Notes and Workflow for SATAY analysis

Contents

***Author**: Gregory van Beek*

Laanlab for Bionanoscience, Delft University of Technology

***Date**: 04-09-2020*

!File under construction!

Summary

Here the general outline of the experimental methods and interpretation of the results using SAturated Transposon Analysis in Yeast (SATAY) is discussed. The introduction explains the purpose of the experiments and what kind of results are expected and how to interpret these results. The Methods and File types section explains the different kind of files used during processing and analysis and how to read and use them. It also explains some custom made software for analyzing the data. This software is stored in the Github page. For processing the data, a workflow is created using different software tools. A detailed step-by-step tutorial on how to use the different software packages and what steps need to be taken for the processing to go from the raw sequencing reads to the location of all the transposon insertion and the number of reads at each location in included at the end.

Introduction

About 20% of the genes in wild type *Saccharomyces Cerevisiae* are essential, meaning that they are required for vital cellular functions and thus cannot be deleted without crippling the cell to such an extent that it either cannot survive (lethality) or proliferate (sterility). Essentiality of genes is not constant between different genetic backgrounds. The underlying mechanism for this change in essentiality is a reorganization of the genetic interaction network. For wild type (WT) cells, an extensive interaction network already exists, although still incomplete [Costanzo et.al. 2016]. But for mutants, such an interaction map is not yet available and therefore the question how exactly the interaction network changes after perturbing one of more genes remains unanswered. With a new experimental technique in yeast we can measure the essentiality for all genes in different genetic backgrounds. Using these results we aim to create an interaction network for multiple backgrounds and with this information we want to create a machine learning algorithm that can predict how the network changes for other backgrounds (see for current progress the github page for machine learning).

To check the essentiality of genes a technique called transposon sequencing is applied that has been used in bacteria many times before (e.g. galaxyproject website). We use an adapted version of this technique that first has been used in *Schizoaccharomyces Pombe* [Guo et.al. 2013] and has recently been applied in *Saccharomyces Cerevisiae* [Michel et.al. 2017] [Segal et.al., 2018]. Transposon sequencing for *S. Cerevisiae* is called SAturated Transposon Analysis in Yeast (SATAY), which uses transposons (small pieces of mobile DNA) that can insert in genes and thereby inhibit the gene function (i.e. it can still be transcribed, but typically it cannot be translated into a functional protein). After a transposon is randomly inserted in the DNA, the growth of the cell is checked. If the cell cannot proliferate, the transposon has likely been inserted in a critical genomic region causing a significant decrease in the fitness. This is done for many cells at the same time. After transposon insertion, the cells are grown and those that have a transposon inserted at a location that causes a significant decrease in their fitness occur less often in population compared with cells with a higher fitness. The DNA of all cells is extracted and, by means of sequencing, the location of the transposon insertion can be determined. Transposons inserted in a vital genomic region will be absent in the sequencing results and regions that important, but not vital, will have less insertions compared with the regions that do not significantly decrease the fitness of the cells. Thus, when the sequencing results of all the cells are aligned to a reference genome, some genomic locations are missing in the sequencing results. These missing locations corresponds to potentially essential genomic regions. The genome of all cells (called the library) is saturated when all possible genomic regions are covered by transposons. In that case all regions of the DNA are checked for their essentiality.

Essentiality

A gene is essential when its function is absolutely required for some vital cellular function. Essentiality can be grouped in two categories, namely type I and type II [Chen et.al. 2016]. Type I essential genes are genes, when inhibited, show a loss-of-function that can only be rescued (or masked) when the lost function is recovered by a gain-of-function mechanism. Typically these genes are important for some indispensable core function in the cell (e.g. Cdc42 in *S. Cerevisiae* that is type I essential for cell polarity). Type II essential genes are the ones that look essential upon inhibition, but the effects of its inhibition can be rescued or masked by the deletion of (an)other gene(s). These genes are therefore not actually essential, but when inhibiting the genes some toxic side effects are provoked that are deleterious for the cells.

Essentiality of genes (both type I and type II), may change between different genetic backgrounds due to a reorganization of the genetic interaction map. For changes in essentiality in different genetic backgrounds, four cases are considered:

1. a gene is **essential** in WT and remains **essential** in the mutant,
2. a gene is **non-essential** in WT and remains **non-essential** in the mutant,
3. a gene is **essential** in WT and becomes **non-essential** in the mutant,
4. a gene is **non-essential** in WT and becomes **essential** in the mutant.

An example is given in the figure below, where an interaction map is shown for WT cells and a possible interaction map for a mutant where both the essentiality and the interactions are changed.

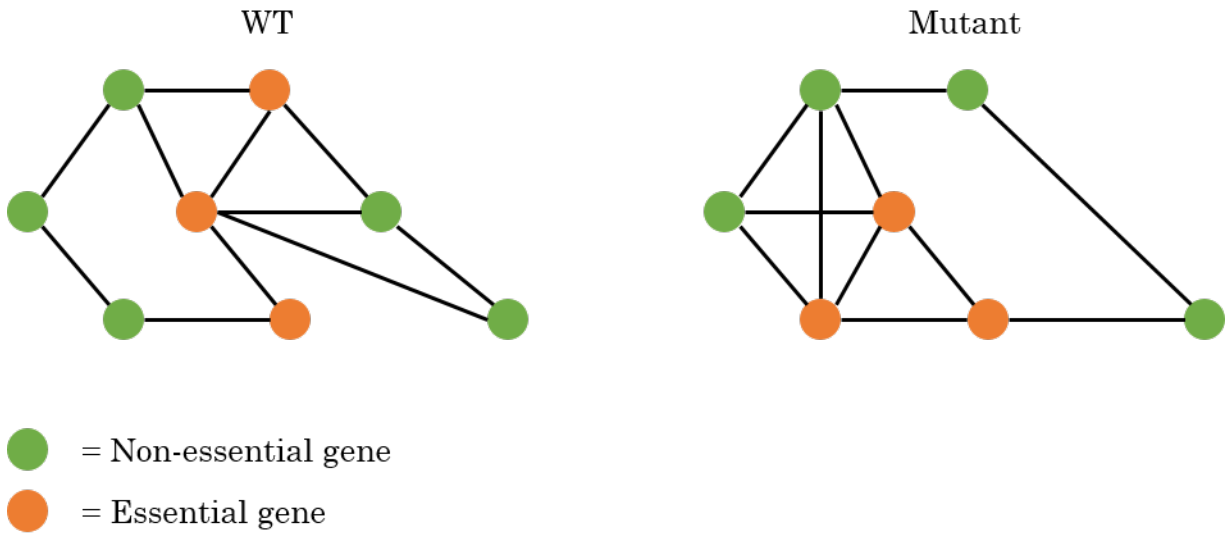


Figure 1: Example interaction network for a WT cell and a mutant with essential and non-essential genes.

It is important to remember that some genes might be essential only in specific growth conditions (see also the next subsection). For example, cells that are grown in an environment that is rich of a specific nutrient, the gene(s) that are requried for the digestion of this nutrient might be essential in this condition. The SATAY experiments will therefore show that these genes are intolerant for transposon insertions. However, when the cells are grown in another growth condition where other

nutrients are present, the same genes might now not be essential and therefore also be more tolerent to transposon insertions. It is suggested to compare the results of experiments with cells from the same genetic background grown in different conditions with each other to rule out conditions specific results.

Because it is not straightforward to label a gene as either essential of non-essential, instead it might be better to define essentiality as continuous variable instead as a boolean. Essentiality will then be a value between 0 and 1 where an essentiality of 0 indicates that the gene is not essential, hence deleting this gene does not have any influence on the cells fitness, and a value of 1 indicates an critical gene that after deleting the cell cannot survive and/or proliferate. This can be determined by measuring the relative number of transposons.

Interpreting Transposon Counts & Reads

Once cells have a transposon inserted somewhere in the DNA, the cells are let to grow so they can potentially generate a few new generations. A cell with a tranposon inserted in an essential part of its DNA grows very slowly or might not grow at all (due to its decreased fitness). Since the sequencing starts only at the location of a transposon insertion (see experimental methods section), the location where each read maps in the reference genome corresponds with the location where the transposon has been inserted. Cells with a transposon inserted in an essential genomic region, will not have divided and therefore does not contribute to the sequencing reads. When the sequencing reads are aligned to a reference genome, missing regions in the alignment might correspond with essential parts. Negative selection can thus be found by looking for empty regions in the reads mapping. When a transposon is inserted in a non-essential genomic region, these cells can still divide and proliferate and after sequencing the non-essential regions will be represented by relatively many reads.

During processing the genes can be analyzed using the number of transposon insertions per gene (or region) or the number of reads per gene. Reads per gene, instead of transposons per gene, might be a good measure for positive selection since locations can be distinquished that have more reads then expected (e.g. more reads relative to the background). It is also more sensitive (bigger difference in number of reads between essential and non-essential genes), but also tends to be nosier. Transposons per gene is less noisy, but is also less sensitive since the number of transposons inserted in a gene does not change in subsequent generations of a particular cell. Therefore it is hard to tell the fitness of cells when a transposon is inserted a non-essential region solely based on the number of transposon insertions.

For each gene, a survival probability can be defined which tells the chances a cell will survive without this gene. The survival probability S is determined as

$$S = \frac{tn/BPInGene}{tn/BPInBackground}$$

where the number of transposons per basepair in background can be determined as the average over all neutral DNA regions (i.e. no CDS, ARS, telomeres, centromeres, terminators etc.). However, the background is not constant over the entire genome. Instead some locations are more likely to have insertions compared to other regions (e.g. see figure 1 in Michel et.al. 2017). To estimate the background number of insertions only the local neutral regions need to be considered.

Ideally only the transposons inserted in non-essential genomic regions will have reads (since only

these cells can create a colony before sequencing), creating a clear difference between the essential and non-essential genes. However, sometimes non-essential genes also have few or no transposon insertion sites. According to Michel et.al. this can have 4 main reasons.

1. During alignment of the reads, the reads that represent repeated DNA sequences are discarded, since there is no unique way of fitting them in the completed sequence. (Although the DNA sequence is repeated, the number of transposon counts can differ between the repeated regions). Transposons within such repeated sequences are therefore discarded and the corresponding reads not count. If this happens at a non-essential gene, it appears that it has no transposons, but this is thus merely an alignment related error in the analysis process.

2. Long stretches of DNA that are without stop codons, called Open Reading Frames (ORF), typically code for proteins. Some dubious ORF might overlap with essential proteins, so although these ORF themselves are not essential, the overlapping part is and therefore they do not show any transposons.

3. Some genes are essential only in specific conditions. For example, genes that are involved in galactose metabolism are typically not essential, as inhibition of these genes block the cell's ability to digest galactose, but it can still survive on other nutritions. In lab conditions however, the cells are typically grown in galactose rich media, and inhibiting the genes for galactose metabolism cause starvation of the cells.

4. A gene might not be essential, but its deletion might cripple the cell so that the growth rate decreases significantly. When the cells are grown, the more healthy cells grow much faster and, after some time, occur more frequently in the population than these crippled cells. In the processing, it might therefore look as if these genes are essential, but in fact they are not. The cells just grow very slowly.

