Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Data Structures for Selective Sequencing

Bachelor Thesis

2020
Andrej Korman

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Data Structures for Selective Sequencing

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. Mgr. Tomáš Vinař, PhD.

Bratislava, 2020
Andrej Korman

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Andrej Korman
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Data Structures for Selective Sequencing
Dátové štruktúry pre selektívne sekvenovanie

Anotácia: Sekvenovacia technológia MinION umožňuje vylúčenie molekúl zo sekvenovania priamo počas sekvenovacieho behu. Problémom však je, že preklad sekvenovacieho signálu to reťazca DNA je príliš pomalý na to, aby ho bolo možné robiť priamo za behu. Cieľom tejto práce je vyvinúť indexovaciu schému a dátové štruktúry, ktoré by umožnili rýchlu identifikáciu jednotlivých molekúl priamo na základe sekvenovacieho signálu namiesto porovnávania reťazcov DNA.

Vedúci: doc. Mgr. Tomáš Vinař, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 30.10.2019

Dátum schválenia: 30.10.2019

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Andrej Korman
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Data Structures for Selective Sequencing

Annotation: MinION sequencing technology allows on-line rejection of molecules that are sequenced. However, the base calling process (translation of the raw sequencing signals to DNA bases) is too slow to perform on-the-fly. The goal of this thesis is to develop an indexing scheme and data structures that would allow fast identification of individual molecules based on raw signals instead of the corresponding DNA sequences.

Supervisor:	doc. Mgr. Tomáš Vinař, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	30.10.2019

Approved: 30.10.2019 doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: Na tomto mieste by som chcel v prvom rade poďakovať môjmu školiteľovi doc. Mgr. Tomášovi Vinařovi, PhD, ktorý mi venoval nemalé množstvo svojho času. Taktiež by som chcel poďakovať super bioinformacikej komunite tu na Matfyze, najmä doc. Mgr. Bronislave Brejovej, PhD. V neposlednom rade sa chcem poďakovať celej mojej rodine, že ma roky podporovala v každom mojom úsilí a tiež mojej priateľke Kristíne Korecovej, ktorá mi po celý čas písania práce dávala cenné rady a návrhy na vylepšenia.

Abstrakt

Selektívne sekvenovanie je veľmi užitočná technológia, umožnená sekvenátorom MinION. Ten nám dovoľuje počas samotného sekvenovania vybrať DNA molekuly, ktoré chceme sekvenovať. V niektorých scenároch toto veľmi zefektívni proces sekvenovania. Avšak, v súčasnosti neexistuje riešenie, ktoré by umožňovalo plne využiť potenciál tejto technológie. Proces prekladania báz nie je stále dostatočne rýchly aby splnil požiadavky tejto metódy. V tejto práci sa pokúsime vybudovať algoritmus diskretizovania signálu aby sme umožnili jednoduchšiu prácu so surovým signálom. Potom sa pozrieme na vlastnosti tejto reprezentácie a bližšie sa pozrieme na vhodnosť tejto reprezentácie pre účely selektívneho sekvenovania.

Kľúčové slová: selektívne sekvenovanie, MinION

Abstract

Selective sequencing is very useful technology enabled by the MinION sequencer that allows us to choose DNA molecules that we want to sequence on-the-fly. In some scenarios, this can make the whole process of the DNA sequencing much more effective. However, currently there is no solution that would be able to use the full potential of this technology for the big reference sequences. The base-calling process is still not fast enough to satisfy the needs of this method. In this work, we try to build a discretization algorithm to allow easier work with the raw squiggles. Then we look at the properties of this representation and discuss its suitability to help achieve the goal of the selective sequencing.

Keywords: selective sequencing, MinION

Contents

Introduction

The sequence of DNA is one of the most important molecules in the living organisms. It consists of the nucleotides adenine (A), cytosine (C), guanine (G), and thymine (T). The MinION sequencer enables us to read this DNA so we can study numerous secrets of the DNA molecule and its implications in our lives. The MinION sequencer, however, does not produce the DNA sequence directly. Instead, it produces a noisy signal created as the DNA passes through one of the MinIONs nanopores. Due to the nature of the nanopores, we are able to reconstruct the DNA sequence from this noisy signal using the process called base-calling.

Another advantage of the MinION sequencer is that it is able to release the DNA fragment that is currently passing through one of his nanopores without further sequencing. Selective sequencing is an idea that based on the signal currently produced by the MinION, we can decide if we continue the sequencing or release the DNA fragment. Selective sequencing enables us to enrich sequencing for the fragments, we are interested in.

To this day, base-calling algorithms are too slow to apply this idea in practice. During the time, we are base-calling the signal obtained from the beginning of the DNA fragment, the fragment could have already passed through the pore and could been already sequenced in its entirity.

This is why, we have to work with the raw signal. This is quite hard as the signal is very noisy. We propose a method to discretize the raw signal and then subsequently attempt to find it in a longer reference discretized signal. The problem of this method is that most of the time we are only given the reference in the form of the DNA so we need to create a simulated signal.

In the first chapter, we present the necessary background needed to understand the main challenges in this area and existing solutions.

In the second chapter, we propose a discretization method and a method for simulating the signal from the reference sequence. We look at different properties of the real and simulated signal and we test several methods to make them to have similar properties. At the end of the second chapter, we summarize experimental results of our efforts in this area.

In the third chapter, we use the methods developed and tested for signal discretiza-

tion to build the index data structure that is able to respond fast to requests whether the query signal is present in the index.

Chapter 1

Selective DNA sequencing

In this chapter, we explain the term selective sequencing. We highlight the advantages of this method and look at the current state of research in this area.

1.1 DNA

Genetics is a branch of biology that studies genes, genetic variation, and heredity [1]. It tries to explain the variability between animals, the source of hereditary diseases, and other important things that influence our lives. The DNA is the key molecule that stores biological information in living organisms. It is contained in almost every living cell. Based on the information from this molecule, our cells can reproduce and create copies of themselves. Nowadays, it is possible to look at the DNA of the organism. This ability is one of the strongest tools of genetics as it helps us tell what are the functions of different parts of DNA by comparing it between organisms and looking at the consequences of different mutations.

DNA stores genetic information in the form of a sequence of nucleotides. Their particular order defines the stored information. There are four types of DNA nucleotides: adenine (A), cytosine (C), guanine (G) and thymine (T). DNA consists of two long strands of these nucleotides that together create the DNA molecule. These two strands are connected in a complementary way. If there is A on the forward strand, then there is T on the reverse strand. If there is a C on the one strand, we can expect its complement G on the other one. In this way, the cell machinery can (to some extent) repair missing nucleotides based on the complement rules. An example of how we can think of DNA is in Figure 1.1.

From now, we will represent the DNA molecule as a sequence of characters A, C, G, T. Such representation stores enough information so we can recover the other complementary DNA strand easily.

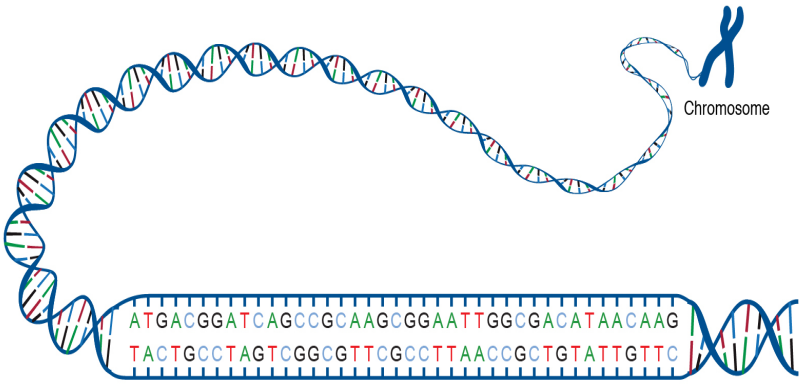

Figure 1.1: DNA Molecule [8]

1.2 DNA sequencing

The process of obtaining DNA sequence is called *DNA sequencing*. The DNA molecule is very long, and processing it whole at once would be very hard. The single DNA molecule in one cell can reach up to 2 meters when untangled [4]. Thus, it is convenient to broke down the whole DNA molecule into many shorter fragments. This can be done chemically. Once we have this mixture of shorter DNA fragments, we need to sequence them individually and then assemble them to obtain the nucleotide string representing our original DNA molecule. One of the devices that can sequence the mixture of the DNA fragments is MinION[13]. MinION is a cheap and versatile DNA sequencer with the size of the larger USB key (see in Figure 1.2).

MinION consists of an active surface filled with many *nanopores*. A nanopore is a small hole with an electric current passing through it. When the positive charge is generated on the other side of this surface, negatively charged DNA molecules would start to pass through the pores. As the molecule of DNA is passing through the pore of MinION, we can observe changes in the flow of an electric current passing through the pore. This electric current is measured over the discrete-time and is called *signal*, *raw signal*.

