RNA-MATE user manual

(preliminary documentation)

Version 1.1
July 2009
Contact:

rna-mate@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).

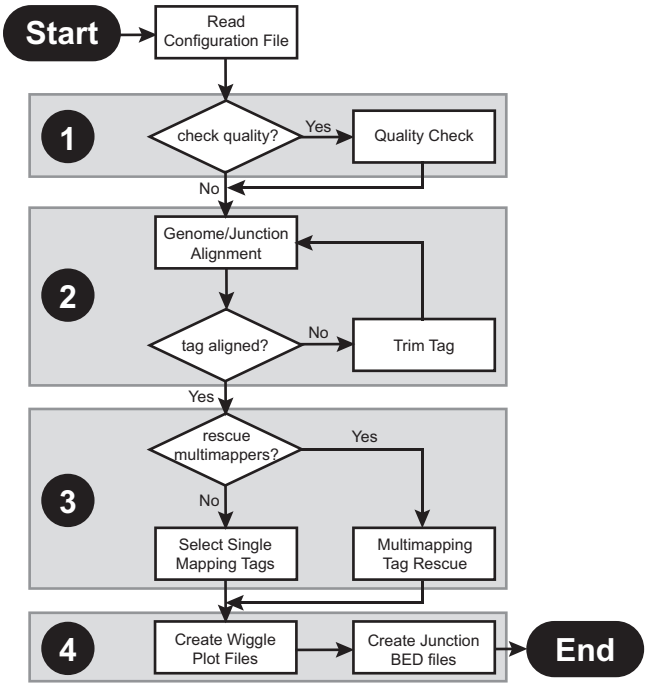


Figure 1. The RNA-MATE recursive mapping pipeline. The pipeline consists of 4 major components. (1) The optional tag quality module filters tags based on the quality values for each basecall. (2) The alignment module attempts to align tags first to the genome, and then to a library of known exon-junction 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 ≥60% likelihood, depending on the relative sequence coverage. The recommended window size for shotgun sequencing is 10 (Cloonan, et al., 2008 Nat Methods **5**, 613-619.), though for disparate data types currently available this can vary. For instance, Cap Analysis of Gene Expression (CAGE) tags are rescued using a window of 100 nt, a size previously shown to optimize mammalian promoter detection (Carninci*, et al.*, 2006, Nat Genet, **38**, 626-635).

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/ (requires registration, which is free).

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. This tool and its installation instructions are available from:

http://solidsoftwaretools.com/gf/project/mapreads/. requires registration, which is free).

Finally, you will need a genome against which to map. The program expects one file per chromosome, with the filename format as "[chromosome_name].fa". Genomes can be downloaded from the UCSC genome browser website at:

http://hgdownload.cse.ucsc.edu/downloads.html

Installation instructions

The instructions given below in courier font are examples of the commands needed to carry out the installation.

1. Move the tarball to the destination directory and decompress it.

```
eg. mv RNA-MATEv1.1.tar.gz /data/
eg. tar –xzf RNA-MATEv1.1.tar.gz
```

2. Add the path of the installation directory to @INC using the command:

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

Where **[full path]** should be replaced with the path of the RNA-MATE directory.

```
eg. export PERL5LIB=${PERL5LIB}:/data/RNA-MATEv1.1/perl/
```

This command can be added to the ~/.bash_profile or ~/.profile files (depending on the shell) for automatic loading, or it can be added to the default profile for all users.

3. Place the script **mask_schemas_mapreads.pl** in the same folder as the mapreads program.

```
eg. mv mask_schemas_mapreads.pl /data/mapreads/
```

Testing RNA-MATE

To ensure that your downloads are functioning correctly, please download the testing data available and testing results available from:

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

Please also download the hg18 version of the human genome, available from:

http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/chromFa.zip

The testing data includes:

- test_500K_tags_50mers.conf (the test configuration file)
- test_500K_tags_50mers.csfasta (the SOLiD testing data)

The testing results folder includes the wiggle plots and junction BED files that were generated from the testing data on our system:

- test_500K_tags_50mers.expect.junction.BED (junction BED file for UCSC genome browser)
- test_500K_tags_50mers.unexpect.junction.BED (antisense matches to junctions)
- test_500K_tags_50mers.junction30.SIM.negativeID (antisense junction matches for 30nt tags)
- test_500K_tags_50mers.junction30.SIM.positiveID (sense junction matches for 30nt tags)
- test_500K_tags_50mers.junction35.SIM.negativeID (antisense junction matches for 35nt tags)
- test_500K_tags_50mers.junction35.SIM.positiveID (sense junction matches for 35nt tags)
- test_500K_tags_50mers.junction40.SIM.negativeID (antisense junction matches for 40nt tags)
- test_500K_tags_50mers.junction40.SIM.positiveID (sense junction matches for 40nt tags)
- test_500K_tags_50mers.junction45.SIM.negativeID (antisense junction matches for 45nt tags)
- test_500K_tags_50mers.junction45.SIM.positiveID (sense junction matches for 45nt tags)
- test_500K_tags_50mers.junction50.SIM.negativeID (antisense junction matches for 50nt tags)
- test_500K_tags_50mers.junction50.SIM.positiveID (sense junction matches for 50nt tags)
- test_500K_tags_50mers.mers30.genomic.collated (all genomic matches for 30nt tags)
- test_500K_tags_50mers.mers35.genomic.collated (all genomic matches for 35nt tags)
- test_500K_tags_50mers.mers40.genomic.collated (all genomic matches for 40nt tags)
- test_500K_tags_50mers.mers45.genomic.collated (all genomic matches for 45nt tags)

- test_500K_tags_50mers.mers50.genomic.collated (all genomic matches for 50nt tags)
- test_500K_tags_50mers.negative.starts (wiggle plot of tag start sites for the –ve strand)
- test_500K_tags_50mers.negative.wiggle (wiggle plot for the –ve strand)
- test_500K_tags_50mers.positive.starts (wiggle plot of tag start sites for the +ve strand)
- test_500K_tags_50mers.positive.wiggle (wiggle plot for the +ve strand)

Edit the configuration file so that it refers to the appropriate directories that you have set up on your system. See the "Configuration Options" section for more details on what each of the parameters does. Run the script, use the following command:

```
nohup [path]/rna-mate-v1.1.pl –c test_500K_tags_50mers.conf &
```

Where **[path]** is the full path to rna-mate-v1.1.pl.

```
eg. nohup /data/rna-mate-v1.1.pl –c test_500K_tags_50mers.conf &
```

Testing will run RNA-MATE on the SOLiD test data in approximately 3 hours (using 3 Blades, each with 16GB of RAM and 2 Dual-Core AMD Opteron(tm) Processor 2218 (4 cores), running RedHat Linux 2.6.18-92.1.17.el5 x86_64). The wiggle plots and BED junction files should be compared to those provided in the test_results folder, to ensure that the pipeline is working as expected.

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:

```
nohup [path]/rna-mate-v1.1.pl –c [configuration file] &
```

Where **[path]** is the full path to rna-mate-v1.1.pl, and **[configuration file]** is the name (and full path, if not in the current working directory) of the 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:

```
raw_tag_length=50
mapping_parameters=50.5.0,45.5.0,40.5.0,35.3.0,30.3.0
mask=111111111111111111111111111111111111111111111111
max_multimatch=10
expect_strand=+
exp_name=test_500K_tags_50mers
chromosomes=chrM,chr2
chr_path=/data/matching/hg18_fasta/
junction_50=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.45.fa.cat
junction_50_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.45.fa.index
junction_45=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.40.fa.cat
junction_45_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.40.fa.index
junction_40=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.35.fa.cat
junction_40_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.35.fa.index
junction_35=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.cat
junction_35_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.index
junction_30=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.25.fa.cat
junction_30_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.25.fa.index
output_dir=/data/RNA-MATEv1.1/test_results/
raw_csfasta=/data/RNA-MATEv1.1/test_data/test_500K_tags_50mers.csfasta
raw_qual=/data/RNA-MATEv1.1/test_data/test_500K_tags_50mers.qual
quality_check=false
run_rescue=false
rescue_window=10
num_parallel_rescue=2
script_chr_wig=/data/RNA-MATEv1.1/chr_wig.pl
script_chr_start=/data/RNA-MATEv1.1/chr_start.pl
f2m=/data/RNA-MATEv1.1/f2m.pl
mapreads=/data/matching/mapreads
rescue=/data/RNA-MATEv1.1/chr_rescueSOLiD.py
master_script=/data/RNA-MATEv1.1/rna-mate-v1.1.pl
```

Configuration options

raw_tag_length=35

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

mapping_parameters=50.5.0,45.5.0,40.5.0,35.3.0,30.3.0

These parameters define the lengths at which matching will occur recursively, the number of mismatches permissible between the tag and the reference sequence, and whether or not to treat valid adjacent errors as a single mismatch. These are comma separated parameters, with the format of **[length].[mismatches].[valid_adjacent]**. **[length**] defines the length of the tag to match at, **[mismatches]** defines the number of mismatches allowed, and **[valid_adjacent]** is set to 1 if valid-adjacent errors are to be treated as a single mismatch, or 0 if they are not.

In the above example, the recursive mapping will match at lengths 50, 45, 40, 35, and 30. For lengths 50-40, there will be 5 mismatches allowed, whereas for lengths 35 and 30 will allow only 3 mismatches. Valid-adjacent errors are not treated as a single mismatch. For a discussion on selecting optimal parameters for analysis see the section "Selecting appropriate parameters".

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_50_5
schema_45_5
schema_40_5
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. For interrogating repeat sequences (such as retrotransposable elements) this value may need to be set higher.

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.

exp_name=test_500K_tags_50mers

Set the experiment name with this parameter.

chromosomes=chrM,chr2

Defines the names of the chromosomes to map against. The filenames that match to these chromosomes are expected to be named as **[chromosome_name]**.fa. In the above example, the files needed to match against should be called:

```
chrM.fa
chr2.fa
```

These should be in a standard single sequence fasta format. There is no requirement for the header line to contain any particular string. Only the filename and the chromosome name will be used in the pipeline.

chr_path=/data/matching/hg18_fasta/

The full path of the chromosome fasta files.

junction_50=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.40.fa.cat
junction_50_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.40.fa.index
junction_45=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.35.fa.cat
junction_45_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.35.fa.index
junction_40=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.cat
junction_40_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.index
junction_35=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.cat
junction_35_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.30.fa.index
junction_30=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.25.fa.cat

junction_30_index=/data/RNA-MATEv1.1/junction_libraries/hg18.junctions.25.fa.index

These parameters define the junction libraries and their associated index files that are used in RNA-MATE. The ability to specify the length of the junction library used for each of the different lengths of the tag means that you can have complete control over the stringency of the junction matching. In this example, by using the 40mer libraries (40nt from the donor exon, concatenated with 40nt from the acceptor exon) with the 50-40mer tag lengths we are requiring a minimum on 10nt of the tag to overlap the exon-exon boundary. For the 35-30mer tag lengths we are requiring an overlap of 5nt. For obvious reasons, the number of nucleotides overlapping should be greater than the number of mismatches allowed in the tag.

