CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR THESIS

Hana Mertanová

Clustering of RNA-seq Reads by Gene Expression Levels

Department of Cybernetics

Thesis supervisor: **Petr Ryšavý**

May 2019

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Mertanová Hana**	Personal ID number:	**465875**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Clustering of RNA-seq Reads by Gene Expression Levels

Bachelor’s thesis title in Czech:

Shlukování RNA-seq reads podle genové exprese

Guidelines:

Differential gene expression analysis is one of the main subjects of interest in bioinformatics. The number of transcripts of a gene correlates with the corresponding protein level in a cell. Current bioinformatics pipelines are based on aligning of RNA-seq reads to reference (and therefore genomes) and consecutive analysis of expression levels. The reference genome alignment is currently the most time and memory consuming task of the analysis. Moreover, the reference may not always be available to the researcher at the time of the analysis. The goal of this thesis is to propose approaches for clustering reads based on estimated expression levels when no reference is available. Such clustering may be used for faster transcriptome assembly or classification of tissues.

1) Get familiar with bioinformatics, including a basic overview of molecular biology, sequencing techniques, sequence assembly, RNA-seq, and differential gene-analysis.
2) Review standard clustering techniques, including k-means, hierarchical clustering, and density-based methods.
3) Provide a brief review of the previous two points.
4) Formally define the problem.
5) Analyze provided data using the standard reference-based approach.
6) Design a prototype of an algorithm that is able to cluster reads based on expression levels.
7) Analyze how the clusters correlate with true genes calculated by the standard reference-based approach. Provide any other necessary experimental evaluation.
8) Discuss possibilities for further usage of clusters in transcriptome assembly or classification tasks.

Bibliography / sources:

[1] Neil C. Jones, Pavel A. Pevzner - An Introduction to Bioinformatics Algorithms - MIT press, 2004
[2] Patro R., Mount S. M., Kingsford C. - Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms - Nature Biotechnology, 2014
[3] Phillip E C Compeau, Pavel A Pevzner, Glenn Tesler - How to apply de Bruijn graphs to genome assembly - Nature Biotechnology, 2011
[4] Jure Leskovec, Anand Rajaraman, Jeff Ullman - Mining of Massive Datasets; chapter 7 - Cambridge University Press, 2014
[5] Dündar, Friederike, Luce Skrabanek, and Paul Zumbo - Introduction to differential gene expression analysis using RNA-seq - Appl. Bioinformatics, 2015

Name and workplace of bachelor’s thesis supervisor:

Bc. Petr Ryšavý, MSc., Intelligent Data Analysis, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **13.12.2018** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Bc. Petr Ryšavý, MSc. Supervisor’s signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Ing. Pavel Ripka, CSc. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisor Petr Ryšavý for the opportunity to work in the area of bioinformatics and also for his encouraging guidance.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

I would also like to thank Elizabeth Thompson, College of Biological Sciences, University of Minnesota for the input data that allowed the experiments to be performed.

Abstract

Sequencing technologies produce a high amount of bioinformatic data. These data are then processed by various algorithms, gathering the information about the DNA structure, the cell condition and many others.
In this thesis, we introduce the basic concepts and methods used to process the sequenced data. Specifically, we focus on the gene expression analysis.
Standard approaches are based on aligning the input sequences to the reference. Unlike these reference-based pipelines, our main goal is to categorize the input sequences according to the membership to the different genes without any reference.
Finally, we compare our solution to the reference-based algorithm.
Keywords: clustering, reads, gene expression, reference-free.

Abstrakt

Technológie na sekvenovanie produkujú veľké množstvo bioinformatických dát. Z týchto dát je možné získať celé spektrum informácií, ako napríklad štruktúru DNA, stav buniek a veľa ďalších.
V tejto práci uvedieme základné koncepty a metódy používané na spracovanie dát získaných sekvenovaním. Zameriame sa najmä na analýzu génovej expresie.
Bežný prístup je založený na priraďovaní sekvencií na úseky v referenčnom reťazci. Na rozdiel od prístupov založených na referencii, naším cieľom bude rozdeliť sekvencie podľa príslušnosti k jednotlivým génom bez znalosti referenčného reťazca.
Na záver porovnáme naše riešenie so štandardným algoritmom založeným na metóde využívajúcej referenciu.
Kľúčové slová: zhlukovanie, ready, génová expresia, bez referencie.

Contents

List of Figures

List of Tables

1 Introduction

The sequencing technologies allow us to analyze the DNA and the genome. As the DNA encodes the essential information to perform the protein synthesis, it is rather desirable to study it and use the information it provides.

Moreover, if the financially available techniques were developed, it would be beneficial also in the medical analysis.

The output of the sequencing technologies consists of sequences that are further processed.

In this thesis, we first provide a brief introduction to molecular biology to explain the basic terminology.

Secondly, we describe the most common methods, such as read-assembly. The task of the read assembly is to compose the longer sequences from the smaller ones or to compose the whole genome. However, the computational complexity of composing long genomes remains enormous. We also offer a brief overview of the RNA-seq.

Thirdly, we discuss the essential clustering methods.

The main part of this thesis begins in Chapter 7. We focus on the read clustering according to the gene expression levels. Since the reads originating from the same region (gene) have a similar expression, we can use a clustering method (by expression) to sort reads into categories. These categories should correspond with the genes, determining which read belongs to which gene. However, if we used only reads from one subject, multiple genes could have similar expression, which would lead to merging reads that belong to different genes. Therefore, we use multiple subject to increase the dimension of the clustering space. The more subjects differ from each other, the easier it is to distinguish between the genes.

Thus, we try to create the collections containing reads that have a similar expression in multiple subjects. However, we do not obtain information about the gene names corresponding to the resulting clusters. After that, we design a pipeline that solves the problem presented above.

We also describe the standard approach. Its main drawback is the requirement of the reference and the time-consuming aligning, which we avoid entirely in our solution.

Finally, we compare our results to the reference-based approach. We also try to estimate the best parameters of our algorithm producing the lowest error.

We conclude the thesis with a discussion of possible future applications of our algorithm.

2 Molecular Biology Background

Human genetic information and genetic information of most living creatures is stored in the DNA. It is inherited over generations, stored in a nucleus and mitochondria of most cells. The importance of DNA is very significant, as its functionality influences many processes and its corruption in the early stages of life leads to various organism dysfunctions. The RNA molecule is very similar to the DNA. Both of them are crucial in the life of a cell (and therefore in a life of a whole organism). However, RNA has a slightly different structure and functionality.

2.1 DNA and RNA Structure

DNA consists of 2 strands that are forming a double-stranded alpha helix as shown in Figure 2.1. Each strand is built from nucleotides (adenine, cytosine, guanine, thymine) attached to deoxyribose and phosphate group. RNA is only single-stranded. It also consists of a phosphate group and nucleotides, but instead of thymine there is uracil, and the deoxyribose is replaced by the ribose. In this thesis, we will represent strands as strings of nucleotides (T-thymine, A-adenine, G-guanine, C-cytosine, U-uracil) in the same order as it is in the DNA-strand.

2.2 3′-5′ end of DNA strand

The carbons of the deoxyribose are numbered such as in Figure 2.2. The phosphate groups are attached to 5′ and 3′ carbons. This results to the difference between the ends of a strand. We denote as 3′-end the one that has „free“ 3′ carbon (another phosphate group is replaced by -OH group). Similarly, we denote the other end as 5′-end, as the last base has „free“ 5′ carbon (bouned with a phosphate group). In our string notation, we start from 5′-end as it is the natural order of reading DNA strands.

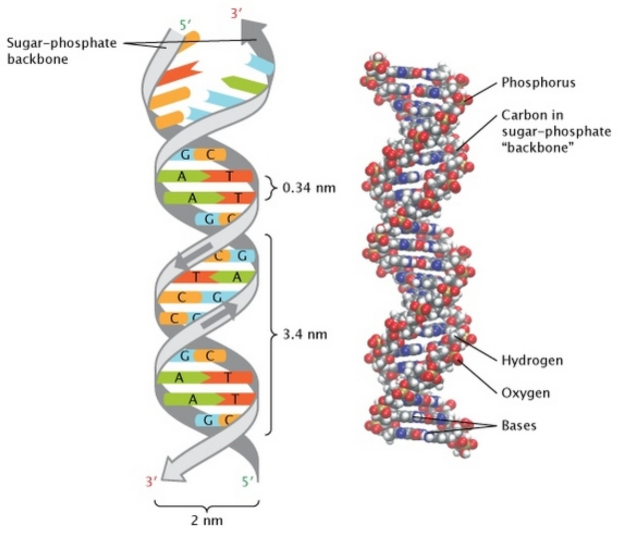


FIGURE 2.1: 3D DNA structure reused from [40]

2.3 Strand Complementarity

Complementarity is an important property of DNA strands. Guanine is complementary to cytosine and adenine to thymine (uracil in RNA), this is also known as Watson-Cricks-rule [40] . If we read one strand from one end to the other (e.g., ACCGTAA), the other strand is its complement (TGGCATT). However, the complement is read from the other end (from 5′-end to 3′-end, resulting in TTACGGT). Figure 2.2 shows the chemical bonding between the paired bases.

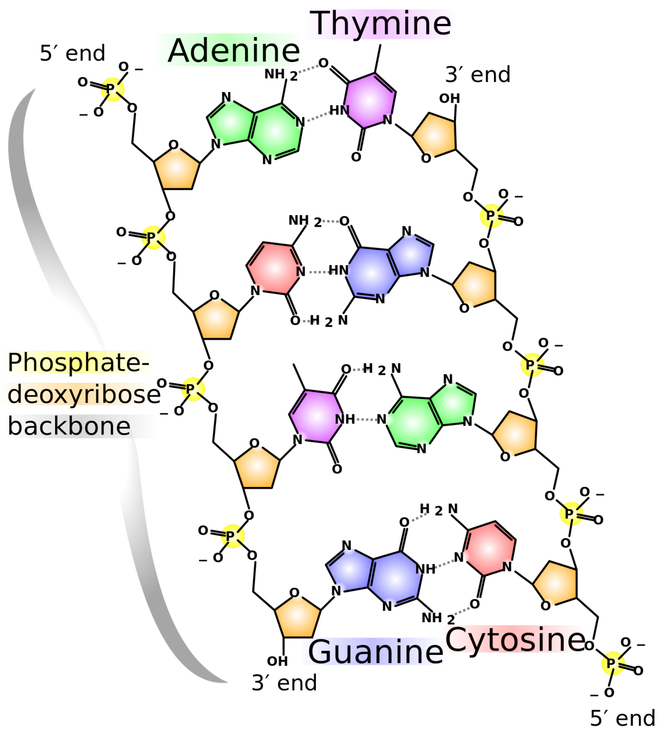


FIGURE 2.2: DNA chemical structure reused from [32]

2.4 DNA - Function

DNA encodes information needed to synthesize proteins. This process is called proteo-synthesis. Proteo-synthesis consists of transcribing and translating DNA.

It starts with transcribing DNA to mRNA in the nucleus of the cell. That means the synthesis of the complementary RNA strand to the original DNA. Then the mRNA is processed by the ribosome in the cell cytoplasm. This step is called the translation. Small segments of RNA called t-RNA containing precisely three bases (these are also called codons) carrying amino-acids are being attached to the m-RNA according to the complementarity rule. The amino-acid is bonded to the growing polypeptide (protein). Hence, the order of amino-acids is determined by order of the nucleotides, as every amino-acid is determined by three succeeding bases. This process is terminated by attaching the stop codon.

The DNA molecule can be further split into smaller segments. It turns out, that not whole DNA molecule is transcripted in the first phase of protein biosynthesis, rather these smaller segments called genes. Thus, corruption of a part of the strand usually influences only the synthesis of some proteins. However, only specific parts of the gene called exons are translated and further processed in the ribosomes.

The result of proteosynthesis is a protein. We will discuss the importance of proteins in the following section.

