED (positive file)
input:
tag_20000_F3.junction30.single_map.positiveID
tag_20000_F3.junction35.single_map.positiveID
output:
tag_20000_F3.positive.junction.BED

sub create_BED (negative file)
input:
tag_20000_F3.junction30.single_map.negativeID
tag_20000_F3.junction35.single_map.negativeID
output:
tag_20000_F3.negative.junction.BED

Modifying the pipeline to work with other queues

In order to make this program compatible with other queue managers the mapping.pm module will need to be edited. Specifically, lines in the genome_mapping and junction_mapping subroutines that contain:

```
$comm = "qsub -l walltime=48:00:00,ncpus=2 -o $mysh.out -e
$mysh.err $mysh > $mysh.id ";
```

will need to be replaced with the appropriate job submission commands and parameters that are specific to your system.

Till Bayer (from the MPI for Evolutionary Biology in Germany) has kindly provided instructions on modifying this script to work on SGE systems. The line above should be changed to:

```
$comm = "qsub -l s_rt=48:00:00,s_cpu=2 -o $mysh.out -e $mysh.err
$mysh > $mysh.id";
```

In addition to modifying the lines above, lines 76 and 133 which read:

```
print OUT $comm;
```

should be changed to include the "#$/bin/sh" line, and a newline after the actual command needs to be inserted.

Optimizing performance on your cluster

The script as written asks for two CPUs per mapping job. As mapreads is not parallelized, this is an inefficient, but necessary throttle if you are running an NFS file transfer protocol. The entire pipeline (including mapreads) is very I/O intensive, and depending on the setup, users may find that NFS will timeout if too much is asked of it. For systems using less archaic protocols this will not be necessary, and the script can be modified to request a single CPU.