For every tag length specified in the mapping_parameters option, there are two files required, the junction_**[length]** file asks for the concatenated junction library, and the junction_**[length]**_index file asks for the file that decodes the concatenated fasta. Junction libraries for human and mouse genomes can be downloaded from:

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

output_dir=/ data/RNA-MATEv1.1/test_results/

The full path of the output directory. This directory must exist prior to running the pipeline.

raw_qual=/data/raw/tag20000.qual

The full path of the file containing the QV file.

raw_csfasta=/data/raw/tag20000.csfasta

The full path of the csfasta file to be matched.

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-v1.1.pl

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

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.

Selecting appropriate parameters

Understanding the two major parameters (the number of mismatches allowed for at every tag length, and the number of nucleotides chopped at each iteration), as well as the smallest mappable size for the genome, is critical to maximizing the efficiency and accuracy of the recursive mapping strategy.

How many mismatches to allow at each length is critical to both the speed and accuracy of tag mapping. The more mismatches allowed, the slower the program will run, however a low number of mismatches may fail to capture mappable tags with sequencing artifacts. Additionally, large numbers of mismatches relative to the tag length will create spurious matching events, and increase the level of noise in your results. For RNAseq data, ideally, the proportion of tags mapping exons should be relatively constant, regardless of the length, and studying this for your genome of interest will provide guidelines as to what levels of mismatching is acceptable for your system. For mouse and human genomes (and presumably, other mammalian genomes), we recommend to use 3 mismatches for lengths from 30-39nt, 5 mismatches for lengths ≥40nt. If additional matching data is required, 25nt matches can be used, but caution should be

used when interpreting the results. Either allow only a single mismatch to ensure specificity of mapping, or filter the final wiggle plots (eg. only look at nucleotide positions that are covered by more than 4 tags) to an extent which removes the noise in this mapping (see Figure 2).

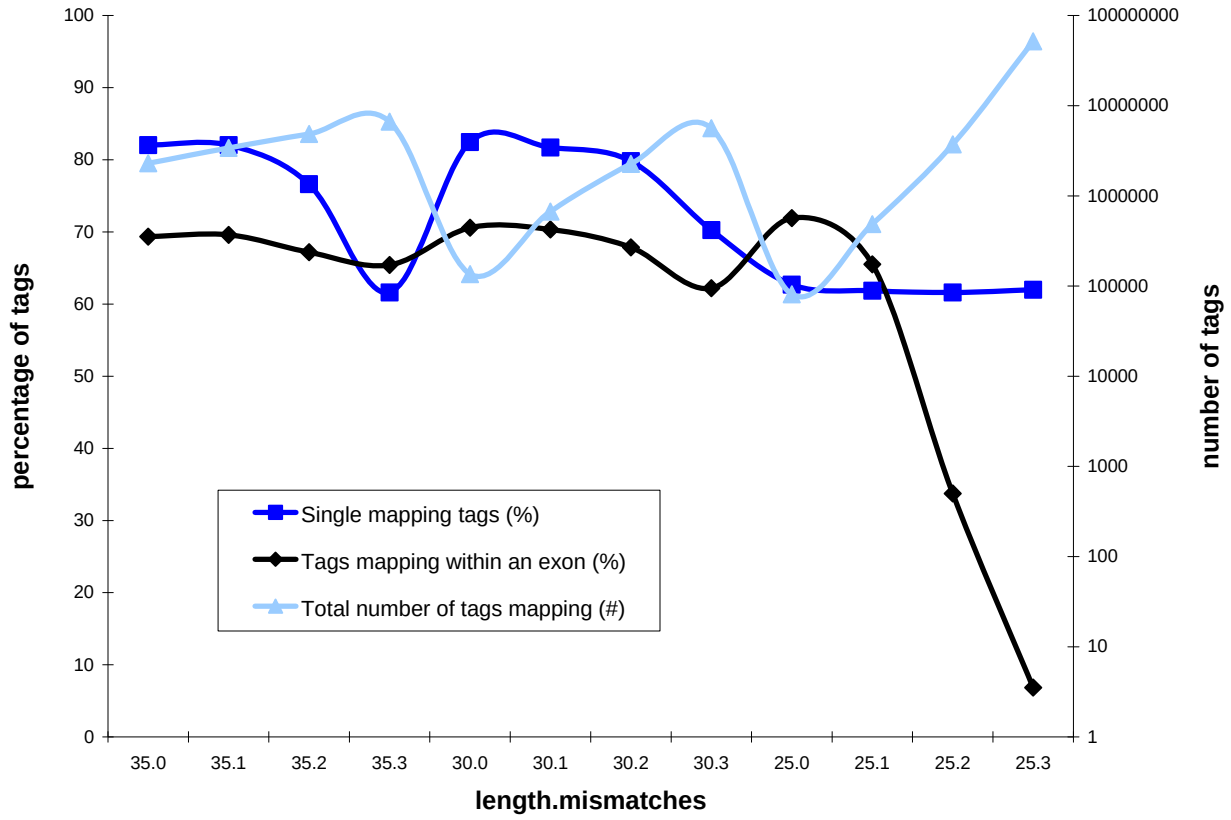


Figure 2. Effect of length and mismatches on the specificity of mapping tags. For each length and mismatch combination, the proportion of tags that map uniquely within the human genome is indicated by the dark blue line (left Y-axis). The black line indicates the proportion of tags that map to exons, and shows a steep decline at 25.3, indicating a drop in specificity at this length (left Y-axis). The light blue line shows the total number of tags mapping at a given length and mismatch combination, showing a sharp increase in the number of mapping tags at 25.3 (right Y-axis; logarithmic scale).

To minimize the computational time used for this approach, the number of nucleotides to be chopped at each iteration should be greater than or equal to the number of mismatches allowed at any iteration. We typically remove 5nt at a time for SOLiD sequencing data for two reasons. First, the sequencing chemistry of SOLiD is performed with five different primers, and the number of cycles will determine the length of the tag (in multiples of five). For example, to generate a 50mer sequence, the data of ten cycles from each of the five primers is added together. Typically, each cycle has roughly the same error profile as the corresponding cycles from other primers (ie. the third cycle on primer one will have the same error rate as the third cycle on primer 5), and the error rate increases as the cycle number increases. This means that typically, error rates jump in multiples of 5nt, so excluding 5nt at a time will minimize the effect of sequencing error on the mapping results. Obviously, this consideration does not apply for non-SOLiD data sets. Secondly, as every iteration takes CPU time, the more iterations that are done the higher the cost of the additional mapping tags. Larger iterations decrease the run time, but they also decrease the

sensitivity of the strategy to detect tags that lie across exon-junctions. Figure 3 shows the effect of alternate mapping strategies on the computational time and the proportion of mappable tags based on two different scenarios (an ideally sized mRNA library, and a smaller than ideal mRNA library).

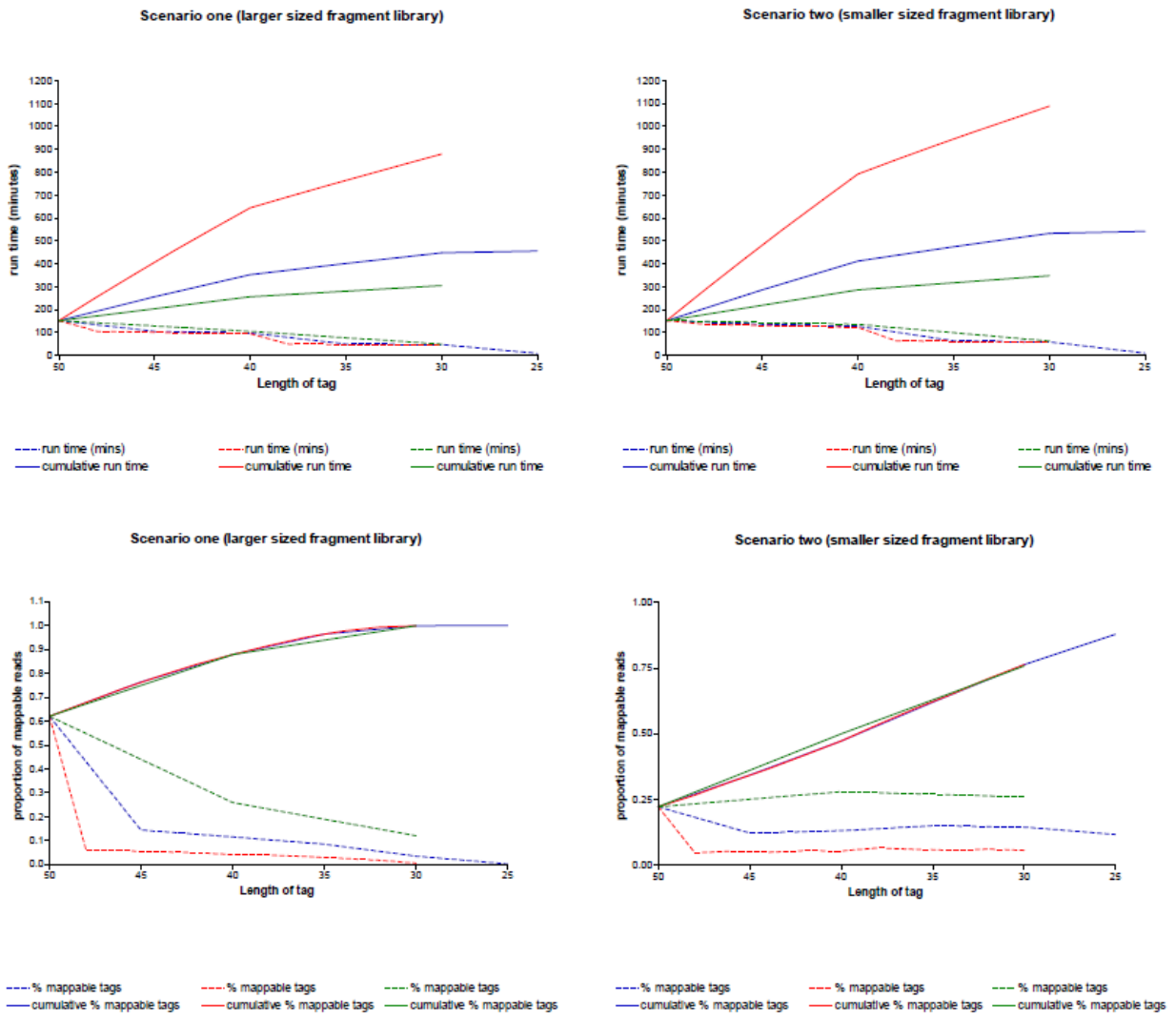


Figure 3. How alternate mapping strategies affect the yield of mappable tags and the computational run time. In all graphs, red lines represent Strategy 1 (2nt interations, 50nt-30nt), blue lines represent Strategy 2 (5nt interations, 50nt-25nt), and green lines represent Strategy 3 (10nt iterations, 50nt-30nt). In all scenarios, 5 mismatches were allowed for tag lengths ranging from 50nt to 40nt, 3 mismatches were allowed for tag lengths ranging from 39nt to 30nt, and 1 mismatch was allowed for tag lengths ranging from 29nt to 25nt. Scenario One is a fragment library with a mode insert size of 54nt; Scenario Two is the same library with the insert size shifted to 39nt. Together, these graphs show that there is more benefit for a recursive strategy when the library insert size is smaller than ideal.

