SETTING UP AND RUNNING THE EXON-QUANTIFICATION PIPELINE (EQP)

This document describes the necessary steps to set-up and run the exon quantification pipeline (version 2.1.0).

THE EXON QUANTIFICATION PIPELINE

The exon quantification pipeline is an integrated alignment workflow and a simple counting-based approach to derive estimates for gene, exon, and exon-exon junction expression. It consists of two distinct parts: the alignment module and the quantification module. The workflow of EQP is designed to provide a high degree of granularity allowing for the use of distributed computational resources as given, for instance, by a cluster with a job scheduling system or a cloud-based infrastructure. It provides the option to split the Fastq input files into blocks of a user-definable read number which can then be processed independently. Appendix 3 provides a more detailed description of the way EQP works.

DEPENDENCIES

EQP runs only on Unix and requires Python (≥ version 2.6.5) and Java (≥ version 1.6) to be able to run.

As tools EQP requires Bowtie2 (≥ 2.0.1), BEDTools (≥ 2.24.0), and SAMTools (≥ 0.1.17). Please check that these programs are available for you and included in the PATH environment variable.

OBTAINING ANG CREATING A PROJECT INSTANCE

The first step to create a project instance is to download the all scripts and tools from GitHub

`http://www.github.com/Novartis/EQP-cluster`

into the directory of your choice which will be denoted as `<project directory>` in the following. After downloading and unpacking the files from GitHub, the directory should contain the following files:

```
$ ls
LICENSE.txt  README.txt  exon-pipeline-files.tgz  samples.tgz
README.md    bin/        exon-pipeline-scripts/
```

Go to the `bin` directory and call the script `create-bowtie2-indices.sh`:

```
  cd bin
  create-bowtie2-indices.sh
```

Note that `create-bowtie2-indices.sh` requires that the programs `samtools` and `bowtie2-build` are installed. See above for the version requirements.

The script `create-bowtie2-indices.sh` extracts the files from the tar file `exon-pipeline-files.tgz` into the directory `exon-pipeline-files` and launches three calls to `bowtie2-build` in the background to construct the Bowtie2 indices of the genome Fasta file, the transcript Fasta file, and a custom junction Fasta file. The computation of the Bowtie2 indices will take several hours.

Limitations: Note that currently EQP-cluster uses human Ensembl76 reference files (on which the files contained in the directory exon-pipeline-files are based) and is, thus, configured to quantify only human data (ftp://ftp.ensembl.org/pub/release-76/fasta/homo_sapiens); furthermore, due to the way the custom junction Fasta file is created EQP-cluster works best with reads of length 101bp or less.
Other reference file package will be made available in the future.

Once the Bowtie2 indices are computed, EQP-cluster is ready to be used.

ADDING SAMPLES TO THE <PROJECT DIRECTORY>

For the processing of samples EQP relies one a fairly rigid directory structure in which reference files are located at predefined locations in the `exon-pipeline-files` subdirectory, scripts and programs are located at predefined locations in the `exon-pipeline-scripts` subdirectory, and Fastq files are located at predefined locations in the `samples` subdirectory. In following we describe how to setup a project specific directory structure in the `samples` subdirectory. When the file `exon-pipeline-files.tgz` is unpacked by `create-bowtie2-indices.sh`, then a `samples` directory is created which contains two sample directories as test data. Please remove these directories before setting up your own sample directories.

For each of your samples you need to create a subdirectory in `<project directory>/samples`. For each sample directory `<sample>` EQP assumes that it contains one or two Fastq files (or links to Fastq files) called either `<sample>.fastq.gz` or `<sample>_1.fastq.gz` and `<sample>_2.fastq.gz` depending on whether you want to process single-read or paired-end reads.[1] By default samples are assumed to be paired-end. See Section “Advanced options” below for the processing of single-read data.

