Institut für Informatik
Praktische Informatik und Bioinformatik
Prof. Dr. Ralf Zimmer
Dr. Gergely Csaba
Dr. Florian Erhard

Wintersemester 2015/2016

GoBi: Excercise 2

Genomic Sequence Extraction, Read simulation

Deadline: Thursday, 26.11.2015, 14:00

Save your solution to /home/proj/biocluster/genprakt/${stud_account}/Solution2. Provide an executable jar file (containing also the sources) in this directory that allows to reproduce your results. The jar should print a usage info if invoked without parameters.

You find the input files in the directory

`/home/proj/biosoft/praktikum/genprakt/assignments/a2/data/`

Task 1 (Genomic Sequence Extraction):

Implement a class GenomeSequenceExtractor with a method:

```
String getSequence(String chr, int start, int end, boolean reverse_complement);
```

You find the human genome in FASTA format here: *Homo_sapiens.GRCh37.75.dna.toplevel.fa*

Keep in mind that in general you will not be able to read the whole human genome into main memory (nor would it make much sense to do so). Also, it is not efficient to read the whole fasta file for each query.

However, when you know where in the fasta file each chromosome starts (i.e. its offset in the file) and how long the sequence lines are, it is straight-forward to calculate the position in the FASTA file, where any sequence is located. Then, you only need to extract each queried sequence by jumping to this position (e.g. using RandomAccessFile.seek in Java.).

Use your implementation and your GTF parser to extract all transcript sequences from the annotation on the standard chromosomes (1-22, X, Y, MT).

Use the GTF: *Homo_sapiens.GRCh37.75.gtf*

Check your results against: *Homo_sapiens.GRCh37.75.cdna.all.fa.gz*

Hint: do not unzip the file, use the class *java.util.zip.GZIPInputStream* to directly iterate over the fasta entries.

Task 2 (Simple read simulator):

The results of RNA-seq experiments are the sequenced reads in FASTQ format
(see: `https://en.wikipedia.org/wiki/FASTQ_format`).

For paired-end experiments the experiment results in two files containing the read pairs in the same order.

Implement a simple read simulator considering the following parameter:

- read length (RL)
- fragment length distribution. The input is the mean (FLM) and standard deviation (FLSD) of the normal distribution. This must be used to draw fragment lengths (you have to round the drawn numbers). For simplicity re-draw the fragment length if the obtained value is smaller than RL or larger than the transcript length.
- list of transcript id, count pairs: (input.counts). For the number of fragments to simulate for a transcript, draw a number from a normal distribution with mean *count* and standard deviation 0.1 *count* and round it. For each read, select a fragment length (see previous step), and draw a start position from a uniform distribution of all possible start positions on the transcript.
- mutation rate (MR): probability for a simulated base position to be changed from the original.

The simulator should write three files, two for the simulated paired-end sequences in fastq format (one fastq file for the first fragment, one for the second, set the quality score to the maximum for all bases), and a tab separated file (simulmapping.info) with the following headers:

- readid: integer simply incrementing from 0
- chr: chromosome
- gene_id
- transcript_id
- fw_regvec: genomic region vector for the forward read
- rw_regvec: genomic region vector for the reverse read
- fw_regvec_transcript: region vector for the forward read in transcript coordinates
- rw_regvec_transcript: region vector for the reverse read in transcript coordinates
- fw_mut: mutated positions in the forward read (, separated integer list)
- rw_mut: mutated positions in the forward read (, separated integer list)

The format for genomic region vectors is;

```
start1-end1(|startx-endx)+
```

Use your implementation and simulate reads with the following settings:

$RL = 75, FLM = 200, FLSD = 80$, $MR = 0.01$ on *readcounts.simulation*,

Plot the features of your simulation based on your simulmapping.info file:

- fragment length distribution
- mutation distribution
- barplot with the number of reads falling in following categories:
  - all reads
  - number of unspliced reads (fw and rw) (unspliced - the corresponding genomic vector consists only of a single region)
  - number of nonspliced reads with no mismatches
  - number of split reads
  - number of split reads with no mismatches
  - number of split reads with no mismatches where all regions are at least 5 basepairs long

Hint: You can use `org.apache.commons.math3.distribution.NormalDistribution` to sample the fragment length.

Task 3 (Read mapping with STAR + evaluation):

Use star:

```
/home/proj/biosoft/software/STAR_2.3.0e.Linux_x86_64/STAR
```

documentation: `https://code.google.com/p/rna-star/`

with the pre-calculated index directory: *star_index* on your simulated data.

Evaluate the mapping performance of STAR on your simulation by analyzing the output SAM file. We reccommend to use the htsjdk library: `https://samtools.github.io/htsjdk/`.

For the evaluation consider only the primary alignments where both mates are mapped. For all mapped reads in the mapped read pairs assign one of the following categories:

- ok: the mapped region vector equals to the input region vector
- partial: the mapped region vector is a part of the input region vector (i.e. the SAMRecord reports clipping for the remaining part)

- wrong-chr: the read is mapped to a wrong chromosome
- everything else

Create two barplots showing the number of ok/partial/wrong-chr mappings for all categories from Task 2 one for each mates, and barplot showing the overall performance (ok/partial/wrong-chr) of the fragments (i.e. a fragment is ok if both reads are ok, partial if both reads are at least partial, wrong chr if either of the reads is wrong chr).

Hint: Sort the output of STAR by read names using:

```
/home/proj/biosoft/software/samtools-0.1.19/samtools
```

This way you can simultaneously iterate over the STAR output and your simulmapping.info file.

Bonus: Task 4 (Comparative evaluation of different mappers)

You find a list of bam files (sorted by read name) in *bams.list* along with the information how you have to handle them in the performance evaluation. The main difference to Task 3 is that they include transcriptomic mappings, i.e. the reads were aligned against the transcriptome.

For such mappings you have basically two possibilities to compare them to the reference from Task 2: Either you map them from transcriptome positions to genomic positions (flag *convert_to_genomic*) or compare them directly to the transcript id/location in the reference. In the first case you have to read a gtf file to get the transcript coordinates, which you can use to convert a position in transcript coordinates (i.e. 0 to transcript length) to the corresponding position on the genome. In the second case you have to check whether the transcript id and region vector in the read mapping is equal to the reference from Task 2.

Evaluate all entries in *bams.list* and create for all categories from Task 2 a barplot showing for each method the number of ok/partial/wrong-chr mappings, and an extra barplot showing the overall fragment performance as in Task 3 for all methods.