The efficiency of the recursive strategy is largely dependant on the median insert size of the RNA fragments. If all fragments are longer than the read length of the tag, then the recursive strategy would only additionally map the 5' end of novel splice junctions, and those with poor quality 3' ends. Depending on the individual sample and sequencing run, this may or may not yield sufficient additional mapping tags to justify the additional computational time.

In contrast, assuming perfect adaptor identification and an ideal sized fragment library, a vector clipping method would use less than 40% of the CPU time than the recursive method, for the same yield of mapping tags. Unfortunately, even under the best circumstances, adaptor identification is not perfect, and there are additional technical challenges for adaptor identification in color-space. Typically, adaptor identification and chopping will yield (under the least stringent conditions), approximately 60% of the tags that will map under a recursive method. On top of this, the SOLiD system can use "Internal Adaptor Blockers" which prevent the ligation of sequencing probes to that region. This causes a drop in accurate base calling (which is not just based on the quality of base calls), and under these circumstances, successful adaptor identification can drop to just over 20%. Whilst these blockers were an optional reagent in SOLiD V2 chemistry, they are now premixed (and therefore not optional) on the V3 plates.

Ideally, neither a recursive method nor an adaptor identification method would be required at all, if we could ensure that the RNA fragment size was always going to be larger than the read size. For microRNA populations, where the mode size is approximately 22nt, this is simply not possible. Due to technical limitations on the maximum insert size possible in an emulsion PCR (ePCR), and the strong amplification bias of small fragments in ePCR, this is not always achievable for fragmented RNA libraries either, even those that have been size selected prior to ePCR.

The primary motivation behind the recursive mapping method was to maximize the number of mapping tags from every sequencing run. The cost of sequencing reagents is considerably more than the cost of server time, so gaining additional depth (between about 1.6 and 3 times the tags mapping at the longest length) represents good value for money. In this respect, it is up to the individual user to decide whether or not to apply a recursive mapping strategy for their own analysis.

How to map to the genome and junctions simultaneously

RNA-MATE has the flexibility to other approaches to junction matching. In some circumstances, one may wish to consider matches to the junction library at the same time as matches to the genome. This can be done by renaming the junction library against which you wish to map to follow the chromosome naming convention (see "Configuration options"). We will sometimes use "chrJ" as the chromosome name (and therefore "chrJ.fa" as the filename). Additionally, you will need to provide a small junction-library file (eg. a string of 35 Ns), as it must go through the process of matching to the junction library, even though there is nothing in it to match to. This is not an efficient or fully automatic process as junction BED files will need to be recreated from the raw mapping files – see "Post RNA-MATE scripts". This inefficiency will be rectified in the next major release of the software, where we will make the hierarchical junction mapping step optional.

How to use RNA-MATE to perform non-recursive mapping

Although designed for recursive mapping, RNA-MATE can also map at a single tag length if preferred. This will speed up the analysis in situations where maximum sequencing depth is not required. To do this, simply adjust the **`mapping_parameters`** option to the length of tags desired.

```
eg. mapping_parameters=50.5.0
```

Modules

`tools_RNA.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.

`RNA_mapping.pm`

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

`single_select.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 uses the new version of the 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

test_500K_tags_50mers.log

This is an example of the output log file for the test_500K_tags_50mers experiment. Each status output includes two lines, the first line is system time and the second is what the system doing at that time.

```
Thu May 28 12:16:57 2009
[PROCESS]: Welcome to our mapping strategy system!
Thu May 28 12:22:02 2009
[SUCCESS]: Created csfasta file for tag with different tag length, in which we
ignore tag quality!
Thu May 28 12:22:03 2009
[PROCESS]: mapping to all chromosomes
 Thu May 28 12:22:09 2009
[SUCCESS]: mapped to all chromosomes
 Thu May 28 12:22:09 2009
[PROCESS]: collating genome mers:35
Thu May 28 22:11:54 2009
[SUCCESS]: collated genome mers:35
Thu May 28 22:11:54 2009
[PROCESS]: mapping to junction (mers 35.2.1)
Thu May 28 22:11:54 2009
[SUCCESS]: mapped to junction (mers 35.2.1)
Thu May 28 22:11:54 2009
[PROCESS]: single selecting junction position & ID! (mers 35)
Thu May 28 22:25:42 2009
[SUCCESS]: single selected junction position & ID (mers 35)
Thu May 28 22:25:42 2009
[PRPCESS]: chopping tag
Thu May 28 22:33:48 2009
[PROCESS]: mapping to all chromosomes
 Mon Jun  1 14:58:04 2009
[SUCCESS]: mapped to all chromosomes
 Mon Jun  1 14:58:04 2009
[PROCESS]: collating genome mers:30
Tue Jun  2 02:56:07 2009
[SUCCESS]: collated genome mers:30
Tue Jun  2 02:56:07 2009
[PROCESS]: mapping to junction (mers 30.2.1)
Tue Jun  2 04:21:10 2009
[SUCCESS]: mapped to junction (mers 30.2.1)
Tue Jun  2 04:21:10 2009
[PROCESS]: single selecting junction position & ID! (mers 30)
Tue Jun  2 04:38:16 2009
[SUCCESS]: single selected junction position & ID (mers 30)
Tue Jun  2 04:38:16 2009
[PRPCESS]: chopping tag
Tue Jun  2 04:52:46 2009
[PROCESS]: mapping to all chromosomes
 Tue Jun  9 12:58:27 2009
[SUCCESS]: mapped to all chromosomes
 Tue Jun  9 12:58:28 2009
[PROCESS]: collating genome mers:25
Tue Jun  9 23:28:09 2009
[SUCCESS]: collated genome mers:25
Tue Jun  9 23:28:09 2009
[PROCESS]: mapping to junction (mers 25.2.1)
Wed Jun 10 00:48:10 2009
```

```
[SUCCESS]: mapped to junction (mers 25.2.1)
Wed Jun 10 00:48:10 2009
[PROCESS]: single selecting junction position & ID! (mers 25)
Wed Jun 10 00:58:18 2009
[SUCCESS]: single selected junction position & ID (mers 25)
Wed Jun 10 00:58:19 2009
[PROCESS]:   combine  data  from  different  tag  length,  and  then  classify  into
different strand and chromosome
 Wed Jun 10 01:20:25 2009
[PROCESS]:  Creating files for wiggle plot!
Wed Jun 10 04:43:31 2009
[PROCESS]:  Creating start file for wiggle plot!
Wed Jun 10 04:44:58 2009
[SUCCESS]:  Created start file:
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.positive.starts
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.negative.starts
 Wed Jun 10 04:46:17 2009
[SUCCESS]:  Created wiggle plot file:
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.positive.wiggle
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.negative.wiggle
 Wed Jun 10 04:46:17 2009
[PROCESS]: Creating BED file for junction mapping!
Wed Jun 10 04:46:24 2009
[SUCCESS]: Created junction BED files
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.expect.junction.BED
 /data/RNA-MATEv1.1/test_results/test_500K_tags_50mers.unexpect.junction.BED
Wed Jun 10 04:46:24 2009
[SUCCESS]: all done! enjoy the data!
```

Checking the finished run

Misconfigured config files or interruptions to the server or queue can cause RNA-MATE to die prematurely, however there is some error catching code that can help you to work out what has gone wrong. The steps you should go through to ensure that the run has finished successfully are listed below.

1. Check the log file. Look for [WARNING] or [DIED] messages that will describe what has gone wrong.

   ```
   eg. grep WARNING test_500K_tags_50mers.log | more
   ```

2. Check the nohup.out file in the directory where you ran RNA-MATE from. A clean nohup.out file should contain only messages that look like:

   ```
   found file /data/RNA-
   MATEv1.1/test_results/hg18_junctions_20.test_500K_tags_50mers.genomic.non_mat
   ched.ma.25.2.adj_valid.success
   sleeped 80 * 60 seconds
   ```

   which simply indicate that the mapping jobs have been found. Other messages will indicate errors in the pipeline.

3. Check that the mapping was completed successfully for each chromosome. The mapreads output (see “Description of output files”) should be at least the same file size as the original csfasta input. A smaller file represents a failed mapreads run for that chromosome. These mappings can

be regenerated by submitting the appropriate shell script (*.sh) to the queue manager, and then restarting the pipeline.

4. Check that the final visualization files are present (see next section for a description of the final output files). The "expect" junction BED file should be much larger than the "unexpect" BED file. The positive and negative files for both "starts" and "wiggle" should be roughly the same size. The final size of the files will depend on the size of the run being analyzed, but for a single slide of SOLiD data you might expect the file sizes to be in the order of 100-300MB.

5. Finally, check that the files upload into the UCSC genome browser without errors. To minimize file size, the wiggle plots and bed tracks can be gzipped. Sometimes the wiggle/starts files are too large to upload even when gzipped, and you may need to apply a post-pipeline filter to the results (see "Post RNA-MATE scripts").

Description of output files

RNA-MATE produces a large number of files in the specified output directory, most of which are temporary working files and can be deleted. Unless the total storage space on your cluster is an issue, it is probably best to wait until the pipeline has finished before deleting files. This allows you to re-enter the mapping pipeline at different points, without needing to start the mapping process from scratch. This can save significant amounts of time in the even of a power outage or similar computational catastrophe. This section takes you through the inputs and outputs generated by running RNA-MATE on the test data, the contents of the output files generated from each of the modules, and whether or not these should be stored or deleted. The bold headings are the perl modules that are called by RNA-MATE. The italicized headings refer to the subroutines within each of the perl modules, and the bracketed information gives examples of the paramaters that are given to this subroutine. Some perl modules are called multiple times (such as RNA_mapping.pm) and particular attention should be paid to the length at which the subroutine has been run (especially if insufficient resources are available and files must be deleted mid-run).

tools_RNA.pm

sub create_csfasta("/data/test_500K_tags_50mers.csfasta")

input:
test_500K_tags_50mers.csfasta
output:
test_500K_tags_50mers.mers35.unique.csfasta
test_500K_tags_50mers.mers30.unique.csfasta

These files contain your original tag sequences and IDs in csfasta format. These are temporary internal files and can be deleted when the genomic_mapping subroutine of the RNA_mapping module has completed successfully.

RNA_mapping.pm (eg. 35mers)

sub genomic_mapping (my $para = [35,3,0])

input:
test_500K_tags_50mers.mers35.unique.csfasta
output:
chr2.test_500K_tags_50mers.mers35.unique.csfasta.ma.35.3
chrM.test_500K_tags_50mers.mers35.unique.csfasta.ma.35.3

These files are the mapreads output for all tags to each chromosome as indicated in the file name. The format is identical to the csfasta input format, except that matches are stored in a comma separated manner after the tag ID in the header line. They are temporary internal files and can be deleted as soon as the genome_collation subroutine of the RNA_mapping module has been completed successfully.

sub genome_collation (my $para = [35,3,0])

input:
chr*.mers35.*.3
output:
test_500K_tags_50mers.mers35.genomic.collated
test_500K_tags_50mers.mers35.genomic.non_matched

The collated file contains all of the genomic matches for the tags, identified by chromosome. The file format is:

[tag ID]<tab>**[number of hits to the genome]**<tab>**[matches]**<newline>
[Colourspace tag]<newline>

The tag **[matches]** themselves are tab delimited, zero based, and are in the format:

[chromosome].[strand][genomic location].[number of mismatches]

eg. chr1.-13000056.2 is a match at chromosome 1, position 13000056 on the negative strand with two mismatches.
eg. chrX_random.64522.0 is a match at chromosome X random, position 64522 on the positive strand, with zero mismatches.