EQP provides a convenience script called `create-fastq-links.sh` in `<project dir>/bin` for the generation of the sample directories. It creates links to Fastq files that are stored elsewhere. To be able to make use of `create-fastq-links.sh`, it is necessary to create a sample annotation file. A sample annotation file has the following format consisting of three columns with the listed headers and contents:

Sample Name	Fastq Path	Fastq Name
<Sample name 1>	<Path to fastq files>	<Fastq file name 1>-2.fastq.gz

[1] Sometimes there is more than one Fastq file for one sample, for instance, if the same sample is sequenced on different lanes. In this case it is possible to create more than one Fastq file or link for one sample in its directory. The naming convention for paired-end data is then to name the files “`<sample>_<copy number>_<read index>.fastq.gz`”, so `<sample>_1_1.fastq.gz, <sample>_1_2.fastq.gz` for the first set of reads, `<sample>_2_1.fastq.gz, <sample>_2_2.fastq.gz` for the second, and so on. The command “`process-samples.sh split`” recognizes this naming convention and then automatically concatenates all the files for the first read and all the files for the second read before splitting them into chunks. If single-read processing is selected, then all (gzipped) Fastq files with the prefix `<sample>` are concatenated.

<Sample name 2>	<Path to fastq files>	<Fastq file name 2>-2.fastq.gz

Note that for paired-end reads only the second file of the two files containing the reads is listed. Please also note that the entries in the "Sample Name" column need to contain a suitable Unix directory name, preferably of the form `[a-zA-Z0-9-_]*`; in particular, they should not contain spaces or special characters like &, ', ", or |. If they do, then this may cause `process-sample.sh` or other scripts to crash. These sample names will later appear as the identifiers for the counts of the samples.

After the creation of the sample directories and the (links to) the Fastq files contained in them, any directory contained in `<project directory>/samples` is considered a sample directory and the script `process-samples.sh` (see below) will loop over these and assume that they contain Fastq files named as outlined above (the exception to this rule is that `process-samples.sh` will ignore any directories with the suffix "-files", e.g. `<project directory>/samples/count-files` which is created by EQP in later stages).

EXAMPLE OUTPUT

```
$ bin/create-fastq-links.sh annotation-file.txt
…
$ ls samples/
sample-01/  sample-02/
…
$ ls samples/sample-01/
sample-01_1.fastq.gz@   sample-01_2.fastq.gz@
…
$ ls -l samples/sample-01/sample-01_1.fastq.gz
lrwxrwxrwx 1 ngs-user ngs-group 127 Dec  1 18:05 samples/sample-01/sample-
01_1.fastq.gz -> /NGS/Data/D00121_0074_BC5FLFACXX/D00121_0074_BC5FLFACXX-s_1-
BC9_GATCAG-1.fastq.gz
```

After the creation of the sample directories the setup of the EQP project is complete.

RUNNING THE PIPELINE

EQP is designed to work with the Univa[2] Grid Engine (UGE, formerly Sun Grid Engine SGE). Most of the processing steps are submitted to the UGE scheduler via the `qsub` command and are then executed asynchronously on a node of the cluster managed by UGE. In Appendix 1 it is shown which resources are used by the `qsub` call of EQP. It is also possible to run EQP without UGE. In this case the option:

```
   -noqsub <number of cores>
```

needs to be supplied to all commands. EQP will then launch processes in the background until the number of cores given by the `–noqsub` option is exhausted. Usually, the different commands require more than one core. The number required for one call is given in the corresponding sections below. Note that the number of cores used by EQP will be at most the number given after `–noqsub`. If this number is

[2] Univa® is a registered trademark of Univa Corporation

smaller than the minimum number of required cores for one process, then, nevertheless, at least one process will be run using the number of cores given by the `–noqsub` option.

SPLITTING THE FASTQ FILES

The first step in the pipeline is to chunk up each Fastq file into chunks of size 25M reads. In order to do this in all sample directories run (in `<project directory>/bin`)

```
 process-samples.sh split
```

which launches a split job running on one core for each sample (via qsub) and divides the Fastq files into chunks of size 25M reads. This creates a number of subdirectories in each sample directory of the form

```
      <project directory>/samples/<sample directory>/<chunk dir>
```

where `<chunk dir>` is `C001, C002, C003`, … etc. Of course, the number of such subdirectories depends on the size of the Fastq file.

