Masaryk University

Faculty of Informatics

A computational pipeline for forward genomics

Master's Thesis

Bc. Peter Javorka

Brno, Spring 2018

Masaryk University

Faculty of Informatics

A computational pipeline for forward genomics

Master's Thesis

Bc. Peter Javorka

Brno, Spring 2018

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Peter Javorka

Advisor: Ing. Matej Lexa, Ph.D

Acknowledgements

I would like to thank to my supervisor Ing. Matej Lexa, Ph.D for his advices, patience, and for guiding of this diploma thesis.

My thanks also goes to colleagues in CEITEC - Vivek Kumar Raxwal, Ph.D. and Mgr. Karel Říha, Ph.D., which came up with the idea of creating of this tool.

Also, I would like to thank Ing. Jan Najvárek from the ARTIN, spol. s r.o. for the opportunity to collaborate on this project.

Last, but not least I would like to thank to my family and Jana Gettová for the support during the hard times.

Abstract

Next-generation sequencing (NGS) is a phenomena in the biological field, which enables to generate large amount of genetic data, through which the biological gene function can be examined. This thesis aims to create a single comprehensive tool, which can be used by non-informaticians to widespread the analysis of the NGS data.

Computational pipeline - the heart of the tool, enables visualization of EMS-induced mutations in Arabidopsis Thaliana. To create mentioned pipeline, we analyzed and used existing software, which solves some of the parts of the pipeline.

Keywords

Arabidopsis thaliana, SNP, pipeline, EMS, NGS

Contents

List of Figures

Introduction

For many years , genetics has relied on genetic crossing, which is a complicated and a long-lasting process. To isolate genes and mutations responsible for a phenotype of an interest, multiple generations need to be grown and crossed. With the arrival of computers and especially gene sequencers, the whole process has been speed up rapidly.

Sequencers allow to reveal the order of nucleotides within DNA or RNA molecules. Traditional sequencers are based on Sanger's method, which is slow and expensive for the current needs. Up to the arrival of next generation sequencers capable of producing large amount of data with a fraction of expenses, DNA analysis was not widespread enough across the scientific community.

Scientific community, which is trying to identify biological function of the genes in many organisms including *Arabidopsis thaliana* is profiting from the next generation sequencers too. By combining forward genetic screens it can help us understand not only the gene functions but also characterize spatial domains of proteins.

However, analysis of the sequencing data to decipher causative mutations is still a challenging task which requires advanced computational infrastructure as well as bioinformatic expertise. The lack of easy and comprehensive tools, which can be used without knowledge of information technology is slowing the progress in this biological field.

1 Biological background

In order to fully understand the purpose of this work, some biological background is required. This chapter tries to provide the minimum biological background necessary.

All living organisms contain genetic code which include information about shape, size and to some extent behavior and govern most of the aspect of their life. This information, called genotype, is stored in DNA as a set of genes present in most individuals of the given species. [1] These hard-coded information is not observable unless they express and show observable characteristics also known as phenotypes. These phenotypes can also be result of interaction between the genotype and environment. [2]

The dynamic nature of interaction between genotype and environment bring about evolution of an organism. Since the environment is a subject to frequent changes, there are mechanisms, through which the species or individual organisms can adapt to the new environment conditions. One of them is mutation combined with natural selection.

Mutation, which is the permanent alteration of one or more nucleotides at a specific site along the DNA strand, is key to the process called evolution. It can be beneficial, neutral, harmful or even lethal for the organism. Mutation can occur naturally in the population or it can be caused experimentally by chemical mutagens, ionizing radiation or insertional mutagenesis. It can be associated with the change of a one

or more nucleotide through substitution, deletion or rearrangement of nucleotide base pairs in the chromosome. [3, 4, 5]

Ethyl methanesulfonat (EMS) is one of the many chemical mutagens, causing specific mutagenesis. It induces chemical modification where the majority (99%) is based on changing C-to-T resulting in T/A to G/C transversions and A/T to G/C transitions. At low frequency EMS also generates G/C to C/G or G/C to T/A transversion. [6] These mutations are randomly located [7] and may therefore be also helpful in changing the biological function of genes. Historically, EMS has been applied to many model organisms including plants where *Arabidopsis thaliana* is preferred owing to availability of genomic and genetic resources. [8, 9, 10]. *Arabidopsis thaliana* is a flowering plant naturally occurring in Western Eurasia [11] and its complete genome was sequenced in 2000 which contains about 125 megabases distributed over five nuclear chromosomes. [12, 13, 14]

To be able to associate gene function to the phenotype of interest we need to define a DNA sequence marker. Single nucleotide polymorphism (SNP) is a single base change marker in DNA sequence. Usually it is an alternation of two possible nucleotides at a given position. SNPs may be responsible for genome evolution as well as for diversion among individuals or some complex diseases. [15, 16]

2 Bioinformatical background

2.1 Pipeline

In forward genetic screens, whole genome sequencing read files are generated from two biological samples, one treated with EMS (mutant) and another, reference sample, not treated with EMS (usually parent). The main goal of this work is to create a pipeline that is capable of identifying EMS-induced mutations, causative to phenotype of interest, in such screens.

The pipeline should perform multiple steps including an alignment of sequencing reads to reference genome, identification of SNPs caused by EMS, isolation of SNPs present in mutant sample only and visualization of the putative causal SNPs in an user friendly way. Important factor is that data generated by Next-generation sequencers can vary in many ways including file formats. Because of this data diversification, a preprocessing step is required.

2.1.1 Preprocessing

Next-generation sequencers have multiple options, which may cause multiple issues in the following steps. First of them, which may significantly change the pipeline workflow, is that reads can be sequenced from both ends of the read. In this situation such as data are called *paired end* reads. The opposite are *single end* data, which are sequenced

in only one way. This information must be provided from user as it cannot be determined from the data itself.

Output quality of produced reads is another possible problem. Information about mentioned quality is stored alongside the produced data and usually represented via PHRED score. PHRED score is logarithmically related to the base-call error probability. [17]

$$q = -10 \log_{10}(p)$$

This issue is related to adapter contamination in the reads, which can appear when sequenced DNA fragment is shorter than the read length and read errors (base calling errors and small insertions or deletions). Mentioned problems can significantly affect final results. [18, 19, 20]

Users may be aware of the quality of their data, or their samples may have good quality, hence the quality control step, in contrast with handling different types of data, is only optional.

2.1.2 Alignment to reference genome

Regardless of the quality and format, sequencer provides short chunks of reads whose location in the genome is not determined. Hence in our scenario, we must align both reference and mutated chunks to the reference genome of *Arabidopsis thaliana*.

It is important to realize that the alignments for short and long reads are different. Aligning short reads correctly may be a problem, since there may be multiple candidate regions where the mapping

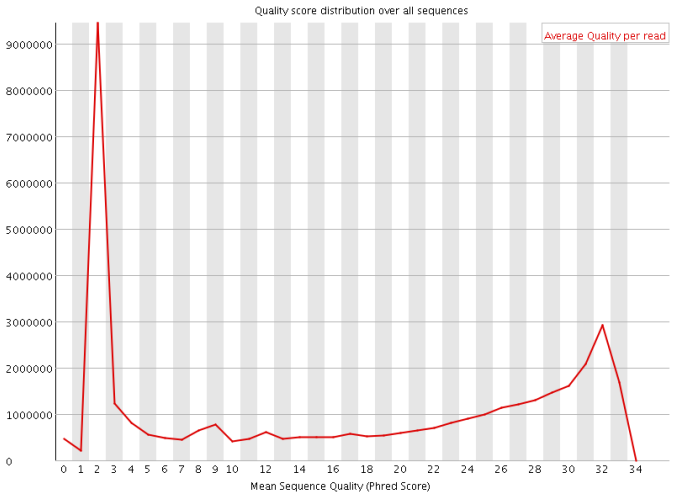

Figure 2.1: PHRED score distribution over reads - before quality control [21]

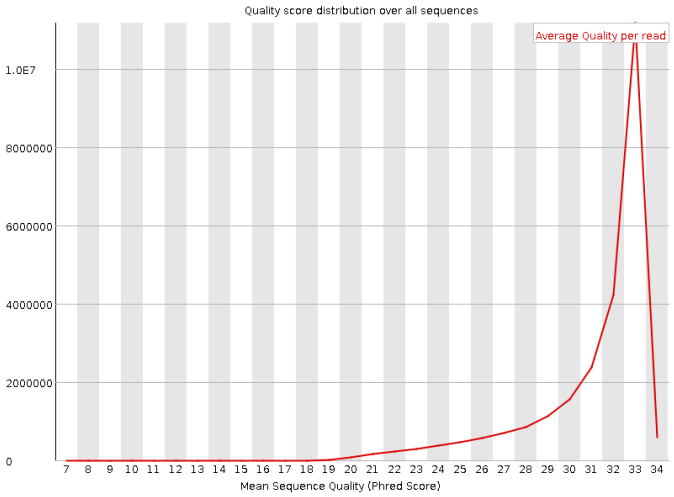

Figure 2.2: PHRED score distribution over reads - after quality control [22]

read can be mapped. Short reads can be aligned using the full-length read or alternatively, a very small threshold can be used to reduce the reference bias caused by the mismatches towards the end of the read. On the other hand, longer reads are more prone to structural variations and misassemblies in the reference genome but are less affected by mismatches close to their ends. [23] Aligning single-end data also differs from paired-end data, because of the nature of this data.

There are numerous approaches to align such a big amount of short reads fast and accurately, one of these is Burrows-Wheeler transformation [24]. The whole process is crucial for identification of exact locations of produced mutations, because with a wrong alignment, the region of the target mutation cannot be peoperly identified. Alignment generally produces BAM files from FASTQ files.

2.1.3 Post-processing

Data from Alignment is represented via SAM format, which is inefficient in compare to BAM format, which is a binary representation of SAM format. Some operations may be more time and memory efficient when using binary representation of data. For above reason data should be converted to BAM format.

Sequenced data may also include Polymerase chain reaction (PCR) duplicates [1] , which may affect the final result as they are affecting the frequency of the reads. Therefore, there should be a mechanism, which can identify and remove the PCR duplicates. However, samples do not need to contain the duplicates and this step should be optional.