Note that tag matches will only be reported for those where the total number of matches (second column) is less than or equal to the number specified with the max_multimatch parameter.

The collated files are very useful, particularly for saving time on the regeneration of wiggle plots or other analyses, and we strongly recommend that all collated files are stored with the final output files.

The non_matched files are a csfasta file containing the tags that did not match to the genomic locations. This is a temporary internal file, and can be deleted after the junction_mapping subroutine of the RNA_mapping module has been completed successfully.

```
sub junction_mapping (my $para = [35,3,0])
```

```
input:
test_500K_tags_50mers.mers35.genomic.non_matched
output:
hg18_junctions_best_quality.test_500K_tags_50mers.mers35.genomic.n
on_matched.ma.35.3
```

These files are the mapreads output for all tags to the junction library specified in the configuration file. The format is identical to the csfasta input format, except that matches are stored in a comma separated manner after the tag ID in the header line. They are temporary internal files and can be deleted as soon as the junction_selection subroutine of the RNA_mapping module has been completed successfully.

```
sub junction_selection (my $para = [35,3,0])
```

```
input:
hg18_junctions_best_quality.test_500K_tags_50mers.mers35.genomic.n
on_matched.ma.35.3
output:
test_500K_tags_50mers.mers35.junction.non_matched
```

The non_matched file is a csfasta file containing the tags that did not match to the junction library specified in the configuration file. This is a temporary internal file, and can be deleted after the chop_tag subroutine of the tools_RNA module has been completed successfully.

```
test_500K_tags_50mers.junction35.negative.stats
test_500K_tags_50mers.junction35.positive.stats
```

These stats files show, for every tag, the number of times a tag matched (in total, and at each level of mismatching) to the junction library specified in the configuration file. The file format is as follows:

>**[tag ID]**<tab>**[total matches]**<tab>**[number of 0 mismatches]**<tab>**[number of 1 mismatches]**<tab>**[number of 2 mismatches]**<tab>**[number of 3 mismatches]**<newline>

These are not required for any further steps in RNA-MATE and can therefore be deleted if they are not required for your analysis.

```
test_500K_tags_50mers.junction35.SIM.negative
test_500K_tags_50mers.junction35.SIM.positive
```

These are temporary internal files that specify the start, end, and number of mismatches for each of the tags against the specified exon junction library. These can be deleted as soon as the junction_selection subroutine of the RNA_mapping module has been completed successfully.

```
test_500K_tags_50mers.junction35.SIM.negativeID
test_500K_tags_50mers.junction35.SIM.positiveID
```

The ID files contain all of the junction library matches for the tags. The file format is:

>**[tag ID]**<tab>**[start coordinate of match]**<tab>**[end coordinate of match]**<tab>**[number of mismatches]**<tab>**[junction ID]**<tab>**[start coordinate of junction]**<tab>**[end coordinate of junction]**<newline>

The ID files are very useful, particularly for saving time on the regeneration of junction BED tracks or other analyses, and we strongly recommend that all ID files are stored with the final output files.

tools_RNA.pm

*sub chop_tag("*test_500K_tags_50mers.mers35.junction.non_matched*",*
*"*test_500K_tags_50mers.mers30.unique.csfasta*",5)*

```
input:
test_500K_tags_50mers.mers35.junction.non_matched
test_500K_tags_50mers.mers30.unique.csfasta
output:
test_500K_tags_50mers.mers30.unique.csfasta
```

These files contain the trimmed tag sequences and IDs in csfasta format. These are temporary internal files and can be deleted when the genomic_mapping subroutine of the RNA_mapping module has completed successfully.

single_select.pm

sub main

```
input:
test_500K_tags_50mers.mers35.genomic.collated
test_500K_tags_50mers.mers30.genomic.collated
output:
test_500K_tags_50mers.mers30.genomic.stats
test_500K_tags_50mers.mers35.genomic.stats
```

These stats files show, for every tag, the number of times a tag matched (in total, and at each level of mismatching) to the genome as specified in the configuration file. The file format is as follows:

>[tag ID]<tab>[total matches]<tab>**[number of 0 mismatches]**<tab>**[number of 1 mismatches]**<tab>**[number of 2 mismatches]**<tab>**[number of 3 mismatches]**<newline>

These are not required for any further steps in RNA-MATE and can therefore be deleted if they are not required for your analysis.

```
chr2.test_500K_tags_50mers.for_wig.negative
chr2.test_500K_tags_50mers.for_wig.positive
chrM.test_500K_tags_50mers.for_wig.negative
chrM.test_500K_tags_50mers.for_wig.positive
```

These are temporary internal files and can be deleted after the successful completion of the wiggle_plot module.

wiggle_plot.pm

sub parallel_wig_fork

input:

```
chr2.test_500K_tags_50mers.for_wig.negative
chr2.test_500K_tags_50mers.for_wig.positive
chrM.test_500K_tags_50mers.for_wig.negative
chrM.test_500K_tags_50mers.for_wig.positive
```

output:

```
chr2.test_500K_tags_50mers.for_wig.negative.sorted
chr2.test_500K_tags_50mers.for_wig.negative.wig
chr2.test_500K_tags_50mers.for_wig.negative.wig.success
chr2.test_500K_tags_50mers.for_wig.positive.sorted
chr2.test_500K_tags_50mers.for_wig.positive.wig
chr2.test_500K_tags_50mers.for_wig.positive.wig.success
chrM.test_500K_tags_50mers.for_wig.negative.sorted
chrM.test_500K_tags_50mers.for_wig.negative.wig
chrM.test_500K_tags_50mers.for_wig.negative.wig.success
chrM.test_500K_tags_50mers.for_wig.positive.sorted
chrM.test_500K_tags_50mers.for_wig.positive.wig
chrM.test_500K_tags_50mers.for_wig.positive.wig.success
```

These are temporary internal files and can be deleted after the successful completion of the wiggle_plot module.

sub start_plot_fork

input:

```
chr2.test_500K_tags_50mers.for_wig.negative
chr2.test_500K_tags_50mers.for_wig.positive
chrM.test_500K_tags_50mers.for_wig.negative
chrM.test_500K_tags_50mers.for_wig.positive
```

output:

```
chr2.test_500K_tags_50mers.for_wig.negative.starts
chr2.test_500K_tags_50mers.for_wig.negative.starts.success
chr2.test_500K_tags_50mers.for_wig.positive.starts
chr2.test_500K_tags_50mers.for_wig.positive.starts.success
chrM.test_500K_tags_50mers.for_wig.negative.starts
chrM.test_500K_tags_50mers.for_wig.negative.starts.success
chrM.test_500K_tags_50mers.for_wig.positive.starts
chrM.test_500K_tags_50mers.for_wig.positive.starts.success
```

These are temporary internal files and can be deleted after the successful completion of the wiggle_plot module.

```
test_500K_tags_50mers.negative.starts
test_500K_tags_50mers.positive.starts
```

These are the final output files containing the genomic locations corresponding to the start sites of each mapped tag, which can be loaded into the UCSC genome browser. These files should be stored.

```
sub collect_data
        input:
        chr2.test_500K_tags_50mers.for_wig.negative.starts
        chr2.test_500K_tags_50mers.for_wig.positive.starts
        chr2.test_500K_tags_50mers.for_wig.negative.wig
        chr2.test_500K_tags_50mers.for_wig.positive.wig
        chrM.test_500K_tags_50mers.for_wig.negative.wig
        chrM.test_500K_tags_50mers.for_wig.positive.wig
        chrM.test_500K_tags_50mers.for_wig.negative.starts
        chrM.test_500K_tags_50mers.for_wig.positive.starts
        output:
        test_500K_tags_50mers.negative.wiggle
        test_500K_tags_50mers.positive.wiggle
```

These are the final output files containing the single nucleotide resolution coverage plots (wiggle) plots, which can be loaded into the UCSC genome browser. These files should be stored.

UCSC_junction.pm

```
sub main()
        inputs:
        test_500K_tags_50mers.junction30.SIM.positiveID
        test_500K_tags_50mers.junction35.SIM.positiveID
        test_500K_tags_50mers.junction30.SIM.negativeID
        test_500K_tags_50mers.junction35.SIM.negativeID
        outputs:
        test_500K_tags_50mers.expect.junction.BED
        test_500K_tags_50mers.unexpect.junction.BED
```

These are the final output files containing the matches to the exon-junction libraries in a BED format which can be loaded into the UCSC genome browser.

For more information on BED or wiggle (bedGraph) file formats, please see the UCSC genome browser website at: http://genome.ucsc.edu.

Re-entering the RNA-MATE pipeline

There may be some occasions where you might wish to regenerate the wiggles or BED tracks from the collated files. For example, you may have initially generated the wiggles without multi-mapping rescue, but now you wish to generate them with rescue turned on. Rather than remapping, you can simply modify your config file (eg. rescue=true) and reenter the pipeline at the rescue stage using the command:

```
nohup ./restart_at_rescue.pl –c test_500K_tags_50mers.conf &
```

To make use of this feature, you must keep the collated files, and the junction ID files (see "Description of output files").

Modifying the pipeline to work with other queues

In order to make this program compatible with other queue managers the RNA_mapping.pm module will need to be edited. Specifically, line 332 in the mapping subroutine that contains:

```
$comm="qsub -l walltime=48:00:00 -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.

For SGE

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 -o $mysh.out -e $mysh.err $mysh > $mysh.id";
```

In addition to modifying the lines above, line 325 which reads:

```
print OUT $comm;
```

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

For LSF

Xuanzhong Li (from the Children's Hospital Boston, USA) has kindly provided instructions on modifying this script to work on LSF systems. Any instances of the "qsub" command need to be replaced with the "bsub" command. All other parameters appear to work fine for both PBS and LSF queue managers.