Once all split processes have finished, it should be checked that the splitting of all samples was successful by calling

```
 process-samples.sh –status split
```

which should give the result

```
Sample      Operation   Job name          Status
<sample 1> split        SP-<sample 1>     complete
…
```

The split script generates a log-file for each sample in the directory

```
  <project directory>/samples/<sample>/log-files
```

The log-files need to be checked for problems with the splitting process if the status is different from `complete`. Please note that this step needs to be executed independent of the size of the Fastq file, i.e. even it is smaller than 25M reads; this is due to fact that the identifiers of the Fastq files are changed in this step (to F0000001, F0000002, …) which is needed by the pipeline for various reasons.

Note that the split jobs need to finish before the alignment and quantification jobs can be launched. The reason is that in the number of chunks per sample is determined in the split step and in the alignment and quantification steps one job per chunk is launched.

ALIGNING THE READS

Once all the split jobs have finished (and the log files have been checked), run

```
 process-samples.sh align-all
```

which launches three alignment jobs (one for the transcripts, one for the junctions, and one for the genome) each of which runs on six cores for each chunk of each sample. This creates the respective SAM files in the directories

```
  <project directory>/samples/<sample>/<chunk dir>/sam-files
```

Calling `process-samples.sh` without any arguments lists the available commands and options.

ALIGNING THE READS SEPARATELY

If desired, alignments against the genome alone can be generated by calling

```
 process-samples.sh align-genome
```

Similarly, alignments against the transcripts (or junctions) alone can be launched by calling

```
 process-samples.sh align-transcripts
```

```
 (process-samples.sh align-junctions)
```

CHECKING THE JOB STATUS AND RESUBMITTING JOBS

The three alignment scripts generate three log-files for each chunk of each sample in the directory

```
  <project directory>/samples/<sample>/log-files
```

In order to check the status of the jobs you can either use `qstat` and view the log-files or use the –`status` option again; similar to split though you do not have to wait for the completion of the jobs:

```
   process-samples.sh –status <operation>
```

where `<operation>` is one of `align-genome`, `align-transcripts`, or `align-junctions`.[3]

The status of a job can be either `not submitted`, `running`, `waiting`, `on hold`, `in deletion`, `SGE-error`, `crashed`, `complete`, or `failed`. If the job has been submitted and is in the UGE queue, then its status is one of `running (r)`, `waiting (w)`, `on hold (h)`, `in deletion (d)`, `SGE-error (E)`, or `crashed`. The first four depend on the status character in the output of `qstat`. The status `crashed` implies that the job is still running in the queue while an error message has been detected in the log-file. In theory this should not happen, but it may occur on rare occasions.

Once a job is not contained in the UGE queue anymore, its status is either `failed` or `completed` depending on whether an error message is detected in the log file or not.

If a job failed because of a technical problem with the cluster, it is necessary to resubmit it. In EQP this can be done via

```
   process-samples.sh –resubmit <operation>
```

which leads to the resubmission of all jobs with status `failed` to the SGE queue.

CREATING COUNTS

After the alignment the count-files can be generated by executing

[3] The `–status` option also works for the operation `compute-counts`. Please note that is does not work on `combine-counts`.

```
process-samples.sh compute-counts
```

For each chunk this launches a job on the cluster running on four cores that computes the (weighted)[4] count files for genes, exons, and junctions in the directories

```
<project directory>/samples/<sample>/<chunk dir>/count-files
```

Again a log-file is generated for each chunk of each sample in the directory

```
<project directory>/samples/<sample>/log-files
```

and the status of the jobs can again be checked with

```
process-samples.sh –status compute-counts
```

COMBINING COUNTS

The separate count files can be joined into one file by running

```
process-samples.sh combine-counts
```

which creates three result files, one for the gene, exon, and junction counts, resp. in the directory

```
<project directory>/samples/count-files
```

These files contain one column per sample plus one column for the id of the quantified object (gene, exon, or junction). Note that this command does not create a job on the cluster but executes directly on the node on which you invoke it. It is recommended to use qlogin before executing `combine-counts` to avoid undue burdening of the cluster head node. Please also note that all `compute-counts` job on the cluster need to successfully finish before `combine-counts` can be called.