Currently, MinION processes around 450 nucleotides per second for each nanopore. The value of the electric current is measured 4000 times per second so each nucleotide produces on average about 8-9 measurements, also called *readouts*. The signal generated from the pass of the single DNA fragment through the single pore is called *squiggle*. The example of the squiggle is shown in Figure 1.3. One of the advantages of the MinION sequencer is that it produces very long squiggles. The length of the squiggles is stated in kb(kilo-bases) where base corresponds to one nucleotide. The

Figure 1.2: MinION sequencer[2]

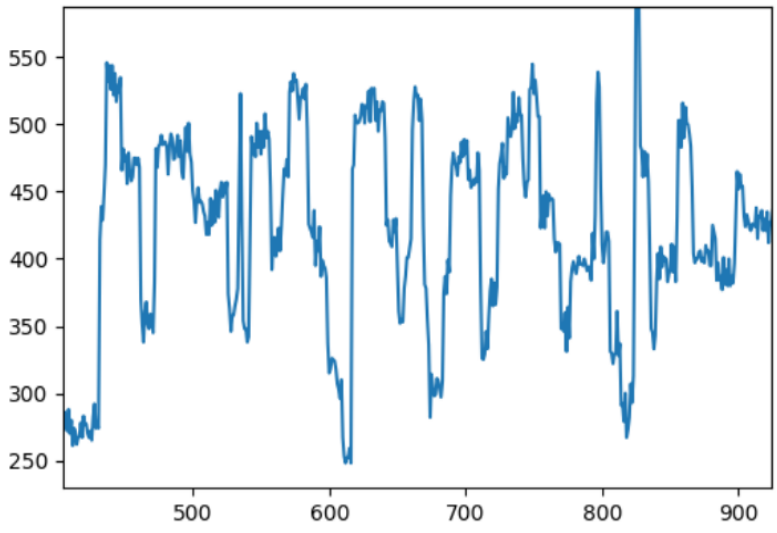

Figure 1.3: Electric current (squiggle) coming from MinION.

mean length of the squiggles in some scenarios ranges from 13kb to 20kb [14]. Additionally, since MinION has 200-500 nanopores, it can produce large amounts of data very quickly.

After obtaining the individual squiggles, these need to be converted into the DNA sequence. Importantly, as the DNA passes through the pore, only a small number of nucleotides in the proximity of the pore influence the current output signal.

The output signal of MinION is mostly dependent only on the context of the k nucleotides (also called *k-mers* present in the proximity of the pore. There are several studies on how much is signal influenced by more distant nucleotides. It is generally accepted that using the $k = 6$ is appropriate for building an accurate model of the signal.

Usually, with the MinION, we are provided with a signal model. In general, the signal model is a list of all possible k-mers for some k that states the mean and standard deviation of a Gaussian distribution that describes expected distribution signal readout for that particular k-mer. Using the measurements of signal over time and a method called base-calling, we can reconstruct the sequence of the DNA molecule which produced this particular signal.

Early base-calling algorithms tried to split the signal into *events*[**?**]. Event is a longer sequence of readouts at the roughly same level. The event corresponds to one particular k-mer present in the nanopore at that time. Using the k-mer model on events, as well as information that k-mers following each should overlap, one can predict the sequence that passed through this pore.

Nowadays, more successful base-calling algorithms are based on deep and recurrent neural networks[15]. Despite significant improvements in the base-calling algorithms, the overall process is quite slow and resource-intensive.

After the base-calling we have hundreds of these base-called squiggles, called *reads*. Read is the squiggle that does not carry only the raw signal but also the information what nucleotide sequence is his signal representing. We need to assemble this reads, to form the resulting DNA sequence. This is quite hard as the base-calling process is not entirely accurate and produces some errors. Thanks to the duplication of the shorter DNA fragments, we are, in most cases, able to reconstruct the original DNA string.

1.3 Selective sequencing

MinION has an ability to reject DNA molecule that is currently passing through the pore. MinION reverses the direction of the molecule and throws it away. *Selective sequencing* is the idea that based on the incoming signal, one can determine whether we are interested in sequencing the current DNA molecule. Subsequently, we can decide

if we want to continue or reject this molecule. This happens on-the-fly, so we need to make the decision as quickly as possible for the process to be effective.

There are many benefits of selective sequencing. In case we are not interested in some DNA that we know is contained in our sample, we can use this technique to reject unwanted DNA molecules. This saves us a lot of time and resources as obtaining nucleotide sequence from the signal is in some scenarios unnecessary and too costly process in terms of performance.

For example, consider a scenario when attempting to sequence a pathogen (bacteria, virus) in human blood. We can reject all human DNA molecules because we are not interested in sequencing human DNA. We could also filter in positive way. So we could say that we are only interested in sequencing DNA that produces a signal in some sense similar to our chosen sample. This all means saving a lot of resources, for example, during the disease diagnosis process.

Naturally, there are some drawbacks to selective sequencing. In order to find out if the currently passing molecule is from human DNA, we have to have some information about the signal from human DNA beforehand. This is limiting as we need a sample signal from the species that we want to filter. The other problem is that in the case of misclassifying some signal as not interesting, we lose information about the corresponding part of the DNA molecule.

The idea of accessing real-time data during the phase of reading the DNA molecules and rejecting the DNA molecule based on this data is called *Read Until*.

1.4 Current state of selective sequencing

One approach to selective sequencing is to base-call the DNA molecule that is currently passing through the pore and try to find it in target, also called *reference*, sequence. If we could not find this DNA sequence in any part of the reference, we can discard this DNA molecule.

The base-calling process, however, is quite slow and our decision must be made on-the-fly. This often means we do not have time to turn our signal into a nucleotide sequence as the squiggles are too short for this and can be already sequenced when we made our decision.

In this thesis, we will work directly with the raw signal to ascertain whether the molecule should be sequenced or not. One approach (see [12]) is to obtain the reference signal using simulation from the reference nucleotide string. We then use an algorithm called the dynamic time warping (DTW) to align the signal to the reference signal. The DTW is a dynamic programming algorithm that takes two signals and aligns them in a way that minimizes the total number of insertions and deletions of the readouts. One

of the limitations of DTW is that it has run time complexity $\mathcal{O}(n \cdot m)$ where n and m are the lengths of the reference signal and query signal, respectively. This would pose a problem for this method if we wanted to use it for longer sequences. This is, however complete, implemented and tested solution that can be used for shorter reference DNA.

Another useful tool that we will heavily use during the testing of our method, with a similar idea as the previous solution is nanopore data variant caller (Nadavca) [5]. It is a convenient tool that requires a reference DNA sequence and squiggle. Nadavca can create the simulated reference signal and search for the squiggle in this simulated signal using the already mentioned DTW algorithm. One drawback is that it is not fast enough to be used for selective sequencing.

In our work, we attempt to improve the results of the work with the squiggles. We focus on the scenario, where we are given a reference sequence of the DNA. We will attempt to distinguish between the signal originating from this reference DNA and the signal unrelated to the reference.

Chapter 2

Squiggle Discretization and Similarity Between Squiggles

As we stated in the previous chapter, we will work directly with the squiggles instead of the basecalled sequences. However, for this to be possible we need to have some reference squiggle that we can sample and use as an index in which we will search for the squiggles once we are sequencing. Unfortunately, we are given the reference only in the form of a DNA sequence. In this chapter we will show how to create the reference squiggle from the reference DNA sequence and also introduce more suitable representation of the signal that enables easy searching of the query signal in the reference.

2.1 Squiggles as Sequences of Discrete Observations

Given a simulated squiggle from the reference DNA, we want to decide whether the squiggle passing through the pore has originated from the reference. This task can be reformulated as finding if the current squiggle is similar to a segment of the reference squiggle.

We propose to discretize the squiggles to facilitate their better representation and also provide easy searching capabilities in this representation. Note that, we are not very limited by the preprocessing time, but very fast processing is required during the sequencing of the DNA. This puts certain speed limitations on our discretizing process during the actual sequencing of the DNA.

We will represent the squiggle as a string of characters. We split the squiggle vertically into several windows so that everything between the minimum and maximum value is allocated to one of these windows. The windows are of a constant width and do not overlap. More formally, let a_i be the readout at the time i and m be the number of vertical windows. Denote the squiggle s of length n as $s = a_1 a_2 a_3 \cdots a_n$. Besides

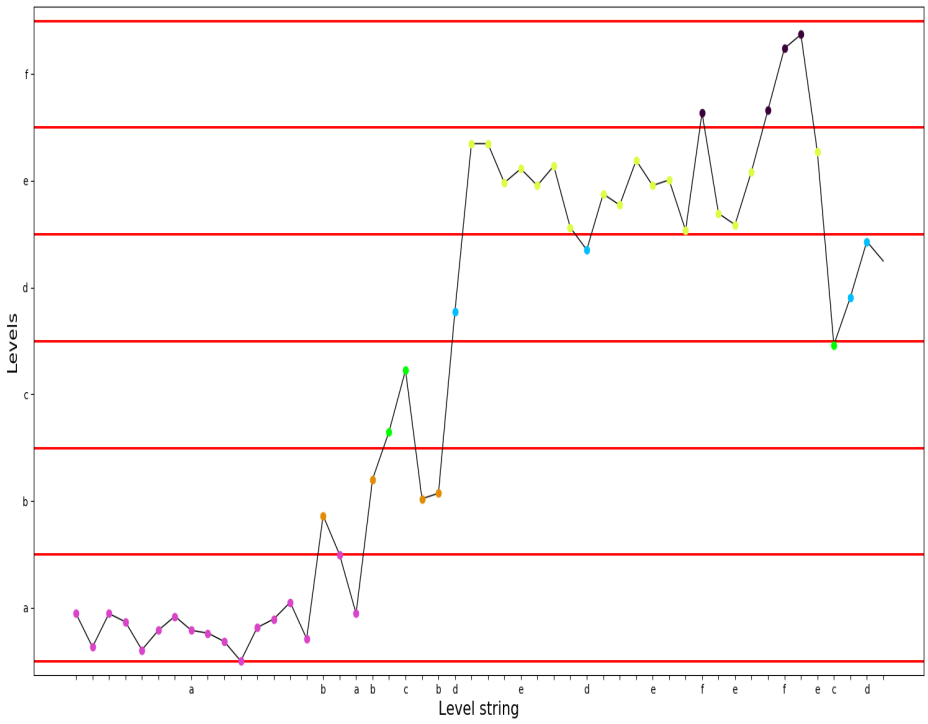

Figure 2.1: Squiggle with the level string displayed on the x-axis

this, we are given values min_a, max_a that $\forall a_i : min_a \leq a_i < max_a$. We say that the readout a_i is in j-th window if it holds that:

$$min_a + j \cdot \frac{max_a - min_a}{m} \leq a_i < min_a + (j+1) \cdot \frac{max_a - min_a}{m}$$

Let the readouts $a_0, a_1, \cdots, a_n$ correspond to the windows $w_0, w_1, \cdots, w_n$ respectively. ls such that $ls = w_1 w_2 \cdots w_n$ is called *level sequence*. Level sequence in which the subsequent appearances of the same character are substituted by the single apperance of that character is called the *level string* of the squiggle s. We will call the transformation from the level sequence to the level string a *contraction*. So if the $ls = aabccaa$ then the level string from this level sequence is "abca". An example of the transformation of the squiggle to the level string for $w = 6$ is shown in Figure 2.1.