The other way around might also occcur, where essential genes are partly tolerant to transposons. This is shown by Michel et.al. to be caused that some regions (that code for specific subdomains of the proteins) of the essential genes are dispensable. The transposons in these essential genes are clearly located at a specific region in the gene, the one that codes for a non-essential subdomain. However, this is not always possible, as in some cases deletion of non-essential subdomains of essential genes create unstable, unexpressed or toxin proteins. The difference in essentiality between subdomains in a single protein only happens in essential genes, not in non-essential genes. Michel et.al. devised an algorithm to estimate the likelihood L of a gene having an essential subdomain:

$$L = \frac{\mathrm{d}\ N_{\mathrm{cds}}}{l_{\mathrm{cds}}}$$

where d is the longest interval (in terms of base pairs) between 5 neighboring transposons in a Coding DNA Sequence (cds) ($\geq$ 300 bp), N_{cds} is the total number transposons mapping in the cds ($\geq$ 20) transposons) and l_{cds} is the total length of the CDS. Additionally, it must hold that $0.1l_{cds} \leq d \leq 0.9l_{cds}$.

It is expected that only essential genes cary essential subdomains, and indeed what was found by Michel et.al. that the genes with the highest likelihood were mostly also genes previously annotated as essential by other studies.

Because of the reasons mentioned before, not a simple binary conclusion can be made solely based on the amount of transposon insertions or the number of reads. Instead, a gene with little reads

might be essential, but to be sure the results from other experiments need to be implemented as well, for example where the cells were grown in a different growth conditions. Therefore, SATAY analysis only says something about the relative fitness of cells where a specific gene is inhibited in the current growth conditions.

Genetic interaction Maps

...

Methods and File types

Many essential genes in wild type cells are already known and published. The results from SATAY experiments can be verified using the known essential genes. A protocol for SATAY is introduced by the Kornmann lab [Michel et.al. 2017] using wild type cells (see also the SATAY users website). For fast processing of many experimental results, a tool needs to be developed that automatically converts the raw sequencing data to a list of the number of transposons and reads for each gene, and potentially list some of their properties (e.g. essentiality, GO-terms, interactions etc.). This section discusses briefly a simplified protocol for SATAY and the analysis.

Experimental method summary

SATAY uses two kinds of transposable elements found in maize, the activator transposable element (Ac) and the dissociation transposable element (Ds). Ac is autonomous, meaning that it can express the transposase that is needed to cut loose the transposon. Ds is nonautonomous and cannot express the transposase and thus it needs the Ac for transposition. In yeast cells, the Ds is used (called MiniDs) that is inserted in the Ade2 gene and thereby disrupting this gene. The cells are induced to express the transposase Ac that cuts out the Ds and repairs the Ade2 gene. Only cells in which the Ade2 gene is repaired are able to form colonies again. The excised Ds transposon reinserts randomly in the DNA. However, since the original location of the Ds transposon is the Ade2 gene, the genomic locations around this Ade2 are more likely to have the transposon reinserted compared with other genomic regions (e.g. see figure 1 in Michel et.al. 2017). All cells are then put in media where only cells grow that have a repaired Ade2 gene, so that only the cells where the transposon is not excised from the Ade2 are diluted since they are outcompeted by the cells that have their transposons excised.

(See next figure for the following section). If the excised Ds transposon is inserted in a gene, then this gene can still be transcribed by the ribosomes, but typically cannot be (properly) translated in a functional protein. The genomic DNA (with the transposon) is cut in pieces for sequencing using enzymes, for example DpnII. This cuts the DNA in many small pieces (e.g. each 75bp long) and it always cuts the transposon in two parts (i.e. digestion of the DNA). Each of the two halves of the cut transposon, together with the part of the gene where the transposon is inserted in, is ligated meaning that it is folded in a circle (circularization). A part of the circle is then the half transposon and the rest of the circle is a part of the gene where the transposon is inserted in. Using PCR and primers, this can then be unfolded by cutting the circle at the halved transposon. The part of the gene is then between the transposon quarters. Since the sequence of the transposon is known, the part of the gene can be extracted. This is repeated for the other half of the transposon that includes the other part of the gene. When both parts of the gene are known, the sequence from the original gene can be determined.

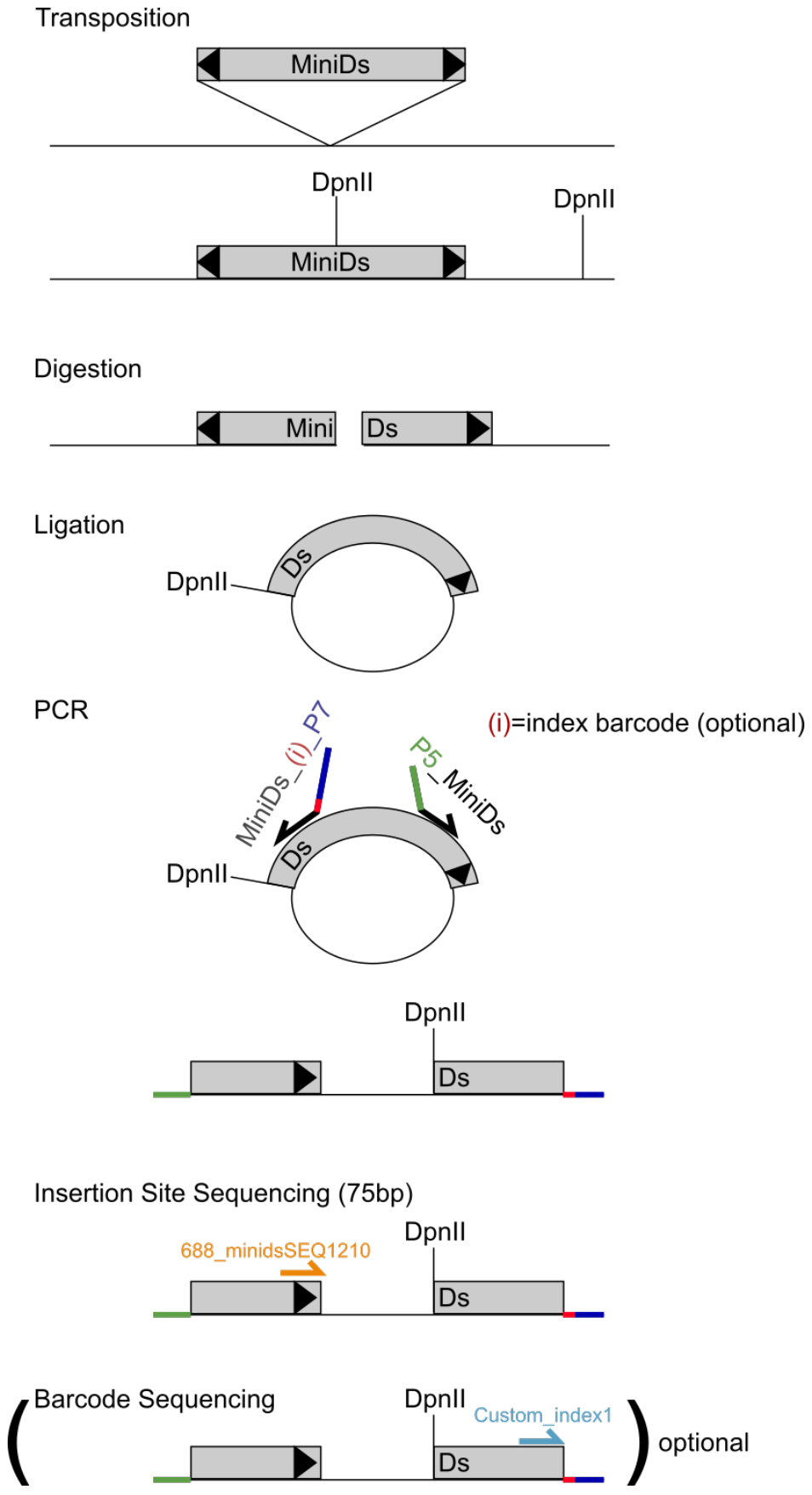


Figure 2: Schematic overview of transposon insertion experiments.

Sequence alignment

To get the order of nucleotides in a genome, shotgun sequencing is used where the genome is cut is small pieces called reads (typically tens to a few hundred basepairs long). The reads have overlapping regions that can be used to identify their location with respect to a reference genome and other reads (i.e. mapping of the reads). Mapping of the reads result in contigs, which are multiple mapped reads that form continuous assembled parts of the genome (contigs can be the entire target genome itself). All contigs should be assembled to form (a large part of) the target genome.

The sequence assembly problem can be described as: *Given a set of sequences, find the minimal length string containing all members of the set as substrings.*

The reads from the sequencing can be single-read or paired-end, which indicates how the sequencing is performed. In paired-end sequencing, the reads are sequenced from both directions, making the assembly easier and more reliable, but results in twice as many reads as single-read reads. The reason of the more reliable results has to do with ambiguous reads that might occur in the single-read sequencing. Here, a read can be assigned to two different locations on the reference genome (and have the same alignment score). In these cases, it cannot be determined where the read should actually be aligned (hence its position is ambiguous). In paired-end sequencing, each DNA fragment has primers on both ends, meaning that the sequencing can start in both the 5'-3' direction and in the 3'-5' direction. Each DNA fragment therefore has two reads both which have a specified length that is shorter than the entire DNA fragment. This results that a DNA fragment is read on both ends, but the central part will still be unknown (as it is not covered by these two particular reads, but it will be covered by other reads). Since you know that the two reads belong close together, the alignment of one read can be checked by the alignment of the second read (or paired mate) somewhere in close vicinity on the reference sequence. This is usually enough for the reads to become unambiguous.

The resulting data from the sequencing is stored in a FASTQ file where all individual reads are stored including a quality score of the sequencing. The reads are in random order and therefore the first step in the processing is aligning of the reads in the FASTQ files with a reference genome.

Note that the quality of the reads typically decreases near the 3'-end of the reads due to the chemistry processes required for sequencing (this depends on the kind of method used). For Illumina sequencing, the main reasons are signal decay and dephasing, both causing a relative increase in the background noise. Dephasing occurs when a DNA fragment is not de-blocked properly. A DNA fragment is copied many times and all copies incorporate a fluorescent nucleotide that can be imaged to identify the nucleotide. If there are 1000 copies of the DNA fragment, there are 1000 fluorescent nucleotides that, ideally, are all the same to create a high quality signal. After imaging, the DNA fragment is de-blocked to allow a new fluorescent nucleotide to bind. This deblocking might not work for all copies of the DNA fragment. For example, 100 copies might not be deblocked properly, so for the next nucleotide only 900 copies agree for the next incorporated nucleotide. For the third round, the 100 copies that were not deblocked properly in the second round, might now be deblocked as well, but now they are lagging behind one nucleotide, meaning that in the coming rounds they have consistently the wrong nucleotide. As the reads increases in length, more rounds are needed and therefore the chances of dephasing increases causing a decrease in the quality of the reads. This gives noise in the signal of the new nucleotide and therefore the quality of the signal decreases. For example, take the next 6bp sequence that is copied 5 times:

1. `GATGTC`
2. `GATGTC`

3. `G ATGT`
4. `GAT GT`
5. `G AT G`

The first two reads are deblocked properly and they give all the right nucleotides. But the third and fourth have one round that is not deblocked properly (indicated by the empty region between the nucleotides), hence the nucleotide is always lagging one bp after the failed deblocking. The fifth copy has two failed deblocking situations, hence is lagging two bp. The first nucleotide is a G for all 5 copies, therefore the quality of this nucleotide is perfect. But, by the end of the read, only two out of five copies have the correct bp (i.e. a C), therefore the quality is poor for this nucleotide. (It can either be a C or a T with equal likelyhood or potentially a G, so determining which nucleotide is correct is ambiguous without knowing which reads are lagging, which you don't know). (See for example this question on seqanswers or the paper by [Pfeifer, 2016])

FASTQ files and FASTA files

The output of sequencing is typically a FASTQ file. This file contains the sequences of all reads (typically 75 to 100 bp long). Each read is represented by four lines:

1. Starts with '@' followed by a sequences identifier and can include some descriptions.
2. Raw sequence letters representing the nucleotides.
3. Starts with '+' for separating the second and third line.
4. Quality score of the sequence represented by ASCII symbols running from '!' (lowest score) to '~' (highest score) [http://www.asciitable.com/]. This is called ASCII-base 33 since '!' has decimal ASCII number 33 and is defined as Q-score 0. This typically runs towards ASCII symbol 'J' (number 74, Q-score 41). The error probability can be calculated based on the Q-score using $P_{error} = 10^{-\frac{Q}{10}}$. This means that '!' has an error of ($P_{error} = 100\%$) and 'J' an error of $P_{error} = 0.008\%$). Typically a Q-score higher than Q=20 (ASCII symbol '5', ($P_{error} = 1\%$)) is acceptable [https://drive5.com/usearch/manual/quality_score.html]. This line has the same length as the second line.