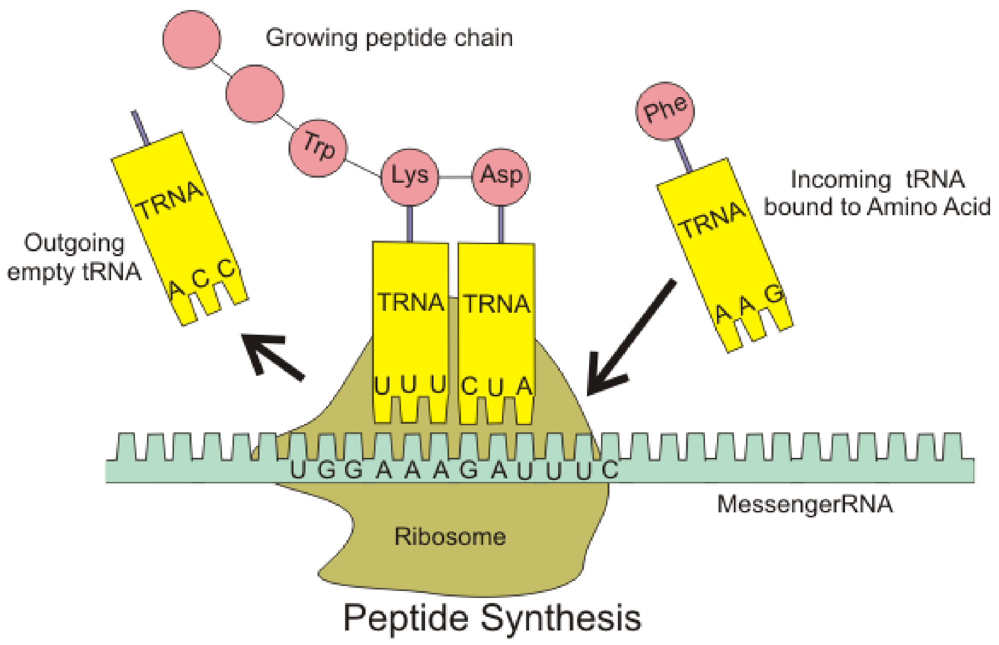


FIGURE 2.3: DNA translation reused from [9]

2.5 Protein Function

Proteins are biocatalysts of chemical reactions (such as sugar decomposition), so it is highly important to synthesize them sufficiently. Enzymes are made of proteins, and these control the whole life cycle of a cell, and decisions it has to make. For example, when the cell should replicate or how to react on the extern stimulus. If this mechanism does not work, it leads to various negative consequences, such as cancer if a cell does not recognize that it should die.

2.6 Applications

Since DNA is so important, it seems reasonable to maintain as much information about it as it is possible. Naturally, scientists wanted to know the exact structure of the human genome, which segments are responsible for which proteins. It is also interesting, whether there are any similarities of DNA through different species. Multiple projects emerged to answer these question, such as Human genome project (international project research focusing on determining the precise sequence of human DNA [6]). Another international project research creating a detailed catalog of the human genome of at least 1000 different ethnic groups named 1000genomes started in 2008 and was completed in 2012 [5] . Therefore, it was necessary to develop some tools to read sequences of nucleotides. We will describe these in the following chapter.

3 Sequencing Methods

In this chapter, we will discuss different approaches to DNA/RNA sequencing. The variety of sequencing methods is huge. As the length of DNA-strand is high and we want to sequence multiple copies to minimize the error probability, the efficiency of the chosen method is essential. Scientists try to maximize the accuracy and speed of sequencing and simultaneously minimize the cost. The possibility of using DNA-sequencing in clinical medicine highly depends on these parameters. As it is hard to optimize all of these requirements at the same time, each method is suitable in a different situation.

3.1 A General Approach to DNA Sequencing

Although the variability of sequencing methods is huge, the overall structure is common. The process of sequencing usually consists of these steps:

- To sequence the DNA molecule we firstly need to extract it from the cell.
- After that, we need to cut DNA into smaller segments.
- Then we proceed by making many copies of the segments. This lowers the probability of the error and enables us to sequence parts of DNA parallely which reduces the required time of sequencing.
- After that, we read the bases of each molecule in the corresponding order. The read sequences are called reads (see 3.2), and they differ in length (the length is typically expressed in the number of base-pairs or bp).
- Last, we assembly the read regions into the original DNA sequence. This is a computationally expensive task, and therefore we use computers to complete the assembly. We will look closely to this part in Chapter 4.

3.2 Terminology

Let us introduce some new terminology (commonly used in DNA/RNA-sequencing).

- Read is a sequence of bases (A, T, C, G) in the same order as in the sequenced DNA region obtained by a seq-method.

- K-mer is a sequence of bases (A, T, C, G) of given length k in the same order as in the sequenced DNA region.
- Contig represents a subsequence of the original sequence that we assembled (we will discuss assembling in Chapter 4).
- Paired reads are 2 reads obtained by reading a single DNA region. Each of the two is read from a different end of complementary strands.
- Mate-pairs is a technique providing additional information about the sequence. We read a segment of sequence and another (paired) segment at a known distance from the first one. This distance should be large enough to avoid obtaining two reads from the same repetitive region. This helps us to determine the order of contigs (as we obtain the distance between some of them).

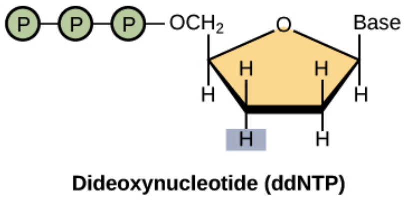
3.3 Sanger Sequencing

One of the first developed methods was Sanger sequencing [44] (also known as chain termination method). It was developed by Fred Sanger and his team in 1977 and was used in the Human Genome Project [7, 5, 6] . The main idea of this approach is based on DNA replication [42] .

Steps of Sanger sequencing:

1. Preparing copies of the desired DNA region.
2. Performing a standard polymerase chain reaction [15]. However, modified (dideoxy) nucleotides are included in addition to the normal ones (adenosine, cytosine, guanine, thymine). They act similarly, except for these modified nucleotides terminate DNA replication. When the dideoxy-nucleotides bond to the growing strand, no further base can be attached to that strand. Comparison between modified and standard nucleotides is shown in Figure 3.1.
3. The result of step 2 is multiple DNA strands that vary in length, according to the termination point ending with the dideoxy-nucleotide. The strands are separated by the gel electrophoresis. The gel electrophoresis is a process in which the shorter molecules move faster than the longer ones in a fluid.
4. The ending base of the strand is detected by the laser excitation and the spectral emission of termination nucleotides analysis.
5. The order of bases is determined by a decreasing length of the DNA strand (one at a time) as can be seen in Figure 3.2.

However, Sanger sequencing is very slow, expensive and thus inappropriate for a larger datasets analysis. Next-generation sequencing (NGS), on the other hand, are:



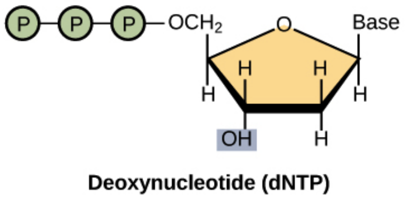


FIGURE 3.1: Dideoxy-nucleotide reused from [42]

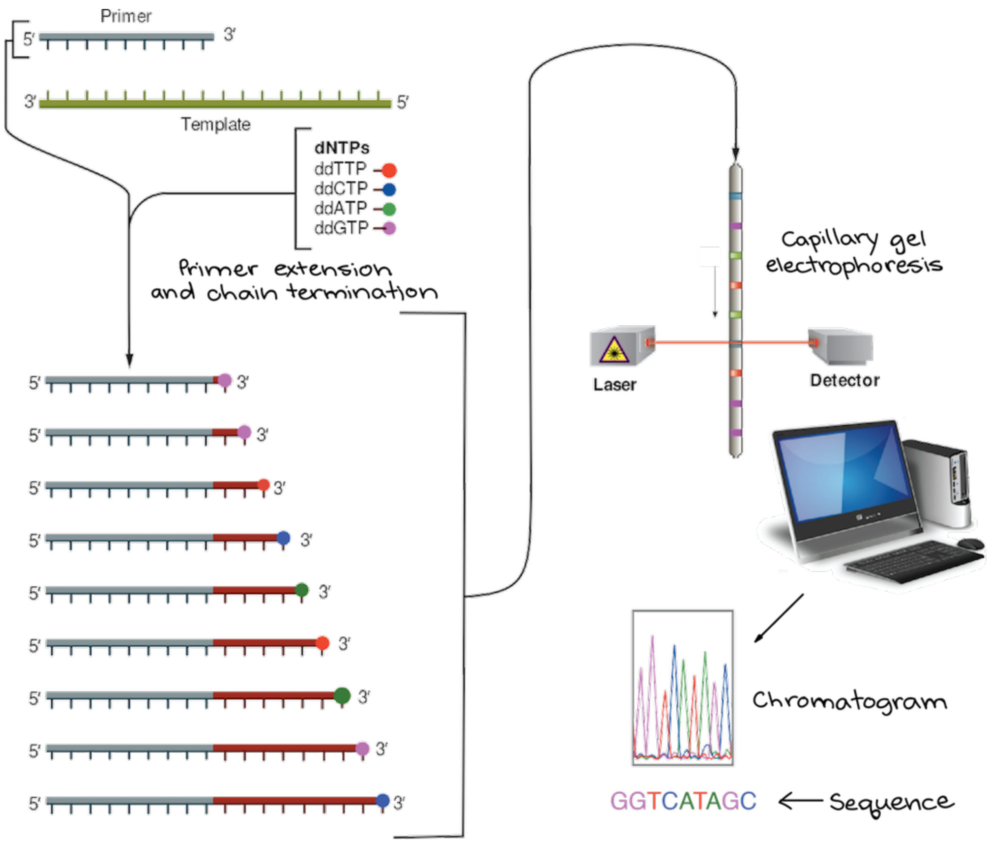


FIGURE 3.2: Sanger sequencing reused from [14]

- highly parallel: many sequences are being read at the same time,
- fast,
- low-cost.

3.4 NGS

The length of the reads is lower than in Sanger sequencing. Generally, there are multiple ways to determine the base in a sequence. A common technique is to detect some fluorescence signal that is unique for every base. Also widely used technique is based on the detection of pH change, but this is generally less accurate than the fluorescence signal. We will mention two approaches: sequencing by synthesis in Section 3.5 and microarrays in Section 3.6.

3.5 Sequencing by Synthesis

Numerous methods belong to this category. We will focus on SBS CRT (Illumina, Qiagen). The principle is similar to Sanger sequencing. Initially, we have multiple DNA single strands that we want to read. The whole process repeats in cycles. In each cycle 3′-blocked deoxynucleotides (dNTPs) are added. DNA polymerase bonds the corresponding dNTP to the replicated strand (just one at a time to every strand, the dNTP is blocking further elongation [17, 24]). The fluorescence signal caused by ligation is analyzed. As every base produces a unique signal, we can determine which nucleotid was bonded. Then the blocking group is cleaved, and the cycle repeats until no more bases can be attached [16].

3.6 Microarrays

Microarrays main idea is rather simple. We have some known sequences of the DNA attached to the distinct areas on a chip. These sequences are called probes. We also have an unknown sequence of the DNA.

Target DNA is labeled with a fluorophore. When it hybridizes to the probe, we detect a signal. The intensity of the signal is used to determine the number of bound molecules [16]. Hence, we obtain the desired information about the unknown sequence. Detection of a signal means that the subsequence complementary to the probe is contained in the unknown strand. The main advantage of microarrays technique is its low cost.

In the next chapter, we will focus on assembling reads obtained by the sequencing methods.

4 Sequence Assembly Methods

In the previous chapter, we obtained reads contained in the sequenced DNA. Now, we need some tool to reconstruct the whole DNA molecule.