Optimizing performance on your cluster

The entire RNA-MATE pipeline (including mapreads) is very I/O intensive, and depending on the cluster setup, users may find that it needs to be modified for optimal performance. For example, those people using NFS filesystems may find that NFS will timeout if too much is asked of it. For these systems, an inefficient but necessary throttle may be to request two or more CPUs per mapping job. This can be done by modifying line 322 as follows (changed text is in red):

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

Post-RNA-MATE scripts

Filtering wiggle plots

This script is for filtering and reducing the size of the wiggle plots (bedGraphs) to be uploaded into the UCSC genome browser.

```
Usage: ./filter_bedGraphs.pl

       REQUIRED:
       -f name of file to be filtered
       -m minimum number of tags to report
```

For example, to remove the data from all nucleotides where there isn't at least 5 tags covering them, use the command:

```
./filter_bedGraphs.pl –f test_500K_tags_50mers.negative.wiggle –m 5
```

Assessing the specificity of mapping

In order to examine the specificity of the mapping by the directionality of the library, this script can be used to examine the junction BED files generated by RNA-MATE. The "sense" strand matches will be your "expected" strand BED file, and the "antisense" matches will be your "unexpected" strand BED file.

```
Usage: ./assess_junctions_for_directionality.pl

        REQUIRED:
        -s name of BED file containing sense matches
        -a name of BED file containing antisense matches
        -o name of outfile
```

For example, to assess the output of the test data set, use the command:

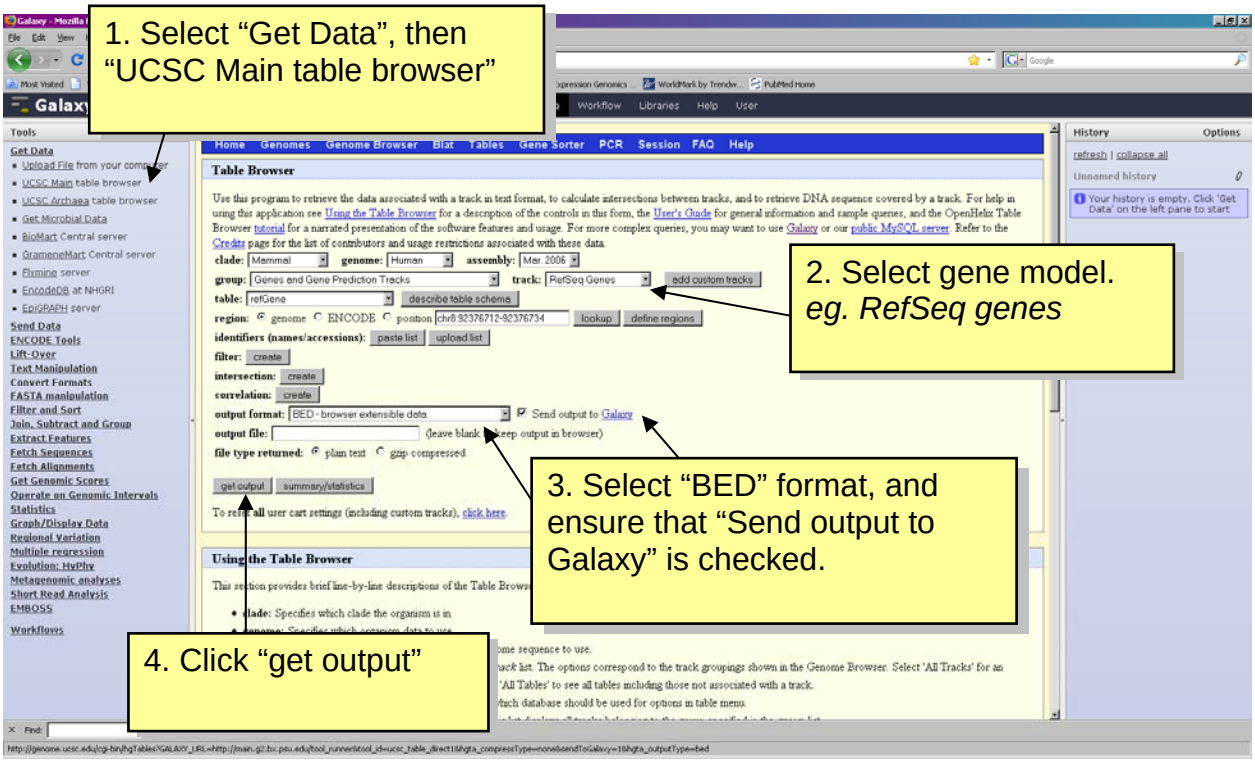
```
./assess_junctions_for_directionality.pl –s test_500K_tags_50mers.expect.junction.BED
–a test_500K_tags_50mers.unexpect.junction.BED –o test.output
```

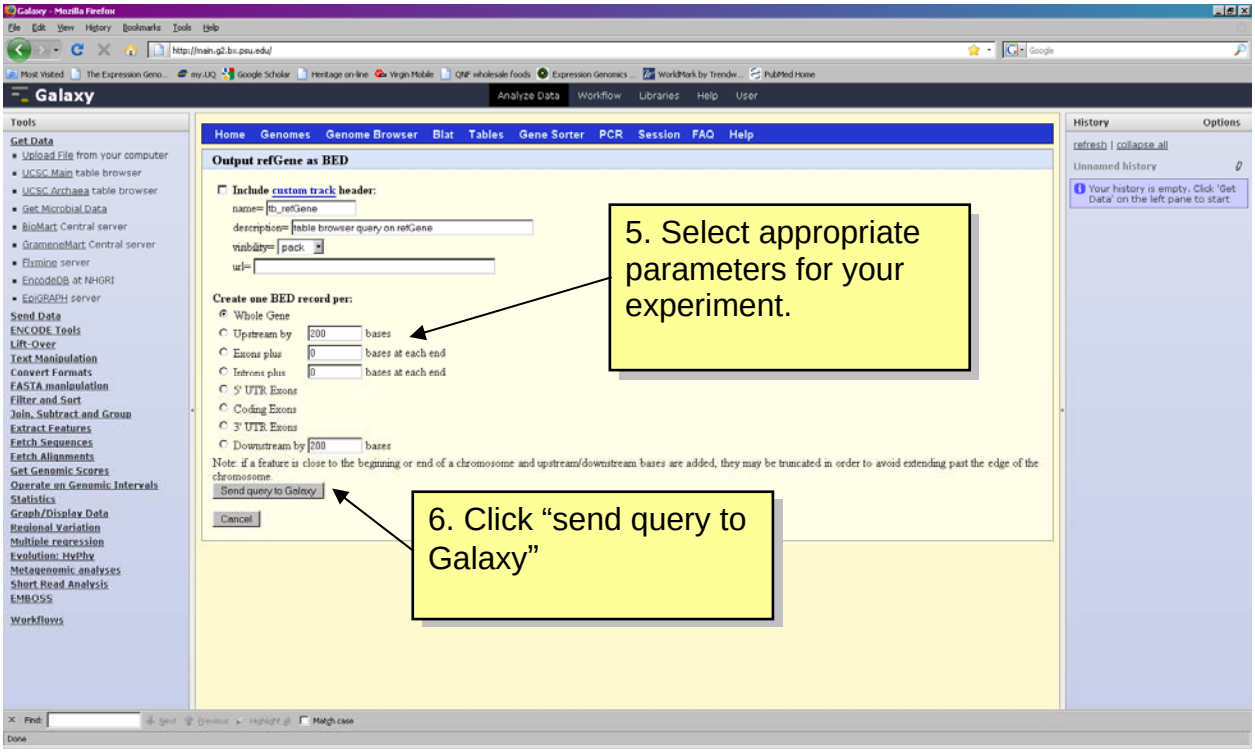
Ideally, more than 99.5% of tags should be in the expected strand. A lower value indicates a problem with the mapping parameters used or (less likely) a problem with the cDNA library generation.

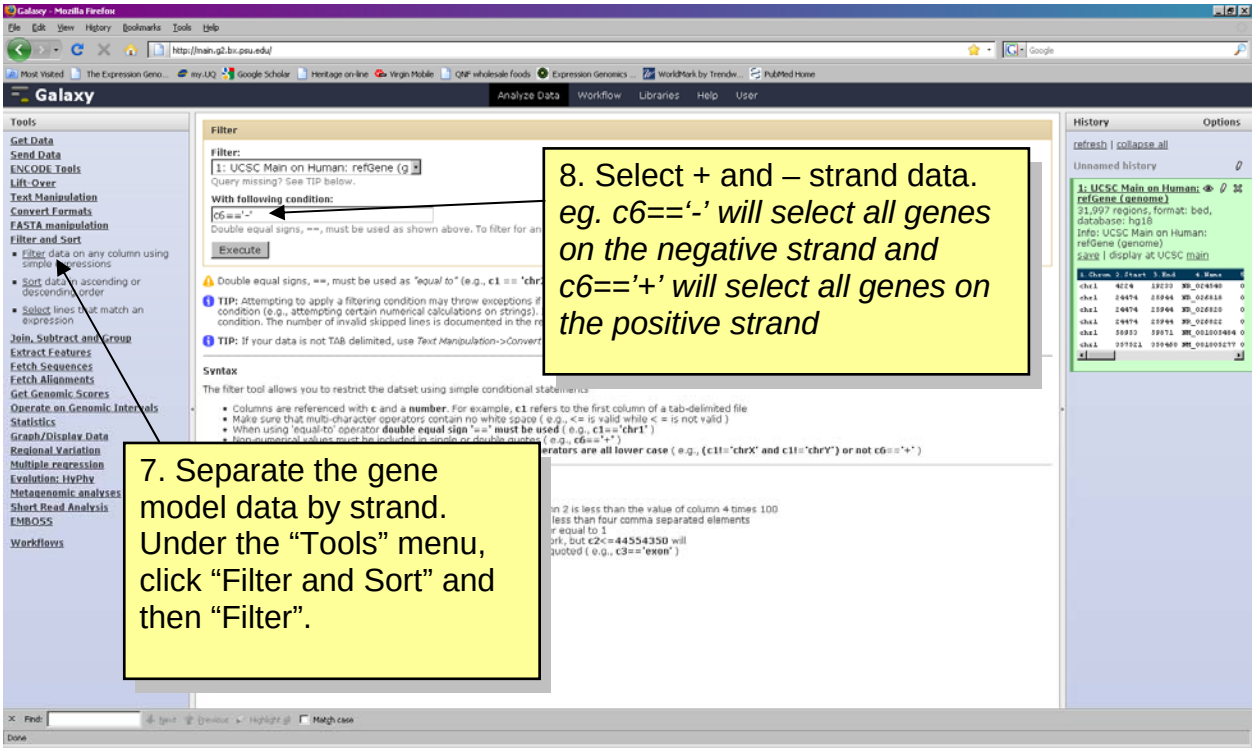
Assigning tags or coverage counts to gene models

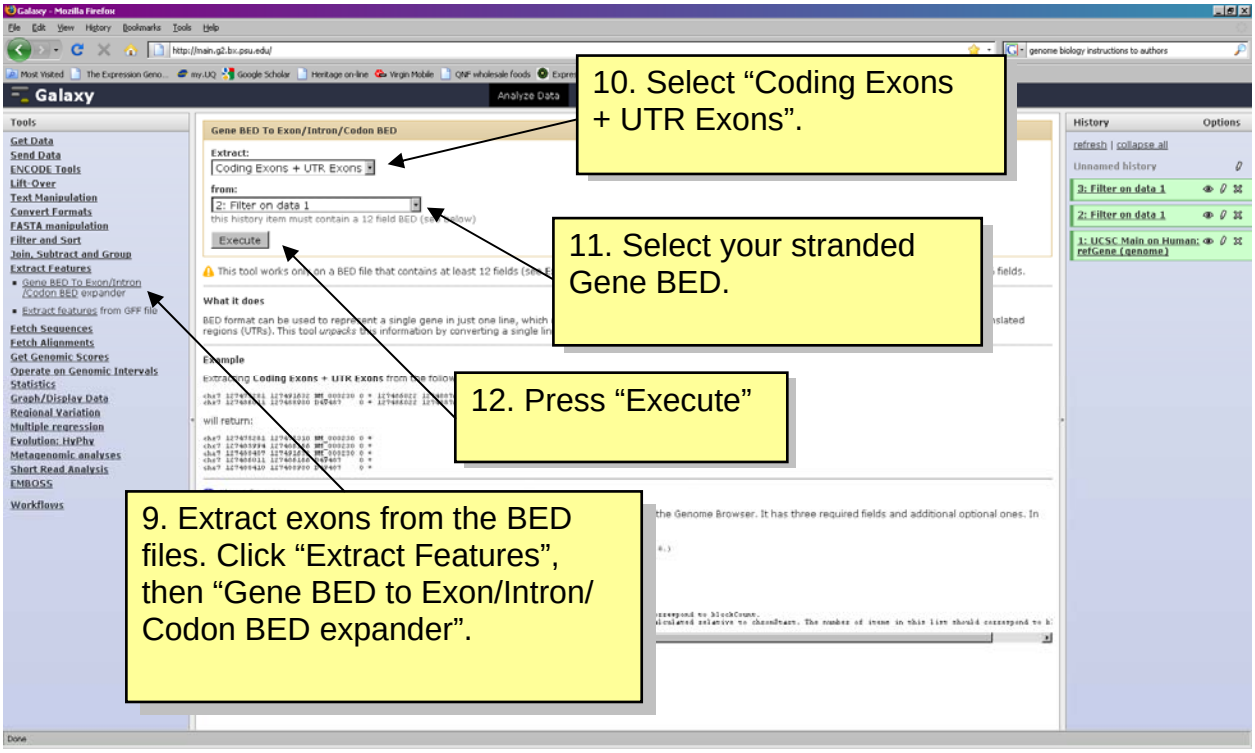
This script has been deprecated, as more extensive tools for manipulation of genomic regions are available from the GALAXY website at http://main.g2.bx.psu.edu/. Local copies of Galaxy can be installed and used, useful to avoid excessive internet usage through the transfer of large files. Downloading and installation guides are available from http://g2.trac.bx.psu.edu/wiki/HowToInstall.