A FASTQ file can be used as an input for sequence alignment using a reference genome. The result of sequence alignment is a SAM file.

Besides FASTQ files, FASTA files are also used for alignment. These are similar to FASTQ files, but do not include the quality string and therefore FASTA files can be created from FASTQ files by removing line 3 and 4 from each read. Depending on the sequencing method, FASTA files may be directly given.

SAM and BAM files

The FASTQ (or FASTA) files contain all the reads in a random order. To determine where each belong relative to a reference sequence, the reads need to be aligned. After alignment of the reads, the results are typically represented in a Sequencing Alignment Mapping (SAM) file. For processing purposes this is typically translated to a Binary Alignment Mapping (BAM) file. This is a compressed, binary version of the SAM file.

The SAM files contain all the short reads from the FASTQ file together with information where and how well the read mapped to a reference sequence. The reads are represented by a header and a

body. The header consists of four fields (note that the headers can be different depending which program made the alignment, see for example cpwardell.com):

1. @HD lines: version number of SAM specification and how the file is sorted (e.g. sorting by genomic coordinates).
2. @SQ: Which reference sequence has been used to align the sequence. There will be one line for every chromosome (contig). It also tells the sequence name (SN) and length (LN).
3. @RG: Read group line with the tags specifying the origins of the input data.
4. @PG: Which programs and commands were used to create the SAM file.

The body of the SAM file contains the aligned data. Every row contains one read. If there is paired data (i.e. a forward reading and a backward reading), then the pair is divided in two rows. Every row consists of at least 11 columns:

1. `QNAME` Name of the query sequence.
2. `FLAG` Bitwise flag. This consists of twelve binary properties. A read typically has multiple flags. These flags are all then translated to a decimal number, given by the third column, and these decimal numbers are added up. Typical values are 99, 147, 83 or 163. To get a proper translation, use the SAM flag decoder. The following flags can be used:
    1. 000000000001 : 1 : read paired
    2. 000000000010 : 2 : read mapped in proper pair
    3. 000000000100 : 4 : read unmapped
    4. 000000001000 : 8 : mate unmapped
    5. 000000010000 : 16 : read reverse strand
    6. 000000100000 : 32 : mate reverse strand
    7. 000001000000 : 64 : first in pair
    8. 000010000000 : 128 : second in pair
    9. 000100000000 : 256 : not primary alignment
    10. 001000000000 : 512 : read fails platform/vendor quality checks
    11. 010000000000 : 1024 : read is PCR or optical duplicate
    12. 100000000000 : 2048 : supplementary alignment
3. `RNAME` Name of the reference contig (chromosome) where the sequence is aligned to (i.e. which chromosome the read is aligned to).
4. `POS` Position of the reference contig that the alignment starts at (given in terms of base pairs).
5. `MAPQ` Mapping quality. Number indicating the chances that the mapping is wrong, based on phred scaling. This is logarithmic scaled where 60 is typically the maximum score meaning that the chance of a wrong mapping is the smallest (so a high number is better). If a value of 255 is shown, that means that the quality is not determined.

6. `CIGAR` Tells how to match the query sequence to the reference sequence using a 'Compact Idiosyncratic Gapped Alignment Report' (CIGAR) string. This contains a sequence of integers and letters. Possible letters are M (Match), N (Alignment gap), D (Deletion) or I (Insertion). Thus 76M means that 76 basepairs match the reference sequence (see JEFworks for more information).
7. `RNEXT` The name of the reference contig (chromosome) that the other read in the pair (i.e. the next or previous line?) aligns to. If the two reads in the pair aligns to the same contig, an '=' sign is used.
8. `PNEXT` Position on the contig where the other read in the pair aligns to. Depending on the size of the DNA fragments in the sequencing library, this is typically a few hundred base pairs away from the current read (i.e. given by the POS column).
9. `TLEN` Total length of the template fragment. This is the distance from the leftmost base of the first read to the rightmost base pair of the second read in the pair. This value is assigned a '+' or '-' to indicate the reading orientation.
10. `SEQ` The DNA sequence of the query sequence. (Identical to the sequence in the FASTQ file that was aligned to the reference genome).
11. `QUAL` Base quality score of the SEQ. (Identical to the scores in the FASTQ file). There are 42 scores, each of which are related to a specific error. See for example phred score conversion table for a conversion table.

Determine essentiality based on transposon counts

Using the number of transposons and reads, the essentiality can be determined. Currently, genes that are taken as essential are the annotated essentials based on previous research (e.g. see the Github page of this research).

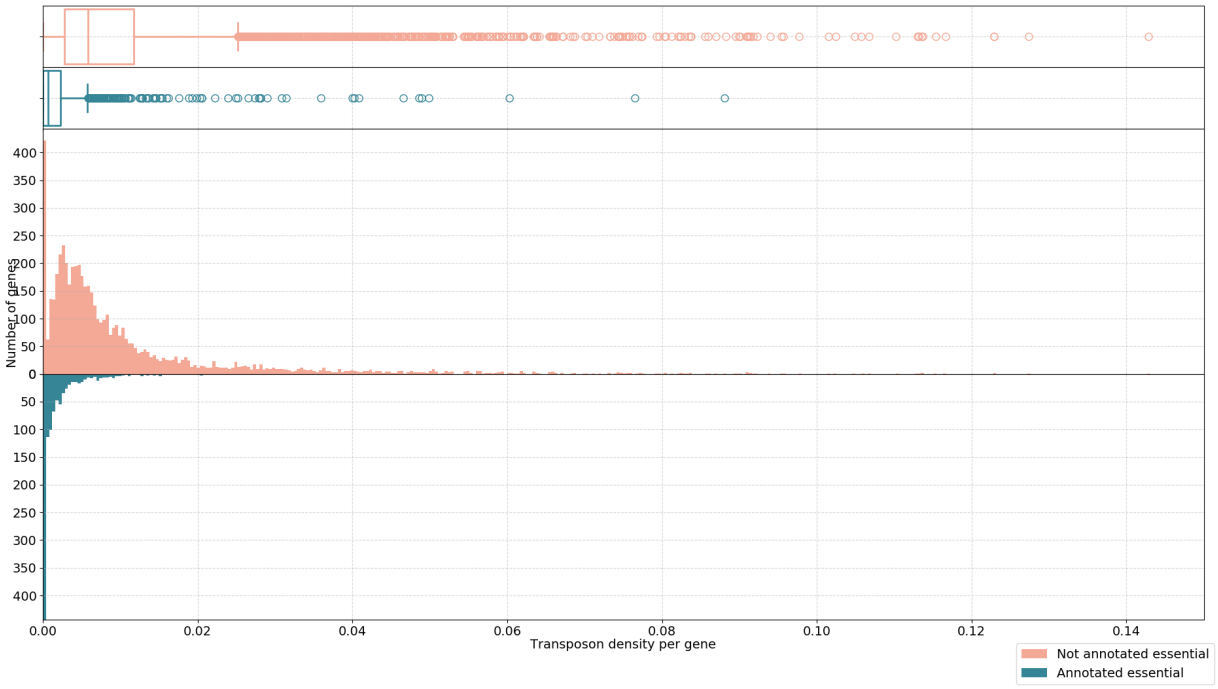
Distribution number of insertions and reads compared with essential and non-essential genes

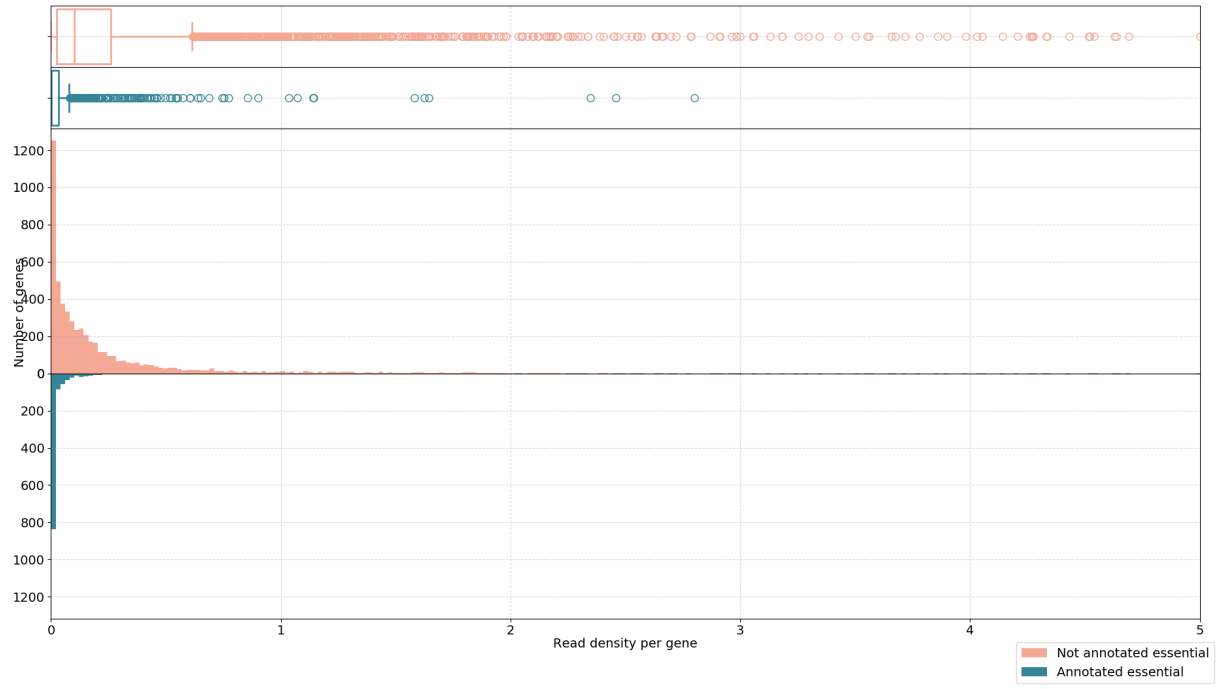
(*The figures in this section are created using* `analyze_pergene_insertions.py` *and* `statistics_pergene.py`)

Ideally, the number of transposon insertions of all essential genes are small and the number of insertions in non-essential genes is large so that there a boundary can be defined that seperates essential from non-essential genes. However, this is not always so clear. For example, the distribution of transposons in WT cells in the data from Michel et. al. looks like this:

In these figures, both the reads and the transposon counts are normalized with respect to the length of each gene (hence the graph represents the read density and transposon density). High transposon counts only occurs for non-essential genes, and therefore when a high transposon count is seen, it can be assigned nonessential with reasonable certainty. However, when the transposon count is low the there is a significant overlap between the two distributions and therefore there is no certainty whether this gene is essential or not (see also the section about 'Interpreting Transposon Counts & Reads').