There are many algorithms that provide multiple different approaches to this problem. They vary in complexity, error rate, and cost.

The human genome was sequenced in 2001 [48]. Though it was a great success, the used method is not sufficient to sequence a huge amount of data that is emerging nowadays. Therefore, next-generation sequencing methods (NGS) are being developed. These methods are minimizing the cost of sequencing and have brought it to 1000$ per genome [39].

There are two main concepts that are widely used for the read assembly: the overlap-layout consensus (OLC) and de Bruijn graphs [38].

4.1 OLC

Suppose we have a set of reads, and our goal is to produce the sequence of nucleotides in the strand from which these reads originated. OLC provides a solution to this problem. It first constructs a graph in which every read of the set is denoted as a vertex. An edge between two vertices exists if and only if there is an overlap in reads they represent. Two reads overlap if the suffix of the first read is the prefix of the second.

For example, ACCGTAA and CGTAACC are overlapping as the last 5 letters of the first sequence are the same as the first 5 letters of the second. The length of the required overlap is scalable, allowing us to set a different length of a suffix and a prefix overlap.

A reconstruction of the original sequence is done as follows: We want to include all reads, and we want to order them so that the reads joined by a directed edge are neighboring in a reconstructed sequence . For example, let us have the same reads as used above: ACCGTAA, CCGTAACC. Hence, the overlapping sequence is CGTAA. The reconstructed sequence would consist of the prefix of the first read (AC), the overlap (CGTAA) and the postfix of the second read (CC), resulting in ACCGTAACC. We can formulate this as finding Hammiltonian path in the constructed graph. It visits each vertex precisely once and reads in the reconstructed sequence are succeeding only if they have overlap. Thus, the order in which we visit the vertices determines their order in the original molecule.

This approach was used in Sanger sequencing reconstruction. Furthermore, k-mers (defined in Section 3.2) can be used instead of whole reads . Though the concept is rather simple, the complexity of finding a Hamiltonian path (or circle) is very high (NP-complete problem) and it might not have any solution.

4.2 DeBrujin Graphs

The main idea of deBrujin graphs is to formulate the problem described above in a way that would provide a less computational expensive solution. This time, we proceed as follows:

1. Firstly, we generate all k-mers from the reads.
2. After that, we denote all k-1 -mers as vertices.
3. Then we add an edge between two vertices if there exists a k-mer having the first one of them as a prefix and the other as a suffix.
4. Finally, we construct the Euler path in the constructed graph.

Euler path is such a path that visits each edge exactly once.
Note that in this case, similarly to the OLC solution, we visit each k-mer present. However, the complexity of finding the Euler path is much lower, and there is a very simple algorithm to find it. Furthermore, we know that a solution exists if every node is balanced (the number of edges connecting a vertex to other vertices is even). On the other hand, it is not guaranteed, that the solution is unique. Therefore, we try to choose the most likely solution from a set of possible solutions.

Although this formulation enables us to process the data faster, it has some negative aspects. DeBrujin assembly is sensitive to the sequencing errors. Sequencing tools usually do not have 0% error rate, and if a nucleotide is misinterpreted, the de Bruijn graph will not reconstruct the sequence correctly. Most of the modern assemblers provide some method to handle errors. We can use the fact, that there are multiple reads of the same region and recognize and remove edges that have low quantity.

The worse problem is that the DNA is repetitive. DeBrujin finds the simplest solution, that means, from sequence AACTAACT (and length of k-mer 3) it would reconstruct AACT, which is not correct. If we could provide k-mers longer than repeats, this problem would not emerge. However, the repeats in DNA are often very long. One possibility to solve this is using the mate-pairs (see Section 3.2). Another is to alter the graph so that we allow multiple edges between two vertices, one for every occurrence of the k-mer (if the number of occurrences is known).

Another deBrujin assembly problem is that some regions of DNA can be unsequenced. That results in the creation of contigs, the parts of the assembly that cannot be further merged as they do not overlap. The contigs also emerge from multiple possible paths in the graph; the assembler has multiple choices where neither is better than the other. Determining the order of these contigs is hard, and mate pairs reads proved to be useful also in this task. For further details about graphs in bioinformatics, the reader is kindly reffered to [23] .

5 Introduction to the Clustering Methods

In this chapter, we will introduce some basic methods of clustering. Clustering is a method allowing us to organize datapoints into multiple groups, each group representing one cluster. Naturally, we put similar points in the same group. Generally, we want to maximize the distance between different clusters and minimize the distance between the points within one cluster.

For example, let us have two kinds of datapoints: women and men. Each datapoint has a vector of features: height and weight. We do not know which datapoint represents a woman and which a man. If we can organize data into two groups (ideally corresponding to men and women) that are similar in these parameters, then we should be able to determine to which group a new unknown datapoint belongs.

Generally, success is not guaranteed. It might turn out, that the provided parameters cannot distinguish between groups. In addition, the number of groups can be unknown. There are multiple approaches to clustering, it depends mostly on the data, which one is the most appropriate to use. One of the best-known methods is k-means. It was independently discovered in different scientific fields by [30, 46, 2, 31]. We will discuss it in the next section.

5.1 k-means

K-means is a very simple and useful clustering algorithm. It divides data into k groups. The brief overview of the algorithm is as follows [30]:

1. choose k different points $N_1 ... N_k$ - centroids,
2. for every datapoint find the nearest centroid and label the datapoint according to the centroid,
3. calculate the mean of every set of common-labeled datapoints and denote them as new centroids,
4. goto step 2. and proceed until no datapoint changes its label during step 2.

The main advantage of this algorithm is its simplicity. However, we can only use it if the mean function is defined. For example, if the datapoints have features such as „yes/no“, it is not possible to count mean. Another drawback is that we have to know the number of the desired clusters in advance.

If we lack information about the number of clusters, we can run the k-means algorithm multiple times with different k-s and try to determine the best one. We will look at this approach in the next section. It also turns out, that the initialization of the k-means algorithm in step 1. might significantly influence the quality of the result. We will discuss the ways to improve the initialization in Section 5.1.2 .

5.1.1 Determining the Best k

As we want to minimize the distance within one cluster, adding more centroids seems to improve the solution. If we had a separate centroid for every datapoint, this distance would be 0. However, such a solution would not be satisfying as it would not show anything about similarities between datapoints. An approach to determine the best k is increasing k and observing how the minimized distance decreases. We stop increasing when the decreasing slows down. For a more detailed explanation see the elbow method [27].

5.1.2 Initialization of k-means

There are multiple ways to choose initial centroids. The easiest way is to randomly pick k centroids among datapoints. We can improve this method by picking the datapoints so that the distance between them is maximized [21].

Despite the problems emerging from the undefined mean function or the unknown number of clusters, the k-means algorithm is a very common clustering technique. It is also guaranteed that it will converge to a solution. To find more details about k-means we recommend [21] .

5.2 Hierarchical Clustering

Hierarchical clustering is a clustering method that either starts with one cluster that is divided into more smaller clusters (in case of divisive clustering) or starts with each datapoint in a separate cluster and then merges them into the bigger clusters (that is called agglomerative clustering) [41].

5.2.1 Divisive Hierarchical Clustering

In the beginning, we add all datapoints into one cluster. Then we recursively divide this cluster into smaller ones. As there are 2^N ways how to divide a set of N points into two groups in each step, this approach is more computationally expensive compared to the agglomerative clustering. We stop dividing when we reach the desired number of clusters.

5.2.2 Agglomerative Hierarchical Clustering

Initially, every datapoint represents one cluster. Then the closest clusters are merged into one. This requires at most $N * (N - 1)/2$ (every cluster compared with all the others) operations where N denotes the number of clusters.

Determining which cluster is the closest differs. The most common criteria are:

1. Single-link clustering
   The distance between two clusters equals the shortest distance from any member of one cluster to any member of the other cluster [45].

2. Complete-link clustering
   The distance between two clusters equals the longest distance from any member of one cluster to any member of the other cluster [26].

3. Average-link clustering
   The distance between two clusters equals the average distance from any member of one cluster to any member of the other cluster. Such clustering algorithms may be found in [49].

4. Centroid-link clustering
   The distance between two clusters is equal to the distance between their centroids.
   A centroid is a point in a cluster (does not necessarily belong to the datapoints) that has some property. For example, it has the minimum sum of square distances to all points in the cluster.

These criteria are common for both agglomerative and divisive clustering.

Similarly to divisive clustering, we stop merging clusters when the desired number of clusters is reached.

If the number of clusters is unknown, a common approach is merging the clusters until just two remain and construct a dendrogram. Dendrogram is a tree showing which clusters were merged together. Then the final clusters are determined from the dendrogram according to some criterion.

All of the previously discussed methods require the desired number of the clusters in advance. In the next section, we will introduce the density-based clustering that does not require that.

5.3 Density-based Clustering

Density-based clustering is an approach allowing us to distinguish between clusters that have a complicated shape. If we have many datapoints in the region, we say that the region has a high point density. Similarly, if there are none or few datapoints in the region, we say that the region has a low point density. The main idea of the density-based clustering is that clusters are regions of high point density and they are separated by space where the point density is low. The data that lie in the separating regions are called outliers. Unlike other clustering methods, we do not need the number of clusters as the input parameter. On the other hand, we need to set distance threshold r and frequency threshold k. The algorithm works as follows [28]:

1. We first evaluate the frequency k of each datapoint. The frequency is the number of points located within a distance r of the datapoint.
2. Then we remove the noise points where the frequency does not reach the threshold.
3. After that, we cluster the remaining points by the single-linkage.
4. Finally, we can add the noise points to the suitable cluster.

One of the best-known density based clustering algorithms is DBSCAN [13] . See [28] for more detailed explanation of DBSCAN.

Figure 5.1 shows how not convex sets of datapoints are clustered by density-based clustering. For more clustering techniques we recommend [29].

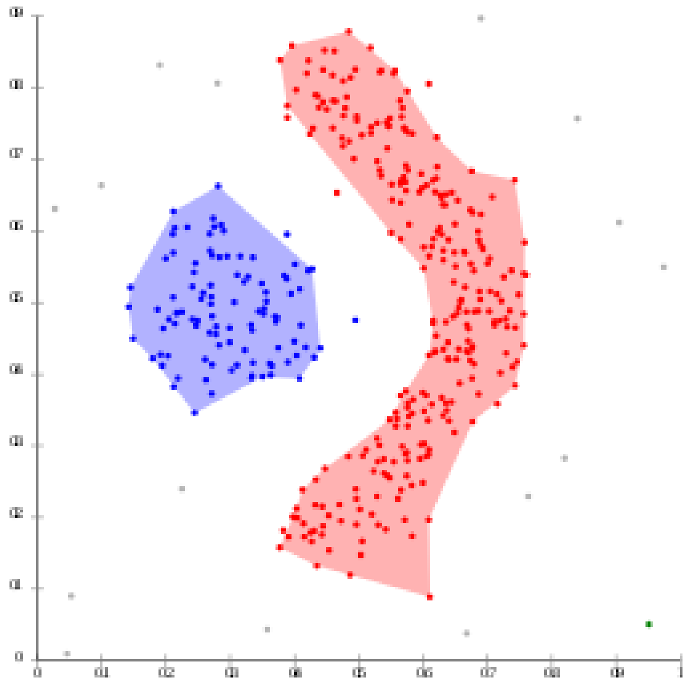


FIGURE 5.1: Density-based clustering reused from [4]

6 RNA-seq

In this chapter, we will discuss RNA-seq. It is a sequencing method that reads RNA from a cell and then processes the gathered information.

Let us introduce a dictionary for this chapter :

- Expressed genes are the active genes in a cell. That means only these genes translate to proteins. See Section 2.4 for more detailed description.
- A gene expression stands for a number that represents the abundance of the transcribed genes.
- Gene isoforms represent multiple forms of gene: mRNAs are produced from the same locus but differ in their transcription.
- Splice isoforms are various forms of genes transcribing. As it is possible that only some sections of gene are transcribed, multiple different transcripts emerge due to ommiting some of these sections.