`combine-counts` also creates a file with suffix `junctions-all.cnt` which contains the junction counts of all junctions which have a non-zero entry in at least one sample. It is usually much smaller than the junction counts file.

If you are only interested in gene counts, then only these can be combined via

```
process-samples.sh combine-gene-counts
```

CREATING FPKM VALUES

Once you have created a combined count file, you can convert the counts into FPKM values by invoking:

```
process-samples.sh compute-fpkm
```

The FPKM file is then created as the combined counts file in the directory

```
<project directory>/samples/count-files
```

[4] The role of read weights for the creation of read counts in EQP is described in Appendix 3.

EQP provides its own gene length file which is based on the genomic footprint of a gene, that is, the length of a bases in the genome that are covered by some exon of the gene. The sequencing depth normalization is based on the sum of the counts in each sample.

CREATING A GENOME SAM FILE WITH SPLICED READS

In order to visualize the alignments in a genomic context EQP provides the possibility to create genome SAM file with spliced reads that is a combination of the transcript, junction, and genome alignments similar to splice-aware aligners such as Tophat or STAR. In order to create a combined genome SAM file with splice reads you need to execute

```
process-samples.sh –eqp align-genome
```

which will create a genome SAM file for each chunk of each sample. In order to combine the chunks of one sample in one BAM file and sort the reads by coordinates call

```
process-samples.sh –eqp merge-genome-alignments
```

The resulting sorted BAM file will be contained in the directory

```
<project directory>/samples/<sample>/sam-files
```

ADVANCED OPTIONS

The operations `align-all`, `compute-counts`, and `combine-counts` usually suffice to generate the counts for a given set of samples. However, EQP provides additional options to either enhance usability or for additional functionality.

SUBMITTING SELECTED SAMPLES OR CHUNKS

Sometimes it is necessary to submit a subset of samples or chunks. In EQP this is possible via

```
process-samples.sh –s <sample-list> -c <chunk-list> <operation>
```

where `<sample-list>` is a concatenation of sample names separated by one of "`:;,|`". Note that since "`;`" or "`|`" are special characters in the shell `<sample-list>` should always be enclosed in quotes (" or '). The same applies to `<chunk-list>`.

SINGLE READ DATA

Single read data can be processed by using the option `–sr` for each operation:

```
process-samples.sh –sr <operation>
```

STRAND-SPECIFIC DATA

If the samples are sequenced by a strand-specific protocol, then EQP can make use of this information and only count reads that align to the correct strand of a feature. In order to specify that the reads are strand-specific you can use the option `–S <direction>` where direction is either `forward` or `backward`.

```
   process-samples.sh –S <direction> <operation>
```

The direction `forward` indicates that the reads are assumed to be aligned with the first read on the correct strand and the second read on the reverse strand. The direction `backward` indicates the opposite orientation of the reads (first read on the wrong strand and second read on the correct strand).

Please note that the specification of strand-specific reads only has implications in the alignment against the transcripts (`align-transcripts`) and in the quantification (`compute-counts`).

COUNTING ONLY UNAMBIGUOUS READS

Some packages for the quantification of reads (e.g. htSeq) recommend counting only reads that can be unambiguously assigned to one feature (gene, exon, or junction). This option is now available in EQP for genes by using the option `–unambig`.

```
      process-samples.sh --unambig compute-counts
```

For exons and junctions we do not consider this option as useful as there exist many overlapping exons, in particular, in Ensembl (in addition to the fact that one exon can have a number of different identifiers).