The data is also sensitive to postprocessing. It is expected that the trimming of the sequences is one of the important steps. When similar graphs are made using different processing steps on the same dataset, the distribution of the transposon and read density can look significantly different.





Profile plot for number of reads

(*See* `TransposonRead_Profile_Plot.py`)

To create a visual overview where the insertions are and how many reads there are for each insertion, a profile plot is created for each chromosome.

The bars indicate the absolute number of reads for all insertions located in the bars (bar width is 545bp). The colored background indicate the location of genes, where green are the annotated

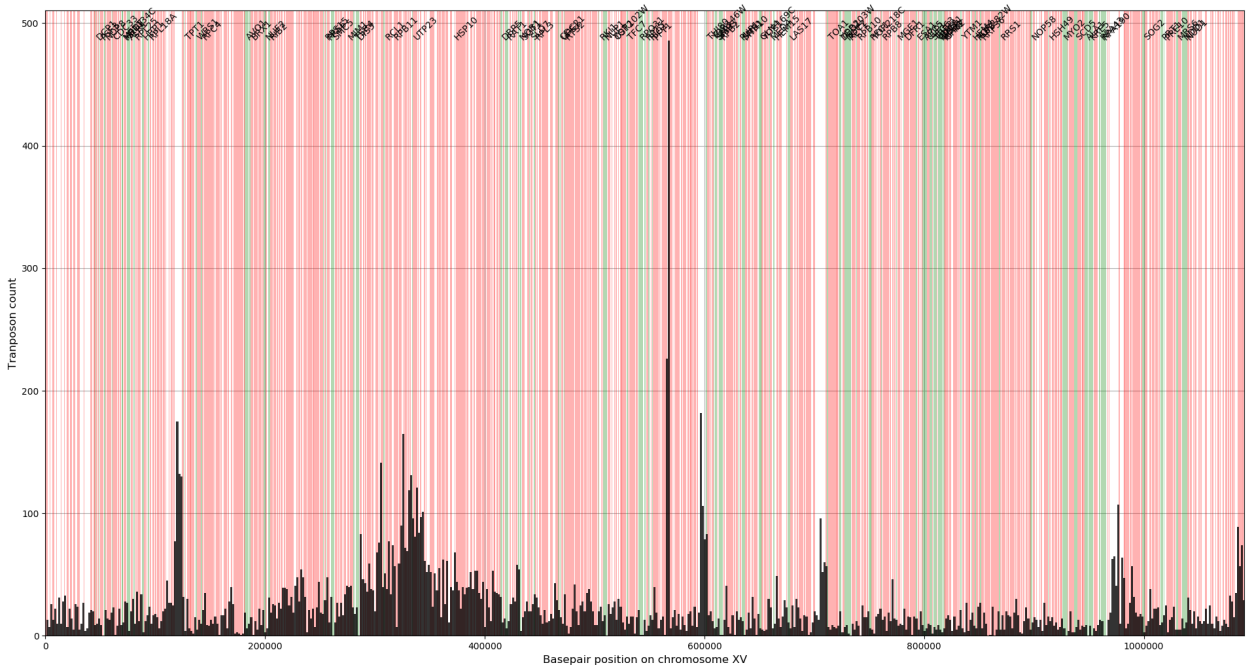


Figure 3: Transposon profile plot for chromosome XV

essential genes and red the non-essential genes. In general, the essential genes have no or little reads whereas the non-essential genes have many reads. Note that at location 564476 the Ade2 gene is located that has significant more reads than any other location in the genome,(see section 'Experimental method summary'). The examples used here are from a dataset discussed in the paper by Michel et.al. 2017 which used centromeric plasmids where the transposons are excised from. The transposons tend to reinsert in the chromosome near the chromosomal pericentromeric region causing those regions to have about 20% more insertions compared to other chromosomal regions.

This figure gives a rough overview that allows for identifying how well the data fits the expectation. Also an alternative version of this plot is made (`TransposonRead_Profile_Compare.py`) that makes a similar plot for two datasets at the same time, allowing the comparison of the two datasets with each other and with the expectation.

Profile plot number of reads per individual genes

(*See* ***gene_reads.py***)

Instead of plotting the number of reads for an entire chromosome, it is also useful to plot the read profile for individual genes to see how the insertion sites and reads are distributed within a gene. For this a bar plot is made where the number of reads per transposon are determined. This also shows the distribution of the distances between subsequent transposon insertions for both the gene and the chromosome the gene is located. It is expected that for essential genes, the median distance between subsequent insertions is larger compared to the distance in the entire chromosome (since important genes have few insertions and large parts will be free of insertions). For non-essential genes, the distribution for the distance between insertions is expected to follow the distribution of the chromomsome more closely.

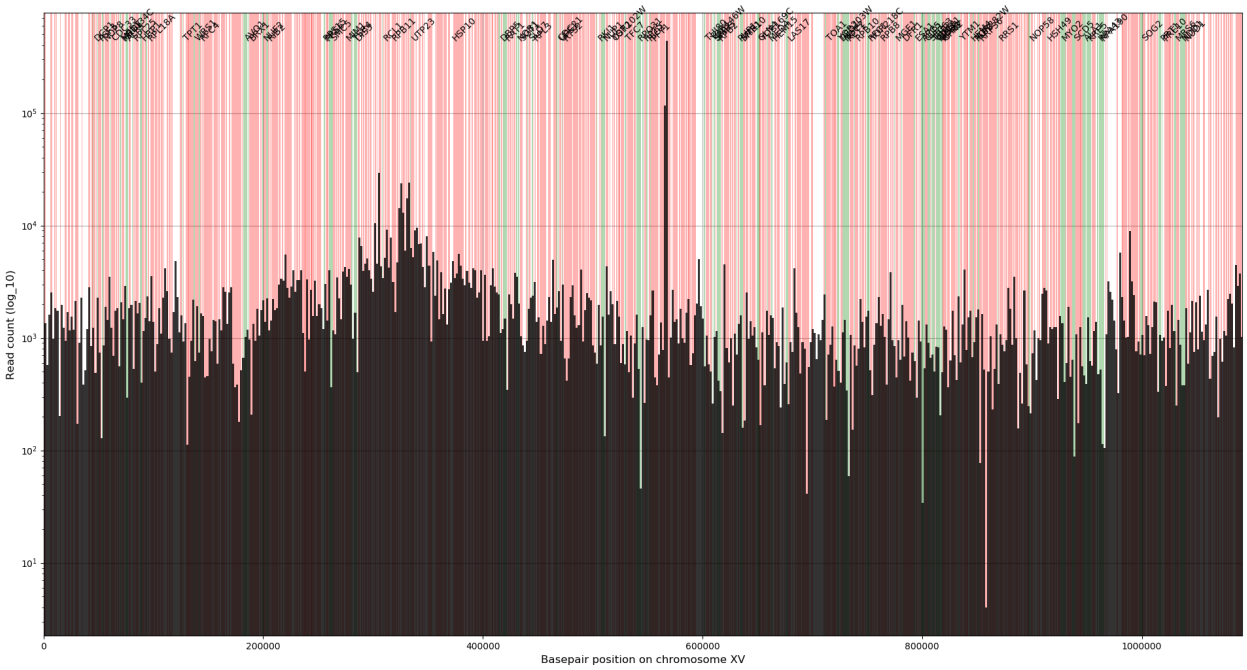


Figure 4: Read profile plot for chromosome XV (note the y-axis is in logarithmic scale).

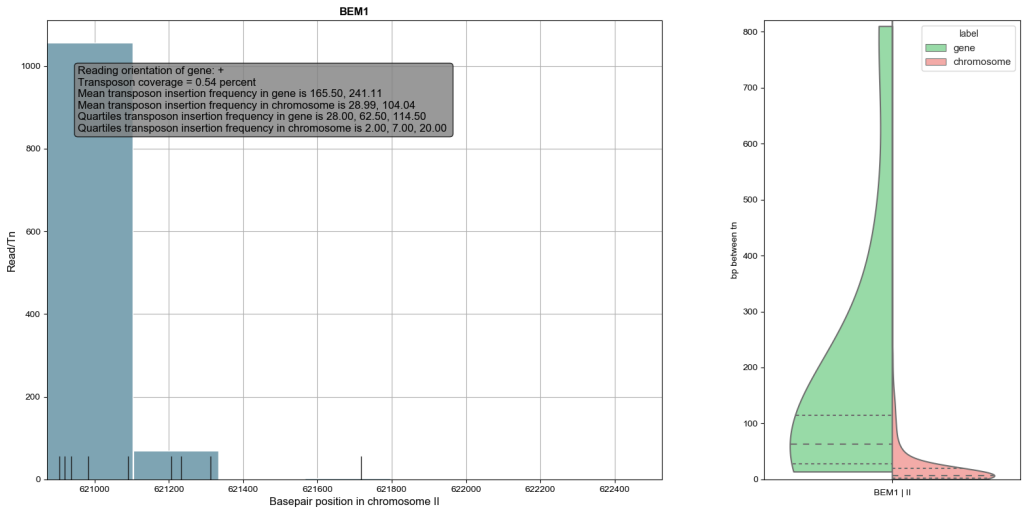


Figure 5: Read per transposon insertion for BEM1. The violin plot gives the distribution for the distance between subsequent insertions for both BEM1 and chromosome II (where BEM1 is located). The small black bars indicate individual transposon insertions.

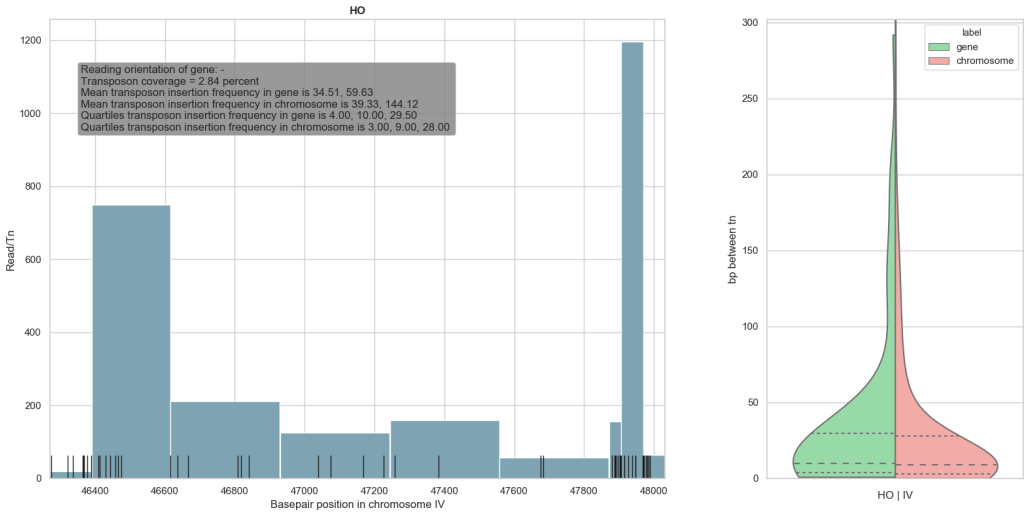


Figure 6: Read per transposon insertion for the HO-locus. Note that the distribution for the distance between insertions follows the distribution for the chromosome more closely compared to BEM1.

The minimum width of the bars are chosen such that each bar should contain 8 transposon insertions. The maximum width is set equal to the length such that the probabilty of finding at least one insertion is more than 99% in the whole chromosome. This is chosen because if now a bar is empty in a gene than this is not a coincidence, but this is an interesting region.