Discretization has several important properties that will be later used. First, it is deterministic and the same squiggle has the same level string each time for the unchanged w. Second, small changes and variance in the signal readouts are unlikely to change the resulting level string most of the time. Note that the squiggle consists of events that are represented by a longer signal that is around the same level. With this method, we hope that each event will remain insde a single window over its duration. Subsequent contraction of the level string thus results in one event being represented by a single character of a level string in most cases.

Changes to the parameter w balance specificity and informativness of our method. With the $w = 2$, we cannot say a lot about two squiggles that have the same level

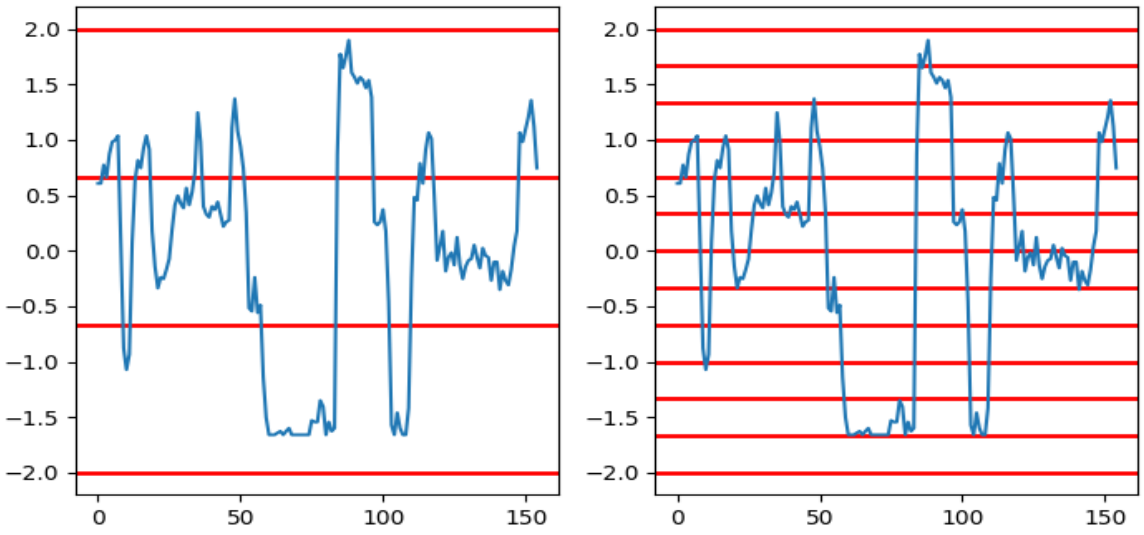

Figure 2.2: Examples of the squiggle divided into 3 and 12 windows

string. With larger w, we can miss two squiggles from the same DNA sequence if there is a too much noise present in one of them.

Consider two scenarios, when the number of levels is small and high. Figure 2.2 shows that for the small number of levels, a lot of different events stay in the same center window even if the higher number of levels could distinguish between these individual events. With a high number of levels, like we see in Figure 2.2 one event is less likely to remain within the same window. We will address the signal oscillations in more detail in Section 2.3.

2.2 Simulating Squiggles from the Reference

We already stated that before the process of the selective sequencing we are most of the time only given the reference DNA sequence. If we want to work directly with the squiggles, we need to obtain the reference squiggle. We need this reference squiggle so that we have some reference which we can sample and use later to split squiggles between that we are interested in and not interested in.

We create a simulated squiggle based on the 6-mer model, where each subsequent 6-mer in our reference DNA will generate a signal (event) equal to the mean of the gaussian distribution for that particular 6-mer. This mean value is the information provided by the manufacturer of MinION through the k-mer model that we mentioned in Section 1.2.

Our simplistic approach to signal simulation does not take into account that the signal from one nucleotide is measured several times. Thus, we replicate every entry in the simulated signal 10 times.

We now compare the simulated squiggle to the real squiggle that arose from the real sequencing. Moreover, we do not want to compare the simulated squiggle with unrelated squiggle but with the squiggle that corresponds to the same DNA region.

We will look now how to obtain the pair of the simulated-real signal that originate from the same DNA region. First, we take some real squiggle. Then, we find the part of the DNA reference that this squiggle corresponds to. After successfully finding this part of the DNA, we take it and simulate the squiggle from it. We will use two ways how of finding the DNA region corresponding to the squiggle in this work. The first, more accurate method is to use Nadavca. This tool, introduced in Section 1.4 takes as the input the reference DNA sequence and the particular squiggle we want to find in it. It then returns the table that accurately maps the nucleotides to the individual events in the squiggle. This possibility is very accurate and gives a lot of information that we do not need most of the time such as the mapping between the nucleotides and events. We are interested only in location of the start and end of the squiggle in the reference. The second option is to take the read instead of squiggle and use information from it. Hovewer, we know that the DNA sequence in read is generated by the base-caller algorithm so it has some error in it. To obtain the sequence of the nucleotides corresponding to this read without errors, we need to use some algorithm that is able to find this erroneous DNA sequence in the reference DNA sequence. For this purpose, we use minimap2 algorithm that we describe more closely in the Section 3.2. This is faster process as the solution using Nadavca and we will use this option anytime when we need speed and do not need any additional information that Nadavca provides.

Figure 2.3 compares the simulated squiggle to the squiggle obtained by sequencing the same region of the DNA.

Now we have a simulated reference squiggle. We can see on Figure 2.3 that the simulated and real squiggle are not identical. This poses a problem as we want to have these squiggles as similar as possible. The first thing that we want to consider is normalization. One of the most wide-spread ways of doing this is to subtract mean and dividing the signal by the standard deviation. The other possible solution is subtracting median value, as the frequent outliers can deform the mean. We will stick to the first solution, as our experiments showed that there is no big difference between squiggle mean and median value in most of the squiggles. After the normalization, we remove outliers in a way that everything over and under certain values will be clamped to the target range. Most of the time, this will be -2 as the lower bound and 2 as the upper bound.

Another visible problem can be seen when we compare events in the simulated and real squiggle. The events in the simulated signal are perfect repeats of the same value. In the real signal, we can sometimes see the phenomenon called *drift*. This is a fact

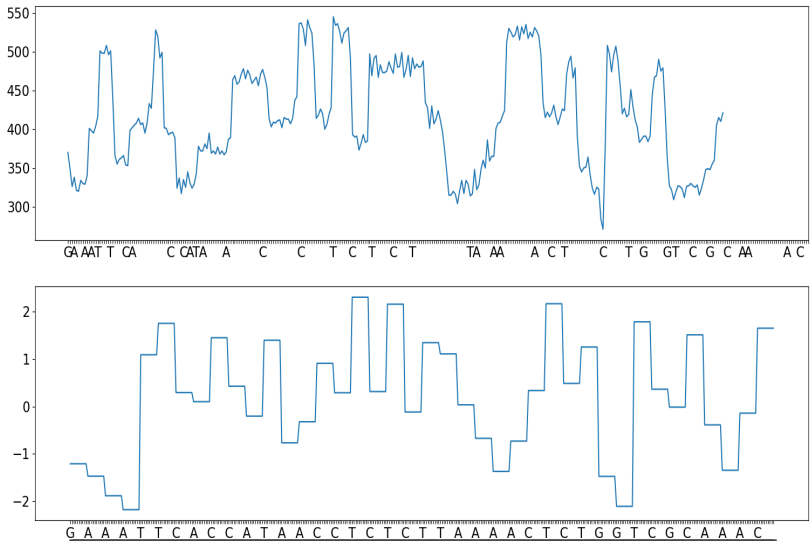

Figure 2.3: Comparison of the real (upper graph) and simulated (bottom graph) signal with the corresponding nucleotid sequence on x-axis

that real event tends to slide up or down over time. Another difference is that real squiggle is sometimes contracted in some places. This is caused by the fact that the DNA molecule is not moving through the pore at a constant speed.

There are two possible ways how to overcome the differences between real squiggle and simulated squiggle. One approach is to make the simulated squiggle more like the real squiggle, including adding a noise and a varying event deviation. In fact there are more advanced squiggle simulators such as DeepSimulator[10].

The opposite approach is to make the real squiggle less noisy. We decided to go the second way so we will now address the most important differences between the real and simulated squiggles.