The next step of post-processing is the SNPs identification - SNP calling. The main goal is the determination of a position of polymorphisms (e.g. SNP) that occurs or in which positions at least one of the bases differs from a reference sequence. In this case, the reference genome is the original genome of *Arabidopsis thaliana* and the one provided by the user is the reference one. Following process - genotype calling, resolves genotype for every individual in which a SNP has already been called. The word 'calling' is used for indication of the estimation of one unique SNP or genotype. [26]

2.1.4 Identification of SNPs specific to mutant samples

The previous step produces two files, one for reference, and one from mutant subsample containing SNPs. These SNPs include all mutations including the ones generated by EMS. As we mentioned in chapter 1 EMS induces G-to-A and C-to-T transitions in DNA, therefore, we need to filter only mentioned transitions from the control and mutant

1. PCR is widely used technique in molecular biology and biomedical research. It is based on specific amplification of a single gene or DNA sequence from a very small sample. [25]

samples and after that, to find the ones that are exclusive to the mutant sample.

Unfortunately, EMS is capable of inducing numerous mutations in the genome, which includes both causative and non-causative SNPs. Hence, we need to provide some mechanism to filter non-causative mutations. Frequency of causative SNPs in the mapping population are significantly higher (close to 100%) in contrast to non-causative. Another criteria, which may help to identify causative SNPs is sequencing depth, which can be defined as an average number of times in which a particular nucleotide is represented in a collection of random raw sequences. [27] As long as the proposed filters can significantly affect the number of excluded SNPs, these filters should be configurable using predefined recommended values.

2.1.5 Annotation of identified SNPs

After we get filtrated SNPs, we have informations about the genes which mutated; such as type of transition EMS induced and more, but we do not have the information about the gene itself and its biological characterization. Therefore, we need to annotate the given SNPs to get more biological information such as gene name or amino acid change.

2.1.6 Visualization

In order to understand the characteristics of EMS induced mutations in the mutated samples, it is handy to visualize the data. Main characteristics indicating causative SNPs like frequency and gene location,

should be used as axis variables. Other informations such as data from the annotation can be used as a description of each point. Another valuable information for researchers may be the position regarding the chromosome. There could be some location, which one should be more or less interested, like centromere [2] or a leg of a chromosome.

2.2 File formats

The whole pipeline presented in the previous chapter contains multiple steps and it is operating with various types of data in multiple file formats. In this section we will describe all the main formats used in the pipeline, including FASTQ, SAM, BAM or VCF and show their structure and linkage to the part of the problem, which is the whole pipeline solving.

FASTQ One of the most common file formats used to hold sequencing data is FASTQ. It is based on a simpler FASTA format, which is capable to store sequence of nucleotides, whereas with the FASTQ, there were also added informations about the quality of each nucleotide in the sequence. All the data are represented in human readable way, where quality of the nucleotides are stored as PHRED score. [29, 30, 31] PHRED score is logarithmically related to the base-call error prob-

2. A constricted region on a chromosome where two sister chromatids are joined, esp. during metaphase; the region on a chromosome to which the spindle is attached during mitosis via the kinetochore. [28]

ability. [17]

$$q = -10 \log_{10}(p)$$

It has emerged as a *de facto* standard data exchange between the tools. Despite of that and due to the lack of the clear definition of the format, there are multiple versions, with at least three major ones (Sanger, Solexa and Illumina), which are incompatible and commonly used. [32] None of these versions, can provide the information if the provided data are single-end or paired-end reads, since they are only holding the information about the sequence and it's quality.

SAM and BAM SAM is an abbreviation for Sequence Alignment/Map, which, as the name may suggest, is a file format designed to hold aligned data of various sequence types. The goal was to create a well-defined interface between the alignment and a downstream analysis, including detection, genotyping, and assembly of variants. SAM is a human-readable format, that consists of two sections - a header and an alignment. Each alignment line has eleven mandatory fields and a variable number of optional fields. However, the mandatory fields must be presented, their values can be set as unknown by setting them as '-' or a zero (depending on the field). [33, 34]

BAM format was developed to increase the performance of the SAM format. It is the binary version of the SAM format (abbreviation of Binary Alignment/Map), therefore, it is designed to hold the same data as the SAM. The mentioned performance improvement was achieved by using a BGZF compression. BGZF compression was

developed by the authors of the SAM format to achieve fast random access with preserving the compatibility with the zlib library. [34, 35, 36] Both, SAM and BAM, formats can be sorted to increase the performance to streamline data processing. Nowadays BAM format is also commonly used to store raw sequencing data, because of its ability to save up space, in comparison to FASTQ, through the compression.

2.2.1 VCF

The variant call format (VCF) is a generic file format for storing DNA polymorphisms data such as SNPs, insertions, deletions and structural variants with rich annotations. Its primary intention was to represent human genetic variations, but it is flexible and extensible enough to represent various genomic variations. [37] The structure of this format consists of optional meta informations, and mandatory header and data lines. As the SAM format, it also has a binary version called BCF, which defines more precise structure. [38]

3 Functional requirements

Functional requirements are an important part of the software specification, as it ensures that the final product will have all desired functions. These requirements originated from the cooperation with scientists from CEITEC (Central European Institute of Technology in Brno), who are actively researching biological purpose of the genes of *Arabidopsis thaliana* using EMS as a mutagen substance.

- since the most users of this tool would probably be biologists, it is necessary to make the installation process simple and provide detailed installation guide understandable to people with average IT skills.
- data handled by the application can be large, so the whole process of mutation identification in causative genes can take quite a long time. Hence, it is essential that the tool, after small modifications, should be able to run on cloud services. It also implies that solution should be platform independent or at least support all major OS - Windows, Mac OS and Linux.
- as next-generation sequencing is spreading more and more, it would be beneficial to apply the pipeline to another species than *Arabidopsis thaliana*. Therefore, the tool should be configurable and extensible.
- there are various tools, which can efficiently solve some parts of the pipeline. Well tested and actively maintained tools should

be used in the pipeline, as developing them beyond the scope of this work.

- the whole process is relatively complicated and users may not understand how the output was produced. Hence, the tool should have documentation on very high level and each step should be described. If an existing tool is being used, one should know how it is used (which parameters, input data etc.), if not, implemented step should be documented and well tested.

- some steps of the solution may be very resource demanding. To increase the efficiency of the pipeline optimize the processes, which can be run simultaneously.

- in order to spread the tool and help biologist with their work, the software solution should provide good graphical user experience.

4 Design

Designing the application is a very important part of the development process since it affects many aspects of the resulting software. The biggest challenge during designing this application is to satisfy the requirements of *using existing tools for solving partial problems* and *providing an easy installation with a good user experience*. Bioinformatical tools are written in various languages, each with their own installation process and dependencies. Because of this, it is very difficult and inefficient to build native application with an installation wizard.

Another aspect of the maintainability of the tool. Used tools may evolve and update their dependencies, which makes usability of the native application even more a bad choice. But on the market, there are various platforms more or less satisfying our needs.

4.1 Platform

Selection of a proper platform during software design may directly affect multiple requirements presented in chapter 3 such as performance or user experience including the installation process.

One platform that is frequently used by bioinformaticians is called Bioconda. It is based on the platform and the language-agnostic package manager Conda. Bioconda provides almost 4000 [1] [39] bioinformatics packages from various language ecosystems like Python, C, C++, R or Perl. [40, 41]

1. at the time of writing this thesis - 18/20/04

Since Bioconda is a primary software for bioinformaticians, it has some drawbacks from the point of view of common and IT unskilled person. First major issue may be relatively complicated installation process. It has multiple requirements including installation of Conda, which itself has multiple dependences like Python, PyYaml, Requests and pycosat. [42] As a second issue, in terms of user experience and usability, can be considered command-line interface.

4.1.1 Virtualization

Another commonly used solution for delivering platform independent solution with many dependencies is virtualized environment. Virtualization (virtual machine, virtual memory etc.) in this context is a high-level abstraction that hides the underlying implementation details. [43] That abstraction is a key point, because it allows to emulate hardware. Hence to support of multiple platforms, the only thing, which is needed is an emulator with already set up environment on the virtual machine.

There are multiple types of virtualization differing by the subject of virtualization. Two major types - Hypervisor-based virtualization is based on emulating at the hardware level and in opposite Containerization, which is emulation operation system. Each of them has their advantages and drawbacks. Hypervisor-based has bigger overhead, since on the top of virtualized hardware (note that emulator is software running on OS), must be installed OS, in order to run various software. However, this isolation may be beneficial in term of security,

since host and guest machines are not sharing kernel. [2].This is complementing Containerization, which is sharing kernel, hence it is less secured, but has significantly better performance. [45]

Although, virtualization is not a solution for everything. The emulator is adding another layer of overhead. It is running on the top of an operation system, so it can not achieve performance as high as the native application.

To satisfy both functional requirements - a pleasant installation guide with good user experience and a platform independent application, we think that the best solution is to pick Container-based visualization technology as it provides good trade-of of security, performance, and user experience.

4.1.2 Architecture

As well as choosing appropriate platform, choosing well suited architecture can solve many issues during development and even more during maintenance. Every architecture has its advantages and disadvantages and in this section we will try to find a suitable one for our problem.

We chose client-server architecture, because of its main characteristic - division user interface from the application logic. Therefore, server-side will be independent from the client-side which is allowing to build a system, which can act like a desktop application, but with

2. The kernel is a program that constitutes the central core of a computer operating system. It has complete control over everything that occurs in the system. [44]

some minor changes, it can be adapted into an external web-based service. However, it may introduce some complications like defining application programming interface (API) or more complicated error handling.

In order to develop a web-based service, which can be run on the local machine, as a desktop application, we chose to build a web-based application. It can be hosted either by the external server or on the localhost exposing a single port, through which a user can connect to the application via a web browser. Therefore, we will use HTML, CSS, JS stack on the client side. A single page application, as a current trend of development of the web application may be a beneficial for user in terms of the user experience.

On the server side we chose a JavaScript too - Node JS, as it is a cross-platform and it has a high performance. Node JS is based on the Google's V8 engine, which are both implemented mostly in C/C++. It does not rely on multithreading, but on asynchronous eventing model, [46] which is an ideal pattern for spawning external tools and handling asynchronous messages from standard output (stdout) of the tool.