These plots can be used for checking if a gene has transposon free regions which might indicate that this gene is essential.

Preprocessing; From raw sequencing data to number of insertions and reads per genomic region

SATAY experiments need to be sequenced which results in a FASTQ file. The sequence reads from this file needs to be aligned to create a SAM file (and/or the compressed binary equivalent BAM file). Using the BAM file, the number of transposons can be determined for each insertion location.

For guides and manuals of the software discussed below, see the Github page for this project.

For testing, the raw data (.FASTQ files) discussed in the paper of Michel et.al. 2017 is used and can be found here.

This section describes how to use the software for preprocessing. To make this more efficient, **it is advised to use an automated workflow that has been created for Linux**. This automated workflow is written in a bash file that performs all steps in the *processing pipeline* described in the next subsection. The workflow in the bash file starts with a section with user defined options in which the name of the fastq-file has to be given and the options can be set that are described in the following sections. It requires the fastq file to be located in the shared folder of the Linux machine where the workflow gets the fastq file to perform all processing steps and, when finished, returns the file together with the output files back to the shared folder. This bash script is located on the desktop

of the Linux Virtual Machine. This Virtual machine can be found on the N-Drive/VirtualMachines. This N-drive folder also contains a readme about how to use the virtual machine and how to run the bash script. When you want to use this VirtualMachine, **please make a copy of the .vhd file, do not use the vhd file directly from the VirtualMachine folder!** You can make your own copy on the N-drive, but when using it make sure you have a very stable internet connection, preferably using an internet cable. To start the Virtual machine, first download VirtualBox. The InstallationGuide (which can also be found in the N-Drive/VirtualMachines) explains how to get started with the virtual machine. The readme file on the N-drive/VirtualMachines folder should be enough to get you started with the processing, but for a more detailed explaination about the used software or how to perform the processing without the workflow, see the following subsections.

Workflow

The results from the sequencing is typically represented in FASTA or FASTQ format. This needs to be aligned according to a reference sequence to create a SAM and BAM file. Before alignment, the data needs to be checked for quality and possibly trimmed to remove unwanted and unnecessary sequences. When the location of the reads relative to a reference sequence are known, the insertion sites of the transposons can be determined. With this, a visualization can be made that shows the number of transposon insertions per gene.

In the description given in this document, it is chosen to do the quality checking and the trimming in windows and the alignment of the reads with a reference genome in a virtual machine running Linux. It is possible to do the quality checking and trimming in Linux as well or to do the whole process in windows. To do quality checking and/or trimming in Linux requires more memory of the Linux machine since both the raw sequencing data needs to be stored and the trimming needs to be stored (both which can be relatively large files). Since a virtual machine is used, both the computation power and the amount of storage is limited, and therefore it chosen to do the trimming on the main windows computer (this problem would not exists if a computer is used running on Linux, for example a computer with an alternative startup disc running Linux). Sequence alignment can be done on Windows machines (e.g. using BBmap), but this is not ideal as many software tools are designed for Unix based computer systems (i.e. Mac or Linux). Also, many tools related to sequence alignment (e.g. converting .sam files to .bam and sorting and indexing the bam files) are done with tools not designed to be used in windows, hence this is performed in Linux.

An overview of the different processing steps are shown in the figure below.

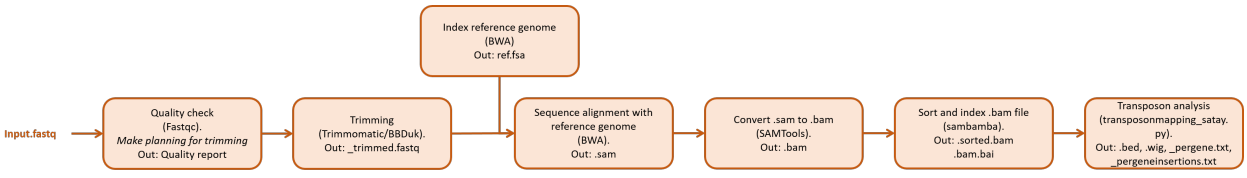


Figure 7: Processing pipeline. Input is a file containing the raw reads from the sequencing saved in a .fastq file.

A short overview is given for different software tools that can be used for processing and analyzing the data. Next, a step-by-step tutorial is given as an example how to process the data. Most of this is done using command line based tools.

An overview of the different steps including some software that can handle this is shown here:

1. Checking the raw FASTA or FASTQ data can be done using the (**FASTQC**) software (Windows, Linux, Mac. Requires Java). This gives a quality report (see accompanying tutorial) for the sequence reads and can be run non-interactively using the command line.

2. Based on the quality report, the data needs to be trimmed to remove any unwanted sequences. This can be done with for example (**FASTX**) (Linux, Mac) or (**Trimmomatic**) (Windows,requires Java). An easy graphical user interface that combines the FASTQC and Trimmomatic is (**123FASTQ**). Also **BBDuk** can be used for trimming (which belongs to BBMap).

3. The trimmed sequence reads need to be aligned using a reference sequence, for example the *S. Cerevisiae* S288C Ref64-2-1 reference sequence or the W303 reference sequence from SGD. Aligning can be done, for example, using (**SnapGene**) (Windows, Linux, Mac. This does not import large files and is therefore not suitable for whole genome sequencing), (**BBMap**) (Linux, Mac, Windows (seems to give problems when installing on windows machines), might be possible to integrate it in Python, (**BWA**) (Linux, Mac), (**Bowtie2**) (Linux, Mac) or (**ClustalOmega**) (Windows, Linux, Mac). This step might require defining scores for matches, mismatches, gaps and insertions of nucleotides.

4. After aligning, the data needs to be converted to SAM and BAM formats for easier processing hereafter. This requires (**SAMtools**) (Linux, Mac) or (**GATK**) (Linux, Mac). Conversion from SAM to BAM can also be done in Matlab if preferred using the 'BioMap' function.

5. Using the BAM file with the aligned sequence reads, the transposon insertion sites can be determined using the **Matlab** script given by Benoit Kornmann Lab (including the name.mat and yeastGFF.mat files). The results from this step are three files (a .txt file, a .bed file and a .wig file) that can be used for visualization.

6. If more processing is required, (**Picard**) (Linux, Mac) might be useful, as well as (**GATK**) (Linux, Mac). Visualization of the genomic dataset can be done using (**IGV**) (Windows, Linux, Mac) or SAMtools' tview function. Also (**sambamba**) (Linux, Mac) can be used, especially for sorting and indexing the bam files.

7. Creating transposon insertion maps for the genome (see the satay users website) and comparison essential genes between different genetic backgrounds using Venn diagrams, customized software needs to be created.

0. Initializing

The steps discussed in this section are not obligatory, but might help in organizing the data. The **bold** printed commands in this and the following sections are put so that they can be copied directly to the bash. (Note to modify the respective paths on your own machine in this Initialization step, though). If the paths below are correctly defined, the boldface commands defined in the different processing steps below can be literally copied and pasted in the bash.

Since this protocol works mostly in the Linux Virtual Machine, the commands below are defined for Linux. First, a possible method for organizing the data is shown, but feel free to change this as you like.

1. Create an empty datafolder.

2. Add the .fastq file to the datafolder.

3. Add the following empty folders for the outcomes of the different processing steps:

   A. filename_QC

   B. filename_Trimmed

   C. filename_Aligned

4. When this is done in Windows, copy the datafolder to the shared folder for processing in the Virtual Machine using the commands (using git-bash)

```
#!/bin/bash

pathwin_sharedfolder='/C/Users/gregoryvanbeek/Documents/ VirtualBox VMs/ VMSharedFolder_Ubuntu6

pathwin_data='C:\Users\gregoryvanbeek\Desktop\Cerevisiae_WT2_Seqdata_Michel2017\Cerevisiae_WT2_

cp -r ${pathwin_data} "${pathwin_sharedfolder}"
```

5. The main processing is done in the Linux Virtual Machine. Since the software tools are mostly commandline based, it might be convenient to be able to define variables, for example for the paths to the difference programs and files so that these do not have te be entered every single time. For this start with the command that enables defining variables **`#!/bin/bash`**.

6. Define the following variables. (copy paste the commands in the bash using `shift+Ins`. Remember to first alter the respective variables given below to the paths and filenames in your computer):

   A. Path to the shared folder used for communicating with the virtual machine running Linux (**`path_sf=/media/sf_VMSharedFolder_Ubuntu64_1/`**).

   B. Name of the folder containing the data (**`foldername='Cerevisiae_WT2_Seqdata_Michel2017_ProcessingTest'`**)

   C. Name of the data file (**`filename='Cerevisiae_WT2_Michel2017.fastq'`**)

   D. Name of the trimmed data file (**`filename_trimmed='Cerevisiae_WT2_Michel2017_trimmed.fastq'`**)

   D. The processing can be performed in shared folder, but this is not recommended. It is better to move the folder temporarily to the hard drive of the Virtual Machine. For this define the location where the processing is performed (in this example located in the Documents directory) (**`pathdata=~/Documents/data_processing/${foldername}`**) If this directory does not already exists, create it using the command `mkdir ${pathdata}`

   E. Move the datafolder from shared folder to data processing folder (**`mv ${path_sf}${foldername} ${pathdata}`**)

   F. Path to the location where the Trimmomatic software is located (**`path_trimm_software=~/Documents/Software/Trimmomatic-0.39/`**).

   G. Path to the location where the BBDuk software is located (**`path_bbduk_software=~/Documents/Software/BBMap/bbmap/`**)

   H. Path to the outcome folder for the fastqc software (**`path_fastqc_out=${pathdata}/Cerevisiae_WT2_Michel2017_QC/`**).

   I. Path to the outcome folder for the trimmomatic software (**`path_trimm_out=${pathdata}/Cerevisiae_WT2_Michel2017_Trimmed/`**).

J. Path to the outcome folder for the aligned software (**path_align_out=${pathdata}/Cerevisiae_WT2_Michel2017_Aligned/**).

K. Path to the reference genome fasta file (**path_refgenome=/home/gregoryvanbeek/Documents/Reference_Sequences/Reference_Sequence_S288C/S288C_reference_sequence_R64-2-1_20150113.fsa**

Some useful commands:

1. `echo`: Print a variable name or some text.
2. `gunzip`: Unzip a .gz file.
3. `bunzip`: Unzip a .bz file.
4. `${}`: when using a variable, the name of the variable should be placed between curly brackets and should start with a dollar sign (`$`), otherwise the bash won't recognize the name as a variable.
5. `less`: open the first few lines of a files that can be read as a text file.
6. When using or defining strings of texts, putting the string between accolades (‘) tells the bash to take the text within the accolades literally. Remember this when using the variables, as `'${var}'` is literally taken as the string *varwhereaswhenusing*‘{var}‘ (without accolades) the bash will try implement the variable 'var' depending on what you have defined before for this variable.
7. Copying texts in the bash does not work using `crtl-v`, instead use `shift-Insert`

A suggestion is to put all the commands with the correct paths in a text file and store this text file together with the raw sequencing file ('Linux_Processing_Commands.txt' located at the SATAY github repository). The variables containing the paths and names in this file can be changed to point at the right files and directories in your computer. Then all the commands can be easily copy-pasted in the bash and all the processing steps can be traced back later using this text file.

1. Quality checking of the sequencing reads; FASTQC (0.11.9)

FASTQC creates a report for the quality of sequencing data. The input should be a fastq (both zipped and unzipped), sam or bam file (it can handle multiple files at once). The program does not need to be installed, but after downloading only requires to be unzipped. FASTQC can be ran as an interactive program (i.e. using a GUI) or non-interactively using the command line options.