RNA-seq is mostly used for observing the gene expression. It also allows us to identify which genes are expressed in a cell at a given time, as the expressed genes are represented in RNA within the cell.

Compared to DNA-microarrays that we mentioned in Chapter 3, RNA-seq detects lowly expressed transcript and provides less false positive rates [50].

There are many applications of RNA-seq, such as the detection of gene expression changes, which is the most common application of RNA-seq, as well as the detection and the quantification of non-genic transcripts or splice isoforms. However, we will focus on the gene expression, for more details see [11].

RNA-seq begins with the data preparation. This process includes:

- RNA extraction: a microbiology procedure gathering the RNA from a cell.
- Library preparation: preparing DNA/RNA fragments we want to sequence.
- Sequencing (discussed in Chapter 3). As only expressed genes are transcribed, the reads we obtain originate from the expressed genes.

As RNA-seq provides information about the gene expression, we can observe its state under different circumstances. We can also compare the expression of a healthy individual to the expression of an unhealthy one.

To avoid biases in such experiments, the following steps are taken:
Blocking – grouping the samples into blocks according to the known sources of variation, such as sex, weight, or the cell cycle status;
Randomization – when testing a factor, such as drug treatment, select the cells randomly to avoid bias caused by subtle differences in the activity of the animals or the growth pattern of cell lines.

After data preparation, we can proceed to the data processing (bioinformatics). In the next section, we will discuss read alignment, which is a commonly used technique to identify the transcripts present in a specific sample.

6.1 Read Alignment

In this section, we will introduce the read alignment. It is a method used to identify which genes are expressed. However, we can perform this operation only if we have the reference genome or transcripts.

In this reference genome, we already know which segments correspond to which genes. We can assign (map) the reads we sequenced to the most likely locus at the known transcriptome or the known genome and thus provide information about the expression of the particular gene. This expression is proportional to the number of reads mapped to the given gene segment.

However, aligning itself is a very complex process. The most problematic aligns are caused by a spliced alignment of exon-exon-spanning reads. Since only exons are expressed (as mentioned in Section 2.4), aligning parts of the read to the different locations in the genome means that we must consider multiple possibilities of the splice region. Figure 6.2 illustrates the exon-exon-spanning reads.

Thus, aligning to the transcriptome (visualized in Figure 6.1) seems to be a better option, but it is limited to known transcripts. Another complication is sequencing error, causing that alignment might to not fit perfectly.

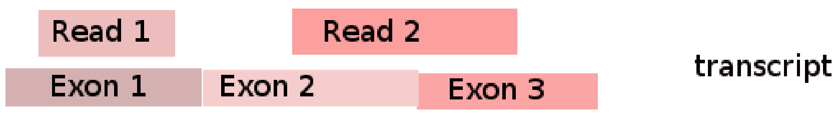


FIGURE 6.1: Mapping to the transcriptome

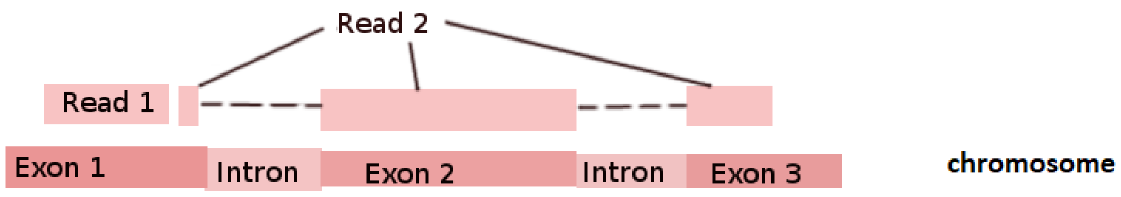


FIGURE 6.2: Mapping to the genome

Moreover, mapping ambiguity appears as many reads overlap with more than one isoform. There exist numerous programs for read aligning, such as STAR [10], Tophat [25], GSNAP (see [12] for a review of RNA-seq aligners). To choose above a variety of these programs, it is important to know the specification of the reads one needs to align as some tools might provide the optimization for the specific sample characteristics.

6.2 Read Quantification

One of the crucial steps of the read analysis is read quantification. By counting reads that were mapped to the particular gene, we can determine the number of transcriptions of that gene to proteins. This value corresponds to the gene expression. The main goal is to compare expression levels of mapped reads, knowing the number of mapped reads. This number corresponds with the expression levels, though it is necessary to perform some normalization (we will discuss normalization later in Section 6.4).

However, computer programs developed for the read quantification may differ in handling the following:

- overlap size,
- reads overlapping multiple genomic features of the same kind,
- reads overlapping introns,
- multi-mapping introns.

Although the read alignment to the reference genome and the following quantification works, the new tools for measuring the gene expression have emerged recently. We will introduce them in the next section.

6.3 Probabilistic Measure – Algorithms

As read alignment is a rather complex and difficult task and it is computationally expensive to find the best alignment, this new group of algorithms avoids aligning reads. Thus, speed improves significantly.

Instead, the algorithms use a probabilistic measure of the number of reads present in each transcript. The main idea is that it is not necessary to know the exact locus in a transcript from which a read originated. It is sufficient to know which transcript is represented by the read. However, these tools cannot be used to detect novel isoforms [11].

Sailfish [35] (or the more updated version – Salmon [36]) and Kallisto [3] are examples of these tools.

6.4 Normalizing Reads Counts

In this section, we will discuss a quantification normalization. As the number of reads aligned to genes is influenced by multiple factors alongside with the expression level, a normalization step is required.

These factors are:

- length of the gene: the longer the gene is, the higher number of mapped reads,
- the sequencing depth: depends on the number of reads within the sample, more reads increase the depth,
- the expression of all other genes within the sample.

To compare the gene expression level, it is therefore necessary to divide it by the total number of reads and the length of the gene.

7 Clustering of RNA-seq Reads by Gene Expression Levels

In this chapter, we will introduce the main idea of read clustering by gene expression. We will formally define the problem and proceed to our solution in the next chapter. There are many approaches and algorithms that analyze RNA-seq data differing in goal and efficiency.

In this thesis, we will focus on the reads clustering. Our overall goal will be to assign reads that originate from the same gene to the same cluster. Hence, each cluster will represent one gene (a region, where the read can be found). However, as genome contains repetitive sections, it is possible, that one read is assigned to multiple clusters (genes). This is considered a normal phenomenon. Moreover, our solution will not require any reference genome. It will avoid aligning reads completely, thus allowing us to use it on reads from species whose genome is not known.

7.1 The Main Idea of Clustering By Gene Expression

As discussed in Chapter 6 , a gene expression reflects the amount of transcripts from a particular gene. In other words, the more the gene is expressed, the more reads originating from it are sequenced. Also, a highly expressed gene means that a cell synthesizes higher levels of a corresponding protein.

A low expression can lead to a lack of some proteins, resulting in a disease. However, too high levels of protein can also have negative consequences.

Proteins, besides hormones, influence, and control many processes in the organism. Therefore, it is highly probable that individuals with a certain dysfunction differ in some gene expression levels.

To illustrate this: imagine we have two individuals, I1 (healthy) and I2 (unhealthy). Also, imagine that only two genes: gene A and gene B are expressed and there are no common sequences of sufficient length amongst them. Now, suppose that:

- Gene A is highly expressed in I1 (meaning that we read many subsequences of gene A multiple times) , and highly expressed in I2.
- Gene B is lowly expressed in I1, and highly expressed in I2 (as visualized in 7.1).

$$\begin{array}{ccc} & I_1 & I_2 \\ A & (10 & 0) \\ B & (0 & 20) \end{array}$$

TABLE 7.1: The different expression example

We can therefore assume, that sequences highly expressed in I1 and simultaneously lowly expressed in I2 belong to the same gene (A), analogically for the second row (gene B).

This is the reason why we need reads from more individuals (increasing the dimension), and it is beneficial if these individuals differ from each other.

7.2 The Formal Definition of the Problem

In this section, we will formally define the problem and introduce important notation that will be used in the following chapters.

- *read* is a structure having two attributes, where
  readID stands for a unique name within the file identifying read
  readSeq is represented by a string $a_1a_2 \dots a_n, n =$ length of the sequence, $a_i \in \{A, C, T, G\}, i \leq n, i \in \mathbb{N}$.

- *k-mer* is a string $a_1a_2 \dots a_k$, k is k-mer length, $k \in \mathbb{N}, a_i \in \{A, C, T, G\}, i \leq k, i \in \mathbb{N}$.

- *k-mer set from a read:* K_m is a function defined on reads. It produces a set of all k-mers contained in a particular read. A k-mer is contained in a read if for read sequence $a_1a_2 \dots a_n$ and k-mer $b_1b_2 \dots b_k$ holds:
  $[\exists l : \forall b_i, i \in [1, k], i \in \mathbb{N} : b_i = a_{l+i}]$. In other words, a read contains a k-mer if the given k-mer is a substring of the read string.

- *nucleotide complement* ***t*** is a function t from $\{A, C, T, G\}$ to $\{A, C, T, G\}$ defined as : t(A) = T, t(T) = A, t(G) = C, t(C) = G

- *reversed complement* : a sequence of nucleotides A $a_1a_2 \dots a_n$ is reversed complement to a sequence B
  $b_1b_2 \dots b_n$ if $n =$ length of A and
  $b_i = t(a_{n-i+1}), i \in [1, n], i \in \mathbb{N}$

- *membership table* : a set of rows *r*, where each row consists of *read-ID* and *vector v*, such that $\sum_{i=1}^{n} v_i = 1, n =$ length of vector
  We introduce two helper functions : ***ID*** , assigning each read the corresponding ID and ***v***, assigning each read corresponding vector v mentioned above.
  Each component of a vector represents how much the read belongs to the corresponding cluster.
  In the following chapters we will refer to two kinds of membership tables :

- $\mathbb{M}$ - **out membership table** obtained by our clustering algorithm,
- $\mathbb{R}$ - **reference membership table** obtained by a reference based approach.

7.3 The Task

The main goal consist of two parts :

1. Our aim is to create *out membership table* for r in reads of one individual. Once created, we should have sufficient information to determine reads belonging to the same gene. Furthermore, we can compare our read assignment to the different clustering method (as we discuss later in Chapter 9). That leads us to the part two:

2. The second part is to minimize the error between our proposed solution and the standard reference-based solution. That includes determinating parameters of our algorithm and developing a metric to compare the read assignment in both our (reference-free) method and the reference-based method.

8 The Proposed Solution

In this chapter, we will describe our prototype of the algorithm solving the problem described in the section above (7.2).

Firstly, let us examine the input data.

8.1 The Datastructure

The input data consists of files containing reads sequenced by Illumina [20] . These reads originate from multiple subjects.

Each subject has exactly two corresponding files. The first file contains forward reads (see Section 3.2), the other contains reverse reads.

However, we have no further information to determine, which one of the two files contains forward reads. [1]

To provide a simple example:

> Suppose Strand1 is ACCTG.
>
> Complementary strand then results to TGGAC.
>
> Forward read of length 5 is ACCTG (corresponding to Strand1)
>
> Reverse read of length 5 is CAGGT (this sequence is reversed complement to ACCTG as defined in Section 7.2)

It will be necessary to keep this in mind when processing files.

Notation: We will refer to forward and corresponding reverse file as to R1-file and R2-file. Remember, that we cannot distinguish which of them contains forward reads.

Now, we will outline the algorithm. It consists of 4 steps, and we will discuss each of them in the following sections.

8.2 Step One : Computing k-mers Occurences in All Files

As we show in Section 7.1, our aim is to assign each read a corresponding gene expression. However, read-sequences are too long (around 75-100 nucleotides, depending on a sequencing technology). Therefore, we will take shorter sequences

[1] Reminder: Forward reads originate from one strand of DNA and reverse reads from the complementary strand. Moreover, reverse reads are sequenced from the opposite end of the strand.

called k-mers (see definition in Section 7.2). A length of k-mer is roughly 25. We will discuss a better estimation of k in Section 11.1. Consequently, we will compute a number of occurrences of each k-mer present for each file.