LICENSE

APPENDIX 1 - EQP-CLUSTER AND UGE

EQP is designed to work with the Univa® Grid Engine (UGE, formerly Sun Grid Engine, SGE). Most of the processing steps are submitted to the UGE scheduler via the `qsub` command (with the notable exception of `process-samples.sh combine-counts` which is executed on the machine on which the command is issued). Once submitted, a job is then executed asynchronously on a node of the cluster managed by UGE. Below we show which resources are used by the `qsub` call of EQP. It is also possible to run EQP without UGE. In this case the option

```
    -noqsub <number of cores>
```

needs to be supplied to the script `process-samples.sh` for all operations (see below). With this option EQP launches processes in the background until the number of cores given by the parameter `<number of cores>` is exhausted. Usually, the different commands require more than one core:

- `split` requires one core
- `align-all` six cores
- `compute-counts` four cores

Note that the number of cores used by EQP will be at most the number given after `-noqsub`. If this number is smaller than the minimum number of required cores for one process, then, nevertheless, at least one process will be launched using the number of cores given by the `-noqsub` option.

The format of the `qsub` command used by EQP is as follows:

```
    qsub -V -l m_mem_free=<mem> -l h_rt=<run time> -N <job name>
      -hold_jid <dependence jobs> -o <log file> -pe smp <num cores> <command>
```

The parameters `-V, -N, -hold_jid, -o`, and are generally accepted by UGE `qsub` whereas the resource parameters `-l m_mem_free=<mem>, -l h_rt=<run time>`, and `-pe smp <num cores>` are dependent on the configuration of the UGE instance. Here, `<mem>` is the memory requirement per core and `<run time>` is either given as the number of seconds (14400 for four hours, default value) or as hours (12:00:00 for compute-counts as it sometimes may take more than four hours).

If your UGE instance does not accept the parameters `m_mem_free, h_rt, smp` or requires other parameters (e.g. a queue name), the recommendation is to edit the script `process-samples.sh` and to adapt the calls to `qsub` in the function `submitJob` which is the only place where `qsub` jobs are actually submitted.

APPENDIX 2 – LIST OF COMMANDS

Command	Effect
`create-fastq-links.sh <sample annotation file>`	Create sample subdirectories and links to Fastq files listed in the annotation file
`process-samples.sh split`	Split Fastq files into chunks of 25M reads each and create chunk subdirectories (runs on one core)

`process-samples.sh align-genome`
`process-samples.sh align-transcripts`
	For each sample and chunk submit a job to the cluster to align the Fastq file(s) of the chunk against the genome (transcripts, junctions, or all three); each alignment job

`process-samples.sh align-junctions`
 	
`process-samples.sh align-all`
	runs on six cores

`process-samples.sh compute-counts`	For each sample and chunk submit a job to the cluster to generate the gene, exon, and junctions counts; runs on four cores and uses about 15G of memory

`process-samples.sh combine-counts`
 	
`(or combine-gene-counts)`
	Combines the gene, exon, and junctions count files of each sample and chunk into one file in the directory samples/count-files; works only if the status of compute-counts for each sample/chunk is complete. Runs directly without creating jobs.

`process-samples.sh compute-fpkm`	Normalizes the gene counts to FPKM values (`combine-counts` needs to be completed)

`process-samples.sh –noqsub <number of cores> <operation>`
 	
`where <operation> is one of align-genome, align-transcripts, align-junctions, compute-counts, merge-genome-alignments.`
	Launches jobs as background jobs on the current machine rather than using qsub. At most <number of cores> cores are used.

`process-samples.sh –status <operation>`
 	
`where <operation> is one of align-genome, align-transcripts, align-junctions, compute-counts, merge-genome-alignments. Does not work without qsub.`
	For each sample and chunk display the status of the job submitted to the cluster for the operation. The status can be `not submitted, running, waiting, on hold, in deletion, SGE-error, crashed, complete, or failed.`

`process-samples.sh –resubmit <operation>`
 	
`where <operation> is one of align-genome, align-transcripts, align-junctions, compute-counts, merge-genome-alignments. Does not work without qsub.`
	For each sample and chunk resubmit a job to the cluster for the operation if the status of the job is `failed`.