We know, that our system will handle relatively large files (tens of GB) and their processing may last a long time. Therefore, we should provide up-to-date information to the user either from running of the external tools, and from our scripts. For this purpose we need to find a mechanism to transfer all logs and information from the server to the client.

4.2 Choosing pipeline tools

In order to create a usable pipeline producing relevant data, it is essential to select right tools for the pipeline. Especially, when some of the steps of the pipeline can be implemented with using one or more external tools capable of desired functionality. For the selection of the external tools we chose three major criteria - adoption in bioinformatic community, performance of the tool, and maintainability.

4.2.1 Preprocessing

External tools used in this steps can be divided into two groups. First group containing tools capable of conversion of the raw data in BAM format to the FASTQ and second group for the tools, which provide quality control of the reads.

BAM Conversion There are three major software products capable of conversion BAM to FASTQ. The first of them - *SAMtools* - is probably the most popular tool. It contains multiple commands for handling various formats including SAM, BAM, FASTQ or VCF. It also contains an utility to convert BAM to FASTQ, but it is unable to handle paired-end data. Therefore, we decided to not to use *SAMtools*. [47]

The second and third, *bedtools* and *Picard*, are both capable of handling single-end as well as paired-end data. These tools are very similar in all selected criteria, however there are some minor differences. One of them, is that *Picard* is JVM based tool in compare to the *bedtools*, which is native module (written in C++). Therefore, we can expect,

that *bedtools* may have better performance especially when dealing with large files. [48, 49]

According to Github [3], where both packages have published their source code, both tools are actively maintained and have similar popularity, where *bedtools* have a slightly more stars, which can indicate a bigger popularity.

Quality control Over time, as the next generation of sequencing is taking popularity, the using of tools capable of quality control increased. Nowadays, there are numerous tools used for the various operations to increase quality of the reads.

We chose three commonly used softwares to solve quality control - *Trimmomatic*, *Cutadapt* and *Trim Galore!*. The first of them - *Trimmomatic* was created to overwhelm existing tools in terms of flexibility, correct handling of paired-end data and high performance. It has multiple modes including palindrome mode for the detection of 'adapter read-through', or options for quality filtering (PHRED). Although, it has been published under GPLv3 public license, we were not able to find any repository, bug tracer, or any information about how the tool is maintained. [20] Therefore, we have chosen not to use *Trimmomatic*.

Cutadapt is on the other hand still actively maintained, despite of the fact, that it is the oldest tool from our selection. *Cutadapt* is capable of searching multiple adapters in a single run and removing the best matching one. It has various options, which can help to find ideal

3. At the time of writing - 18/01/05

ratio for sensitivity-specificity or remove low-quality ends of reads. Ability to process paired-end reads is included too. With regards to the performance, it was implemented into the Python, while core aligning algorithm was implemented in C. [50, 51, 52]

On top of *Cutadapt* and another quality control tool *FastQC* is build *Trim Galore!*. It is a wrapper script to automate quality and adapter trimming. *Trim Galore!'s* main functions include handling paired-end date, filtering based on PHRED score, or removing sequencing too short for trimming process. [53] implementation was written in Perl and the whole codebase is actively maintained and available on the Github. [54]

We chose to use *Trim Galore!* over *Cutadapt* because *Trim Galore!* extends the functionality of the *Cutadapt* with addition of *FastQC*. Both tools are actively maintained and the performance should by similar, since Perl scripts used in *Trim Galore!* are relatively lightweighted.

4.2.2 Alignment to the reference genome

For alignment to the reference genome, there are again three major tools used within target community - *Bowtie 2*, *BWA* and*SOAP2*. Probably the most used tool nowadays is *Bowtie 2*. As it was described by the authors, it is an ultrafast and memory-efficient tool for aligning sequencing reads to long reference sequences. [55] The performance is achieved by using hardware-accelerated dynamic programming algorithms in combination with full-text minute indexing. This tool is excellent for aligning longer reads (starting in 50 characters), where it

outperforms the competitors. [24] For the shorter reads they suggest to compare the results with an older version - *Bowtie 1*, and by results to choose an appropriate one. [56]

Burrows-Wheeler Transformation is method for string matching, which is used by *BWA* and *SOAP2*. The first mentioned tool - *BWA* (abbr. from Burrows-Wheeler Aligner) is using backward search for exact matching and backtracking for inexact. BWA is built around three algorithms: BWA-backtrack, BWA-SW and BWA-MEM, which are specialized for various length of reads. The tool is performing well on the short reads, but the performance is degrading on the long reads, especially in the reads with the high error rates. In comparison to the *SOAP2*, *BWA* has significantly lower error rates on the simulated data. [57, 58]

Short Oligonucleotide Alignment Program 2 (SOAP2) is a second version of the popular aligning tool. It was designed for fast alignment of short reads onto a reference sequence of large-scale re-sequencing projects. [59] It is able to filter reads by the quality (PHRED), long insertions, or setting number of allowed mismatches in one read. The only big problem is, that this is probably no longer maintained, as we were not able to find any codebase and the latest update was released in 2011. [60]

From above analysis of the selected tools, we will use BWA for its ability to handle reads of variable lengths and good performance - error rate ratio. It is also relatively maintained and well documented.

4.2.3 Post-processing

This step is composed of multiple steps, where some of them are so specific to our problem, that they cannot be implemented by existing tools. An example of that step are filtering mutations induced by EMS, or filtering of the existing SNPs by the user defined filter (frequency and read depth). Before the mentioned specific filtration, we need to generate variants (SNPs) for both - reference and mutant samples.

Variant calling Tools capable of variant calling are many, but the most popular and used are *GATK* and *samtools GATK - Genome Analysis Toolkit*. It is a structured programming framework, which was designed to simplify the process of creating tools for next-generation sequencing projects. The *GATK* provides a rich set of data access patterns satisfying the majority of analysis tools needs. It is based on the MapReduce paradigm, which enables easy way to parallelize or distribute the most computational extensive problems. [36] However, the *GATK* is well-established and maintained framework, which includes other tools like *Picard*, [61] using *GATK* may include big overhead to this project for using only single function - variant calling.

Creators of the SAM and BAM file specification also developed a software for parsing and manipulating alignments in the SAM/BAM format called *SAMtools*. *SAMtools* is a small library capable of simple conversions within the mentioned formats, with some other functions including sorting and merging an alignment, removing PCR duplicates, generating per-position information, or calling SNPs. It consist

of two major components - *samtools* and *bcftools*, which are based on *htslib* - a library for high-throughput sequencing data formats implemented in C language. [62, 34, 63] That is a good prerequisite for a good compute performance.

Since *SAMtools* is well maintained and established tool, we prefer it over *GATK*, especially because *GATK* is a extensive software including many other tools and functions unnecessary in this project. *SAMtools* has a good performance, hence we chose it for variant calling.

Removing PCR duplicates The removal of PCR duplicates is a small step, which can help better process the data from next generation sequencers. This function is usually a part of a larger software packages, which are handling quality control or providing some general functionality. It is also part of the *Picard* package or *SAMtools*. Therefore, we will use a function of selected software like *SAMtools*, and do not introduce a new dependency.

4.2.4 Annotation

SNPs from the variant calling need to be annotated in a order to provide an additional information - especially biologic information related to the mutations. These biologic information are commonly stored in public databases, from which annotation tools are gathering data.

ANNOVAR, a software tool for functional annotation of genetic variants, is one of the many tools for annotation. It is capable to an-

notate various genomes and its main features include gene-based, region-based, as well as filter-based annotation. It needs a downloaded dataset, from which it can import data necessary for its function. It is still actively maintained and well documented. [64, 65]

Another popular program for annotating and predicting the effects of SNPs is *SnpEff*. It annotates variants based on their genomic locations and predicts coding effects. This genomic location includes various biological information like intronic, untranslated region, upstream, downstream and others. The main characteristics of *SnpEff* are speed, flexibility, and option to integrate with an open access platform for computational bioinformatic research - *Galaxy*. *SnpEff* differs from *ANNOVAR* in an ability to handle VCF data natively and permits annotation of more genome versions, whereas the performance of both tools are comparable. [66]

SnpEff is still maintained and used in larger projects including *Galaxy* or *GATK*. Because of the spread of the *SnpEff* and its performance, it is an ideal option for our purposes.

4.2.5 Visualization

An appropriate way of data visualization may increase the software's usability and may reveal the interesting statistical information about the visualized data. We need to choose a tool, which is capable of displaying very specific information, like a graph, which has a chromosome displayed below the x-axis and enables interactivity like zooming

or showing detailed information about the single entry within the graph.

As we chose a client-server architecture with a HTML base website on the client side, a JavaScript library is de facto the only way how to bring the interactivity to the graph. With the rising popularity of the JavaScript, the number of libraries capable of graph visualization are numerous. The most appropriate to our problem would be *D3.js - Data-Driven Documents*. None of the existing libraries, like *Highcharts*, *Google Charts* or *Chart.js*, established in the community, have such a big variability of graph visualization as the *D3.js*. It enables the direct inspection and manipulation of a native representation - the HTML DOM (document object model).*D3.js*. It also enables a very good performance alongside with the transformations and the immediate evaluation of the operators. [67].

4.3 Designing the process of the pipeline

In this chapter we will try to describe and model the whole process of visualization of EMS induced mutations from the given reference and the mutant samples. When we created the model, we took into consideration multiple factors including performance optimization and extensibility. The whole model is based on the previous analysis of the required steps for processing and tools capable of given functionality. For the modeling we used UML notation, specifically Activity diagram, which is suitable for the description of whole process.