As we do not know, if a k-mer originated from a forward or a reverse read, we will merge the k-mer and its reverse complement (defined in 7.2), keeping only one of them and assigning it a sum of both occurrences. It is common to keep the k-mer that comes first lexicographically.

To compute counts for each file in the dataset we will use jellyfish [33]. The main advantage of this tool is efficient and parallel counting all k-mers.

In the next section, we will process these k-mers and create a table.

8.3 Step Two : Creating a Table of k-mers

In this step, we will construct a table, where rows will represent all k-mers found in all files of datasets. The occurrences of a particular k-mer in each column will refer to one subject from the dataset, containing occurrences of a particular k-mer in reads associated with that subject.

That means summing a k-mer occurrences in R1-file and R2-file for every subject. Moreover, we pad zeroes to k-mers that do not occur in every subject. However, the k-mers that occur only few times in each file are not significant, as they provide a few information. Hence, we filter out k-mers that do not exceed lower bound in any of the input files.

A result of this step is similar to Figure 7.1, only A and B are k-mers, and the dimension corresponds to the number of subjects. Finally, k-mer counts represent expression of a particular k-mer (similarly to a gene expression).

8.4 Step Three : Clustering k-mers

In the previous step, we generated a table of k-mer counts. We can imagine this table as a set of rows, where each row consists of a k-mer and N-dimensional vector (N is be the number of subjects).

In this step, it is essential to realize, that gene expression (amount of a gene transcribing) almost equally increases all k-mer counts present in that gene. Except for k-mers at the ends of the reads as illustrated in Section 8.4.1.

Thus, similar counts at all positions of two k-mer vectors in the previously constructed table would indicate membership to the common gene.

Note, that this approach does not distinguish between uniformly distributed genes. That is the reason, why it is important to have multiple different subjects.

So, relating to the idea that k-mers belonging to the same gene are close to each other in the vector space we created, we proceed to the clustering of k-mers.

8.4.1 Clustering Method Selection

We considered multiple vector-based clustering algorithms (see Chapter 5). For instance, agglomerative clustering, k-means clustering, and DBSCAN clustering discussed in Chapter 5. All of them are implemented in sklearn python library [37].

However, due to the huge amount of data it is impossible to cluster all of the k-mers present in the table. Luckily, it is not necessary. As one read sequence has length [2] l , it contains $l - k + 1$ k-mers (where k usually varies from 20 to 30). However, these k-mers overlap. In fact, most of the read sequence positions are covered by k k-mers (except for the ends of read sequence). Therefore, we do not need to cluster all of the k-mers, as the read can be determined by a smaller portion of them.

Nonetheless, k-means and agglomerative clustering run out of memory very fast, even when taking only a small sample from the table. On the other hand, DBSCAN, unlike the previous two algorithms, has an eps-bound, causing a rapid decrease of comparisons. The eps-bound determines the maximum distance within a cluster. That means dividing the vector space into subsections that do not need to be compared.

Due to these reasons, we decided to use sklearn [37] DBSCAN clustering, allowing us to take 10% of the table constructed in step two and process it in a reasonable time. Determining eps will be discussed later in Section 11.1.

Then, results of DBSCAN are the labels for each vector and -1 label for noise. Finally, we sort clusters by size.

8.4.2 Selecting the Imporant Clusters

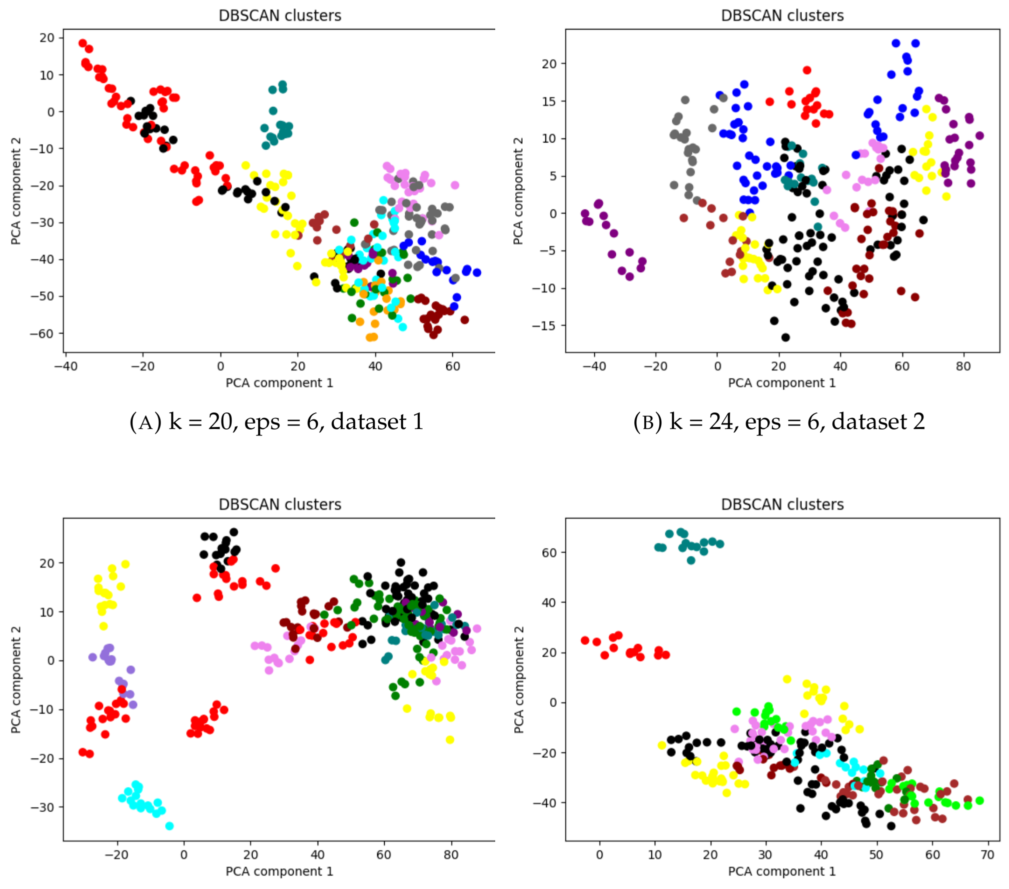
When provided with clusters from DBSCAN clustering, we sort them by size. The largest cluster is noise and we do not further process it.

The second cluster should contain k-mers that are uniformly distributed and does not offer much information (roughly [3] 10000) k-mers belong to that cluster). On the other hand, we are interested in middle-sized clusters. These follow right after the second cluster and contain roughly 10-50 k-mers.

We will keep in memory which clusters belong to this ***important cathegory***.

Finally, we define it as follows:
important cathegory $\mathbf{I} = \{\forall n : n \in \mathbb{N} \wedge n \in [v, u]\}, v < u$, where $u - v$ is a number of important clusters. Here u stands for the upper bound and v for the lower bound, in our case the upper bound is 50 and lower bound 2. This means that we mark the second, the third $\cdots$ the 50-th largest cluster as important.

[2] l is usually roughly 100, in our case l equals 76
[3] while the number of human genes is more than 40000 [43]



(A) k = 20, eps = 6, dataset 1

(B) k = 24, eps = 6, dataset 2

(C) k = 25, eps = 10, dataset 2

(D) k = 25, eps = 12, dataset 2

FIGURE 8.1: Clusters of k-mers projected to 2D

8.4.3 Visualizing Results

As the vectors have more dimensions than 3, it is hard, though not impossible to visualize result clusters. Hence, we use PCA-dimension reduction implemented in sklearn [37] to transform vectors to 2-dimensions and plot some of the clusters (each cluster having a distinct color in the transformed vector space).

However, the outliers squish the axes so that the most of the points merge. Therefore, we use the z-score filtering to remove these distant points (with z equals 3).

The number of plotted clusters can be set, but for illustration purposes, we choose 17 clusters from the ***important cathegory*** discussed above (omitting noise and uniformly distributed k-mers).

We need to remember that we squished vector space to a smaller dimension. Hence some clusters project to the similar place.

In the Figure 8.1, we show the PCA-reduced graphs of first 17 k-mer clusters with multiple different setting of *k* and *eps* and dataset. The same colored points belong to one cluster. We can observe that the points belonging to the same cluster are projected into the similar position.

However, as the points originally have more dimensions (10), some of the points of different clusters overlap in the 2D space.

To illustrate the difference between the clusters including and excluding noise, we show the DBSCAN plot with noise in Figure 8.2. Obviously, the noise provides no information about the clusters. Furthermore, it completely shadows the other points (as the noise contains much more points than the not-noise points that we plot).

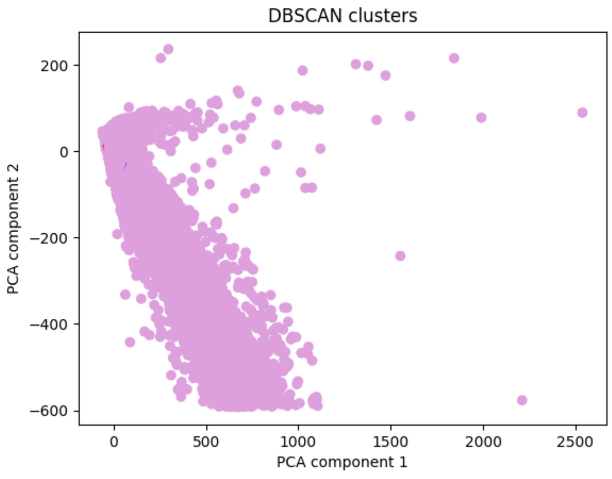


FIGURE 8.2: Clusters of k-mers projected to 2D including the noise

8.5 Step Four : Clustering Reads

In the previous step, we created clusters of k-mers based on an expression. However, our goal is to cluster the original reads in files. Hence, we need to cluster reads. As a read can belong to multiple genes, we will use soft clustering. That means a read can be partially assigned to multiple clusters, each with some percentage summing to 1.

The result will be a membership table as defined in Section 7.2. Each read will have a vector of membership to the previously constructed clusters.

Firstly, we will process all data files containing reads (as specified in Section 8.1). We will process data from R1-file and R2-file simultaneously. For processing reads, we will use the python library Bio [8]. For each read A, we determine to which clusters it belongs.

We will do that as follows:

1. If k-mer $\in R_m(A)$, then add 1 to membership vector on position corresponding with the cluster containing k-mer.

2. We act the same if k-mer $\in R_m$(reverse complement of A). That is because k-mer and the reverse complement of k-mer are equivalent (as described in step one).

3. After that, we will normalize the vector of membership by the sum of the elements to obtain percentual membership.

As a result, we will have a table of membership, determining for each read to which clusters it belongs.

To analyze if these clusters correspond with real genes, we will use the standard reference-based approach for comparison. We will describe the analysis by STAR [10] in the next chapter.

9 Analyzing Data Using the Standard Reference-Based Approach

In this chapter, we will discuss the standard approach for RNA-seq analysis. The standard reference-based approach aligns reads to a reference genome. Hence, reference is always required. We will introduce the STAR algorithm [10].

STAR consists of 2 steps:

1. Firstly, it generates index to provide faster aligning in the next step. This phase requires a reference. For our purpose, we used the reference genome: Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz and the anotation: Homo_sapiens.GRCh38.86.gtf.gz from the ensembl database [18]

2. After that, aligning takes place. This step is parallel and results to aligned `.BAM` files. These files contain the index in reference, where the read aligns for each subject (merging R1-file and R2-file). However, not all reads are aligned. This can be caused by sequencing errors, as well as the uniqueness of each organism.

Next, we need to create a *reference membership table* (defined in Section 7.2). We take output `.BAM` files produced by STAR.