`process-samples.sh –eqp align-genome`	For each sample and chunk submit a job to the cluster to combine the three alignment files (for the genome, transcripts, and junctions) and create a genome alignment file with spliced reads.
`process-samples.sh –eqp merge-genome-alignments`	For each sample and chunk submit a job to the cluster to combine the spliced genome alignment files into one file for each sample.
`process-samples.sh –sr <operation>`	Execute `<operation>` for single read data
`process-samples.sh –S <direction> <operation>`	Execute `<operation>` for strand-specific data which direction `<direction>` (`forward` or `backward`).
`process-samples.sh –unambig compute-counts`	Based gene counts only on reads that can be

	unambiguously assigned to a single gene.

`process-samples.sh –A <aligner> <operation>`
 	
`where <operation> is align-genome, compute-counts, combine-counts, or compute-fpkm`
	Create or use the alignments based on aligner `<aligner>`. (beta version)

`createPicardMetrics.sh <type>`	For each sample run Picard on the alignments of type `<type>` (transcript or genome).
`combinePicardMetricFiles.sh <type>`	For each Picard metric, combine the sample results into one file for all samples for alignments of type `<type>` (transcript or genome).

APPENDIX 3 – THE ALIGNMENT AND QUANTIFICATION MODULE OF EQP

The workflow of EQP is designed to provide a high degree of granularity allowing for the use of distributed computational resources as given, for instance, by a cluster with a job scheduling system or a cloud-based infrastructure. It provides the option to split the Fastq input files into blocks of a user-definable read number which can then be processed independently; at the same time it also filters out reads with too few non-A or non-T bases to filter out polyA tail reads. Generally all processing operations of EQP are based on a read by read ordering – rather than a genome-coordinate based ordering after the alignment step.

THE EQP ALIGNMENT MODULE

Before EQP can be run, it is necessary to create a number of auxiliary files that are needed in the alignment module and for the generation of the counts. As input for the setup EQP requires a genome Fasta file, a transcript Fasta file, and a GTF file. The main output of the setup step is a file containing the mapping of the genomic exons to the transcripts in BED format (1) and an exon-exon junction Fasta file. The latter file contains all possible junctions of exons that belong to the same gene. More precisely, only the parts of the exons up to the read length on each side of the junction are used in the creation of the Fasta file. If an exon is shorter than the read length, EQP pads junction entries with all possible upstream or downstream exons (depending on whether the exon is upstream or downstream of the skipped intron) in order to avoid imbalanced junction entries. This procedure can lead to a very high number of combinations for genes with many small exons as several padding steps may be required on each side until the sum of the exon lengths exceeds the read length. In EQP such genes are identified based on a threshold on the number of junction entries and for these genes short exons are only padded by their direct upstream and downstream flanking exons.

The alignment module of EQP aligns the Fastq files containing the single or paired-end reads against the externally provided transcript and genome Fasta file as well as the custom created junction Fasta file using Bowtie2 (2); the three alignment computations can be run independently, thus providing an additional opportunity for parallelization. For each read and alignment file, the type of alignment (single-read or paired-end), the number of alignments, and the sum of the number of mismatches of the alignments (taken from the optional SAM NM field (3)) are computed. Then the best set of alignments (either the transcript, junction, or genome alignments) is selected based on three criteria (similar to (4)): the alignment type (paired-end is preferred over single-read), the mean number of mismatches, and the number of alignments; a slight preference is given to transcript alignments by adding a small penalty per read to genome alignments. If no paired-end alignment exists, then the best alignments are chosen separately for each read. The result of the selection process is a combined SAM file consisting of SAM entries that originate from the three different original alignment files. As an optional step, EQP can also generate a genome alignment file containing spliced genome alignments for all mapped reads which can then be used to visualize the alignments in a genome browser such as IGV (5).