2.3 Signal Oscillations and Continuity

Figure 2.3 clearly shows that one of the biggest differences between simulated and real squiggles is the noise which can be seen as small oscillations of the signal. This is one of the reasons that our discretization method use windows - so small oscillations within the window are dealt with. We can also easily adapt the number of windows. However, when our method is used without any modifications it produces much longer level strings for the real squiggles than for simulated squiggles. The problem arises when the oscillations are located on the borders of the windows and produce too long level string of the form "cdcdcdcd". To address this problem, we tried several smoothing techniques to reduce this oscillations. *Moving average* is the smoothing technique that for every readout s_i in some signal s calculate its new value $snew_i$ as $snew_i =$

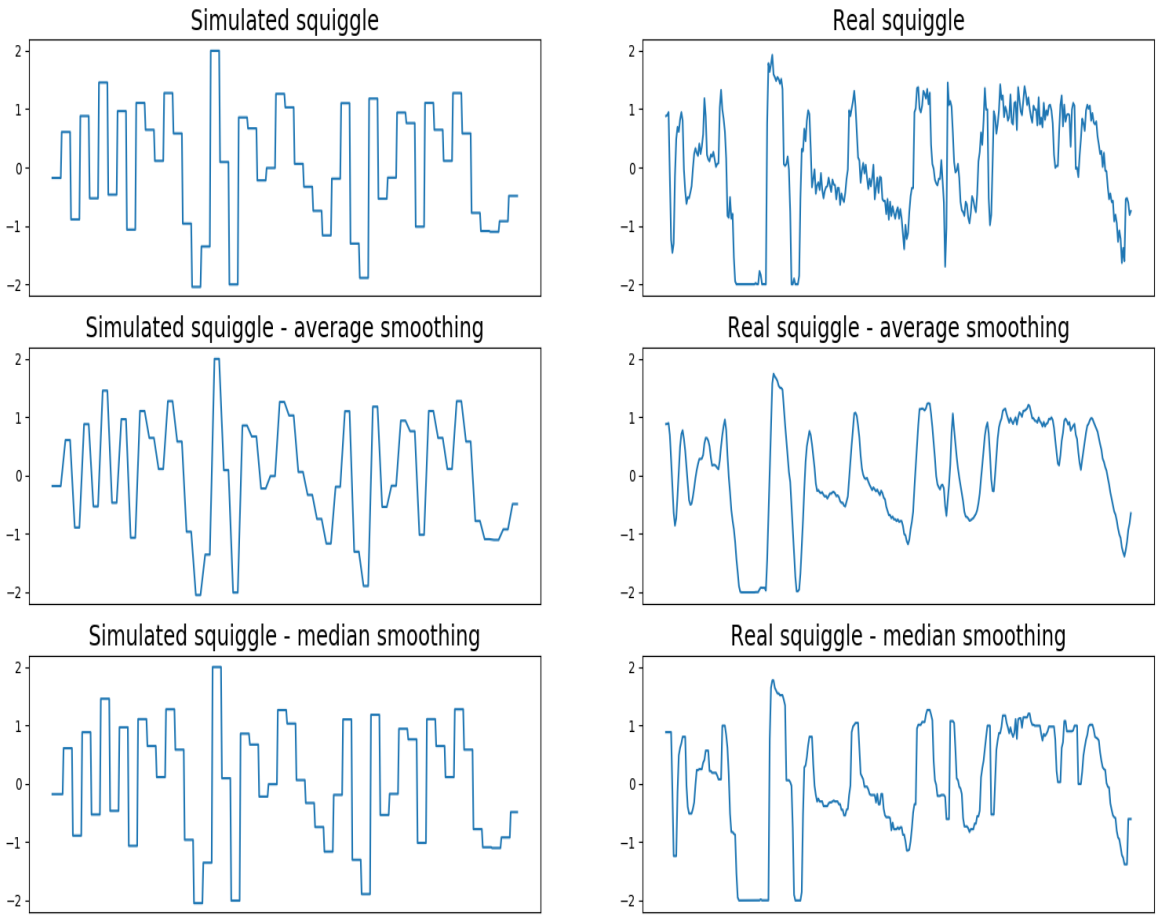

Figure 2.4: Comparison of simulated and real squiggle smoothed by different techniques

$avg(s_i, s_{i-1}, s_{i-2}, \ldots, s_{i-k+1})$. We will call k the *smoothing parameter*. We can also simply change the average function to the median to obtain the *moving median*. For now, we will stick to faster yet simpler smoothing techniques.

Another difference between the real signal level string and the simulated signal level string is that the real signal is more continuous instead of a simulated signal that jumps. Again, we can use the moving average. Instead of the moving median, it has a nice advantage that it causes signal not to jump from one level to another but rather move continuously.

In Figure 2.4 we can see part of the simulated and corresponding real squiggle. Then, in the subsequent two graphs, we apply the moving average with a smoothing parameter equal 5. We can see that this smoothed the simulated squiggle but also smoothed the noise that we could observe in the events. With the median smoothing, the outcome is quite similar but lacks the quality of smoothing the simulated signal.

In Figure 2.5 we see the result of our smoothing techniques applied on the simulated reference squiggle and real squiggle of lengths 10 000. We can see how big impacts the smoothing techniques have on the lengths of the resulting level strings. In most of

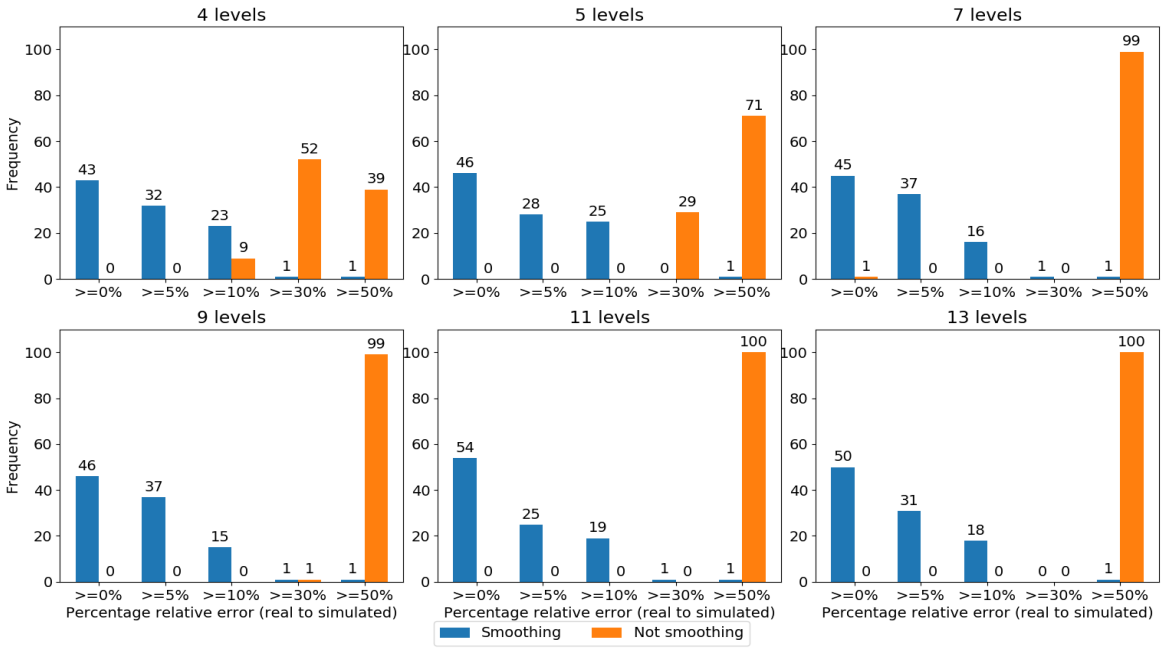

Figure 2.5: The impacts of the smoothing on the underlying squiggles of length 10 000. We see the individual frequency of reads level string length to reference level string length

the cases, the majority of level strings from the real squiggles without smoothing are more than half time bigger in size. After the smoothing we can see big changes in the distribution of the length ratios.

2.4 Identification of Reads Based on Shared k-mers

In this section, we will demonstrate how to distinguish between the cases when two squiggles originate from the same sequence and cases when we are comparing unrelated squiggles. We take the pair real squiggle - simulated reference squiggle from the same DNA region and one unrelated real squiggle. We will then try to distinguish which of the two real squiggles is from the same region as the simulated squiggle and which is not based only on the level strings derivated from the respective squiggles. We emphasized the speed many times, so we will try to come up with a fast test. We split the individual level strings into overlapping k-mers (any k subsequent characters from the string). The size of the overlap between k-mers from the reference squiggle and the sample squiggles should distinguish which squiggle is from the reference and which is not.

Table 2.1: Basic characteristic of the used datasets

Name	# reads	mean read length (nucleotides)	median read length (nucleotides)	mean squiggle length (readouts)	median squiggle length (readouts)
sapIng	3 000	14 663	6 965	151 002	71 964
sapFun	3 000	10 912	4 775	113 550	51 266

2.5 Experimental Evaluation

In our experiments, we want to test:

- How accurately the discretization method that we proposed in Section 2.1 represents the squiggle.
- How the proposed identification from Section 2.4 works on real data

In both of these cases we are interested in the results for the different set of parameters such as the number of levels, length of the k-mers and impact of the methods that we discussed in Section 2.3.

2.5.1 Experimental Data

Data that we use for these experiments come from two organisms. One is *Saprochaete ingens* (dataset sapIng) and the other one is *Saprochaete fungicola* (dataset sapFun). These organisms were sequenced using R9.4.1 MinION flow cells at the Department of Biochemistry of the Faculty of Natural Sciences. We have also obtained the reference sequences of both of these organisms that were previously published [7] [6]. We basecalled the raw squiggles using the Albacore basecaller provided by Oxford Nanopore Technologies, company that also develops the MinION sequencer.