Firstly, we translate each read index in gtf-file to gene name corresponding with the index position using python library HTSeq [1] . Then we make *a list of genes L*.

Secondly, we construct the *reference table of membership* where positions of vector correspond to genes. $vector_i$ $(i \in [1, N]$, N = length of the list L$)$ is assigned 1, if read belongs to a gene on i-th position in *the list of genes L*. As the genome contains repetitive regions, one read can map to multiple different regions (genes). After that, the membership vector is normalised by the number of assigned genes.

As a result, we obtain the reference membership table that can be used for comparison with the table constructed in Section 8.5. The following chapter will discuss it further.

10 Comparing our Algorithm With the Reference-based Approach

In this chapter, we will analyze the results from our solution presented in Chapter 8.

Firstly, we prepare *the reference-membership table* described in Chapter 9. Then, we generate *the membership-table* with our algorithm discussed in Chapter 8. We also prepare the list of the ***Important category I***. This list contains clusters, that should represent the well-distinguished genes (see Section 8.4.2 for more details).

Now, we have all the necessary information to compute the error defined in Section 7.2. This error shows, how much our reads assignment differs from the reference-based approach for each subject. We define the error as 3-dimensional vector $E = (E_1, E_2, E_3)$. In addition, we define the normalized error $N = (N_1, N_2, N_3)$. We will explain each component of the error and provide the corresponding formulas. For required notation to understand formulas see the formal definition in Section 7.2.

10.1 Error Explained

10.1.1 The First Component of the Error

Let us suppose that read A has a vector v_1 assigned by our algorithm and vector v_2 assigned by STAR. These vectors should ideally provide the same information.

However, the components of v_1 and v_2 do not necessarily have the same order. This is caused due to the fact that though components of v_1 express the membership to clusters representing genes, the order of these genes is not the same as in v_2. Moreover, we do not know which elements in v_1 correspond to which elements in v_2.

Therefore, we take 2 reads, read A and read B. For these reads holds that if they belong to the same gene in the *reference-membership table*, they should be in the same cluster in our table.

To provide a simple example:
Suppose we have two vectors in the *reference-table*, A : $A_{ref} = (1, 0, 0)$ and B : $B_{ref} = (1, 0, 0)$. It is clear, that if the dot product of A and B equals 1, both of reads belong to the same gene and only one gene. On the contrary, if the dot product is 0, they do

not share any gene membership. If A and B partially belong to multiple genes, the product of components referring to these genes is weighted by the percentage.

All in all, if read vectors in *reference-membership table* and *out-membership table* differ only in the order of elements (this order is the same amongst one table), the dot product of two reads with IDs A and B from one table should stay the same as the dot product from the other one with the same IDs A and B .

Hence, we sum the squares of differences. Squares will prevent the error from being negative.

If the domain of variables is obvious, we omit it. For further purposes the notation read $\in \mathbb{R}$ or read $\in \mathbb{M}$ means that such a read exists in the corresponding membership-table.

Formally,

Sample $S_1 = \{(i,j,k,l) | \, i \neq j, k \neq l, r_i \in \mathbb{R}, r_j \in \mathbb{R}\}$

$$E_1 = \sum_{(i,j,k,l) \in S_1} (v(r_i) \cdot v(r_j) - d_1)^2, d_1 = \begin{cases} v(m_k) \cdot v(m_l) & \text{if } m_k \in \mathbb{M} \text{ and } m_l \in \mathbb{M}, \\ & \quad ID(m_k) = ID(r_i), \\ & \quad ID(m_l) = ID(r_j) \\ 0 & \text{otherwise} \end{cases} \tag{10.1}$$

The variable d_1 in 10.1 represents the dot product in the *out membership table*. Hence, we can only compute it if the reads with corresponding IDs appear in the *out membership table*. If we removed the reads during our solution (in the table creation with threshold or during DBSCAN sampling), the E_1 increases (as we subtract 0 from the positive number instead of another positive number).

Consequently, we normalize E_1 to obtain N_1 by the number of summed elements (couples of compared reads). Moreover, as we powered the difference that is between -1 and 1, we decreased the error. Therefore, we will take the square root of the normalized error. Hence, we obtain the quadratic mean

$$N_1 = \sqrt{\frac{E_1}{|S_1|}}.$$

10.1.2 The Second Component of the Error

The second component is highly similar to the first one. Except, we focus on reads we consider well-distinguished by our method. These reads are defined by having non-zero elements on the position corresponding to any cluster from *the important cathegory* (described in Section 8.4.2).

This enables us to avoid computing error on uniformly expressed genes and noise. However, as the goal is to minimize the error, this could result to not marking any of the clusters as important. Therefore, the third component is necessary.

Formally,

$$\begin{aligned}\text{Sample } S_2 = \{(i,j,k,l) |\ & i \neq j, k \neq l, \\ & m_i \in \mathbb{M}, m_j \in \mathbb{M}, \\ & \exists o \in \mathbb{I} : v(m_i)_o \neq 0, \\ & \exists p \in \mathbb{I} : v(m_j)_p \neq 0\}\end{aligned}$$

$$E_2 = \sum_{(i,j,k,l) \in S_2} (v(m_i) \cdot v(m_j) - d_2)^2, d_2 = \begin{cases} v(r_k) \cdot v(r_l) & \text{if } r_i \in \mathbb{R} \text{ and } r_j \in \mathbb{R}, \\ & \quad ID(m_i) = ID(r_k), \\ & \quad ID(m_j) = ID(r_l) \\ 0 & \text{otherwise} \end{cases} \tag{10.2}$$

The variable d_2 in (10.2) works the same as d_1 in the first element of the error. Except, this time we pick the reads from the *out membership table* and we are searching for the same IDs in the *reference membership table*. As the *reference membership table* is composed only of reads that were aligned to the reference, it is possible that some of the reads exist only in the *out membership table*.

Similarly to the normalization of E_1, we normalize E_2 as follows:

$$N_2 = \sqrt{\frac{E_2}{|S_2|}}.$$

10.1.3 The Third Component of the Error

Imagine that read A is marked important. All reads belonging to the same gene should also be marked important.

The third component of error expresses the number of reads that should be marked important, but are not. In this step, for each important read **A** from *reference-membership table* we find all reads belonging to the same gene and not marked important.

To obtain error caused by omitting some important cluster, we firstly sum membership to clusters of reads marked unimportant. As a result, we have the partial error. After that, we normalize the partial error by the number of summed elements, obtaining the normalized partial error. We weigh these partial errors by the percentage of membership of the particular read. After that, we sum the obtained results of all reads marked important, gaining the infromation about the rate of the reads marked unimportant despite being important. Finally, we multiply this rate by number of unimportant reads.

Formally, we will denote significant reads from the *reference table* as SR and non significant reads as NS.

$$\mathrm{SR} = \{r \mid \exists m \exists k : v(m)_k \neq 0, ID(m) = ID(r), m \in \mathbb{M}\}$$

$$\mathrm{NS} = \{r \mid r \in \mathbb{R}, r \notin \mathrm{SR}\}$$

$$\mathrm{avgnonS} = \frac{1}{|\mathrm{NS}|} \sum_{r \in \mathrm{NS}} v(r)$$

$$E_3 = |\mathrm{NS}| \sum_{r \in \mathrm{SR}} v(r) \cdot \mathrm{avgnonS} \tag{10.3}$$

To avoid a long table search, we implement it more efficiently, though equivalent. See Chapter 11 for more details.

Lastly, to get the normalized error we perform a normalization process similarly to previous errors.

$$N_3 = \frac{E_3}{|\mathrm{SR}|}$$

for E_3 from (10.3).

10.2 Sampling

However, it would be very computationally expensive, if we computed dot products for every pair of reads. As there are roughly 4000000 reads in *the reference-membership table*, it would result in $\binom{4000000}{2}$ pairs. That is roughly $7 \cdot 10^{12}$ pairs.

Therefore, we will use sampling and take only 0.0009% of pairs (thus normalizing error by the sample size, which is around 250000).

11 Implementation

In this chapter, we will provide information about implementation. All of the scripts are implemented in python3.6.7. We used the following libraries : HTSeq [1], Bio [8], sklearn [37], scipy [22], numpy [34], matplotlib [19]. We also used jellyfish software [33] for counting k-mers.

All of the scripts with a brief description, ordered by the position in the pipeline can be found in Appendix A.

In the next section, we will discuss setting the parameters to obtain better results.

11.1 Parameter Setting

The result clusters strongly relate to eps used in the DBSCAN clustering. Eps determines maximum distance within a cluster. Thus, changing eps could have a high influence on results.

Similarly, the length of k-mers (k-size) could also influence results. As too low k-size would cause assigning (almost) all reads to all clusters. To illustrate this, imagine the extreme value of k where k equals 1). Likewise, too long k-mers would provide little information about each read.

As the results depend on multiple parameters, we will design experiments that will try to estimate the optimal values of eps and k-size in the next chapter.

12 Experiments

In this chapter, we will describe the performed experiments. Firstly, we will examine the relation between the error and the parameter setting in our pipeline, as discussed in Section 11.1.

As we want to avoid a biased result, we will perform the experiments on two different datasets. Admittedly, the size of both datasets (a number of contained subjects) is the same amongst them. For further purposes we will denote them as *dataset 1* and *dataset 2*. The source of the data is described in [47].

Overall time of running the whole pipeline on Metacentrum fluctuates between 14 and 19hours. For more detailed time description of all steps see Appendix A.

12.1 Eps setting

Initially, we will run our algorithm with multiple values of eps, while k is fixed. We will observe the influence of eps on all components of the normalized error described in Section 10.1. In the performed experiment we set the k to 25.

Figure 12.1 shows all components of the normalized error for multiple values of eps. As each subject has its own values of N_1, N_2, and N_3, we show the averaged error through the particular dataset.

At this point, we will discuss all of the error components.

N_1 does not hint any relation to the eps changes. It remains around 0.38 through all values of eps. N_2 stands for the error of the read assignment amongst all reads.

On the contrary, the second component focuses on the reads from the *important cathegory*. Our algorithm should distinguish better amongst these reads. And truly, if we compare N_1 with N_2, N_2 is much smaller, it does not even exceed 0.2 in any of the datasets.

However, neither N_1 nor N_2 seems to have decreasing or increasing tendency. Furthermore, the local minima differ amongst datasets. That leads us to the conclusion that either we chose the wrong interval of eps, or eps has only a minor impact on the resulting error.

Moreover, the third component of the error does not show any trend. We will discuss the results of the third component of the error in Section 12.4.

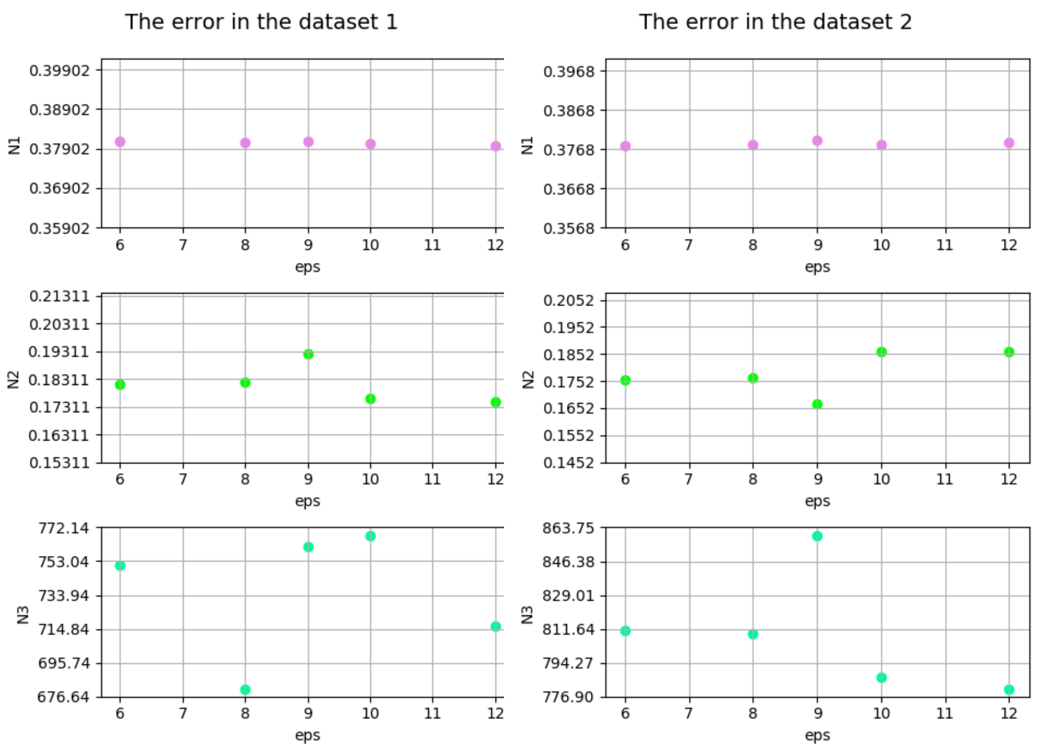


FIGURE 12.1: Errors of both datasets with k set to 25

12.2 K setting

In this section, we will choose multiple values of k (representing the k-mer length) and perform the experiments similar to those described in Section 12.1. Due to the previous results, we fix eps to the value 6, as the smaller eps speeds up the algorithm and consumes significantly less memory.