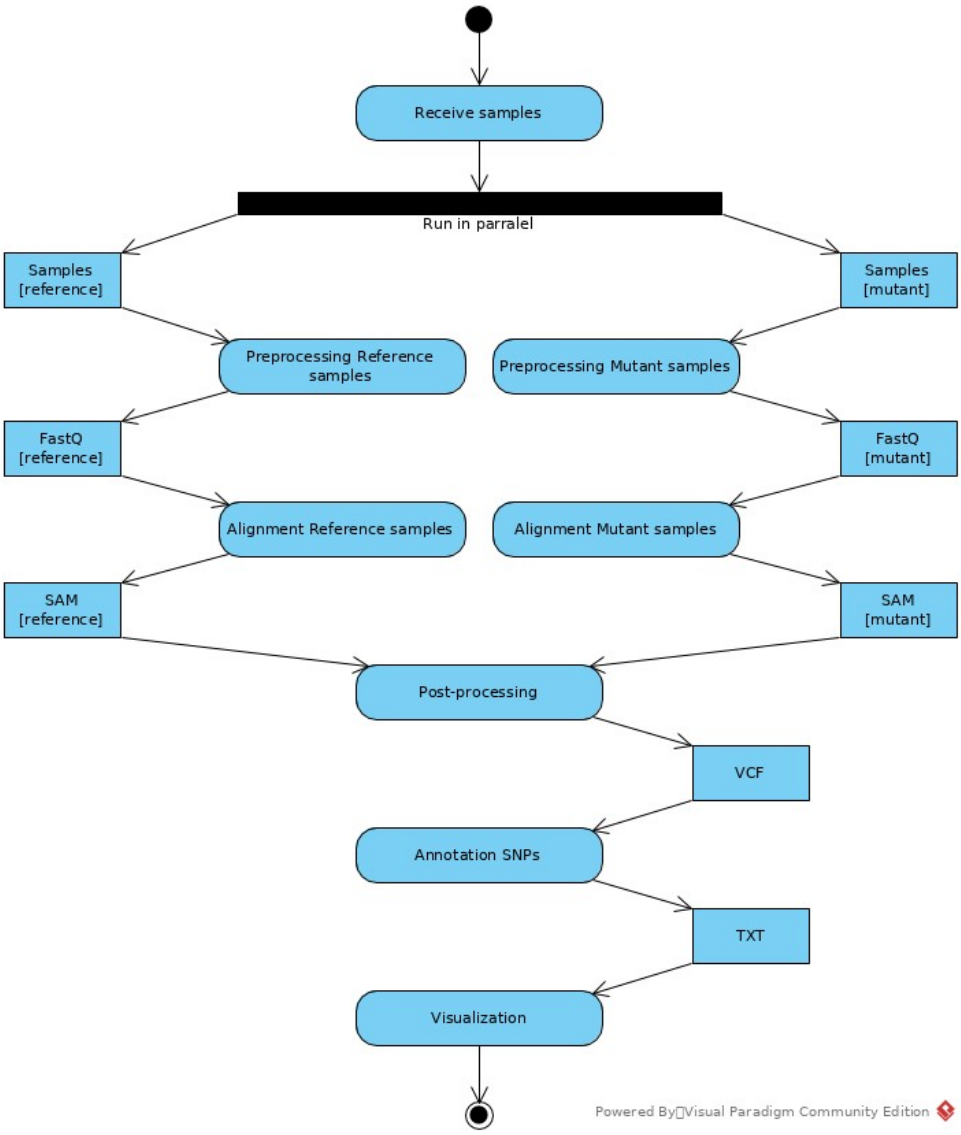

Figure 4.1: Pipeline - Activity diagram

Firstly, we describe the main actions and activities of the whole pipeline. We also provided basic data objects, which are available after all of the steps. In the following picture (figure 4.1), we can see, that the process starts with the *Receiving samples*, which include both - reference and mutant samples. These samples can be in the BAM or FASTQ format and can be either single-ended or paired-ended.

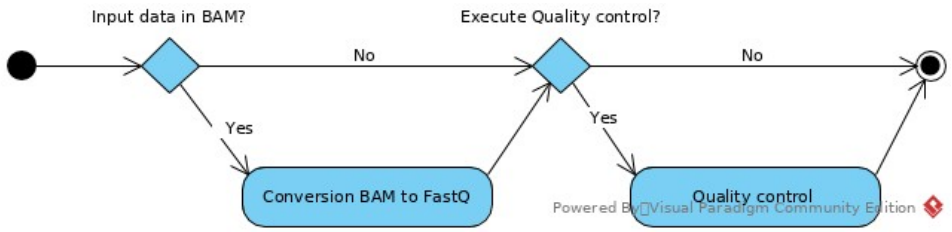

Figure 4.2: Pipeline - Preprocessing

Further steps are similar to reference and mutant samples and are processed in parallel branches. *Preprocessing* step (figure 4.2), can be divided in multiple actions, including *Conversion BAM to FASTQ* and *Quality control*, which both are optional. Preprocessing step is followed by the *Alignment*, which cannot be decomposed into smaller steps and *Post-processing*.

In the *Post-processing* step (figure 4.3), are concurrent branches merging into a single one. Thus, before that, the *PCR duplicates removal*, which is an optional step, and *Variant calling* must be executed on the both - control and mutant alignments. After that, we can filtrate SNPs, either by read's depth, frequency, exclusivity for the mutant sample or EMS induced transversions. The mentioned mutant exclusivity, is the main reason, why this step is a synchronization step and both

branches must be finished. Last two steps include the *Annotation* of the filtrated SNPs and *the Visualization*.

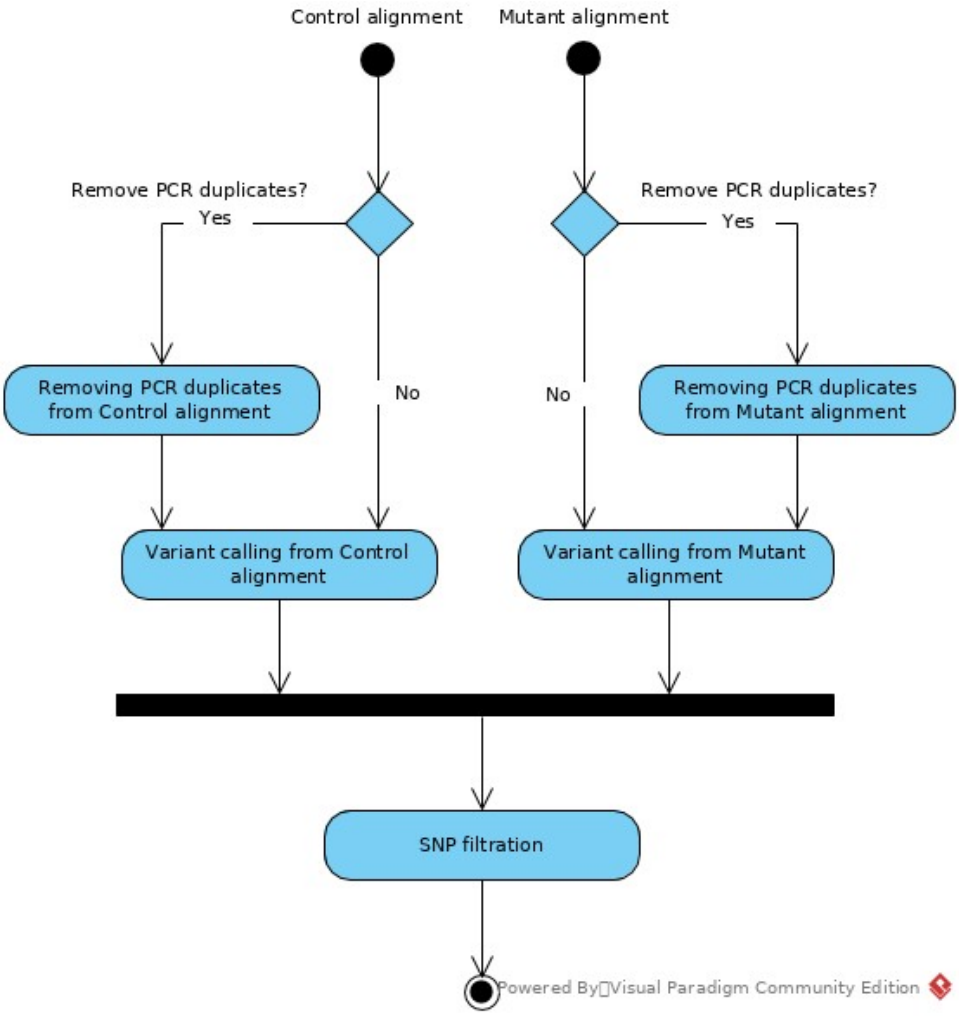

Figure 4.3: Pipeline - Post-processing

5 Implementation

The core of this thesis is the implementation of the previously designed pipeline, while satisfying all the requirements mentioned in chapter 3. This chapter is dedicated to the pipeline implementation, solving major implementation challenges, like portability, maintainability, and transparency of the processes.

5.1 Tool integration

Probably the biggest challenge during the pipeline implementation was integration of the all chosen tools into a single package, which is available on the all major platforms (Windows, Linux and MacOS), while keeping the installation process simple. We needed to integrate bedtools, SAMtools, Trim Galore!, BWA and SnpEff with all of their dependencies.

Preprocessing For conversion of BAM files to the FASTQ format, we used bedtools utility `bamtofastq`. It can be used in two ways - `bedtools bamtofastq` or `bamToFastq`. To execute conversion we use:

```
bamToFastq -i <BAM> -fq <FASTQ>
```

for single end reads and for paired end reads:

```
bamToFastq -i <BAM> -fq <FASTQ_1> -fq2 <FASTQ_2>
```

TrimGalore!, which was chosen for the quality control is implemented via command:

```
trim_galore -o <OUTPUT_DIR> --no_report_file <FASTQ>
```

respectively:

```
trim_galore -o <OUTPUT_DIR> --no_report_file --paired
    <FASTQ_1> <FASTQ_2>
```

The both TrimGalore! commands execute the quality control, which consists of three steps: quality trimming, which removes low-quality base calls; adapter trimming, which is implemented via Cutadapt, auto-detect the adapter and then removes adapter sequences; and removing short sequences, which removes sequences shorter then 20 bases. These short reads may also be generated via the previous steps. The variant with the pair end reads is slightly different in terms of removing short sequences. If at least one of the two sequences became shorter than a threshold, then the entire read pair is removed. [54]

Alignment BWA, used for alignment, has various commands for aligning different types of data. In this case, we have four possible types of incoming data:

- short single end reads
- long single end reads
- short paired end reads
- long paired end reads

For long reads, we use BWA's `mem` command. It is suitable for 70bp-1Mbp query sequences, which are aligned by seeding alignments with maximal exact matches and then extending seeds with Smith-Waterman algorithm. [68] So we use:

```
bwa mem <ARABIDOPSIS_REF.fa> <FASTQ>
```

for single end reads; and for paired end:

```
bwa mem <ARABIDOPSIS_REF.fa> <FASTQ_1> <FASTQ_2>
```

Alignment of the short reads is different although it is using the same tool. It consist of two steps: firstly we find suffix array (SA) coordinates of the input reads, and secondly we generate alignments from them. So the alignment is implemented via following commands:

```
bwa aln <ARABIDOPSIS_REF.fa> <FASTQ> > <COORDS>
bwa samse <ARABIDOPSIS_REF.fa> <COORDS> <FASTQ>
```

for single end reads; and for the paired end:

```
bwa aln <ARABIDOPSIS_REF.fa> <FASTQ_1> > <COORDS_1>
bwa aln <ARABIDOPSIS_REF.fa> <FASTQ_2> > <COORDS_2>
bwa sampe <ARABIDOPSIS_REF.fa> <COORDS_1> <COORDS_2>
    <FASTQ_1> <FASTQ_2>
```

Difference between commands used for alignment single end reads with long and short sequences is minimal. In the paired end reads we

need to create SA coordinates for both ends and provide them to the `bwa sampe`, which is paired end version of the `bwa samse` command.