To see if these two organism do not share some big similarities, we align the reads from *Saprochaete fungicola* to the reference sequence of *Saprochaete ingens* using the Minimap2 algorithm. We run this test on the 500 randomly selected reads from each of these organism. We studied the ratio of the reads that were successfully matched in the reference sequence. Figure 2.6 shows the number of reads that were alligned with the respective percentage of their length that was successfully aligned. Observe that only 12 reads out of 500 in case of the *Saprochaete fungicola* had more than 50% of their length aligned.

For our experiment, we use only the squiggles from the *Saprochaete ingens* if they are successfully aligned into the reference using the Nadavca. We then create the

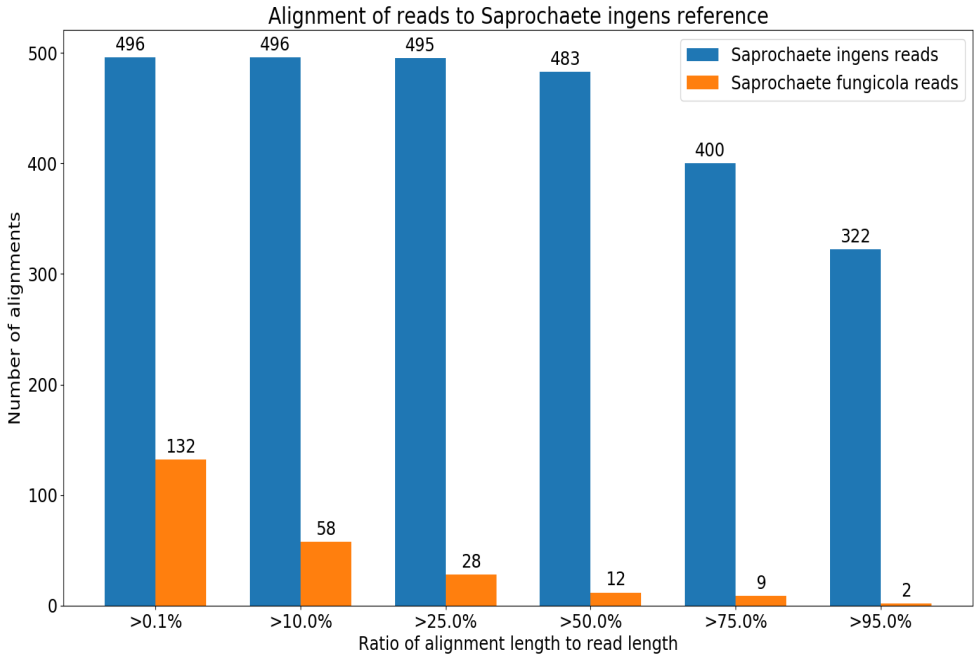

Figure 2.6: Number of reads that aligned at least on $x\%$ of their length to the reference DNA of the *Saprochaete ingens*

simulated reference signal from the reference part that Nadavca identified and pick some random read from the sequencing of the *Saprochaete fungicola* and run our analysis. We run this cycle many times over. As the ultimite goal of our work is to achieve the ability of selective sequencing, we must also restrict ourselves to only work with the beginning part of the squiggle. We reflect this in our experiments so we work only with the first 5 000 readouts from the squiggle segment that Nadavca identified as matching to our reference.

2.5.2 ROC curve

A receiver operating characteristic curve is a graphical illustration of how well the *binary classificator* performs on the data for a particular threshold. The binary classifier system is a system that tries to predict the binary output from the inputs. We can look at our experiment as predictor that predicts based on the number of hits, if the squiggle is from the simulated reference squiggle or not. We can choose some threshold value t and say that our classificator will predict all the squiggles with the number of hits bigger or equal than t as coming from the reference squiggle. When working with binary predictor there are two important rates *true positive rate* TPR and *false positive rate* FPR.

TPR is a ratio of the correctly classified positive squiggles to the number of the all

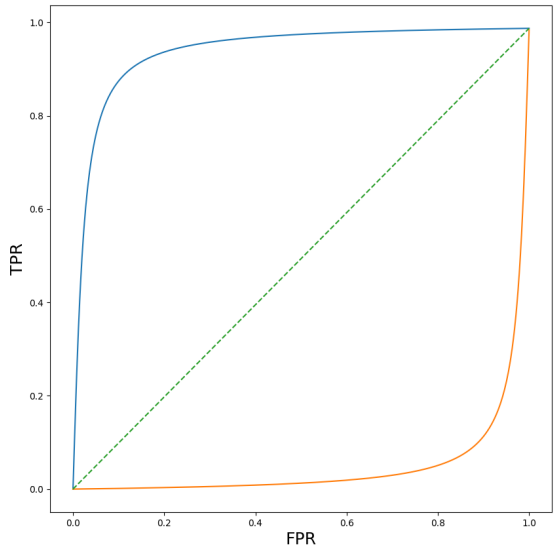

Figure 2.7: In this picture we see the example of a ROC curve. Blue line represents very good binary estimator. Random binary estimator is expected to have his ROC curve somewhere around the green dashed curve.

squiggles.

FPR is a ratio of the squiggles that we incorrectly classified as a belonging to the reference squiggle.

The ROC is curve that represents the dependency of TPR on FPR for various threshold values t. If the binary classificator was random generator the ROC curve would be roughly linear. Very good classificator obtains the curve that is very close to the point $FPR = 0, TPR = 1$. We can see the example of a curve that represents very good binary classificator on Figure 2.7. The binary classificator that randomly guesses his output is expected to have an ROC curve somewhere around the green line.

2.5.3 Level string alignment

To see how two level strings compare to each other we will align them. The input to the alignment algorithm are two strings s_1, s_2: $s_1 = a_1a_2 \cdots a_n$, $s_2 = b_1b_2 \cdots b_m$. The alignment are strings s_3, $s_4 : s_3 = c_1c_2c_3 \cdots c_k$, $s_4 = d_1d_2d_3 \cdots d_k$ such that:

1. s_3 and s_4 was created from s_1, s_2 respectively by inserting dashes
2. $c_i \neq d_i$ then $c_i = - \vee d_i = -$

Example of allignment of $s_1 = AAACTGC$ and $s_2 = AACGTC$ is:

$$s_3 = \text{AAAC - TGC}$$
$$s_4 = \text{AA - CGT - C}$$

There are a lot of alignments between the two strings. Most of the time we are interested in some particular alignment. For this purpose, we will create a scoring system and we will try to find the maximal (minimal) alignment in this scoring system. We can, for example, minimize the number of dashes. Using the alignment defined this way we can see how two-level strings compare to each other and how similar they are. Finding the best alignment is not trivial. We use dynamic programming for solving this problem. Let $T[i][j]$ be the best alignment of string $s_{1_i} = a_1 a_2 \cdots a_i$ and $s_{2_j} = b_1 b_2 \cdots b_j$. We will for simplification say that $\forall i, j : T[i][0] = i, T[0][j] = j$.

$$T[i][j] = \begin{cases} 0, & \forall i, j : i \leq 0 \vee j \leq 0 \\ max(T[i-1][j-1] + 1, T[i-1][j], T[i][j-1]) & \text{only if } a_i = b_i \\ max(T[i-1][j], T[i][j-1]) & \text{otherwise} \end{cases}$$

2.5.4 Results

We performed three experiments that relate to the goals that we set at the start of this Chapter. We use the squiggles from the datasets introduced in Section 2.5.1. We stated that the selective sequencing must be done using the information from the start of the squiggle. In this all three experiments, we always use the 5 000 readouts from the any squiggle mentioned.

The first experiment evaluates the number of shared k-mers between the simulated reference squiggle and corresponding real squiggle. To put the number of shared k-mers into perspective, we also look at the number of shared k-mers between the same simulated reference and an unrelated squiggle. For easier description, we label the simulated reference squiggle the reference, real squiggle that corresponds to this reference the positive squiggle and the random unrelated squiggle the negative squiggle. In all individual test cases, we calculate the ratio of the number of hits in negative squiggle to the number of hits in the positive squiggle. The ideal case is that most of the test cases will have the ratio as small as possible. If the number of hits of the positive squiggle is 0 or the ratio exceeds 2:1, we mark this ratio as a 2:1.