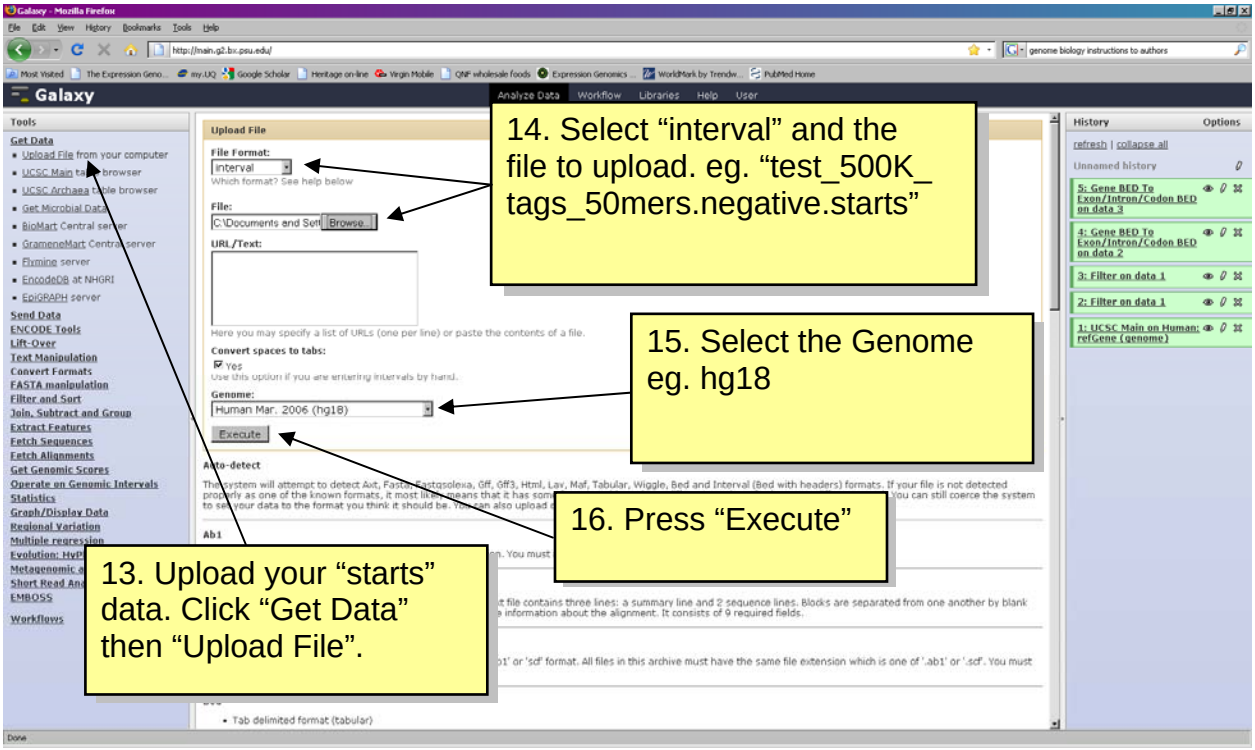
The following is a tutorial on how to assign tags to genes using Galaxy. In this step, particular care needs to be taken to ensure that different RNAseq protocols are processed with the strand of capture in mind. For example, serial-ligation approaches will generate sequences from the sense strand, relative to the annotated gene, whereas the random-primed strand-specific protocols will generate tags mapping to the anti-sense strand. Assigning tags to the wrong strand of gene models will result in relatively low numbers of tags assigned to the gene models, and subsequently very low correlations between array data and sequence data.