THE EQP QUANTIFICATION MODULE

The counts computed by EQP are based on weighted reads where the weight *w* of a read that aligns *n* times to the genome is given as *w* = 1 / *n* similar to the treatment of multi-reads by Cufflinks (6). EQP allows setting an upper bound *B* on the number of allowed genomic alignments of a read. The default is 100 but, for instance, setting *B* = 1 results in counting only uniquely aligning reads and setting *B* = 10 in counting quasi-unique alignments (7).

For the computation of the read weights EQP takes the combined SAM file as input, converts it into a BED file, and intersects the converted BED file with a BED file containing the transcript, the genome, and the junction coordinates of the genomic exons using BEDTools (1). The resulting intersection file contains the relevant information to calculate the reads' spliced genomic alignments.

In EQP a read is counted for a feature, e.g. a gene or exon, if and only if the induced exon boundaries of its alignment completely agree (are *compatible*) with the exon boundaries of the feature. This is illustrated in Figure 1. In particular, compatibility implies that no intron spanned by the read overlaps with the feature and vice versa.

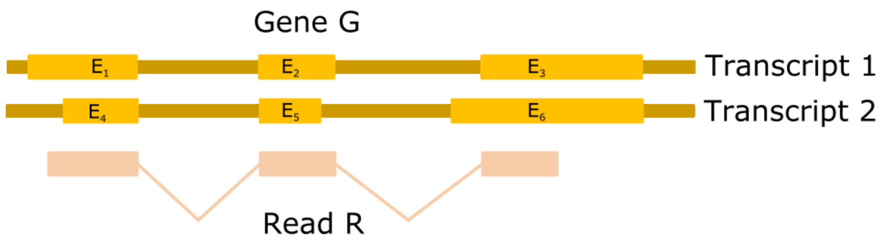

Figure 1 Compatibility of a read alignment

The compatibility requirement ensures that the counted reads represent evidence for the fully matured expression of the feature and excludes reads originating from the sequencing of pre-mRNA transcripts. This is especially important for the computation of exon and junction counts as these are mainly used in the analysis of differential splicing and, thus, should only capture the mRNA signal.

Based on the intersection BED file and a file containing the mapping of the genomic exons to genes, exons, or junctions, the respective counts are then computed. The generation of junction counts (as well as gene and exon counts) is independent of aligning the reads against the junction Fasta file. All three types of counts can be generated from transcript alignments alone.

1. Quinlan, A.R. and Hall, I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841-842.

2. Langmead, B. and Salzberg, S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat Meth*, **9**, 357-359.

3. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R. and Subgroup, G.P.D.P. (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078-2079.

4. Grant, G.R., Farkas, M.H., Pizarro, A., Lahens, N., Schug, J., Brunk, B., Stoeckert, C.J., Hogenesch, J.B. and Pierce, E.A. (2011) Comparative Analysis of RNA-Seq Alignment Algorithms and the RNA-Seq Unified Mapper (RUM). *Bioinformatics*.

5. Robinson, J.T., Thorvaldsdottir, H., Winckler, W., Guttman, M., Lander, E.S., Getz, G. and Mesirov, J.P. (2011) Integrative genomics viewer. *Nat Biotech*, **29**, 24-26.

6. Trapnell, C., Williams, B.A., Pertea, G., Mortazavi, A., Kwan, G., van Baren, M.J., Salzberg, S.L., Wold, B.J. and Pachter, L. (2010) Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat Biotech*, **28**, 511-515.

7. Wang, C., Gong, B., Bushel, P.R., Thierry-Mieg, J., Thierry-Mieg, D., Xu, J., Fang, H., Hong, H., Shen, J., Su, Z. *et al.* (2014) The concordance between RNA-seq and microarray data depends on chemical treatment and transcript abundance. *Nat Biotech*, **32**, 926-932.