If using interactively, open the 'run_fastqc.bat' file in the FASTQC folder and load a file to be checked. Alternatively using the 123FASTQ (version 1.1) program, open this and use the 'Quality Check' menu on the right. The advantage of using 123FASTQ is that it can also do trimming (using Trimmomatic).

If using the command line for checking a single file use the command:

```
fastqc --outdir ${path_fastqc_out} ${pathdata}/${filename}
```

(Note that the output directory should already exist, as the program does not create directories). In the output directory, a .html file and a (zipped) folder is created, both with the same name as the input file. The .html file can be used to quickly see the graphs using a browser. Also, a zipped folder is created where the raw data of the quality check is stored. For explanation about the different graphs, see the fastqc_manual pdf or [https://www.bioinformatics.babraham.ac.u

k/projects/fastqc/Help/3%20Analysis%20Modules/] (or the paper 'Preprocessing and Quality Control for Whole-Genome' from Wright et.al. or the 'Assessing Read Quality' workshop from the Datacarpentry Genomics workshop).

For more commands, type `fastqc --help`. Some useful commands might be:

- –contaminants Reads a file where sequences are stored of (potential) contaminants. The .txt-file should be created before running the software. Each contaminant is presented on a different line the text file and should have the form name 'tab' sequence.
- –adapters Similar as the contaminants command, but specifically for adapter sequences. Also here a text file should be created before running and this file should have the same layout as the contaminants file.
- –min_length where a minimal sequence length can be set, so that the statistics can be better compared between different reads of different length (which for example might occur after trimming).
- –threads Preferably leave this unchanged, especially when an error is shown that there 'could not reserve enough space for object heap' after setting this command.
- –extract Set this command (without following of parameters) to extract the zipped folder from the results.

The output of the FASTQC program is:

- **Per base sequence quality**: Box and whisker plot for the quality of a basepair position in all reads. The quality should be above approximately 30 for most reads, but the quality typically drops near the end of the sequences. If the ends of the reads are really bad, consider trimming those in the next step.
- **Per tile sequence quality**: (Shows only when Illumina Library which retains there sequence identifier). Shows a heat map of the quality per tile of the sequence machines. Blueish colours indicate that the quality score is about or better than average and reddish colours indicates scores worse than average.
- **Per sequence quality score**: Shows an accumulative distribution to indicate which mean quality score per sequence occurs most often.
- **Per base sequence content**: Shows the percentage of nucleotide appearance in all sequences. Assuming a perfectly random distribution for all four nucleotides, each nucleotide should be present about 25% over the entire sequence. In the beginning this might be a bit off due to for example adapters present in the sequences. If this is present, it might difficult/impossible to cut these during the trimming part, but it should typically not seriously affect further analysis.
- **Per sequence GC content**: Indicates the distribution of the G-C nucleotides appearances in the genome. The ideal distribution that is expected based on the data is shown as a blue curve. The red curve should, ideally follow the blue curve. If the red curve is more or less a normal distribution, but shifted from the blue curve, is might indicate a systematic bias which might be caused by an inaccurate estimation of the GC content in the blue curve. This does not necessarily indicate bad data. When the red curve is not normal or show irregularities (peaks or flat parts), this might indicate contaminants in the sample or overrepresented sequences.

- **Per base N content**: Counts the number of N appearances in the data for each basepair position of all reads. Every time a nucleotide cannot be accurately determine during sequencing, it is flagged with a N (No hit) in the sequence instead of one of the nucleotides. Ideally this should never occur in the data and this graph should be a flat line at zero over the entire length. Although at the end of the sequences it might occur few times, but it should not occur more than a few percent.
- **Sequence length distribution**: Shows the length of all sequences. Ideally all reads should have the same length, but this might change, for example, after trimming.
- **Sequence duplication level**: Indicates how often some sequences appear the data. Ideally, all reads occur only few times and a high peak is expected near 1. If peaks are observed at higher numbers, this might indicate enrichment bias during the sequencing preparation (e.g. over amplification during PCR). Only the first 100000 sequences are considered and when the length of the reads is over 75bp, the reads are cut down to pieces of 50bp. Some duplication might not be bad and therefore a warning or error here does not need to concern.
- **Overrepresented sequences**: List of sequences that appear in more 0.1% of the total (this is only considered for the first 100000 sequences and reads over 75bp are truncated to 50bp pieces). The program gives a warning (when sequences are found to be present between 0.1% and 1% of the total amount of sequences) or an error (when there are sequences occurring more 1% of all sequences), but this does not always mean that the data is bad and might be ignored. For Illumina sequencing for satay experiments, the sequences often start with either 'CATG' or 'GATC' which are the recognition sites for NlaIII and DpnII respectively.
- **Adapter content**: Shows an accumulative percentage plot of repeated sequences with a positional bias appearing in the data. So if many reads have the same sequence at (or near) the same location, then this might trigger a warning in this section. Ideally this is a flat line at zero (meaning that there are no repeated sequences present in the data). If this is not a flat line at zero, it might be necessary to cut the reported sequences during the trimming step. If this section gives a warning, a list is shown with all the repeated sequences including some statistics. It can be useful to delete these sequences in the trimming step.
- **Kmer content**: indicates sequences with a position bias that are ofen repeated. If a specific sequence occurs at the same location (i.e. basepair number) in many reads, then this module will show which sequence at which location turns up frequently. Note that in later editions (0.11.6 and up) this module is by default turned off. If you want to turn this module on again, go to the Configuration folder in the Fastqc folder and edit the limits.txt file in the line where it says 'kmer ignore 1' and change the 1 in a 0.

2. Trimming of the sequencing reads

Next is the trimming of the sequencing reads to cut out, for example, repeated (adapter) sequences and low quality reads. There are two software tools adviced, Trimmomatic and BBDuk. Trimmomatic is relative simple to use and can be used interactively together with FASTQC. However, the options can be limiting if you want more control over the trimming protocol. An alternative is BBDuk, which is part of the BBMap software package. This allows for more options, but can therefore also be more confusing to use initially. Both software packages are explained below, but only one needs to be used. Currently, it is advised to use BBDuk (see section 2b).

For a discussion about trimming, see for example the discussion in MacManes et.al. 2014, Del

Fabbro et.al. 2013 or Delhomme et. al. 2014 or at basepairtech.com (although this discussion is on RNA, similar arguments hold for DNA sequence analysis).

2a. Trimming of the sequencing reads; Trimmomatic (0.39) Trimmomatic alters the sequencing result by trimming the reads from unwanted sequences, as is specified by the user. The program does not need to be installed, but after downloading only requires to be unzipped. Trimmomatic can be ran as an interactive program (for this 123FASTQ needs to be used) or non-interactively using the command line options.

If using interactively, use 123FASTQ (version 1.1) and run the 'Runner.bat' file in the 123FASTQ folder. Use the 'Trimmer' in the 'Trim Factory' menu on the right.

If using non-interactively in the command line use the command:

```
java -jar ${path_trimm_software}'trimmomatic-0.39.jar'
```

Before running Trimmomatic, a .fasta file needs to be created that allows clipping unwanted sequences in the reads. For example, the 'overrepresented sequences' as found by Fastqc can be clipped by adding the sequences to the .fasta file. Easiest is to copy an existing .fasta file that comes with Trimmomatic and adding extra sequences that needs to be clipped. For MiSeq sequencing, it is advised to use the TruSeq3 adapter file that needs to be copied to the data folder (see below for detailed explanation). For this use the command:

```
cp ${path_trimm_software}'adapters/''TruSeq3-SE.fa' ${pathdata}
```

Open the .fa file and copy any sequences in the file using a similar style as the sequences that are already present. Typically it is useful to clip overrepresented sequences that start with 'CATG' or 'GATC' which are the recognition sites for NlaIII and DpnII respectively. Note that the trimming is performed in the order in which the steps are given as input. Typically the adapter clipping is performed as one of the first steps and removing short sequences as one of the final steps.

A typical command for trimmomatic looks like this:

```
java -jar ${path_trimm_software}'trimmomatic-0.39.jar' SE -phred33 ${pathdata}${filename}
${path_trimm_out}${filename_trimmed} ILLUMINACLIP:'TruSeq3-SE.fa':2:30:10 LEADING:14
TRAILING:14 SLIDINGWINDOW:10:14 MINLEN:30
```

Check the quality of the trimmed sequence using the command:

```
${path_fastqc_software}fastqc --outdir ${path_fastqc_out} ${path_trimm_out}${filename_trimmed}
```

The following can be set to be set by typing the following fields after the above command (the fields must be in the given order, the optional fields can be ignored if not needed, see also http://www.usadellab.org/cms/?page=trimmomatic):

- `SE` (Single End) or `PE` (Paired End) [required];
- `-phred33` or `-phred64` sets which quality coding is used, if not specified the program tries to determine this itself which might be less accurate. usually the phred33 coding is used. If not sure, check if the .fastq file contains, for example, an exclamation mark (!), a dollar sign ($), an ampersand (&) or any number (0-9) since these symbols only occur in the phredd33 coding and not in the phred64 coding [optional];
- `Input filename`. Both forward and reverse for paired end in case of PE [required];

- `Output filename`. Both paired and unpaired forward and paired and unpaired reverse for paired end (thus 4 output in total) in case of PE. In case of SE, a single output file needs to be specified. Needs to have the same extension as the input file (e.g. .fastq) [required];
- `ILLUMINACLIP:TruSeq3-SE.fa:2:30:10` or `ILLUMINACLIP:TruSeq3-PE.fa:2:30:10` (for Single End reads or Paired End reads respectively). This cuts the adapter and other Illumina specific sequences from the reads. The first parameter after `:` indicates a FASTA file (this should be located in the same folder as the sequencing data). The second parameter is the Seed Mismatches which indicates the maximum mismatch count that still allows for a full match to be performed. The third parameter for PE sets the Palindrome Clip Threshold specifies how accurate the match between the two 'adapter ligated' reads must be for PE palindrome read alignment (Works only for PE, but needs to be set for SE as well). The fourth parameter is the Simple Clip Threshold which specifies how accurate the match between the adapter and the read.

A number of adapters are stored in the 'adapters' folder at the location where the trimmomatic program is saved. In case of MiSeq sequencing, the TruSeq3 adapter file is advised. The way the adapter sequences are aligned is by cutting the adapters (in the FASTA file) into 16bp pieces (called seeds) and these seeds are aligned to the reads. If there is a match, the entire alignment between the read and the complete adapter sequence is given a score. A perfect match gets a score of 0.6. Each mismatching base reduces the score by Q/10. When the score exceeds a threshold, the adapter is clipped from the read. The first number in the parameter gives the maximal number of mismatches allowed in the seeds (typically 2). The second value is the minimal score before the adapter is clipped (typically between 7 (requires $\frac{7}{0.6} = 12$ perfect matches) and 15 (requires $\frac{15}{0.6} = 25$ perfect matches)). High values for short reads (so many perfect matches are needed) allows for clipping adapters, but not for adapter contaminations. Note a bug in the software is that the FASTA file with the adapters need to be located in your current folder. A path to another folder with the adapter files yields an error. [optional] [https://wiki.bits.vib.be/index.php/Parameters_of_Trimmomatic];

- `SLIDINGWINDOW` Sliding window trimming which cuts out sequences witin the window and all the subsequent basepairs in the read if the average quality score within the window is lower than a certain threshold. The window moves from the 5'-end to the 3'-end. Note that if the first few reads of a sequence is of low quality, but the remaining of the sequence is of high quality, the entire sequence will be removed just because of the first few bad quality nucleotides. If this sitatuation occurs, it might be useful to first apply the HEADCROP option (see below). Parameters should be given as `SLIDINGWINDOW:L_window:Q_min` where `L_window` is the window size (in terms of basepairs) and `Q_min` the average threshold quality. [optional];
- `LEADING` Cut the bases at the start (5' end) of a read if the quality is below a certain threshold. Note that when, for example, the parameter is set to 3, the quality score Q=0 to Q=2 will be removed. Parameters should be given as `LEADING:Q_min` where `Q_min` is the threshold quality score. All basepairs will be removed until the first basepair that has a quality score above the given threshold. [optional];
- `TRAILING` Cut the bases at the end (3' end) of a read if the quality is below a certain threshold. Note that when, for example, the parameter is set to 3, the quality score Q=0 to Q=2 will be removed. All basepairs will be removed until the first basepair that has a quality score above the given threshold. [optional];
- `CROP` Cuts the read to a specific length by removing a specified amount of nucleotides from the tail of the read (this does not discriminate between quality scores). [optional];

- `HEADCROP` Cut a specified number of bases from the start of the reads (this does not discriminate between quality scores). [optional];
- `MINLEN` Drops a read if it has a length smaller than a specified amount [optional];
- `TOPHRED33` Converts the quality score to phred33 encoding [optional];
- `TOPHRED64` Converts the quality score to phred64 encoding [optional].