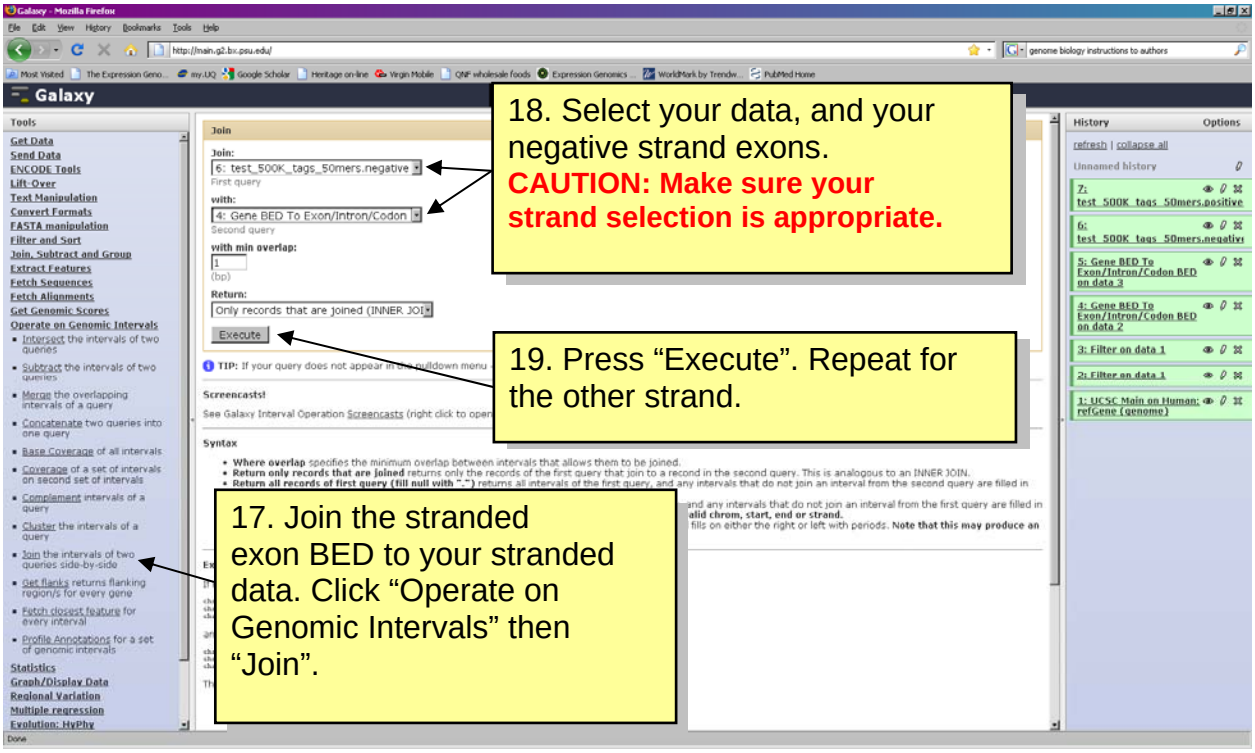


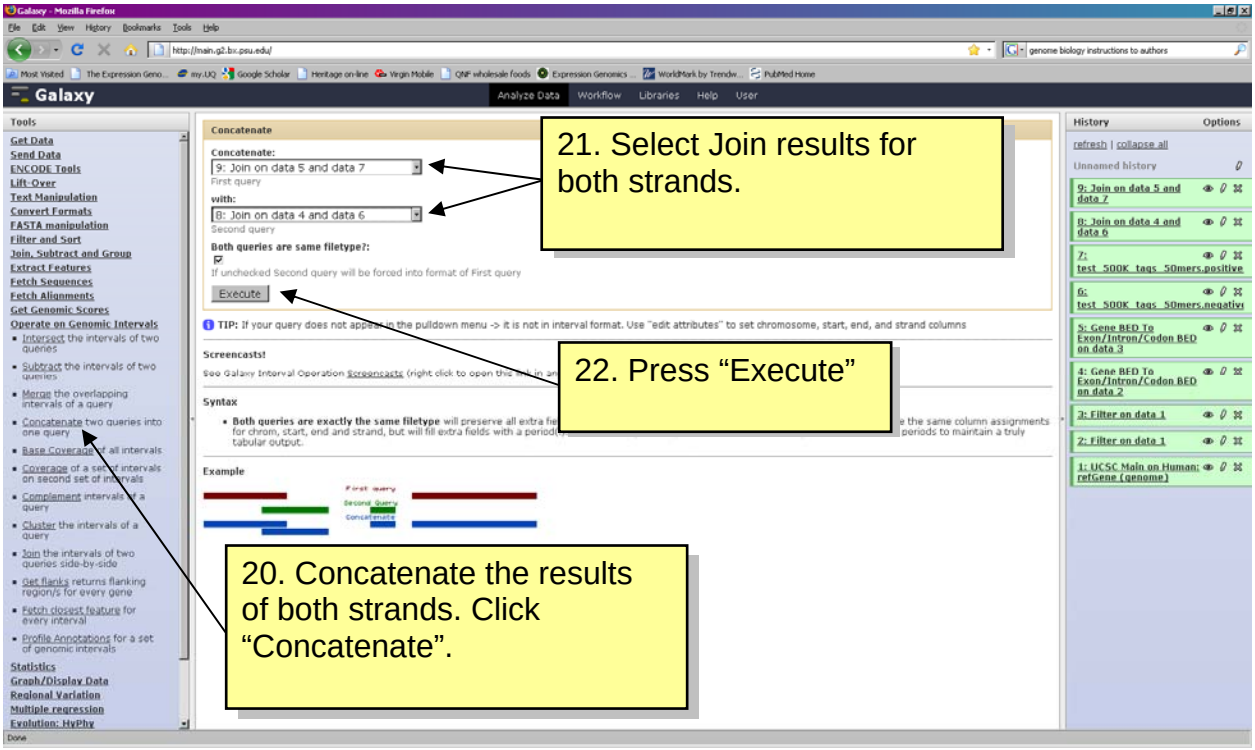

5. Select appropriate parameters for your experiment.
6. Click “send query to Galaxy”

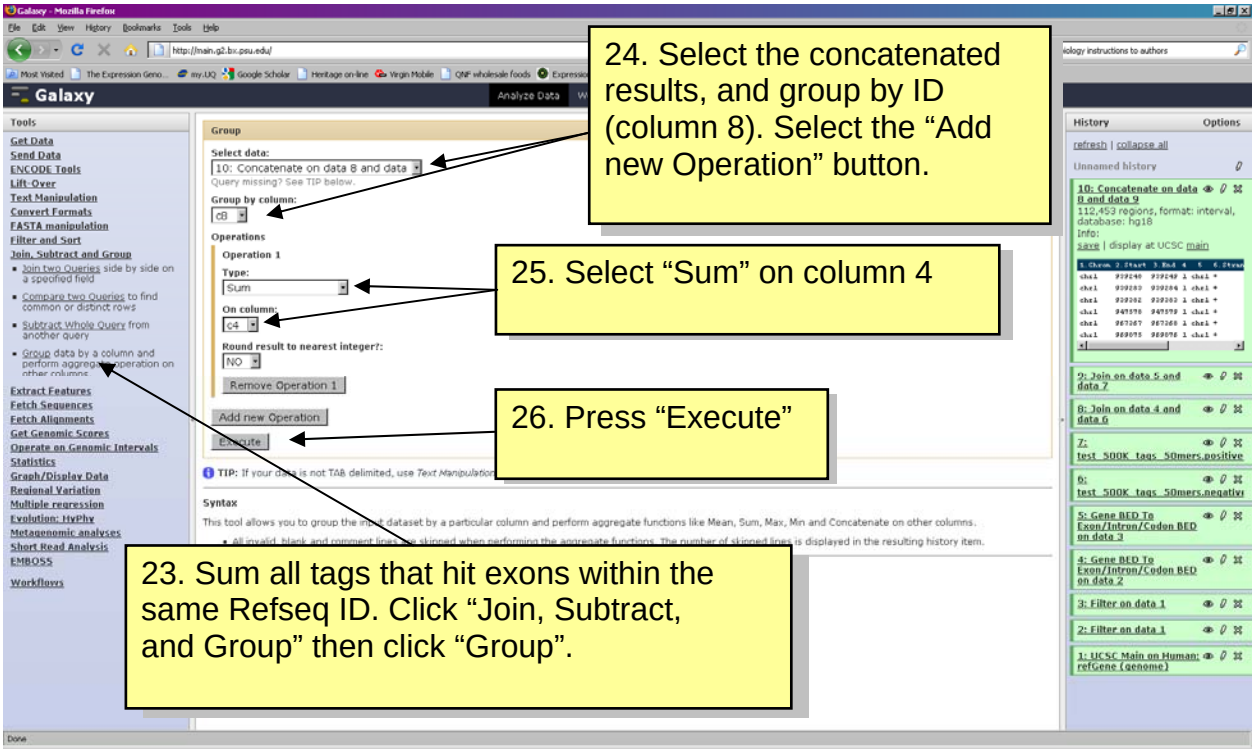

8. Select + and – strand data.
eg. c6==‘-’ will select all genes on the negative strand and c6==‘+’ will select all genes on the positive strand
7. Separate the gene model data by strand. Under the “Tools” menu, click “Filter and Sort” and then “Filter”.

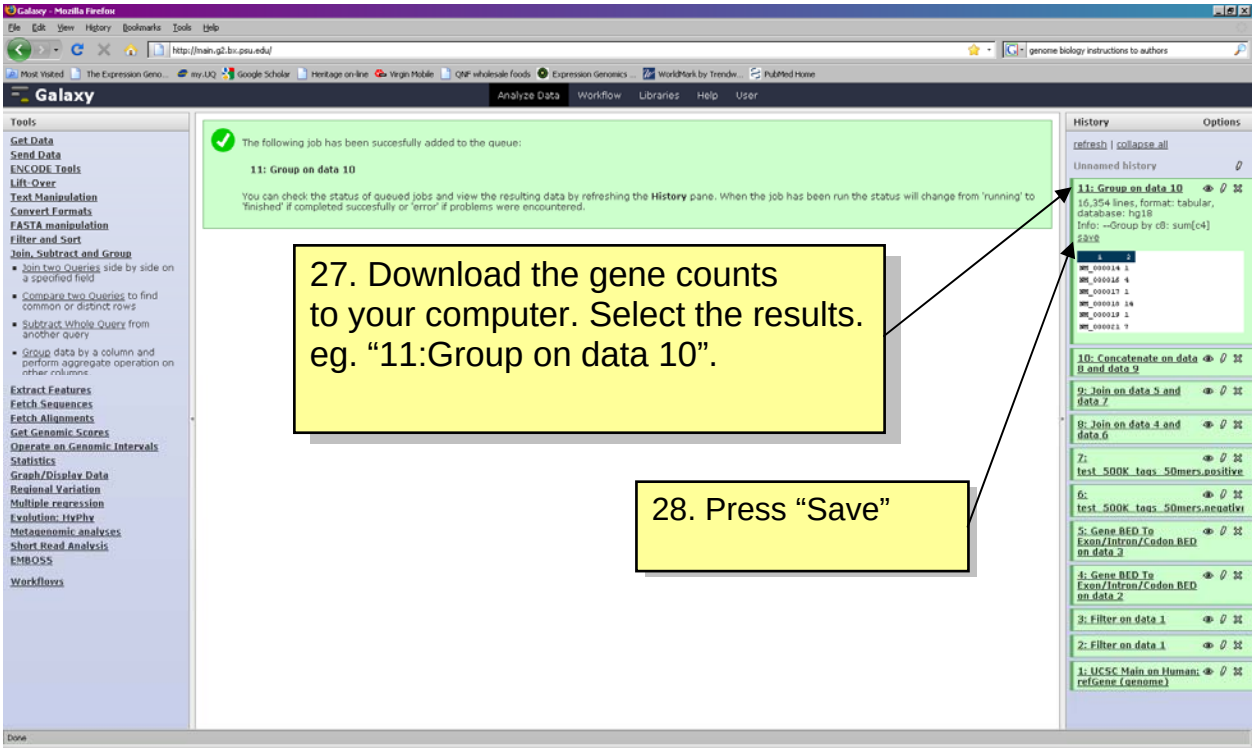

10. Select “Coding Exons + UTR Exons”.
11. Select your stranded Gene BED.
12. Press “Execute”
9. Extract exons from the BED files. Click “Extract Features”, then “Gene BED to Exon/Intron/ Codon BED expander”.


14. Select “interval” and the file to upload. eg. “test_500K_ tags_50mers.negative.starts”
15. Select the Genome eg. hg18
16. Press “Execute”
13. Upload your “starts” data. Click “Get Data” then “Upload File”.

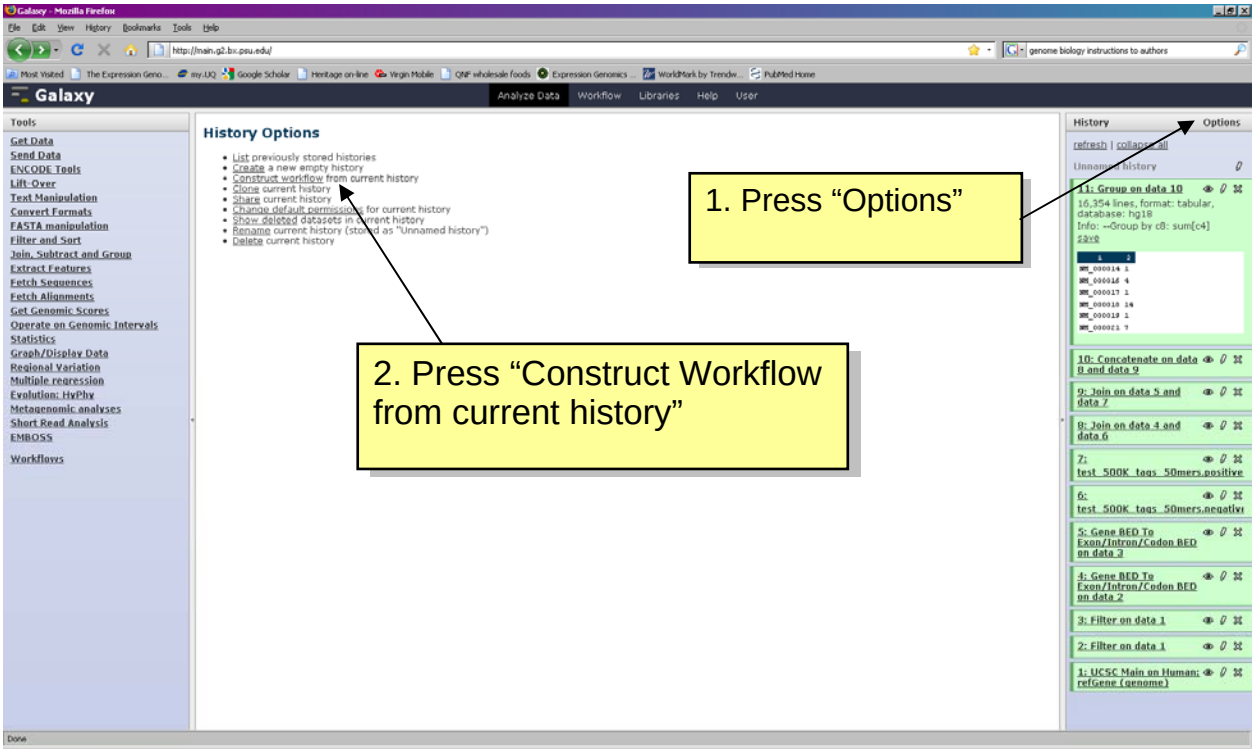

18. Select your data, and your negative strand exons.
CAUTION: Make sure your strand selection is appropriate.
19. Press “Execute”. Repeat for the other strand.
17. Join the stranded exon BED to your stranded data. Click “Operate on Genomic Intervals” then “Join”.

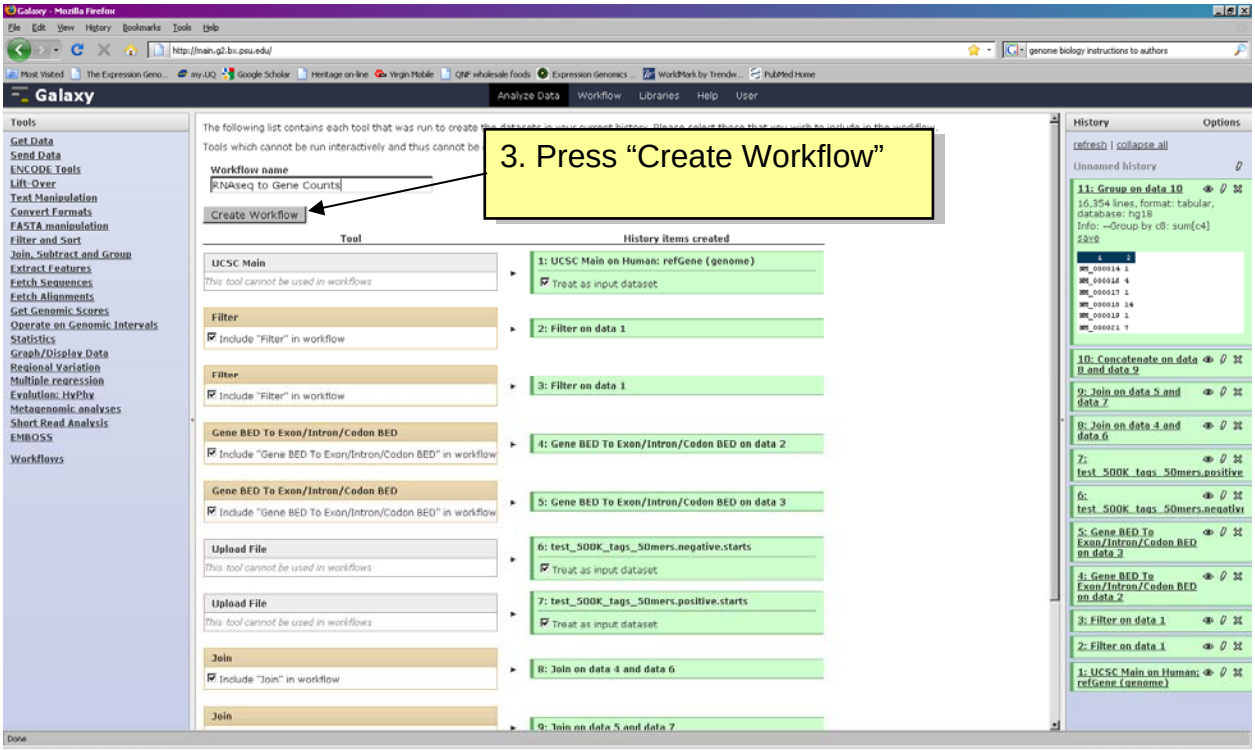

21. Select Join results for both strands.
22. Press “Execute”
20. Concatenate the results of both strands. Click “Concatenate”.