Figure 2.8 shows the results of the experiment using all smoothing methods introduced in Section 2.3. There are parameters that obtain the ratio at most 1/2 in more than 80% of test cases (for example 4 levels, 28 k-mer length). There is phenomenon of the lines that start with the perfect record and then quickly go to the right part of the graph (for example 9 levels, 28 k-mer length). This happenes to the curves representing the high k-mer number. That is because the number of hits in random squiggle is usually zero for high k-mer number but it is also hard for the real squiggles to obtain high number of hits so for a lot of positive squiggles, the number of hits is 0 which shifts the curve to the right quickly. For comparison, Figure 2.9 shows the

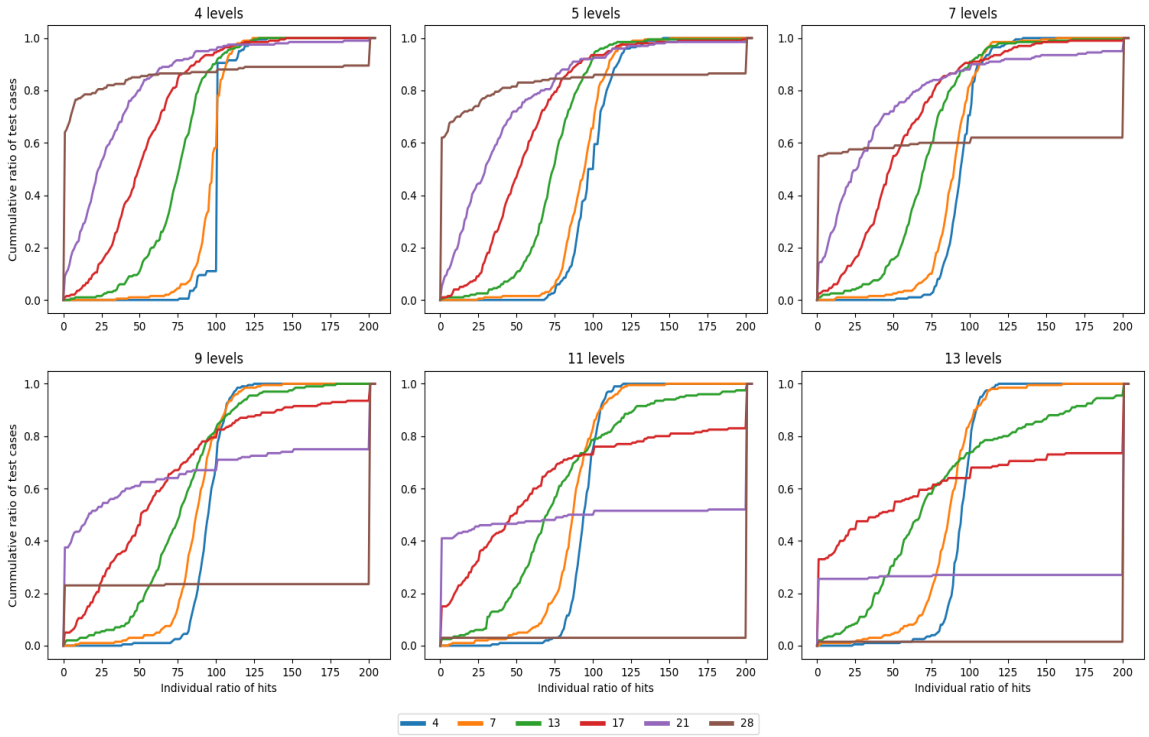

Figure 2.8: Cummulative distribution of ratios of hits in positive and negative squiggles, smoothed. The y axis shows cummulative count of the ratios up to the ratio x. Individual figures represent results for the different number of levels. Individual curves represent different length of k-mer.

results for the squiggles not processed by our smoothing techniques. We can clearly see that with no smoothing there are no parameters usable for the effective discrimination between positive and negative squiggles.

The second experiment is motivated by a hidden drawback of the first experiment. If two positive squiggles have the bigger number of hits than their two negative counterparts in their respective test cases, we consider it as a success in the first experiment. However, there can be problem that even if the both of these test cases are considered successful the number of hits of the negative squiggle in the second test case can be bigger than the number of hits of the positive squiggle in the first testcase. This will cause that even if we have two successful, test cases in the first scenario, we cannot predict if the squiggle is from the reference based on some threshold number of hits. This can happen for example, if a lot of squiggles have considerably lower number of hits due to the discretization algorithm. The second experiment examine how often this happens and if the number of hits in positive squiggles is globally larger than the number of hits in the negative squiggles. We will again take the reference squiggle, positive squiggle that comes from the reference segment and a randomly chosen negative squiggle from the other sequencing. We plot the data using the ROC curve. Note

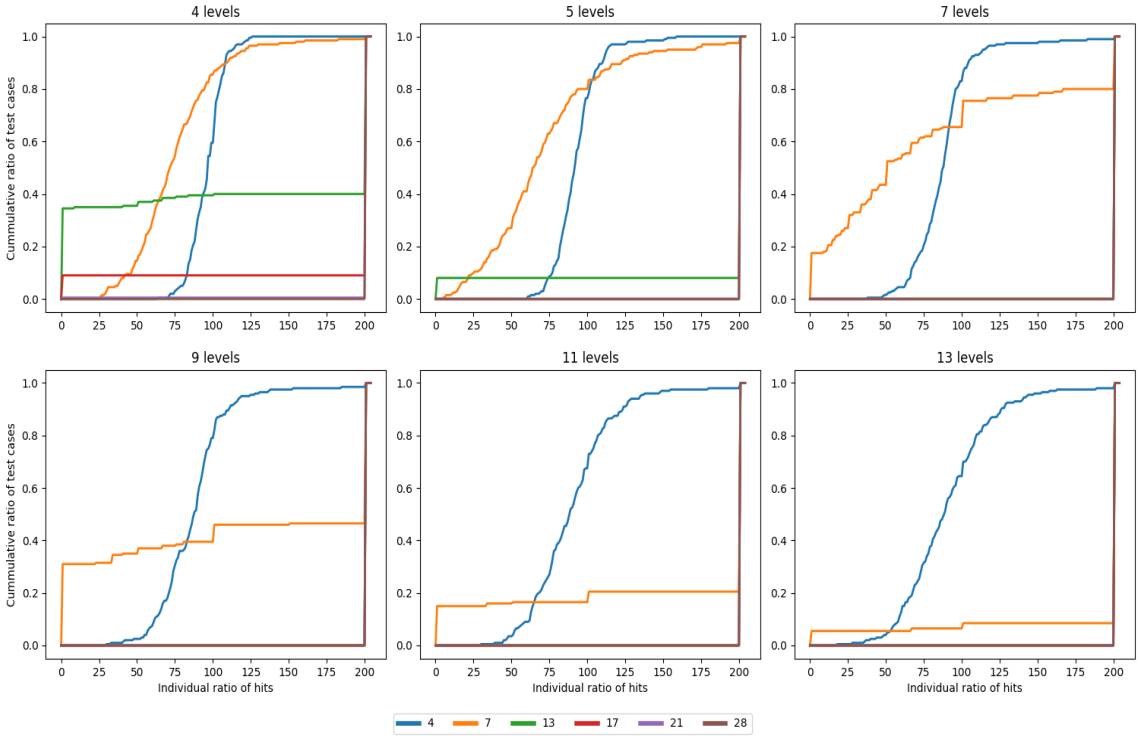

Figure 2.9: Cummulative distribution of ratios of hits in positive and negative squiggles, unsmoothed. The y axis shows cummulative count of the ratios up to the ratio x. Individual figures represent results for the different number of levels. Individual curves represent different length of k-mer.

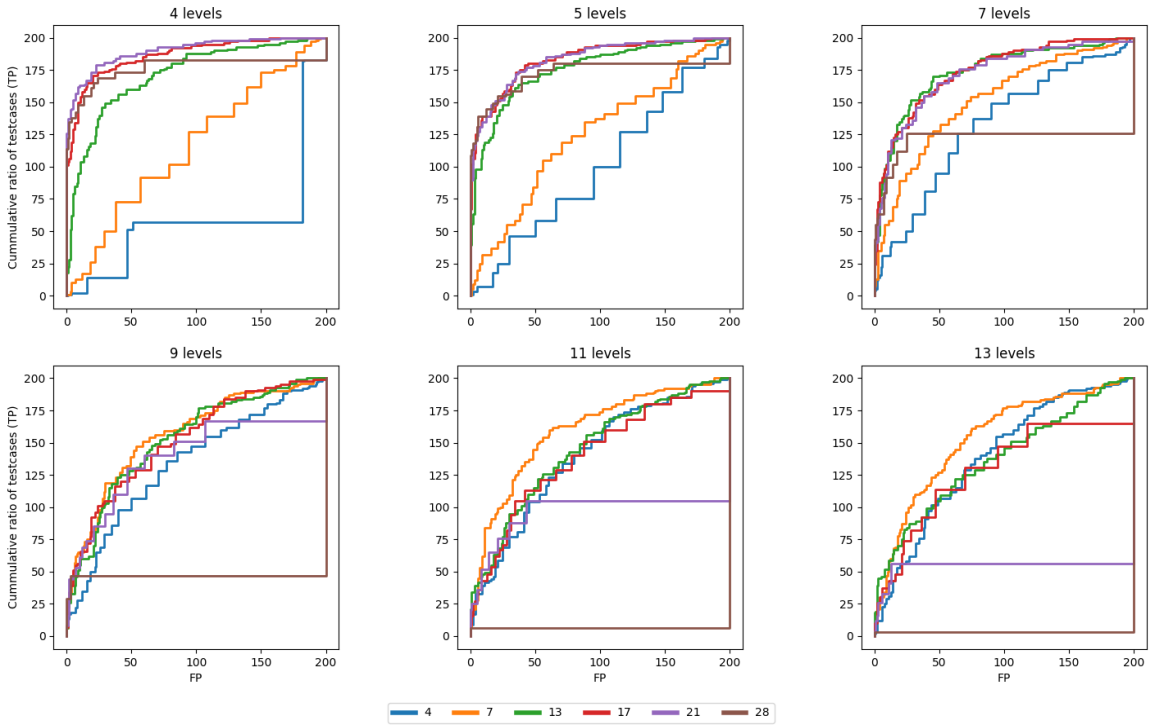

Figure 2.10: ROC curve of the hits with smoothing

that if two squiggles have the same number of hits, we favour the negative squiggle to present objective results.

The ROC curves in Figure 2.10 represent how overall number of hits in the positive squiggles compares to the number of hits of the negative squiggles. We can see very good results for low numbers of levels and higher length of k-mers (4 levels, k-mer length ≥ 17). Figure 2.11 shows that without smoothing techniques applied, only the lower lengths of k-mers (k-mer length ≥ 17) have meaningful results but the results with the use of smoothing techniques are much more successful. This is the case because it is unlikely for the unsmoothed squiggle to obtain the longer match due to the reasons outlined in Section 2.3.

