# Lab 2: Simple Comparative studies of DNA-read aligners with simulated reads

In this lab, we will take a look at the performance of several popular short-read aligners in the era of next-generation-sequencing (NGS).

Before doing this, I should state that this cannot work as the benchmark for choosing the best aligner since

- the analysis can be misleading due to the choice of the genome as well as the read length;
- we only use the default settings without tuning for the parameters for these aligners.

Instead, our intention is just to provide hints for some of the practical issues when conducting analysis of the available tools.

We will choose the following aligners for conducting comparisons:

1. bwa (both `bwa mem` and `bwa aln`)
2. Bowtie2
3. SOAPaligner
4. SubRead
5. STAR
6. blast

In fact, STAR is an RNA-Seq aligner, and we mix this tool here to work as the baseline.

In addition, you need to pre-install the following tools for assistance of the analysis:

1. samtools
2. bedtools
3. EMBOSS

Of course, you can include the other tools for comparison, like BLAST. But to keep in mind that you should convert the output into SAM format for comparison. For example, you need to use the tool, Blast2Bam, in order to transform the BLAST output into SAM records. Or if you use NCBI BLAST+ newer than 2.2.31, it can produce SAM result by setting `-outfmt=15`.

When doing the analysis, you need to specify the version of the above tools, since different version will produce a somewhat different result.

## 1. Genome preparation

In order to save time, here we will choose some of the smaller genomes, like some bacteria and some simple eukaryotes such as *Arabidopsis*.

The first thing you need to do is to download your genome of interest (GOI) from either NCBI-ftp or ENSEMBL-ftp site and build the index for the above tools, respectively.

If you don't know how to create the index, please turn to their documentations for help.

## 2. Reads simulation

The next step is to generate the pseudo-reads that uniformly cover the GOI in user-selected read lengths and intervals. And then output them in **FASTA format**. Here is an example:

```
>1-0-50

CCCTAAACCCTAAACCCTAAACCCTAAACCTCTGAATCCTTAATCCCTAA

>1-10-60

TAAACCCTAAACCCTAAACCTCTGAATCCTTAATCCCTAAATCCCTAAAT

>1-20-70

ACCCTAAACCTCTGAATCCTTAATCCCTAAATCCCTAAATCTTTAAATCC
```

Here the reads are named in the format of **CHROM-START-END** for the convenience of further analysis of accuracy.

This can be done by chaining some awk and bedtools commands:

```
#!/bin/bashCHR=Arabidopsis_thaliana.TAIR10.23.dna.genome_fmt.g GENOMEFA=Ara
bidopsis_thaliana.TAIR10.23.dna.genome_fmt.fa

READLENGTHRANGE='50 100 200'STEP=10PFX=simDNA_

for RDLEN in $READLENGTHRANGEdo

    echo $RDLEN

    bedtools makewindows -w $RDLEN -s $STEP -g $CHR \

        | awk -v L=$RDLEN '($3-$2)==L' \

        | bedtools getfasta -fi $GENOMEFA -bed /dev/stdin -fo /dev/stdout \

        | tr ':' '-' | gzip > ${PFX}${RDLEN}bp.fasta.gz &donewait
```

You can change the settings by modifying the PARAMETERS like *CHR*, *GENOMEFA*, *READLENGTHRANGE*, *STEP* and *PFX*.

# 3. Reads mutations

Next you need to incorporate mutations into the reads with msbar, a neat little EMBOSS tool which introduces mutations (SNP/insertions/deletions) into sequences at a pre-set rate.

```
#!/bin/bashfor FQZ in simDNA_50bp.fasta.gz simDNA_100bp.fasta.gz simDNA_200b
p.fasta.gzdo

    for COUNT in 1 2 4 8 16

    do
```

```
BASE=`echo $FQZ | sed 's/.fasta.gz//'`

gzip -cd ${FQZ} \
    | msbar -sequence /dev/stdin -count $COUNT \
    -point 4 -block 0 -codon 0 -outseq /dev/stdout 2>/dev/null \
    | sed -n '/^>/!{H;$!b};s/$/ /;x;1b;s/\n//g;p' \
    | sed 's/ /\n/' | gzip > ${BASE}_${COUNT}snp.fasta.gz &

gzip -cd ${FQZ} \
   | msbar -sequence /dev/stdin -count $COUNT \
   -point 2 -block 0 -codon 0 -outseq /dev/stdout 2>/dev/null \
   | sed -n '/^>/!{H;$!b};s/$/ /;x;1b;s/\n//g;p' \
   | sed 's/ /\n/' | gzip  > ${BASE}_${COUNT}ins.fasta.gz &

gzip -cd ${FQZ} \
    | msbar -sequence /dev/stdin -point 3 -count $COUNT \
    -block 0 -codon 0 -outseq /dev/stdout 2>/dev/null \
    | sed -n '/^>/!{H;$!b};s/$/ /;x;1b;s/\n//g;p' \
    | sed 's/ /\n/' | gzip  > ${BASE}_${COUNT}del.fasta.gz &

wait
donedone
```

Here we only consider the SNP, 1-base inserts and deletions respectively. But in real-world problems, these may be much more complex. If you like, you can change the settings for msbar to obtain more complicated situations.

# 4. Alignment of the simulated reads against the genome

Here we give a bwa-mem example to illustrate the process:

```
#!/bin/bashREF=/path/to/bwa/index/Arabidopsis_thaliana.TAIR10.23.dna_sm.genome.faf
or FQZ in *fasta.gzdo

    FQ=`basename $FQZ .gz`

    gzip -cd $FQZ | tee $FQ \

        | bwa mem -t 8 $REF - \

        | samtools view -uSh - \

        | samtools sort - ${FQ}_bwamem.sortdone

for BAM in *_bwamem.sort.bam ; do

    samtools index $BAM &done

for BAM in *_bwamem.sort.bam ; do

    samtools flagstat $BAM > ${BAM}.stats &done

wait
```

# 5. Analysis of the results

To determine the number of correctly and incorrectly assigned reads, use samtools and awk to check the sequence header matched the mapping location.

We set a **MAPQ threshold of 10** which is one that is commonly used.

Have a look at how these aligners fared with unmutated, perfect reads, and also the mutated ones. Compare their performance using the following metrics in the form of either tables or diagrams:

- (1) **efficiency**: running time
- (2) **accuracy**: accurately mapped vs. inaccurately mapped
- (3) **robustness** to mutations
- (4) unmapped vs. mapped

# 6. Discussion

Comments on the above results and also, both the advantages and limitations of our approach.