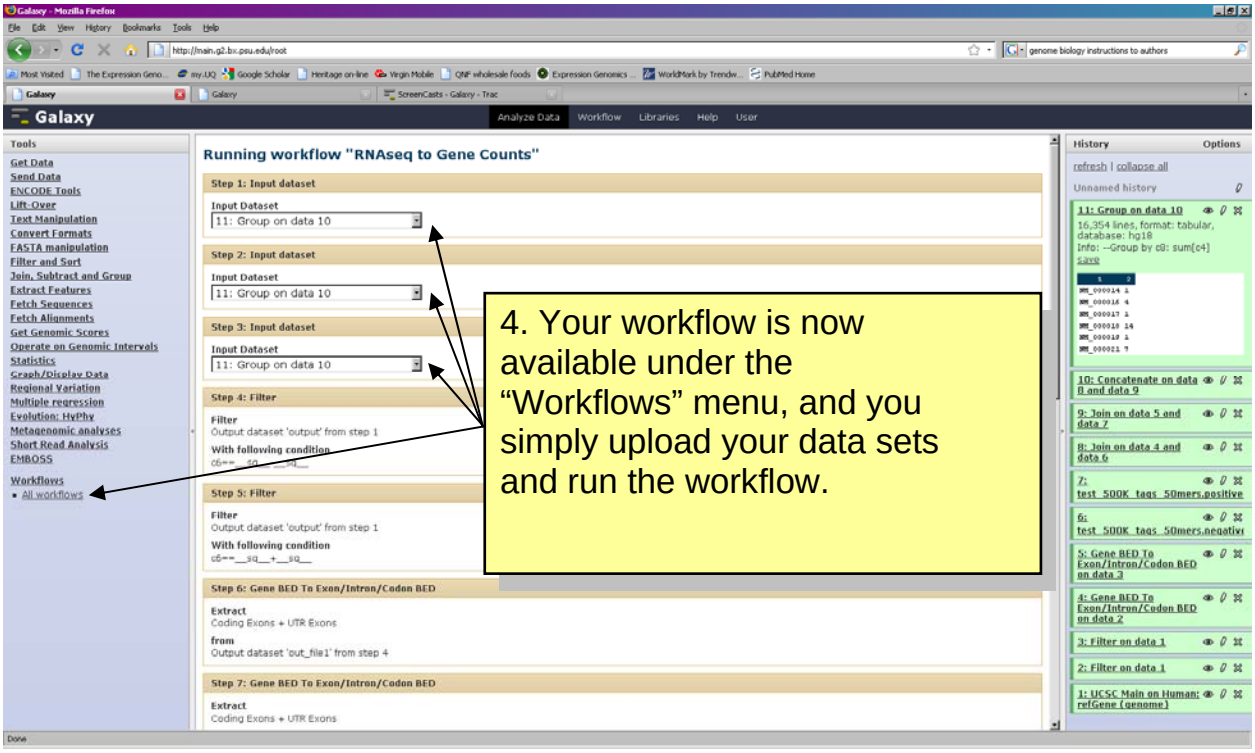

24. Select the concatenated results, and group by ID (column 8). Select the “Add new Operation” button.
25. Select “Sum” on column 4
26. Press “Execute”
23. Sum all tags that hit exons within the same Refseq ID. Click “Join, Subtract, and Group” then click “Group”.


27. Download the gene counts to your computer. Select the results. eg. “11:Group on data 10”.
28. Press “Save”

All these steps can be automated into what Galaxy calls a “Workflow”. This is particularly useful if you have multiple data sets to analyze. To create a workflow from the steps generated above, follow the steps below.






Galaxy
Analyze Data
Workflow
Libraries
Help
User
Tools
Get Data
Send Data
ENCODE Tools
Lift-Over
Text Manipulation
Convert Formats
FASTA manipulation
Filter and Sort
Join, Subtract and Group
Extract Features
Fetch Sequences
Fetch Alignments
Get Genomic Scores
Operate on Genomic Intervals
Statistics
Graph/Display Data
Regional Variation
Multiple regression
Evolution: HyPhy
Metagenomic analyses
Short Read Analysis
EMBOSS
Workflows
All workflows
Running workflow "RNAseq to Gene Counts"
Step 1: Input dataset
Input Dataset
11: Group on data 10
Step 2: Input dataset
Input Dataset
11: Group on data 10
Step 3: Input dataset
Input Dataset
11: Group on data 10
Step 4: Filter
Filter
Output dataset 'output' from step 1
With following condition
Step 5: Filter
Filter
Output dataset 'output' from step 1
With following condition
Step 6: Gene BED To Exon/Intron/Codon BED
Extract
Coding Exons + UTR Exons
from
Output dataset 'out_file1' from step 4
Step 7: Gene BED To Exon/Intron/Codon BED
Extract
Coding Exons + UTR Exons
History
Options
refresh | collapse all
Unnamed history
11: Group on data 10
16,354 lines, format: tabular, database: hg18
Info: --Group by c0: sum[c4]
10: Concatenate on data 8 and data 9
9: Join on data 5 and data 7
8: Join on data 4 and data 6
7: test_500K_tags_50mers.positive
6: test_500K_tags_50mers.negative
5: Gene BED To Exon/Intron/Codon BED on data 3
4: Gene BED To Exon/Intron/Codon BED on data 2
3: Filter on data 1
2: Filter on data 1
1: UCSC Main on Human: refGene (genome)
4. Your workflow is now available under the "Workflows" menu, and you simply upload your data sets and run the workflow.

Junction libraries

Description of the available junction libraries

Each available exon-junction library contains two components. The first is the concatenated fasta file of the exon junction sequences, and the second is the index file that details the genomic coordinates of the exonic sequences. In both available sets, the genomic coordinates form the unique ID of the junction, and are defined as:

```
[chromosome]_[first base of intron]_[last base of intron]_[strand]
```

Note: All junction sequences are provided in the sense orientation (ie. 5' to 3') and all coordinates are zero based. This means that hits to these libraries should be predominantly on the one strand, and which strand will depend on the laboratory based library preparation method used.

In the case of the "mammalian_exon_junction_libraries" each junction contains the last 30nt of the donor exon joined to the first 30nt of the acceptor exon to create a 60mer sequence. Exon sequences were derived from all known genes, gene predictions, mRNA evidence, and EST evidence available at the time of creation (early 2007). Redundant sequences were removed, as were 60nt sequences that matched to the genome in their entirety, or matched to other exon-junctions within the library. The file list for this junction set is as follows:

mammalian_exon_junction_libraries.tar.gz
- hg18_junctions_best_quality.fasta.cat
- hg18_junctions_best_quality.fasta.index
- mm9_junctions_best_quality.fasta.cat
- mm9_junctions_best_quality.fasta.index

In the case of the "junction_libraries" different lengths of the donor and acceptor sequences are provided to allow full customization of matching stringency (the number in the file name represents the number of nucleotides from the donor and acceptor – eg. hg18.junctions.25.fa.cat contains 25nt form the donor and 25nt from the acceptor). In this case, exon sequences were defined from UCSC known genes, Refseq, Ensembl, Aceview, GeneID, GenScan, and N-Scan. The file list for this junction set is as follows:

junction_libraries.tar.gz
- hg18.junctions.25.fa.cat
- hg18.junctions.25.fa.index
- hg18.junctions.30.fa.cat
- hg18.junctions.30.fa.index
- hg18.junctions.35.fa.cat
- hg18.junctions.35.fa.index
- hg18.junctions.40.fa.cat
- hg18.junctions.40.fa.index
- hg18.junctions.45.fa.cat
- hg18.junctions.45.fa.index
- hg18.junctions.50.fa.cat
- hg18.junctions.50.fa.index
- hg18.junctions.55.fa.cat

hg18.junctions.55.fa.index
hg18.junctions.60.fa.cat
hg18.junctions.60.fa.index
hg18.junctions.65.fa.cat
hg18.junctions.65.fa.index
mm9.junctions.25.fa.cat
mm9.junctions.25.fa.index
mm9.junctions.30.fa.cat
mm9.junctions.30.fa.index
mm9.junctions.35.fa.cat
mm9.junctions.35.fa.index
mm9.junctions.40.fa.cat
mm9.junctions.40.fa.index
mm9.junctions.45.fa.cat
mm9.junctions.45.fa.index
mm9.junctions.50.fa.cat
mm9.junctions.50.fa.index
mm9.junctions.55.fa.cat
mm9.junctions.55.fa.index
mm9.junctions.60.fa.cat
mm9.junctions.60.fa.index
mm9.junctions.65.fa.cat
mm9.junctions.65.fa.index

How to create your own junction libraries

To create custom exon-junction libraries, you need the coordinates and sequences of your exons junctions. For some species, you may be able to download these from the UCSC genome browser "Tables" pages at http://genome.ucsc.edu/cgi-bin/hgTables?command=start. RNA-MATE makes no requirement for the minimum or maximum number of nucleotides required on the donor or acceptor side of the junction, and there is no requirement to keep these lengths the same. However, it may be beneficial for your own analysis to ensure that these are symmetrical, so that when performing an analysis you can be sure of the minimum overlap of a tag on the junction sequence. ie. If you require a minimum of 10nt in a 50nt tag to cross an exon-junction, then the donor and acceptor sequences should be 40nt long.

Once the sequences and coordinates have been assembled (ensuring that the unique IDs are in the same format as the provided above), there are two scripts provided to format the libraries the way RNA-MATE is expecting them. The first script is concatenate_sequences.pl, and is used to convert the multi-fasta format into a single concatenated fasta format.

```
eg ./concatenate_sequences.pl -f [fasta file] -o [output file] -h [header]
```

The second is make_index.pl, and this script creates the index file required for decoding the matches to the concatenated junction files.

```
eg ./make_index.pl <[fasta_file] >[output file]
```