The third experiment will focus on mutual alignment of the level strings coming from the real squiggle and the reference squiggle respectively. Again, as in previous experiments, we add randomly chosen squiggles to highlight the differences between alignments. We align the level strings using the scoring scheme presented in Section 2.5.3, minimizing the number of the gaps. While in previous experiments we focused on the shared k-mer counts, such measure does not take into account the relative positions of matched pairs. This means that we can easily count as the matching pair also two k-mers that do not correspond to the same signal but happened to be in the same level string. For a reasonable number of levels, this is unlikely to happen often for a short read. The big advantage of the alignment of the level strings is that it

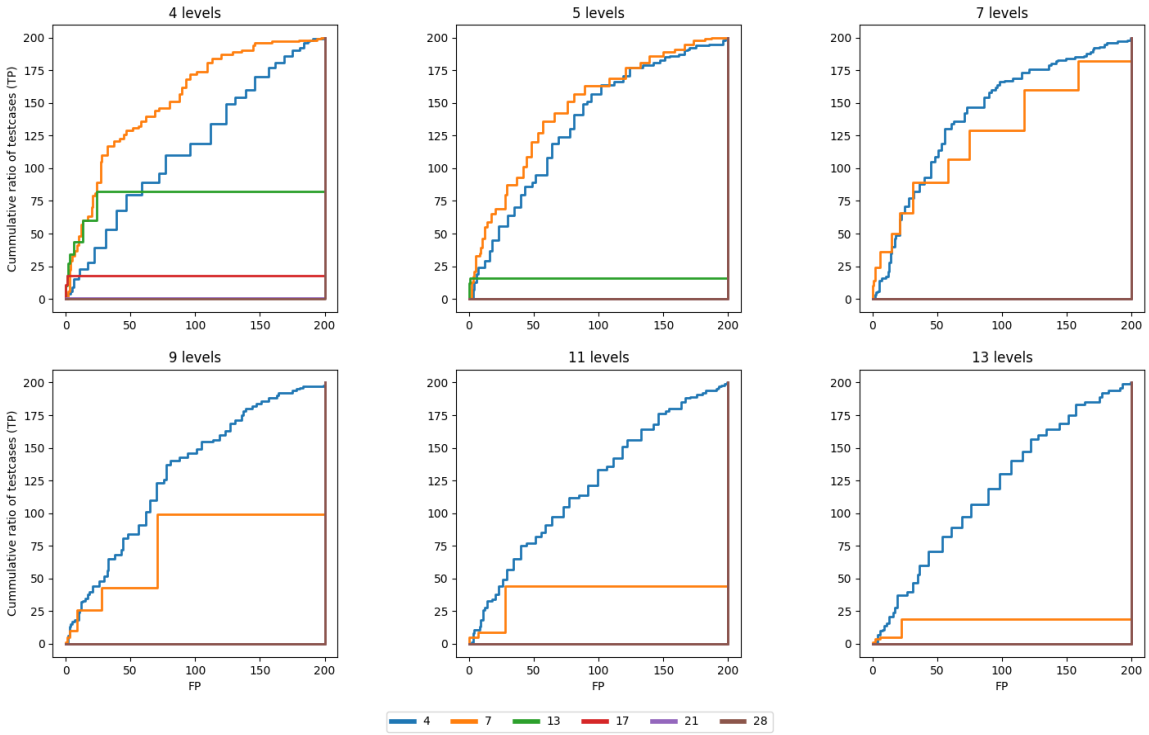

Figure 2.11: ROC curve of the hits without smoothing

tells us more accurately what is the overall similarity of the level strings. We found, that the distribution of lengths of the successive gaps in the alignments (gap lengths) particularly highlights the difference between positive and negative squiggles.

We split the gaps into two groups, those with gap length shorter than 5 (short gaps) and those longer than 8 (long gaps). For each alignment, we count the number of gaps in these two groups.

Figure 2.12 shows the ratio between the number of the gaps of given length divided by the length of the level string for both positive and negative squiggles. For better visualization, we show number of short gaps per 100 level string characters and number of long gaps per 1 000 level string characters. We can see that the number of short gaps favors a lower number of levels and gradually increases with the number of levels. The number of long gaps behaves in the opposite way. As the number of levels is increasing, the number of gaps in the negative reads rises faster that the number of the gaps in the positive reads. The longer gaps are better indicator of the alignment of a bad squiggle as it is common for both positive and negative alignments that there are small gaps due to the noise in the underlying signal. Most of the time, the large gaps are a sign of the two unrelated level strings aligned or very uncommon unsimilarity between positive read and its reference. Figure 2.13 presents results for the unsmoothed squiggles. We can see that for unsmoothed squiggles, there are more larger gaps even in the case of the positive squiggles and the differences between the positive and negative squiggles

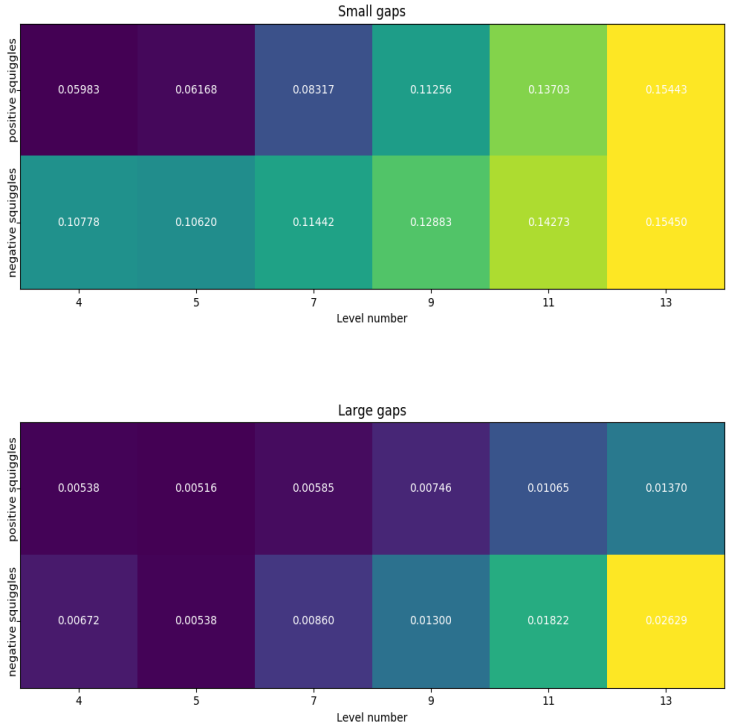

Figure 2.12: (top): The number of short gaps per 100 level string characters of smoothed signal.
(bottom): The number of long gaps per 1 000 level string characters of for smoothed signal.

are much less pronounced.

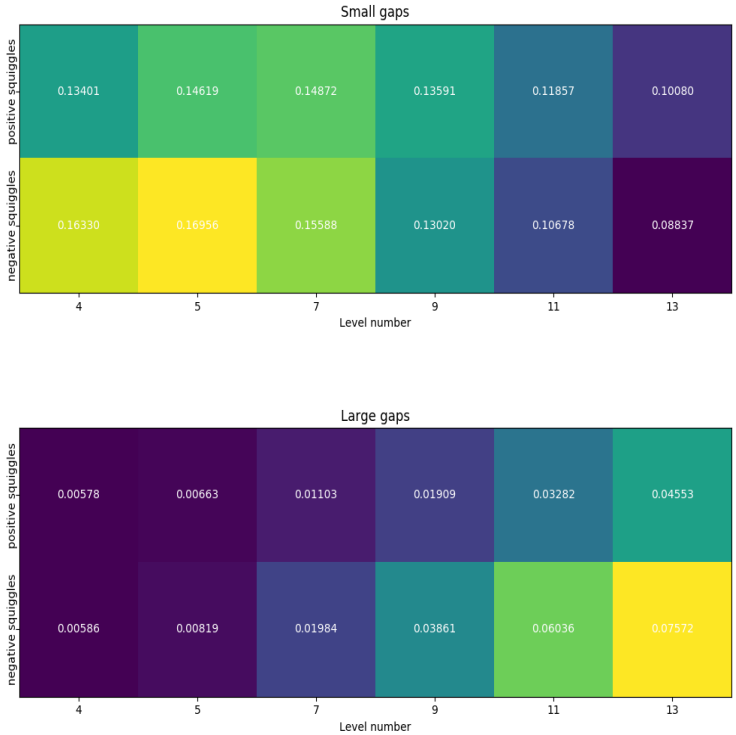

Figure 2.13: (top): The number of short gaps per 100 level string characters of unsmoothed signal.
(bottom): The number of long gaps per 1 000 level string characters of for unsmoothed signal.

Chapter 3

Towards Building a Squiggle Index

In this chapter, we use the results and findings from the previous chapter to build the index data structure that can find the query squiggle in the simulated reference squiggle.

3.1 Building a Squiggle Index

As mentioned in Section 1.3, as an input to the selective sequencing we are usually only provided the reference DNA sequence. The reference DNA sequence is transformed to a simulated signal using the process introduced in Section 2.2. We choose the number of the windows w as a parameter of our discretization algorithm and create the reference level string. Then, when the real signal from the squiggle arrives, we perform the same process on the real signal. Now, we have one reference level string and one query level string which we want to find in the reference. As we see in Section 2.5.4, there is only a small probability that the level string of the longer squiggle will match perfectly to some area in the reference level string. Instead of looking for the exact match in the reference, we will cut the reference level string into overlapping subsequences of length k and put them into a hash table. Hash table is a data structure that allows insertion and search of an element in amortized time complexity $O(1)$. This hash table will now serve as a reference index. To process a squiggle, we build its level string and cut it into the overlapping subsequences of the same length k. As the next step, we compute how many of these subsequences can be found in the index. We call all the subsequences of length k from our query squiggle that are present in the prepared hash table *hits*. Our initial assumption is that the number of hits will be considerably higher for the squiggle that belongs to the reference.