The `ARABIDOPSIS_REF.fa` file is the reference genome of the *Arabidopsis thaliana*, not the provided user samples. This reference genome is provided, therefore user does not need to provide one. We also indexed the reference genome to boost the performance of the BWA. For the indexing we used `bwa index` command.

Post-processing To speed up the processes of all the operations within the post-processing step, firstly we convert the SAM input into BAM. For this we use SAMtools:

```
samtools view -bS -o <OUTPUT_BAM> <INPUT_SAM>
```

We use option `-b`, which ensures, that output will be in the BAM format, and `-S`, which we added for historical purposes, which means that input is in the SAM format. Current version of SAMtools (version 1.8) does not need this parameter and recognizes the input type automatically. After that, we sort given alignments by the leftmost coordinates using:

```
samtools sort -o <SORTED_BAM> <INPUT_BAM>
```

Optional operation - removing PCR duplicates, follows these steps. It is also implemented via SAMtools, concretely command `rmdup`. We used it in the following form:

```
samtools rmdup -s <SORTED_BAM> <OUTPUT_BAM>
```

for single end reads; for paired end reads we change `-s` for `-S` parameter. After, we index the output file for faster random access, which improves performance of the next step - variant calling. For indexing we use:

```
samtools index <BAM>
```

For the last step of the post-processing, variant calling, we will use both components of the SAMtools - samtools and bcftools. Firstly, we generate variants from the index BAM file, which will be piped into bcftools, which will extract SNPs from all the type variants.

```
samtools mpileup -Q 30 -C 50 -P Illumina
    -t DP,DV,INFO/DPR,DP4,SP,DV
    -Buf <ARABIDOPSIS_REF.fa> <INDEXED_BAM>
    | bcftools view -vcg --types snps
    > <OUTPUT>
```

Command `mpileup` which is responsible for the variant calling, where options are ensuring our requirements: minimum base quality (**-Q** 30), needed information (`-t` - format and info tags to output), reducing false SNPs caused by misalignments (`-B`) with utilities: downgrading mapping quality for reads containing excessive mismatches

(`-C`) and providing uncompressed output for piping (`-u`). The reference file is passed via option `-f`. All these variants are piped to the bcftools, provides variant calling and also filters only SNPs from all the variants and everything is redirected to the output file. [1]

For filtration of the EMS induced mutations within the given read depth boundaries satisfying minimal frequency, we do not have any existing tool, therefore it must be implemented. However, we can filter SNPs that are exclusive to the mutant samples. For this we use bedtools utility `subtractBed`, which searches features overlapping features and removes them. [69]

```
subtractBed -a <MUTANT_SNPS> -b <CONTROL_SNPS>
```

Annotation The annotation of the filtrated SNPs using SnpEff generates two files. One text document with annotated SNPs and one HTML file, which provides some information, mostly statistical, about the composition of the SNPs. Both files are accessible for the user and they are generated via:

```
snpEff Arabidopsis_thaliana <SNPS>
```

1. The complete description of the options can be found in SAMtools documentation

5.1.1 Docker

Providing the simple installation process requires setting up an environment, which has installed all the required tools with their dependencies. We chose a virtualized environment based on containers - Docker.

During implementation we have created a reference image, satisfying all the above requirements with integrated application server - Node.js, which can be used to create the containers. This image is based on the community image of Miniconda and an official image of Node.JS. We have installed all required bioinformatical tools via Miniconda and added indexed genome files of the *Arabidopsis thaliana*.

After the setup of all the necessary dependencies we expose the port 3000 outside, therefore, our Node.JS server can be accessible on that port. We also setup some of the environment variables, which are affecting the behavior of the application. Environment variables `REFERENCE_GENOME` and `REFERENCE_GENOME_DIR` are used for configuration, respectively through them we can easily adapt the whole pipeline to another genome than *Arabidopsis thaliana*. Variable `REFERENCE_GENOME` is used in all commands in above chapter, for alignment and variant calling. Together with `REFERENCE_GENOME_DIR`, we provide all necessary information about the location of the reference genome an it's name for the mentioned pipeline steps.

For the last we define the "entrypoint" of the image, which is command, which will be executed every time the container is started. Here we run `CMD npm install -g artin-dna@next && artin-dna`, which

globally can install our application and run it. The installation is using Node package manager (npm), where we published our application. We chose to always install our application because of the maintenance. During installation, the npm checks for the newest published version and installs it. Also, a Docker has it's own mechanism to deliver the latest image. With the combination of those two mechanisms we can always guarantee a piece of software, which is up-to-date. The complete Dockerfile can be found in the appendix C.

5.2 Server side application

On the server side we used Node.js an application server with the Express framework, which provides only thin layer of the fundamental web application features. [70] Using these technologies we expose an application programming interface, through which a user can control the whole application. We also used a Reactive Extensions, which are an API for asynchronous programming with observable streams. RxJS, which is a JavaScript implementation of the Reactive Extensions, which helped us to work with the processes in terms of their parallelization, concatenation or correct cancellation.

5.2.1 Application programming interface

For the communication between client and server we created an application programming interface. The whole interface consists of parts:

the first one is using the representational state transfer (REST) and the second one is communicating via the WebSockets.

We created only two REST endpoints, one for retrieval of the file structure of the server (for choosing control and mutant samples, which are later processed in the pipeline) and one for the visualization (the data from the output text file need to be parsed in order to visualize them).

For retrieval of the file structure of the server we chose the command `ls -lA`, which is Unix command for listing all files and directories including the hidden ones. To provide command above to multiple platforms, we used library ShellJS, which is implementation of Unix shell commands on top of the the Node.js API. We expose this functionality on the endpoint `GET: /api/file`, with the optional path variable `path`. The `path` is the location, from where the script `ls -lA` is called. Default location is the home one.

The parsing and the gathering of the data needed for the visualization to be exposed on the another endpoint, which is available through `GET /api/visualization`, with the required path parameter `file`. This parameter contains the absolute path of the file to be parsed. [2] The given file is then parsed, and extracted data are grouped by the chromosome (excluding the mitochondrial), and the important data such as frequency are calculated. The result is afterwards send back to the requester.

2. Only the final result in the textual format, can be parsed

Alongside the REST API we created an API, which is using a WebSocket for the communication. For the WebSocket, we used a popular socket.io library, which is using another technologies, if the WebScoket is not available in the given web browser (typically the older browsers).

We created 6 channels, through which the whole communication related to the pipeline processing passes. The first one - `pipeline-start`, is used to start the pipeline execution. The client is sending the information with files to process (control and mutant samples), information about them (length of reads, format...) and the options for the processing (providing quality control, removing PCR duplicates, frequency threshold...). The server is listening on the mentioned channel and immediately starts the pipeline execution.

After the server starts the execution of the pipeline, it receives information from the spawned processes. In our application, we distinguished 2 types of the information from the processes - informational messages produced by the tools, and the result of the operation. Both data are mapped to the operation (like BAM Conversion, Alignment) and passed to the client via `pipeline-operation-info`, respectively `pipeline-operation-result` channel. After any of the operations ended successfully, server also sends the information about the current progress of the whole pipeline through `pipeline-progress` the channel. Client side is listening on the mentioned channels and displays actual information to the user.

Processing can be also canceled, when the server receives the message in the channel `cancel-pipeline`. Then it stops the current processes, removes intermediate file structures and sends the confirmation message to the client via `pipeline-result`. In the other cases - processing successfully finished or failed, the same channel is used with the different message within it.

5.2.2 SNP Filtration

The only bigger step, for which the external tool cannot be used is SNP filtration. There is no tool, which is capable of desired functionality - filtering EMS induced transitions satisfying user selected demands, like the minimum frequency and read's depth. For above said reason, we implemented simple CLI tool fitting our requirements.

The implementation of the tool is based on the parsing the VCF input file and by the iteration of one single line at a time we processed the whole file. Naturally we skipped the comment lines, which we automatically included to the output. After that, every single line containing variant data, we parsed and extracted all the necessary information for the filtration. From these data, we compute the frequency of the mutation. When we have all the information needed for decision, whether or not to include given line into the output, we exclude all the lines which are not satisfying at least one requirement.

During implementation, we wanted to stay consistent, hence we used commander.js - a JavaScript library for building CLI tools in Node.js. We created a simple CLI tool, with the input parameters:

mutation frequency, minimum and maximum read depth, input file and output file. It can be used in the following way:

```
snpsPicker -i <INPUT> -o <OUTPUT> -f <MIN_FREQUENCY_PERCENT>
    -d <MIN_DEPTH> -D <MAX_DEPTH>
```

5.2.3 Management of the processes

Asynchronous processes, which can run simultaneously are hard to manage, therefore we used Reactive Extension, respectively its JavaScript implementation - RxJS. To utilize the advantages of the RxJS, we firstly needed to transform the spawned process into the Observable. [3] Transformation was done by sending all the messages from the process's standard output - `stdout` to the subscribed observer. To achieve above described functionality we used `Rx.Observable.create()` function.