Figure 12.2 shows all three components of the error. The figure illustrates that the first component of the error has the decreasing tendency in both datasets. However, the values of extreme values of k do not vary significantly (the difference is only 0.03).

As for the second component of the error, in both cases, the graph hints some kind of a periodic function. However, the local minima do not correspond amongst the datasets. Therefore, the N_2 does not imply any optimal value of k. Similarly to the eps-setting, the third component of the error does not suggest any trend.

All in all, according to the first component of the error in both datasets, the optimal k from our range is 30. The decreasing tendency of N_1 suggests that with increasing k, we get the lower error.

Despite these results, it is improbable that the maximum value of k should be optimal. As we explained in Section 11.1, both too short and too long k-mers should cause the error incrementation. This leads us to the conclusion that we chose too small interval of k.

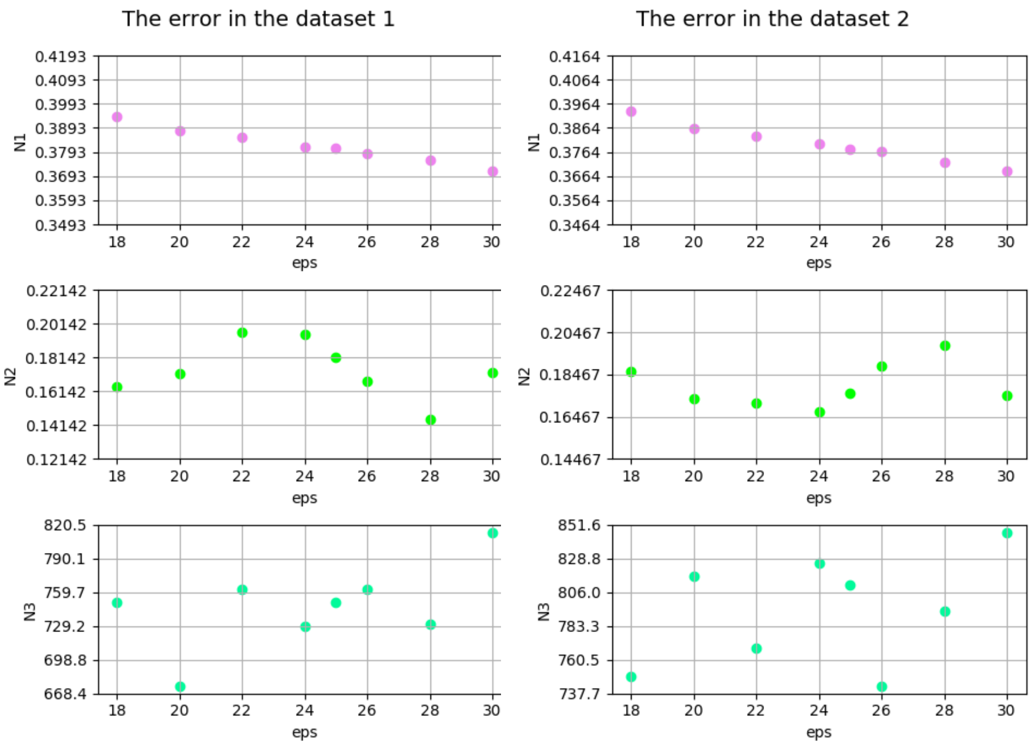


FIGURE 12.2: Errors of both datasets with eps set to 6

12.3 Unaveraged error

Since we only showed the average of the errors of all subject in Figures 12.2 and 12.1, we will look closely to the second component of the error amongst all subjects in the dataset.

As Figure 12.3 shows, all of the subjects have a similar error. In other words, the standard deviation is low and averaging did not distort the previous results.

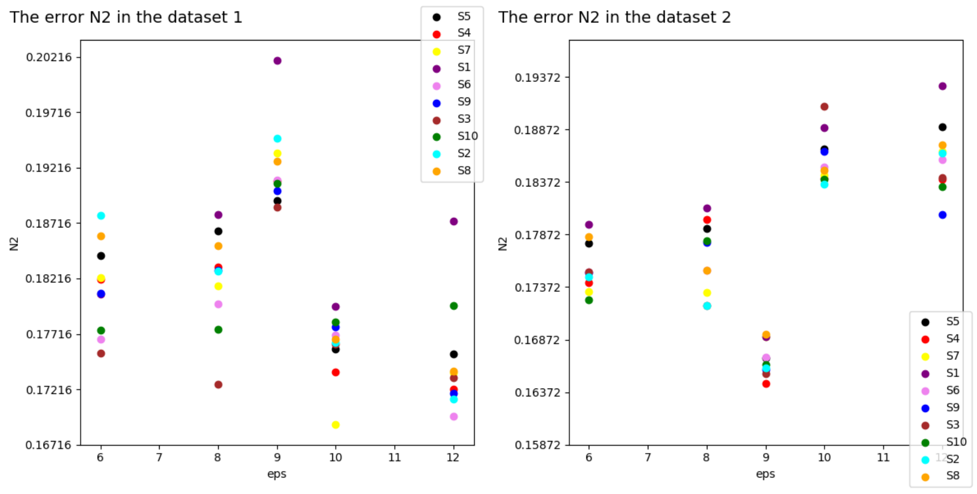


FIGURE 12.3: Errors amongsts the subjects with k set to 25

12.4 The results of N_3

In this section, we will analyze the results of the third component of the error. As Figures 12.1 and 12.2 illustrate, the value of N_3 varies from 668 to 883 in all cases. This means that for one read marked significant exist 668-863 reads that should be marked significant but are not. On the other hand, these unmarked reads might overlap.

To explain these values, we will look at Figure 12.4

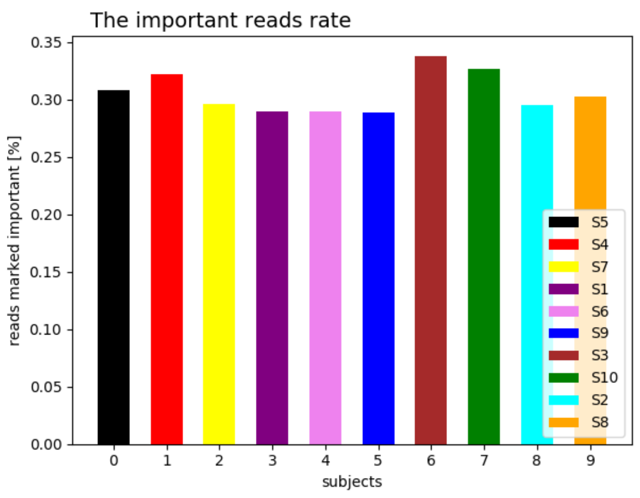


FIGURE 12.4: Histogram of the significant reads rate

This figure shows the percentage of the reads marked important in the reference table amongst the subjects from the dataset 1. The eps is set to 6 and the k-value equals 25. We can see that the percentage of the significant reads does not exceed 0.35%. Due to this rate combined with the values of the third component, we suppose that we marked too few reads as significant.

13 Future Work

In this chapter, we will discuss how to improve our solution. We will also suggest the possibilities of the application of our algorithm.

As for the improvement of our algorithm, there are several features that we could optimize.

Firstly, we could perform an experiment on all the parameters that are set to a constant. These are for instance the upper bound in the second step of the proposed solution (see Section 8.3) and the range of the *Important cathegory*. Unlike the upper-bound, which would probably not highly improve the outcome, setting the *Important cathegory* offers more space for the improvement.

The range of the *Important cathegory* is now set to 50, meaning that we take the 48 largest clusters (excluding the first two) of k-mers in the third step (see Section 8.4) of our pipeline. Moreover, we exclude the noise and equally distributed k-mers.

Now, changing the range of the *Important cathegory* should have a rather high impact on the second component of the errors. If the *Important cathegory* contained fewer clusters; we would mark fewer reads as important. However, this would result in fewer distinguished reads.

Another improvement to our algorithm could be another modification in the third step of our solution. As we sampled the k-mers, taking only 10% of the data, some information could be lost. Thus, we could assign each unassigned k-mer the cluster number of the nearest neighbor. On the contrary, this would increase the memory usage and might not significantly increase efficiency due to the explanation in Section 8.4.1.

We could also expand the range of k in our experiment to find the global extreme.

To decrease the required time of our pipeline we could also rewrite it from python to C++. If optimized, it could take less time than the standard approach.

In the next section, we will discuss the further applications of our algorithm.

13.1 Further Possibilities of Application

In this section, we will introduce the disciplines, where our algorithm could be well-used. Obviously, as our approach does not use any reference to cluster reads, it could be highly appreciated in processing reads from species despite having no information about its genome.

As annotating the whole genome (meaning we know where in the DNA-sequence the particular gene starts and where it ends) and creating the reference genome is non-trivial, our solution can spare time and money necessary for creating it.

Our algorithm could improve the transcriptome assembly, as it could divide the input reads into multiple groups corresponding to different regions (genes). Therefore, if the transcriptome-assembly method searched for the succeeding reads, it would not take into consideration reads from a completely different sector.

Hence, the read assembly efficiency would increase.

Finally, our solution could be helping in regards to the subject classification. We could determine whether the subjects gene expression levels of important clusters is closer to the sample of healthy or unhealthy subjects. Moreover, we could determine the difference between the samples of healthy and unhealthy subject and thus reveal the reads (genes) responsible for the particular disease.

14 Conclusion

The importance of efficient data processing is rather high in the field of bioinformatics. The information gathered from DNA sequencing can tell us a lot about organism it originated from.

This thesis consists of two parts. In the first part, we introduced a microbiology background. After that, we reviewed the current methods of DNA or RNA sequencing processing, such as the read assembly and read alignment.

We also provided a brief overview of the clustering methods.

In the second part, we focused on read clustering according to their membership to genes. We described the STAR algorithm, the method which uses the known reference genome to align reads.

However, the main disadvantage of this approach is the requirement of the reference. Therefore, we presented the concept of read clustering by gene expression. The main idea of the concept is, that reads from one gene have similar expression through multiple subjects.

Consequently, we compared the reference-base pipeline to our solution. Unfortunately, our method had the error of around 0.34. On the other hand, when taking into account only reads that our algorithm distinguishes well, the error decreased rapidly (to around 0.18).

Then we designed the experiments to estimate the optimal parameters. As a result, we suggest to set the value of k to 30 or perform more experiments on setting k to find the global minimum of the error.

The further experiments, primarily changing the range of k-mers we consider well-distinguished, could also improve the efficiency of our algorithm.