Note that the input files can be either uncompressed FASTQ files or gzipped FASTQ (with an extension fastq.gz or fq.gz) and the output fields should ideally have the same extension as the input files (i.e. .fastq or .fq). The convention is using field:parameter, where 'parameter' is typically a number. (To make the (relative long commands) more readable in the command line, use \ and press enter to continue the statement on the next line) (See 'Datacarpentry workshop > data Wrangling and Processing for Genomics > Trimming and Filtering' for examples how to run software). Trimmomatic can only run a single file at the time. If more files need to be trimmed using the same parameters, use

```
for infile in *.fastq
  do
  base=$(basename $(infile) .fastq)
  trimmomatic xxx
done
```

Where `xxx` should be replaced with the commands for trimmomatic.

2b. Trimming of the sequencing reads; BBDuk (38.84) BBDuk is part of the BBMap package and alters the sequencing result by trimming the reads from unwanted sequences, as is specified by the user. The program does not need to be installed, but after downloading only requires to be unzipped.

Before running BBDuk, a .fasta file can to be created that allows clipping unwanted sequences in the reads. For example, the 'overrepresented sequences' as found by Fastqc can be clipped by adding the sequences to the .fasta file. A .fasta file can be created by simply creating a text file and adding the sequences that need to be clipped, for example, in the form:

> Sequence1

CATG

> Sequence2

GATC

Or a .fasta can be copied from either Trimmomatic software package or the BBDuk package, both which are provided with some standard adapter sequences. In the case of Trimmomatic it is advised to use the TruSeq3 adapter file when using MiSeq sequencing. To copy the .fasta file to the data folder (see below for detailed explanation) use the following command:

```
cp ${path_trimm_software}'adapters/''TruSeq3-SE.fa' ${pathdata}
```

When using the adapter file that comes with BBMap, use the command:

```
cp ${path_bbduk_software}'resources/adapters.fa' ${pathdata}
```

The adapters.fa file from the BBDuk package includes many different adapter sequences, so it might be good to remove everything that is not relevant. One option can be to create a custom fasta file where all the sequences are placed that need to be trimmed and save this file in the bbmap/resources folder. To do this, in order to let everything work properly it is best to alter one of the existing .fa files. First copy that file as a backup using a different name (e.g. in the `#{path_bbduk_software}/resources` directory type the command `cp adapters.fa adapters_original_backup.fa`). Then, alter the adapters.fa file with any sequences you want to get trimmed. Note to not put empty lines in the text file, otherwise BBDuk might yield an error about not finding the adapters.fa file.

Typically it is useful to clip overrepresented sequences that were found by FASTQC and sequences that start with 'CATG' or 'GATC' which are the recognition sites for NlaIII and DpnII respectively. Note that the trimming is performed in the order in which the steps are given as input. Typically the adapter clipping is performed as one of the first steps and removing short sequences as one of the final steps.

BBDuk uses a kmers algorithm, which basically means it divides the reads in pieces of length k. (For example, the sequence 'abcd' has the following 2mers: ab,bc,cd). For each of these pieces the software checks if it contains any sequences that are present in the adapter file. The kmers process divides both the reads and the adapters in pieces of length k and it then looks for an exact match. If an exact match is found between a read and an adapter, then that read is trimmed. If the length k is chosen to be bigger than the length of the smallest adapter, then there will never be an exact match between any of the reads and the adapter sequence and the adapter will therefore never be trimmed. However, when the length k is too small the trimming might be too specific and it might trim too many reads. Typically, the length k is chosen about the size of the smallest adapter sequence or slighty smaller. For more details, see this webpage.

A typical command for BBDuk looks like this: `${path_bbduk_software}/bbduk.sh -Xmx1g in=${pathdata}/${filename} out=${path_trimm_out}${filename_trimmed} ref=${path_bbduk_software}/resources/adapters.fa ktrim=r k=23 mink=10 hdist=1 qtrim=r trimq=14 minlen=30`

Next an overview is given with some of the most useful options. For a full overview use call `bbduk.sh` in the bash without any options.

1. `-Xmx1g`. This defines the memory usage of the computer, in this case 1Gb (`1g`). Setting this too low or too high can result in an error (e.g. 'Could not reserve enough space for object heap'). Depending on the maximum memory of your computer, setting this to `1g` should typically not result in such an error.
2. `in` and `out`. Input and Output files. For paired-end sequencing, use also the commands `in2` and `out2`. The input accepts zipped files, but make sure the files have the right extension (e.g. file.fastq.gz). Use `outm` (and `outm2` when using paired-end reads) to also save all reads that failed to pass the trimming. Note that defining an absolute path for the path-out command does not work properly. Best is to simply put a filename for the file containing the trimmed reads which is then stored in the same directory as the input file and then move this trimmed reads file to any other location using: `mv ${pathdata}${filename::-6}_trimmed.fastq ${path_trimm_out}`
3. `qin`. Set the quality format. 33 for phred 33 or 64 for phred 64 or auto for autodetect (default is auto).
4. `ref`. This command points to a .fasta file containg the adapters. This should be stored in the location where the other files are stored for the BBDuk software (`${path_bbduk_software}/resources`) Next are a few commands relating to the kmers

algorithm.

5. `ktrim`. This can be set to either don't trim (`f`, default), right trimming (`r`), left trimming (`l`). Basically, this means what needs to be trimmed when an adapter is found, where right trimming is towards the 5'-end and left trimming is towards the 3'-end. For example, when setting this option to `ktrim=r`, then when a sequence is found that matches an adapter sequence, all the basepairs on the right of this matched sequence will be deleted including the matched sequence itself. When deleting the whole reads, set `ktrim=rl`.
6. `kmask`. Instead of trimming a read when a sequence matches an adapter sequence, this sets the matching sequence to another symbol (e.g. `kmask=N`).
7. `k`. This defines the number of kmers to be used. This should be not be longer than the smallest adapter sequences and should also not be too short as there might too much trimmed. Typically values around 20 works fine (default=27).
8. `mink`. When the lenght of a read is not a perfect multiple of the value of `k`, then at the end of the read there is a sequence left that is smaller than length k. Setting `mink` allows the software to use smaller kmers as well near the end of the reads. The sequence at the end of a read are matched with adapter sequences using kmers with length between mink and k.
9. `minkmerhits`. Determines how many kmers in the read must match the adapter sequence. Default is 1, but this can be increased when for example using short kmers to decrease the chance that wrong sequences are trimmed that happen to have a single matching kmer with an adapter.
10. `minkmerfraction`. A kmer in this read is considered a match with an adapter when at least a fraction of the read matches the adapter kmer.
11. `mincovfraction`. At least a fraction of the read needs to match adapter kmer sequences in order to be regarded a match. Use either `minkmerhits`, `minkmerfraction` or `mincovfraction`, but setting multiple results in that only one of them will be used during the processing.
12. `hdist`. This is the Hamming distance, which is defined as the minimum number of substitutions needed to convert one string in another string. Basically this indicates how many errors are allowed between a read and and an adapter sequence to still count as an exact match. Typically does not need to be set any higher than 1, unless the reads are of very low quality. Note that high values of hdist also requires much more memory in the computer.
13. `restrictleft` and `restrictright`. Only look for kmers in the left or right number bases.
14. `tpe` and `tbo`: This is only relevant for paired-end reads. `tpe` cuts both the forward and the reverse read to the same length and `tbo` trims the reads if they match any adapter sequence while considering the overlap between two paired reads.

So far all the options were regarding the adapter trimming (more options are available as well, check out the user guide). These options go before any other options, for example the following:

15. `qtrim`. This an option for quality trimming which indicates whether the reads should be trimmed on the right (`r`), the left (`l`), both (`fl`), neither (`f`) or that a slidingwindow needs to be used (`w`). The threshold for the trimming quality is set by `trimq`.
16. `trimq`. This sets the minimum quality that is still allowed using phred scores (e.g. Q=14 corresponds with $P_{error} = 0.04$).
17. `minlength`. This sets the minimum length that the reads need to have after all previous trimming steps. Reads short than the value given here are discarded completely.
18. `mlf`. Alternatively to the `minlen` option, the Minimum Length Fraction can be used which determines the fraction of the length of the read before and after trimming and if this drops below a certain value (e.g 50%, so `mlf=50`), then this read is trimmed.
19. `maq`. Discard reads that have an average quality below the specified Q-value. This can be

useful after quality trimming to discard reads where the really poor quality basepairs are trimmed, but the rest of the basepairs are of poor quality as well.

20. `ftl` and `ftr` (`forcetrimleft` and `forcetrimright`). This cuts a specified amount of basepairs at the beginning (`ftl`) or the last specified amount of basepairs (`ftr`). Note that this is zero based, for example `ftl=10` trims basepairs 0-9.
21. `ftm`. This force Trim Modulo option can sometimes be useful when an extra, unwanted and typically very poor quality, basepair is added at the end of a read. So when reads are expected to be all 75bp long, this will discard the last basepair in 76bp reads. When such extra basepairs are present, it will be noted in FastQC.

Finally, to check the quality of the trimmed sequence using the command:

```
fastqc --outdir ${path_fastqc_out} ${path_trimm_out}/${filename_trimmed}
```

3. Sequence alignment and Reference sequence indexing; BWA (0.7.17) (Linux)

The alignment can be completed using different algorithms within BWA, but the 'Maximal Exact Matches' (MEM) algorithm is the recommended one (which is claimed to be the most accurate and fastest algorithm and is compatible with many downstream analysis tools, see documentation for more information). BWA uses a FM-index, which uses the Burrows Wheeler Transform (BWT), to exactly align all the reads to the reference genome at the same time. Each alignment is given a score, based on the number of matches, mismatches and potential gaps and insertions. The highest possible score is 60, meaning that the read aligns perfectly to the reference sequence (this score is saved in the SAM file as MAPQ). Besides the required 11 fields in the SAM file, BWA gives some optional fields to indicate various aspects of the mapping, for example the alignment score (for a complete overview and explanation, see the documentation). The generated SAM file also include headers where the names of all chromosomes are shown (lines starting with SQ). These names are used to indicate where each read is mapped to.