After we transformed spawned process into an Observable, we could use `Rx.Observable.merge()` for running multiple processes at once; `Rx.Observable.concat()` for chaining processes or catching exceptions and providing controlled cancellation.

An important factor in the management processes is that they produce multiple temporary files, which need to be removed, when the process finishes, crashes or is canceled. To provide this functionality we used `Rx.Observable.finally()`, which executes a callback function

3. Observable is an internal structure of RxJS, which have defined various operations

after the Observable is completed or an error occurs. The mentioned callback function is passed as a parameter.

5.3 Client side application

We created a single page application (SPA) in the client-side. It reduces the overhead for calling the server for the current page view, and therefore it minimizes the latency and improves the user experience. We chose React-Redux stack for the creation of the client-side SPA.

Libraries used React is an open source JavaScript library for building user interfaces created by Facebook and released in 2013. [71] Its main advantage is a simple and declarative API, which is easy to learn and apply in many used cases. However, it does not contain any single component, therefore we chose an external library containing almost all of the visual components used within our application - Material-UI.

As the name may suggest, Material-UI provides components, which implement Google's Material Design. We only needed to customize some its components to satisfy our needs. As Material-UI is providing only visual components, to customize them, we used Cascading Style Sheets (CSS).

The SPA needs a mechanism, to a handle internal state of the application. Internal state is responsible for data passed to the components, which are affecting the whole application. For example, when the internal state of the application does not contain any information about the user, it may show the login page. When the user provides the login

credentials, it may validate the data against the server and then save them into the internal state.

Managing application state We chose Redux - predictable state container for JavaScript applications, for managing the application state. The whole concept is designed to help building applications, which behave consistently and are easy to test. [72] The main idea behind Redux is that the whole state of the application is stored in a single store, which acts as a single source of truth. That is the main feature, which is responsible for its characteristics.

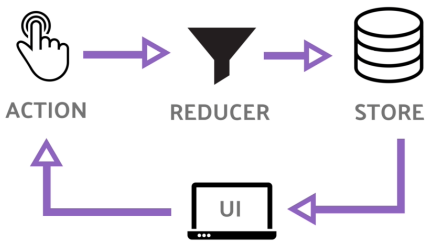

Figure 5.1: Redux - data flow [73]

The interaction within the application can be divided into two groups - one for interactions which are affecting the global state of applications (like the getting data from the server), and one which are not (expanding the panel, to see more info). Interactions, which are not affecting the global state of the application are handled within the React's component state. For the other interactions there is a simple, but effective mechanism for updating a global store and a project the updates into the UI.

The whole state of the application is stored within the global *Store*. This *Store* is passed to the *UI* components, which are rendering elements based on the *Store*. Components, which are reacting to the user's activity (like click), are dispatching *Actions* to the *Reducers*. *Reducers* are changing the *Store* by given *Actions*. It is the only place, where the *Store* can be changed.

5.3.1 Structure

The client-side application consists of components, containers, actions, reducers, prop-types, middleware, socket and utilities. Components are simple pieces of code, which are responsible for rendering of some simple parts of the application. They does not contain any important functionality and usually they are stateless. Main advantage of the components is their reusability. On the other hand, statefull components, which are responsible for "how the application works" rather than "how the application looks", we call containers. Containers typically consists of multiple components and can be linked to the Redux store. For both, containers and components, we created the definitions for the most common objects across the application. Therefore, we can provide a type checking through the library prop-types.

```
client
├── action
├── component
├── container
├── middleware
├── propTypes
├── reducer
├── socket
├── util
└── store.js
```

```
├── app.jsx
└── index.html
```

Actions are the simple objects, which are dispatched from the components. All of our actions are the same in terms of a structure - they consists of type and a payload. These simple objects are passed to the reducers, which are the most important part of the application in terms of functionality. They handle all the changes to the state. The reducers are mutating state based on type of the action. For example the action with type increment will increment the counter within the state.

We also created a mechanism, that will handle all errors, which may occur during using of the application. All errors are caught, processed, and then the error message will be shown to the user. For this mechanism we created a custom Redux middleware. Redux middleware is a point between dispatching the action and processing that action in the reducer. The main concept of the middleware is that any dispatched action will execute the middleware function. Therefore, we created a single error handler, which handles all the errors. We also created a middleware for handling incoming socket messages. The main advantage of this implementation is, that all socket messages are handled in a single point and are forwarded to the reducer within appropriate action.

Except these main packages, this project includes other less important packages like the utility, which is holding some functionality, which is too complex to be stored inside of the component or the container, or it is commonly used across the application.

5.3.2 Visualization

To visualize the output of the pipeline, we created several scatter graphs (one graph corresponds to a single chromosome), which are interactive and can be downloaded in the PNG format. However, we used D3 charting library, the implementation was a nontrivial task. D3 library enables to manipulate the HTML DOM elements (in our case the SVG elements), but the whole interactivity needs to be implemented explicitly.

The single scatter graph consists of following main components: axes, chromosome picture and the points of the graph. We have two axes - x-axis is a representation of the region, where the mutation occurs, while the y-axis is visualizing the frequency of the mutation in percent. Both axes are scalable and react to the zooming.

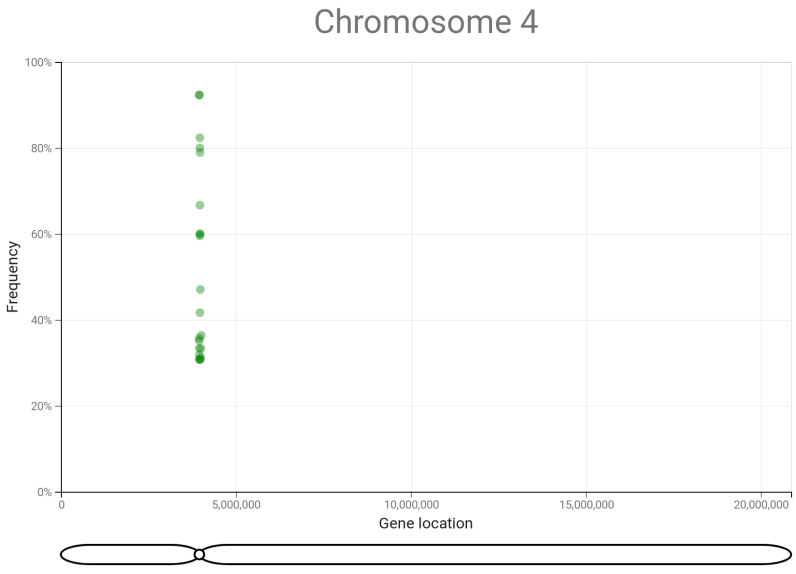

Figure 5.2: Visualization - scatter plot

Chromosome picture, which is positioned below the axis, is for orientation within the chromosome. It consists of the legs and centromere, which is the point, where the legs of the chromosome are meeting each other. This relative location of the mutation within the chromosome is a valuable information for the biologists. It is also reacting to zooming event - the zoomed area is highlighted.

Mutations, which are represented as points in the graph, are positioned relative within the axes bounds. When some area of the graph is zoomed, we need to recalculate the position of the points and hide the points, which are withing the graph area (whole SVG) and out of the aces bounds. Each point has also a function, which is showing detailed info, when one hover on the point.

6 Testing and Verification

To prove, that our pipeline is producing correct results, we need to provide a mechanism for verification. The goal of the verification was not to test the used bioinformatical tools, like BWA or SAMtools, but their integration and our implementation of the tools (SNPs filtration, visualization).

The pipeline and its parts was tested on multiple levels. The first and the most simple was the unit testing. We created numerous unit tests to prove that each step is executing the desired script in the correct order. After that, we have created several unit tests to verify, that the pipeline is executing the steps in the correct order. The main point was to test the differences within the steps, related to the input data. Steps like alignment behaves differently based on the read length and for the single and paired end data. We tested all the possible data flows in the pipeline.

For the unit testing we created an utility function, which is mimicking the spawning of the script. It is asynchronous and simplified to not spawn the tool, but only return the information about how the script was executed. This utility is usable in the unit tests, as it is fast, simple and it is mediating the information about the execution.

Filtration of the SNPs, which was a manually implemented step, was the most tested piece of the pipeline. We created unit tests for various use cases and options. The tests are covering functionality of

the parameter parsing, filtration an the application of the parameters on it, handling edge cases like missing required parameters.

Another type of testing - Beta testing, took place in the in CEITEC (Central European Institute of Technology) in Brno, Czech Republic, where it was continuously tested on the real data. During Beta testing, multiple issues with various severity was discovered and feedback was used to improve the final software.

7 Conclusion

We successfully created an easy to use software, which finds an EMS-induced mutations in the provided mutant samples. Pipeline is capable of the processing single end sequnces as well as the paired end ones. On the input it accepts the two most common file formats: BAM and FastQ.

To create the pipeline, we used various specialized tools with the custom implemented scripts, which together formed a pipeline. Choosing of appropriate tools was based on the deep analysis, and for bundling them together we used a container virtualization - Docker.

Virtualization also simplifies the installation process and portability of the final pipeline. It also enables to port the whole application to the server without any major code changes. We used an environment variables, for configuration of the application, which enables to use the pipeline with another genome than *Arabidopsis thaliana*. This configuration is available to the Docker, therefore the whole configuration is placed in a single file.

The whole software consists of two parts - client and server. Server side is responsible for the execution of the pipeline and sending the information to the client. Client is on the other hand visualizing the output in the form of scatter plots, and showing to the user current state of the operation in the real time.

Bibliography

1. OXFORD ENGLISH DICTIONARY. *genotype, n.2.* Oxford University Press, 2018. Available also from: `http://www.oed.com.ezproxy.muni.cz/view/Entry/77623`. Accessed 2018-08-04.

2. OXFORD ENGLISH DICTIONARY. *genotype, n.2.* Oxford University Press, 2018. Available also from: `http://www.oed.com.ezproxy.muni.cz/view/Entry/142359`. Accessed 2018-08-04.

3. PAREKH, S.; VINCI, V. A.; STROBEL, R. J. Improvement of microbial strains and fermentation processes. *Applied Microbiology and Biotechnology*. 2000, vol. 54, no. 3, pp. 287–301. ISSN 1432-0614. Available from DOI: `10.1007/s002530000403`.