Finally, we discussed the future application of our algorithm. The main advantage of our pipeline is no need for the reference genome. Therefore, it might be used when no reference is available. It also has the potential to increase the performance of the read assembly.

A Implementation

1. The first step (Proposed solution Section 8.2) :
   To make usage of jellyfish easier, we wrote python script to run jellyfish on all input files (path is set to mypath by default).
   Provided script: genJellyCom.py
   Parameters: –datadir (path to the input data folder) –ksize (k-mer length)
   Input: folder containing R1 and R2 files for each subject
   Output: script (`.sh`) that runs jellyfish on input data using 4 threads and saves results to Results folder, each input file name with corresponding output-file name. Each output file contains k-mers present in the input file with its count.
   Time: this step runs approximately 40 minutes

2. The second step (Meging counts as described in the proposed solution Section 8.3):
   Provided script: dictionary.py
   Parameters: –restable (result table name)
   Input: folder with the output of jellyfish
   Output: table.txt described in Section 8.3
   Time: this step takes roughly 25 minutes

3. The third step (Clustering k-mers as described in the proposed solution Section 8.4):
   Provided script: DBSCAN-clustering2.py
   Parameters: –inptable (input table name) –eps (number)
   Input: table name from output of the second step, eps - parameter of DBSCAN
   Output: clusters.txt, each line containing k-mers in one cluster, lines sorted by size,
   graph showing some of the resulting clusters after PCA-dimension reduction,
   sizes.txt containing information about the size of clusters,
   plot.pkl containing data necessary to generate graph
   Time: this step takes around 3 hours

4. The fourth step (Clustering reads according to contained k-mers described in Section 8.5):
   Provided script: readClustering3.py
   Parameters: –ksize (the length of k-mers) –dbtable (result table name) –datadir (the folder containing data as described in Section 8.1)

Input: clusters.txt from the previous step, original data described in the proposed solution Section 8.1
Output: membership-table for every subject in data
Ressig folder containinf ID-s of reads in *Important cathegory* for every subject
Time: this step runs approximately 11 hours

5. Generating the reference-membership table as described in Chapter 9.
   Provided script: reads2Gene2.py
   Parameters: None
   Input: Aligned `.BAM` files (output from STAR)
   gtf - reference file
   Output: reference-membership table
   Time: this step takes roughly 7 hours

6. Comparing results (generates file for one dataset, on each line containing a name of subject-reads file and error between our assignment of reads and reference-based approach as described in Section 7.2).
   Provided script: comparing3.py
   Parameters: –refclust (the name of folder containing ref-membership-tables) –myclust (the name of folder containing our-membership-tables) –ressig (the name of folder containing the *Important cathegories*)
   Input: folders described in parametres
   Output: results2.txt (containing error for each subject from dataset)
   Time: this step takes 3 to 6 hours (when sample size is 0.0009%)

B CD content

- Mertanova_BP.pdf containing the text of the thesis
- README with information about running scripts
- all of the script described in Appendix A (comparing3.py, DBSCAN-clustering2.py, dictionary.py, genJellyCom.py, readClustering3.py, reads2Gene2.py)
- scripts to run STAR: IndexGen.sh, Alignment.sh
- script to generate the reference-membership table from the STAR output: ref2pkl.sh
- script for running our pipeline on metacentrum including comparing with the reference-membership table: runAll.sh
- the final results (folder Results) for various parametres of dataset1 (resfin1.zip) and dataset2 (resfin2.zip)
- the folder Visualizing with python scripts + manual for visualizing the results (DBSCAN-plotting.py, plotFinHist.py, plotFinNEps.py, plotFinN2Eps.py, plotFinNKsize.py, README)

Bibliography

[1] Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. “HTSeq—a Python framework to work with high-throughput sequencing data”. In: *Bioinformatics* 31.2 (2015), pp. 166–169.

[2] Geoffrey H Ball and David J Hall. *ISODATA, a novel method of data analysis and pattern classification*. Tech. rep. Stanford research inst Menlo Park CA, 1965.

[3] Nicolas L Bray et al. “Near-optimal probabilistic RNA-seq quantification”. In: *Nature biotechnology* 34.5 (2016), p. 525.

[4] Chire. *Density-based clustering*. [online accessed Dec-19-18]. 2018. URL: https://commons.wikimedia.org/w/index.php?curid=1848174.

[5] International Human Genome Sequencing Consortium. “A global reference for human genetic variation”. In: *Nature* (2015). URL: https://doi.org/10.1038/nature15393.

[6] International Human Genome Sequencing Consortium. “Finishing the euchromatic sequence of the human genome. Nature Education 1(1):100”. In: (2004). URL: https://doi.org/10.1038/nature03001.

[7] International Human Genome Sequencing Consortium et al. “Initial sequencing and analysis of the human genome”. In: *Nature* 409.6822 (2001), p. 860.

[8] Andrew Dalke et al. “Biopython: freely available Python tools for computational molecular biology and bioinformatics”. In: *Bioinformatics* 25.11 (Mar. 2009), pp. 1422–1423. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp163.

[9] *DNA translation*. [online accessed Dec-4-18]. 2018. URL: https://commons.wikimedia.org/wiki/File:Peptide_syn.png.

[10] Alexander Dobin et al. “STAR: ultrafast universal RNA-seq aligner”. In: *Bioinformatics* 29.1 (2013), pp. 15–21.

[11] Friederike Dündar, Luce Skrabanek, and Paul Zumbo. “Introduction to differential gene expression analysis using RNA-seq”. In: *Appl. Bioinformatics* (2015), pp. 1–67.

[12] Pär G Engström et al. “Systematic evaluation of spliced alignment programs for RNA-seq data”. In: *Nature methods* 10.12 (2013), p. 1185.

[13] Martin Ester et al. “A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon: AAAI Press, 1996, pp. 226–231. URL: http://dl.acm.org/citation.cfm?id=3001460.3001507.

[14] Estevezj. *Sanger picture*. [online accessed Dec-21-18]. 2018. URL: https://commons.wikimedia.org/wiki/File:Sanger-sequencing.svg.

[15] Lilit Garibyan and Nidhi Avashia. “Polymerase chain reaction”. In: *J Invest Dermatol* 133.3 (Mar. 2013). S0022-202X(15)36139-X[PII], pp. 1–4. ISSN: 1523-1747. DOI: 10.1038/jid.2013.1.

[16] Sara Goodwin, John Douglas Mcpherson, and W. Richard McCombie. "Coming of age: Ten years of next-generation sequencing technologies". English (US). In: *Nature Reviews Genetics* 17.6 (June 2016), pp. 333–351. ISSN: 1471-0056. DOI: 10.1038/nrg.2016.49.

[17] Jia Guo et al. "Four-color DNA sequencing with 3'-O-modified nucleotide reversible terminators and chemically cleavable fluorescent dideoxynucleotides". In: *Proceedings of the National Academy of Sciences* 105.27 (2008), pp. 9145–9150.

[18] Tim Hubbard et al. "The Ensembl genome database project". In: *Nucleic Acids Research* 30.1 (2002), pp. 38–41. DOI: 10.1093/nar/30.1.38.

[19] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[20] Illumina, Inc. *Data Sheet: Sequencing*. [Online; accessed 7-May-2019]. Oct. 2010. URL: https://www.illumina.com/documents/products/datasheets/datasheet%5C_genomic%5C_sequence.pdf.

[21] Anil K Jain. "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8 (2010), pp. 651–666.

[22] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 3-May-2019]. 2001–. URL: http://www.scipy.org/.

[23] Neil C Jones, Pavel A Pevzner, and Pavel Pevzner. *An introduction to bioinformatics algorithms*. 2004.

[24] Jingyue Ju et al. "Four-color DNA sequencing by synthesis using cleavable fluorescent nucleotide reversible terminators". In: *Proceedings of the National Academy of Sciences* 103.52 (2006), pp. 19635–19640.

[25] Daehwan Kim et al. "TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions". In: *Genome biology* 14.4 (2013), R36.

[26] Benjamin King. "Step-wise clustering procedures". In: *Journal of the American Statistical Association* 62.317 (1967), pp. 86–101.

[27] Trupti M Kodinariya and Prashant R Makwana. "Review on determining number of Cluster in K-Means Clustering". In: *International Journal* 1.6 (2013), pp. 90–95.

[28] Hans-Peter Kriegel et al. "Density-based clustering". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.3 (2011), pp. 231–240.

[29] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. 2nd ed. Cambridge University Press, 2014. DOI: 10.1017/CBO9781139924801.

[30] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489.

[31] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[32] Madprime. *DNA structure*. [online accessed Nov-28-18]. 2018. URL: https://commons.wikimedia.org/w/index.php?curid=1848174.

[33] Guillaume Marçais and Carl Kingsford. “A fast, lock-free approach for efficient parallel counting of occurrences of k-mers”. In: *Bioinformatics* 27.6 (2011), pp. 764–770. DOI: 10.1093/bioinformatics/btr011.

[34] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[35] Rob Patro, Stephen M Mount, and Carl Kingsford. “Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms”. In: *Nature biotechnology* 32.5 (2014), p. 462.

[36] Rob Patro et al. “Salmon provides fast and bias-aware quantification of transcript expression”. In: *Nature methods* 14.4 (2017), p. 417.

[37] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[38] Pavel A Pevzner & Glenn Tesler Phillip E C Compeau. “How to apply de Bruijn graphs to genome assembly”. In: (2011).

[39] Pavel A Pevzner & Glenn Tesler Phillip E C Compeau. “Veritas Genetics. Veritas genetics launches $999 whole genome and sets new standard for genetic testing — Press Release.” In: (2016). URL: https://www.veritasgenetics.com/documents/VG-launches-999-whole-genome.pdf.

[40] L. Pray. “Discovery of DNA structure and function: Watson and Crick. Nature Education 1(1):100”. In: (2008). URL: https://www.nature.com/scitable/topicpage/discovery-of-dna-structure-and-function-watson-397.

[41] Lior Rokach and Oded Maimon. “Clustering methods”. In: *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[42] Connie Rye. *Biology*. [online] reviewed 7-December-2018. URL: http://cnx.org/contents/185cbf87-c72e-48f5-b51e-f14f21b5eabd@10.4.

[43] Steven L. Salzberg. “Open questions: How many genes do we have?” In: *BMC Biology* 16.1 (Aug. 2018), p. 94. ISSN: 1741-7007. DOI: 10.1186/s12915-018-0564-x.

[44] F. Sanger, S. Nicklen, and A. R. Coulson. “DNA sequencing with chain-terminating inhibitors”. In: *Proc Natl Acad Sci U S A* 74.12 (Dec. 1977), pp. 5463–5467. ISSN: 0027-8424. URL: https://www.ncbi.nlm.nih.gov/pubmed/271968.

[45] Peter HA Sneath, Robert R Sokal, et al. *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[46] Hugo Steinhaus. “Sur la division des corp materiels en parties”. In: *Bull. Acad. Polon. Sci* 1.804 (1956), p. 801.

[47] Elizabeth Thompson, Jakub Tolar, and Eric A. Hendrickson. “FANCN Hypomorphic Mutation Retains BRCA1 Binding Domain”. In: *Blood* 128.22 (2016), pp. 2676–2676. ISSN: 0006-4971. URL: http://www.bloodjournal.org/content/128/22/2676.

[48] J. Craig Venter, Mark D. Adams, and Myers. “The Sequence of the Human Genome”. In: *Science* 291.5507 (2001), pp. 1304–1351. ISSN: 0036-8075. DOI: 10.1126/science.1058040. eprint: http://science.sciencemag.org/content/291/5507/1304.full.pdf. URL: http://science.sciencemag.org/content/291/5507/1304.

[49] Joe H Ward Jr. “Hierarchical grouping to optimize an objective function”. In: *Journal of the American statistical association* 58.301 (1963), pp. 236–244.

[50] Shanrong Zhao et al. “Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells”. In: *PloS one* 9.1 (2014), e78644.