Before use, the reference sequence should be indexed so that the program knows where to find potential alignment sites. This only has to be done *once* for each reference genome. Index the reference genome using the command

```
bwa index /path/to/reference/sequence/file.fasta
```

This creates 5 more files in the same folder as the reference genome that BWA uses to speed up the process of alignment.

The alignment command should be given as

```
bwa mem [options] ${path_refgenome} ${path_trimm_out}${filename_trimmed} > ${path_align_out}
${filename_trimmed::-6}'.sam'
```

where `[options]` can be different statements as given in the documentation. Most importantly are:

- `-A` Matching scores (default is 1)
- `-B` Mismatch scores (default is 4)
- `-O` Gap open penalty (default is 6)
- `-E` Gap extension penalty (default is 1)
- `-U` Penalty for unpaired reads (default is 9; only of use in case of paired-end sequencing).

Note that this process might take a while. After BWA is finished, a new .sam file is created in the same folder as the .fastq file.

4. Converting SAM file to BAM file; SAMtools (1.7) and sambamba (0.7.1) (Linux)

SAMtools allows for different additional processing of the data. For an overview of all functions, simply type samtools in the command line. Some useful tools are:

- `view` This converts files from SAM into BAM format. Enter samtools view to see the help for all commands. The format of the input file is detected automatically. Most notably are:
  - `-b` which converts a file to BAM format.
  - `-f` int Include only the reads that include all flags given by int.
  - `-F` int Include only the reads that include none of the flags given by int.
  - `-G` int Exclude only the reads that include all flags given by int.
- `sort` This sorts the data in the BAM file. By default the data is sorted by leftmost coordinate. Other ways of sorting are:
  - `-n` sort by read name
  - `-t` tag Sorts by tag value
  - `-o` file Writes the output to file.
- `flagstats` Print simple statistics of the data.
- `stats` Generate statistics.
- `tview` This function creates a text based Pileup file that is used to assess the data with respect to the reference genome. The output is represented as characters that indicate the relation between the aligned and the reference sequence. The meaning of the characters are:
  - . :base match on the forward strand
  - , :base match on the reverse strand
  - </> :reference skip
  - AGTCN :each one of these letters indicates a base that did not match the reference on the forward strand.
  - Agtcn : each one of these letters indicates a base that did not match the reference on the reverse strand.
  - + [0-9]+[AGTCNagtcn] : Denotes (an) insertion(s) of a number of the indicated bases.
  - -[0-9]+[AGTCNagtcn] : Denotes (an) deletion(s) of a number of the indicated bases.
  - ^ : Start of a read segment. A following character indicates the mapping quality based on phred33 score.
  - $: End of a read segment.
  - * : Placeholder for a deleted base in a multiple basepair deletion.

- `quickcheck` Checks if a .bam or .sam file is ok. If there is no output, the file is good. If and only if there are warnings, an output is generated. If an output is wanted anyways, use the command `samtools quickcheck -v [input.bam] &&echo 'All ok' || echo 'File failed check'`

Create a .bam file using the command

`samtools view -b ${path_align_out}${filename_trimmed::-6}'.sam' > ${path_align_out}${filename_trimmed::-6}'.bam'.`

Check if everything is ok with the .bam file using

`samtools quickcheck ${path_align_out}${filename_trimmed::-6}'.bam'`.

This checks if the file appears to be intact by checking the header is valid, there are sequences in the beginning of the file and that there is a valid End-Of_File command at the end. It thus check only the beginning and the end of the file and therefore any errors in the middle of the file are not noted. But this makes this command really fast. If no output is generated, the file is good. If desired, more information can be obtained using `samtools flagstat ${path_align_out}${filename_trimmed::-6}'.bam'` or `samtools stats ${path_align_out}${filename_trimmed::-6}'.bam'`. Especially the latter can be a bit overwhelming with data, but this gives a thorough description of the quality of the bam file. For more information see this documentation.

For many downstream tools, the .bam file needs to be sorted. This can be done using SAMtools, but this might give problems. A faster and more reliable method is using the software sambamba using the command

`sambamba-0.7.1-linux-static sort -m 500MB ${path_align_out}${filename_trimmed::-6}'.bam'`

(where `-m` allows for specifying the memory usage which is 500MB in this example). This creates a file with the extension .sorted.bam, which is the sorted version of the original bam file. Also an index is created with the extension .bam.bai. If this latter file is not created, it can be made using the command

`sambamba-0.7.1-linux-static index  ${path_align_out}${filename_trimmed::-6}'.bam'`.

Now the reads are aligned to the reference genome and sorted and indexed. Further analysis is done in windows, meaning that the sorted .bam files needs to be moved to the shared folder.

`mv ${pathdata} ${path_sf}`

Next, the data analysis is performed using custom made codes in Matlab in Windows.

5. Determining transposon insertions: Python or Matlab (Matlab code from Kornmann lab [Michel et. al. 2017])

After creating the .bam file, the location of the transposon insertions and the number of reads per insertion needs to be determined. For this, no standard software is available, instead a custom made python script is used. This python script is heavily based on the Matlab script created by the Kornmann lab as described in the paper by Michel et.al., 2017.

The goal of this software is to create an overview of all insertion locations in the genome with the number of reads at those locations and to determine the number of insertions and reads for each gene.

In order to run the python script in Linux, go to the location where the script is stored and enter the command

```
python3 transposonmapping_satay.py /location/to/file.bam
```

The python script loads a .bam file together with its .bam.bai index file (the .bam.bai script is required). For this it requires the pysam package which partly relies on the SAMTools software, hence this is only available on Linux (or Mac) systems (also when running the python on its own, so without the workflow, this python script will only work in Linux or Mac). Additional files required for the python script to work are (see also `N:\tnw\BN\LL\Shared\VirtualMachines\`):

- Yeast_Protein_Names.txt; includes all names for each gene including any aliases and different naming conventions.
- Saccharomyces_cerevisiae.R64-1-1.99.gff3; includes for each gene the location in the genome.
- Cerevisiae_AllEssentialGenes_List.txt; This file is a combination of the essential genes found in Cerevisiae_EssentialGenes_List_1and Cerevisiae_EssentialGenes_list_2.

The python scripts starts with loading the .bam file using pysam and determining some properties of the file, like the lengths and names of the chromosomes as used in the file and the number of mapped reads. After that, the script loops over all reads in each chromosome and gets the insertion location, orientation and length of the reads. The orientation of the reads are given by a flag, which typically is either 0 (forward orientation) or 16 (reverse orientation). The position of the reads is corrected if it is in reverse orientation. The most important results are stored in the variables `tncoordinates_array` (which stores the exact location of each read) and `readnumb_array` which stores the number of reads for each of the insertions. To count the number of reads, it is assumed that all insertions that are within two basepairs of each other belong together and hence the reads of those insertions are all summed up. This is to allow for small uncertainties during sequencing and alignment. The position of the total number of reads are determined by averaging the locations of the summed reads. When taking the sum of multiple reads, the highest read count is discarded (e.g. when the following number of reads are summed, [3,7,14], the value 14 is discarded and thus the total number of reads of those three insertions is 3+7=10 reads). This is orignally considered in the Matlab code by the Kornmann lab to reduce noise in the read data. See the discussion on the SATAY user forum for a more detailed explaination.

Next, all the essential genes are retrieved from the additional files that were loaded in the beginning. Then, the chromosomes are concatenated into one large genome, so that the numbering of the basepair positions does not start at 0 for each chromosome, but rather continues counting upwards for the subsequent chromosomes. Finally, for all genes the number of insertions and reads are determined by checking the position of the genes and counting all transposons and reads that fall within the range of the gene.

The data is stored in multiple files

- .bed
- .wig
- _pergene.txt
- _pergene_insertions.txt
- _peressential.txt
- _peressential_insertions.txt

The .bed file (Browser Extensible Data) is used for storing the locations of the insertions and the number of reads. The different columns in the file are separated by spaces. The first column

indicates the chromosome number (e.g. `chrI`), the second and third column the start and end position respectively, the fourth column includes a dummy variable (this is needed to comply the standard layout of the bed format) and the fifth column is the number of reads. For the number of the reads, the equation 20*reads+100 is used that linearly scales the values to enhance the contrast (e.g. 4 reads is represented as 180).

The .wig file (Wiggle) contains similar information as the .bed file, but the layout is different. This file contains two columns separated by a space where the first column represents the location of the insertion and the second column the number of reads (the actual number of reads, thus not the using the equation as used in the .bed file). A difference between the .bed file and the .wig file is that in the in .wig file the insertions with different orientations are summed. In the .bed file a distinction is made between reads that come from transposons with different orientation, but this is not done in the .wig file.

Finally four more file are created that include the number of insertions and reads per gene. Files that end with the _insertions.txt are similar to the files without this extension, but include a list of the exact location (in terms of bp) of all insertions within the gene. The files include all genes (or only all annotated essential genes in case of _peressential.txt and _peressential_insertions.txt). In case of _pergene.txt and _peressential.txt, these files contain three tab separated columns where in the first column the gene name is given (standard name is, e.g. Cdc42 or Bem1), the second column contains the number of insertions within the gene and the third column includes the number of reads. In case of the _insertions.txt files, they consist of six columns where the first represent the gene name, the second the chromosome where the gene is located and the third and fourth the start and end position of the gene, respectively. The fifth column includes a list of all insertion locations within the gene and the sixth column represents the number of reads for all insertions shown in the fifth column.

Bibliography

Chen, P., Wang, D., Chen, H., Zhou, Z., & He, X. (2016). The nonessentiality of essential genes in yeast provides therapeutic insights into a human disease. Genome research, 26(10), 1355-1362.

Costanzo, M., VanderSluis, B., Koch, E. N., Baryshnikova, A., Pons, C., Tan, G., ... & Pelechano, V. (2016). A global genetic interaction network maps a wiring diagram of cellular function. Science, 353(6306).

Del Fabbro, C., Scalabrin, S., Morgante, M., & Giorgi, F. M. (2013). An extensive evaluation of read trimming effects on Illumina NGS data analysis. PloS one, 8(12).

Delhomme, N., Mähler, N., Schiffthaler, B., Sundell, D., Mannepperuma, C., & Hvidsten, T. R. (2014). Guidelines for RNA-Seq data analysis. Epigenesys protocol, 67, 1-24.

Guo, Y., Park, J. M., Cui, B., Humes, E., Gangadharan, S., Hung, S., ... & Levin, H. L. (2013). Integration profiling of gene function with dense maps of transposon integration. Genetics, 195(2), 599-609.

MacManes, M. D. (2014). On the optimal trimming of high-throughput mRNA sequence data. Frontiers in genetics, 5, 13.

Michel, A. H., Hatakeyama, R., Kimmig, P., Arter, M., Peter, M., Matos, J., ... & Kornmann, B. (2017). Functional mapping of yeast genomes by saturated transposition. Elife, 6, e23570.

Pfeifer, S. P. (2017). From next-generation resequencing reads to a high-quality variant data set. Heredity, 118(2), 111-124.

Segal, E. S., Gritsenko, V., Levitan, A., Yadav, B., Dror, N., Steenwyk, J. L., ... & Kunze, R. (2018). Gene essentiality analyzed by in vivo transposon mutagenesis and machine learning in a stable haploid isolate of Candida albicans. MBio, 9(5), e02048-18.

Usaj, M., Tan, Y., Wang, W., VanderSluis, B., Zou, A., Myers, C. L., ... & Boone, C. (2017). TheCellMap. org: A web-accessible database for visualizing and mining the global yeast genetic interaction network. G3: Genes, Genomes, Genetics, 7(5), 1539-1549

"I want to be a healer, and love all things that grow and are not barren" - J.R.R. Tolkien