4. EHLING, UH. Comparison of radiation-and chemically-induced dominant lethal mutations in male mice. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*. 1971, vol. 11, no. 1, pp. 35–44.

5. PRENTKI, Pierre; KRISCH, Henry M. In vitro insertional mutagenesis with a selectable DNA fragment. *Gene*. 1984, vol. 29, no. 3, pp. 303–313.

6. KIM, YongSig; SCHUMAKER, Karen S.; ZHU, Jian-Kang. EMS Mutagenesis of Arabidopsis. In: *Arabidopsis Protocols*. Ed. by SALINAS, Julio; SANCHEZ-SERRANO, Jose J. Totowa, NJ: Humana Press, 2006, pp. 101–103. ISBN 978-1-59745-003-4. Available from DOI: `10.1385/1-59745-003-0:101`.

7. HU, Mei-Rong; CHAO, Ya-Peng; ZHANG, Guo-Qing; YANG, Xiu-Qing; XUE, Zhi-Quan; QIAN, Shi-Jun. Molecular evolution of Fome lignosus laccase by ethyl methane sulfonate-based random mutagenesis in vitro. *Biomolecular Engineering*. 2007, vol. 24, no. 6, pp. 619–624. ISSN 1389-0344. Available from DOI: https://doi.org/10.1016/j.bioeng.2007.08.020.

8. JAMES, D. W.; DOONER, H. K. Isolation of EMS-induced mutants in Arabidopsis altered in seed fatty acid composition. *Theoretical and Applied Genetics*. 1990, vol. 80, no. 2, pp. 241–245. ISSN 1432-2242. Available from DOI: 10.1007/BF00224393.

9. UCHIDA, Naoyuki; SAKAMOTO, Tomoaki; KURATA, Tetsuya; TASAKA, Masao. Identification of EMS-Induced Causal Mutations in a Non-Reference Arabidopsis thaliana Accession by Whole Genome Sequencing. *Plant and Cell Physiology*. 2011, vol. 52, no. 4, pp. 716–722. Available from DOI: 10.1093/pcp/pcr029.

10. KOORNNEEF, M.; VEEN, J. H. van der. Induction and analysis of gibberellin sensitive mutants in Arabidopsis thaliana (L.) heynh. *Theoretical and Applied Genetics*. 1980, vol. 58, no. 6, pp. 257–263. ISSN 1432-2242. Available from DOI: 10.1007/BF00265176.

11. HOFFMANN, Matthias H. Biogeography of Arabidopsis thaliana (l.) heynh.(Brassicaceae). *Journal of Biogeography*. 2002, vol. 29, no. 1, pp. 125–134.

12. INITIATIVE, Arabidopsis Genome et al. Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. *nature*. 2000, vol. 408, no. 6814, pp. 796.

13. VALVEKENS, Dirk; VAN MONTAGU, Marc; VAN LIJSEBETTENS, Mieke. Agrobacterium tumefaciens-mediated transformation of Arabidopsis thaliana root explants by using kanamycin selection. *Proceedings of the National Academy of Sciences*. 1988, vol. 85, no. 15, pp. 5536–5540.

14. GIBEAUT, David M; HULETT, John; CRAMER, Grant R; SEEMANN, Jeffrey R. Maximal biomass of Arabidopsis thaliana using a simple, low-maintenance hydroponic method and favorable environmental conditions. *Plant Physiology*. 1997, vol. 115, no. 2, pp. 317.

15. VIGNAL, Alain; MILAN, Denis; SANCRISTOBAL, Magali; EGGEN, André. A review on SNP and other types of molecular markers and their use in animal genetics. *Genetics Selection Evolution*. 2002, vol. 34, no. 3, pp. 275.

16. SHASTRY, Barkur S. SNPs: Impact on Gene Function and Phenotype. In: *Single Nucleotide Polymorphisms: Methods and Protocols*. Ed. by KOMAR, Anton A. Totowa, NJ: Humana Press, 2009, pp. 3–22. ISBN 978-1-60327-411-1. Available from DOI: 10.1007/978-1-60327-411-1_1.

17. EWING, Brent; GREEN, Phil. Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Research*. 1998, vol. 8, no. 3, pp. 186–194. Available from DOI: 10.1101/gr.8.3.186.

18. LINDGREEN, Stinus. AdapterRemoval: easy cleaning of next-generation sequencing reads. *BMC research notes*. 2012, vol. 5, no. 1, pp. 337.

19. PATEL, Ravi K.; JAIN, Mukesh. NGS QC Toolkit: A Toolkit for Quality Control of Next Generation Sequencing Data. *PLOS ONE*. 2012, vol. 7, no. 2, pp. 1–7. Available from DOI: 10.1371/journal.pone.0030619.

20. BOLGER, Anthony M; LOHSE, Marc; USADEL, Bjoern. Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*. 2014, vol. 30, no. 15, pp. 2114–2120.

21. BABRAHAM BIOINFORMATICS. *Before Quality Trimming*. Available also from: https://raw.githubusercontent.com/FelixKrueger/TrimGalore/master/Docs/Images/poor_q_per_sequence.png. Accessed 2018-09-04.

22. BABRAHAM BIOINFORMATICS. *After Quality Trimming*. Available also from: https://raw.githubusercontent.com/FelixKrueger/TrimGalore/master/Docs/Images/fixed_quals_per_sequence.png. Accessed 2018-09-04.

23. LI, Heng; DURBIN, Richard. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*. 2010, vol. 26, no. 5, pp. 589–595. Available from DOI: 10.1093/bioinformatics/btp698.

24. LANGMEAD, Ben; SALZBERG, Steven L. Fast gapped-read alignment with Bowtie 2. *Nature methods*. 2012, vol. 9, no. 4, pp. 357.

25. SAIKI, RK; SCHARF, S; FALOONA, F; MULLIS, K; HOORN, GT; ARNHEIM, N. Polymerase chain reaction. *Science*. 1985, vol. 230, pp. 1350–1354.

26. NIELSEN, Rasmus; PAUL, Joshua S; ALBRECHTSEN, Anders; SONG, Yun S. Genotype and SNP calling from next-generation sequencing data. *Nature Reviews Genetics*. 2011, vol. 12, no. 6, pp. 443.

27. SIMS, David; SUDBERY, Ian; ILOTT, Nicholas E; HEGER, Andreas; PONTING, Chris P. Sequencing depth and coverage: key considerations in genomic analyses. *Nature Reviews Genetics*. 2014, vol. 15, no. 2, pp. 121.

28. OXFORD ENGLISH DICTIONARY. *centromere*. Oxford University Press, 2018. Available also from: `http://www.oed.com/view/Entry/38543553`. Accessed 2018-30-04.

29. COCK, Peter J. A.; FIELDS, Christopher J.; GOTO, Naohisa; HEUER, Michael L.; RICE, Peter M. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*. 2010, vol. 38, no. 6, pp. 1767–1771. Available from DOI: `10.1093/nar/gkp1137`.

30. COCK, Peter J. A.; FIELDS, Christopher J.; GOTO, Naohisa; HEUER, Michael L.; RICE, Peter M. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*. 2010, vol. 38, no. 6, pp. 1767–1771. Available from DOI: `10.1093/nar/gkp1137`.

31. KODAMA, Yuichi; SHUMWAY, Martin; LEINONEN, Rasko. The Sequence Read Archive: explosive growth of sequencing data. *Nucleic acids research*. 2011, vol. 40, no. D1, pp. D54–D56.

32. COX, Murray P; PETERSON, Daniel A; BIGGS, Patrick J. SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. *BMC bioinformatics*. 2010, vol. 11, no. 1, pp. 485.

33. SAM/BAM FORMAT SPECIFICATION WORKING GROUP AND OTHERS. *Sequence alignment/map format specification*. Samtools organisation for next-generation sequencing developers, 2017. Available also from: `https://www.ncbi.nlm.nih.gov/sra/docs/submitformats`. Accessed 2018-09-04.

34. LI, Heng et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*. 2009, vol. 25, no. 16, pp. 2078–2079. Available from DOI: `10.1093/bioinformatics/btp352`.

35. NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION, U.S. NATIONAL LIBRARY OF MEDICINE. *File Format Guide*. National Center for Biotechnology Information. Available also from: `https://www.ncbi.nlm.nih.gov/sra/docs/submitformats`. Accessed 2018-08-04.

36. MCKENNA, Aaron et al. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*. 2010, vol. 20, no. 9, pp. 1297–1303. Available from DOI: `10.1101/gr.107524.110`.

37. DANECEK, Petr et al. The variant call format and VCFtools. *Bioinformatics*. 2011, vol. 27, no. 15, pp. 2156–2158. Available from DOI: 10.1093/bioinformatics/btr330.

38. SAMTOOLS ORGANISATION FOR NEXT-GENERATION SEQUENCING DEVELOPERS. *The Variant Call Format Specification*. 2017. Available also from: http://samtools.github.io/hts-specs/VCFv4.3.pdf. Accessed 2018-09-04.

39. *Anaconda Cloud bioconda* [https://anaconda.org/bioconda/]. Accessed 2018-04-15.

40. GRÜNING, Björn; DALE, Ryan; SJÖDIN, Andreas; ROWE, Jillian; CHAPMAN, Brad A; TOMKINS-TINCH, Christopher H; VALIERIS, Renan; KÖSTER, Johannes, et al. Bioconda: A sustainable and comprehensive software distribution for the life sciences. *bioRxiv*. 2017, pp. 207092.

41. *Using Bioconda - Bioconda documentation* [https://bioconda.github.io/index.html]. Accessed 2018-04-17.

42. *Installation - Conda documentation* [https://conda.io/docs/user-guide/install/index.html]. Accessed 2018-04-17.

43. ANDERSON, Thomas; PETERSON, Larry; SHENKER, Scott; TURNER, Jonathan. Overcoming the Internet impasse through virtualization. *Computer*. 2005, vol. 38, no. 4, pp. 34–41.

44. *Kernel Definition* [https://expressjs.com/]. Accessed 2018-05-01.

45. MERKEL, Dirk. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, vol. 2014, no. 239. ISSN 1075-3583. Available also from: http://dl.acm.org/citation.cfm?id=2600239.2600241.

46. TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*. 2010, vol. 14, no. 6, pp. 80–83.

47. *samtools(1) manual page* [http://bedtools.readthedocs.io/en/latest/content/tools/subtract.html]. Accessed 2018-05-13.

48. *bedtools: a powerful toolset for genome arithmetic* [http://bedtools.readthedocs.io/en/latest/]. Accessed 2018-05-01.

49. *Picard Tools - By Broad Institute* [http://broadinstitute.github.io/picard/]. Accessed 2018-05-01.

50. MARTIN, Marcel. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet. journal*. 2011, vol. 17, no. 1, pp. pp–10.

51. *cutadapt - cutadapt 1.16 documentation* [https://cutadapt.readthedocs.io/en/stable/index.html]. Accessed 2018-05-03.

52. *cutadapt: cutadapt removes adapter sequences from sequencing reads* [https://github.com/marcelm/cutadapt]. Accessed 2018-05-03.

53. *Babraham Bioinformatics - Trim Galore!* [https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/]. Accessed 2018-05-03.

54. *TrimGalore: A wrapper around Cutadapt and FastQC to consistently apply adapter and quality trimming to FastQ files, with extra functionality for RRBS data* [https://github.com/marcelm/cutadapt]. Accessed 2018-05-03.

55. *Bowtie 2: Fast and sensitive read alignment* [http://bowtie-bio.sourceforge.net/bowtie2/index.shtml]. Accessed 2018-05-03.

56. *Bowtie 2: Fast and sensitive read alignment - FAQ* [http://bowtie-bio.sourceforge.net/bowtie2/faq.shtml]. Accessed 2018-05-03.

57. LI, Heng; DURBIN, Richard. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*. 2009, vol. 25, no. 14, pp. 1754–1760. Available from DOI: 10.1093/bioinformatics/btp324.

58. *Manual Reference Pages - bwa (1)* [http://bio-bwa.sourceforge.net/bwa.shtml]. Accessed 2018-05-13.

59. LI, Ruiqiang; YU, Chang; LI, Yingrui; LAM, Tak-Wah; YIU, Siu-Ming; KRISTIANSEN, Karsten; WANG, Jun. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*. 2009, vol. 25, no. 15, pp. 1966–1967. Available from DOI: 10.1093/bioinformatics/btp336.

60. *SOAP :: Short Oligonucleotide Analysis Package* [http://soap.genomics.org.cn/soapaligner.html]. Accessed 2018-05-04.

61. *Official code repository for GATK versions 4 and up* [https://github.com/broadinstitute/gatk/]. Accessed 2018-05-05.

62. LI, Heng. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*. 2011, vol. 27, no. 21, pp. 2987–2993. Available from DOI: 10.1093/bioinformatics/btr509.

63. *htslib: C library for high-throughput sequencing data formats* [https://github.com/samtools/htslib]. Accessed 2018-05-05.

64. WANG, Kai; LI, Mingyao; HAKONARSON, Hakon. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Research*. 2010, vol. 38, no. 16, pp. e164. Available from DOI: 10.1093/nar/gkq603.

65. *ANNOVAR Documentation* [http://annovar.openbioinformatics.org/en/latest/]. Accessed 2018-05-05.

66. CINGOLANI, Pablo; PLATTS, Adrian; WANG, Le Lily; COON, Melissa; NGUYEN, Tung; WANG, Luan; LAND, Susan J; LU, Xiangyi; RUDEN, Douglas M. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*. 2012, vol. 6, no. 2, pp. 80–92.

67. BOSTOCK, Michael; OGIEVETSKY, Vadim; HEER, Jeffrey. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*. 2011, vol. 17, no. 12, pp. 2301–2309.

68. *Burrows-Wheeler Aligner* [http://bio-bwa.sourceforge.net/]. Accessed 2018-05-04.

69. *subtract - bedtools 2.27.0 documentation* [http://bedtools.readthedocs.io/en/latest/]. Accessed 2018-05-01.

70. *Express - Node.js web application framework* [http://www.linfo.org/kernel.html]. Accessed 2018-05-01.

71. OCCHINO, Tom. React. js Conf 2015 Keynote-Introducing React Native. *Facebook Developers-YouTube*. 2015.

72. *ANNOVAR Documentation* [https://redux.js.org/]. Accessed 2018-05-11.

73. FERNANDES, Rodrigo F. *Integrating semantic-ui modal with Redux*. Available also from: https://itnext.io/integrating-semantic-ui-modal-with-redux-4df36abb755c. Accessed 2018-05-12.

A Source code

The source code of the project can be found in the attached zip archive within the information system of Masaryk University.

B Installation guide

The installation of the pipeline on a local computer consists of four simple steps:

1. First, install Docker in your computer. Docker can be installed from `https://docs.docker.com/install/`
2. Install Docker Compose from `https://docs.docker.com/compose/install/`
3. Create a file with file name "docker-compose.yml" and placed it in folder from where you want to run the pipeline. This file should contain following lines:

```
# docker-compose.yml
version: '3'
services:
  web:
    image: "javorka/artin-dna:next"
    ports:
      - "3000:3000"
    volumes:
      - /:/sharedFolder
    network_mode: host
    environment:
      - REFERENCE_GENOME=
          Arabidopsis_thaliana
```

4. Open your command prompt and run `docker-compose up` from the directory, where the docker-compose.yml is stored. Now the application should be accessible on localhost:3000

C Dockerfile

```
1  FROM continuumio/miniconda3
2  MAINTAINER Peter Javorka <peter.javorka@artin.
       cz>
3  RUN echo "Installing␣bioconda"
4
5  RUN conda config --add channels conda-forge \
6      && conda config --add channels bioconda
7
8  RUN echo "Installing␣required␣tools"
9
10 RUN conda install bwa \
11     samtools \
12     snpeff \
13     trim-galore \
14     cutadapt \
15     bedtools \
16     bcftools
17
18 RUN groupadd --gid 1000 node \
19   && useradd --uid 1000 --gid node --shell /
         bin/bash --create-home node
20
```

```
21 # gpg keys listed at https://github.com/nodejs
     /node#release-team
22 RUN echo "Installing␣Node␣JS"
23
24 RUN set -ex \
25   && for key in \
26     94AE36675C464D64BAFA68DD7434390BDBE9B9C5 \
27     FD3A5288F042B6850C66B31F09FE44734EB7990E \
28     71DCFD284A79C3B38668286BC97EC7A07EDE3FC1 \
29     DD8F2338BAE7501E3DD5AC78C273792F7D83545D \
30     C4F0DFFF4E8C1A8236409D08E73BC641CC11F4C8 \
31     B9AE9905FFD7803F25714661B63B535A4C206CA9 \
32     56730D5401028683275BD23C23EFEFE93C4CFFFE \
33     77984A986EBC2AA786BC0F66B01FBB92821C587A \
34   ; do \
35     gpg --keyserver pgp.mit.edu --recv-keys "
         $key" || \
36     gpg --keyserver keyserver.pgp.com --recv-
         keys "$key" || \
37     gpg --keyserver ha.pool.sks-keyservers.net
          --recv-keys "$key" ; \
38   done
39
40 ENV NODE_VERSION 9.2.0
41
```

```
42 RUN ARCH= && dpkgArch="$(dpkg␣--print-
     architecture)" \
43   && case "${dpkgArch##*-}" in \
44     amd64) ARCH='x64';; \
45     ppc64el) ARCH='ppc64le';; \
46     s390x) ARCH='s390x';; \
47     arm64) ARCH='arm64';; \
48     armhf) ARCH='armv7l';; \
49     i386) ARCH='x86';; \
50     *) echo "unsupported␣architecture"; exit 1
         ;; \
51   esac \
52   && curl -SLO "https://nodejs.org/dist/
     v$NODE_VERSION/node-v$NODE_VERSION-linux-
     $ARCH.tar.xz" \
53   && curl -SLO --compressed "https://nodejs.
     org/dist/v$NODE_VERSION/SHASUMS256.txt.
     asc" \
54   && gpg --batch --decrypt --output SHASUMS256
     .txt SHASUMS256.txt.asc \
55   && grep "␣node-v$NODE_VERSION-linux-$ARCH.
     tar.xz\$" SHASUMS256.txt | sha256sum -c -
      \
```

```
56    && tar -xJf "node-v$NODE_VERSION-linux-$ARCH
         .tar.xz" -C /usr/local --strip-components
         =1 --no-same-owner \
57    && rm "node-v$NODE_VERSION-linux-$ARCH.tar.
         xz" SHASUMS256.txt.asc SHASUMS256.txt \
58    && ln -s /usr/local/bin/node /usr/local/bin/
         nodejs
59
60  ENV YARN_VERSION 1.3.2
61
62  RUN set -ex \
63    && for key in \
64      6A010C5166006599AA17F08146C2130DFD2497F5 \
65    ; do \
66      gpg --keyserver pgp.mit.edu --recv-keys "
           $key" || \
67      gpg --keyserver keyserver.pgp.com --recv-
           keys "$key" || \
68      gpg --keyserver ha.pool.sks-keyservers.net
            --recv-keys "$key" ; \
69    done \
70    && curl -fSLO --compressed "https://yarnpkg.
         com/downloads/$YARN_VERSION/yarn-
         v$YARN_VERSION.tar.gz" \
```

```
71    && curl -fSLO --compressed "https://yarnpkg.
         com/downloads/$YARN_VERSION/yarn-
         v$YARN_VERSION.tar.gz.asc" \
72    && gpg --batch --verify yarn-v$YARN_VERSION.
         tar.gz.asc yarn-v$YARN_VERSION.tar.gz \
73    && mkdir -p /opt/yarn \
74    && tar -xzf yarn-v$YARN_VERSION.tar.gz -C /
         opt/yarn --strip-components=1 \
75    && ln -s /opt/yarn/bin/yarn /usr/local/bin/
         yarn \
76    && ln -s /opt/yarn/bin/yarn /usr/local/bin/
         yarnpkg \
77    && rm yarn-v$YARN_VERSION.tar.gz.asc yarn-
         v$YARN_VERSION.tar.gz
78
79  RUN echo "Adding␣genome␣basis"
80  COPY data /genome
81
82  ENV DEBUG dna:*
83  ENV REFERENCE_GENOME_DIR /genome
84  EXPOSE 3000
85
86  CMD npm install -g artin-dna@next && artin-dna
```