To evaluate this idea experimentally we have chosen one *contig* (contig3) from the *sapIng* reference sequence. Contig is a standalone DNA sequence, most of the time representing a single chromosome of the organism. It is smaller than the whole reference

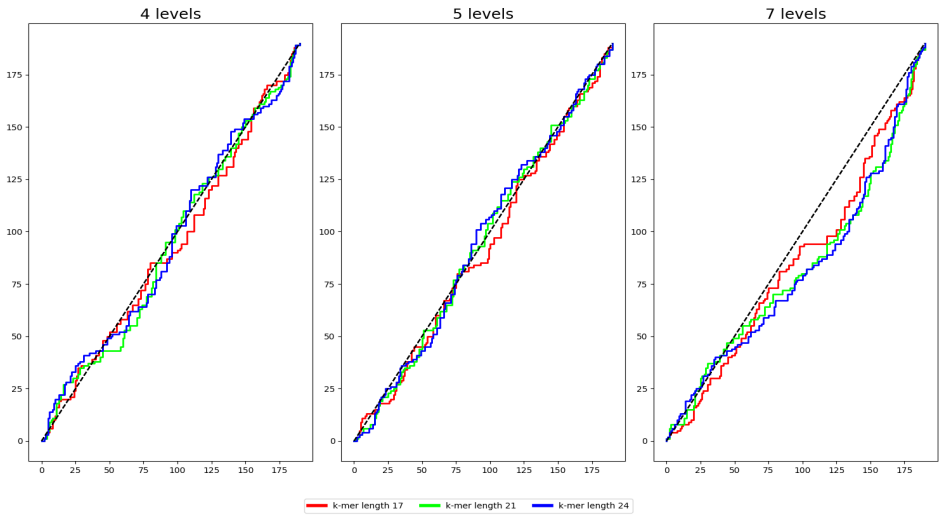

Figure 3.1: ROC curve describing how good predictor is our index. Its decisions are based on the number of hits of the respective squiggle

which consists typically of multiple contigs. We will build the index over the signal simulated from this reference sequence. We will predict, based on the number of hits in our index if the read is from the reference contig or not. We build the index in the slightly different way than we described previously. At first, we will simulate the signal from the whole contig creating one, very long simulated contig squiggle. Then we use the sliding window of size 5 000 with the window step equal to 3 000. For every of this small windows, we will normalize the signal that is contained in it, smooth it using the techniques we described and then create the level string from this signal. We will cut this smaller level string into overlapping k-mers that we insert into the hash table. We will be working with the 2 000 readouts from the query squiggles. If this query squiggle matches simulated signal somewhere, it will be covered by one of our windows.

Beside the reference we have two sets of reads, one containing reads from the sapIng dataset and the other one containing reads from the *sapFun* dataset. Both of these datasets are described in Section 2.5.1. We call the reads from the first set the positive reads and the other negative reads. We use 190 reads from the both datasets. From the whole sapIng dataset, we only choose the reads that aligned to the contig3 and also have at least 95% of their length aligned. We also make sure that the reads from the sapFun dataset have 0 alignments of at least 90% of their length. We are interested in finding if our index can work as a good predictor. It will be deciding if the squiggle is from the reference based only on the number of hits. Figure 3.1 shows what is the ROC curve of our index as a classificator.

We see that our index is not performing very well. After looking at our index, we can identify one of the problems. We see that some of the k-mers have very big

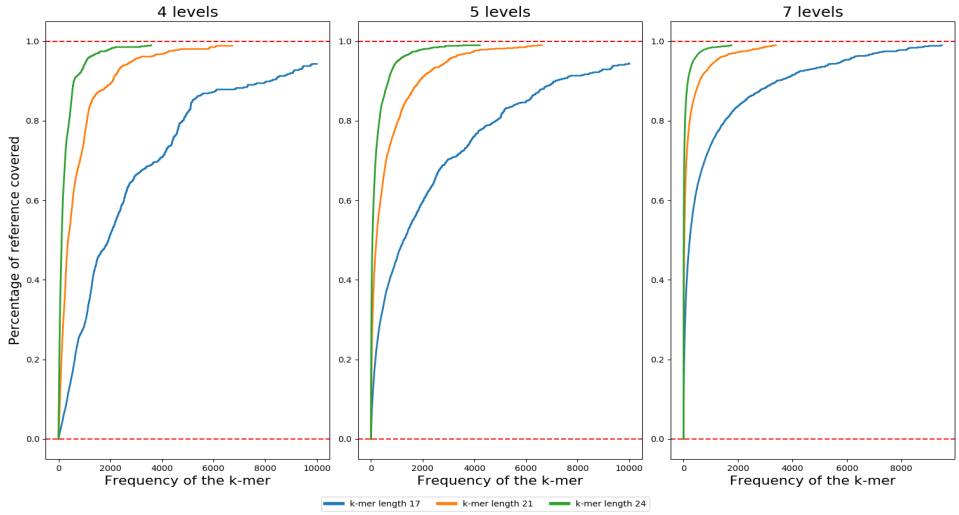

Figure 3.2: On y axis is the percentage of the reference covered by the k-mers up to specific frequency denoted on the x-axis.

number of occurences in the indexed reference level string. We suspect that these k-mers are not that informative because they are very common and they carry more information about the specific properties of the level string rather than information about the underlying signal. Figure 3.2 shows what part of our reference is covered by the k-mers up to a certain frequency. We can see that a lot of k-mers are in our reference level string more than 2 000 times.

We will use the data from the Figure 3.2 to try to remove the most frequent k-mers. We will remove that part of the most frequent k-mers such that at least 50% of our reference is still covered by the remaining reads. We also call this process *cut-off*.

We see in Figure 3.3 that this did not helped at all. This can be because with the most frequent k-mers we also removed a lot of so called fake hits. This could be hits that amounted to the big percentage of the previous hits so the overall number of the hits in our reference decreased because of this fact.

3.2 Aligning the squiggle level string

After failing to build the index straightly, we will look at an easier experiment that can hint us if we are going in the good direction. We would want to see how our discretization suits this usage and how similar is level string of the whole squiggle comparing it to the whole reference level string. For this purpose, we will try to find our query level string in the reference level string.

We already know that it does not make sense to look for the exact match of the query string in the reference so we will need some string searching technique that

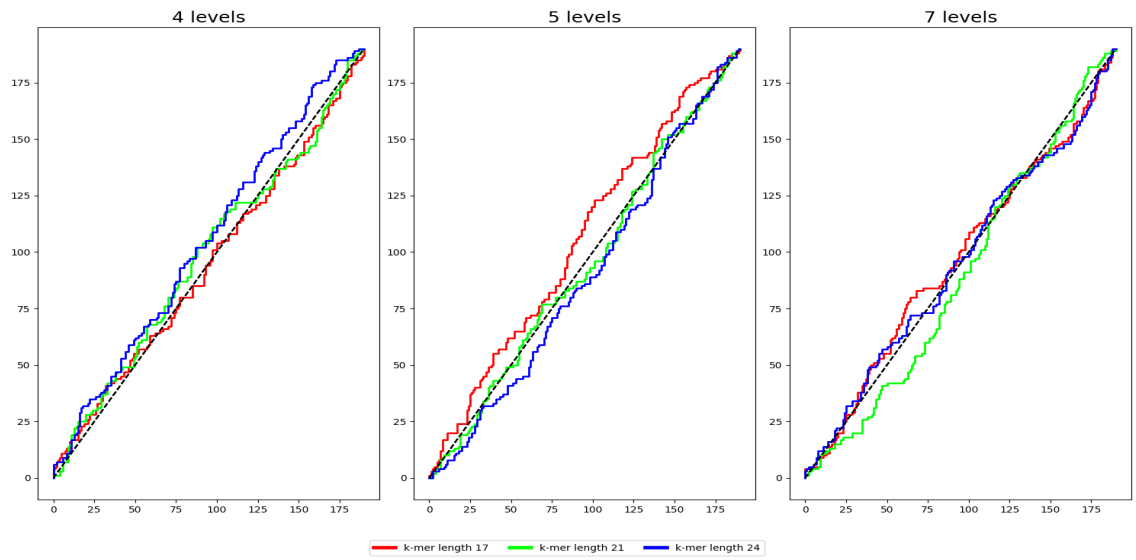

Figure 3.3: ROC curve describing how good predictor our index is after the 50% cut-off. We can see the information for the numerous combinations of levels and kmer lengths.

can deal with few errors. Our alignment algorithm presented in Section 2.5.3 could be adapted but it lacks the neccessary speed. This algorithm works in the time complexity $O(r \cdot s)$ where r is the length of the reference sequence and q is the length of the query string. Many algorithms are able to do this fast and with very high accuracy. We decided to tweak a Minimap2 [9] algorithm. This is one of the most popular DNA sequence aligning algorithms. This algorithm can take the query DNA sequence and find it in a long reference DNA sequence. It is also able to find the query DNA string even if it does not exactly match any subsequence precisely. Hovewer, this algorithm works only with the DNA sequences. What we will do is that we choose the number of levels $w = 4$. This will cause that the reference and query level strings will both consist of the characters 'a', 'b', 'c', 'd'. We then substitute 'a', 'b', 'c', 'd' by 'A', 'C', 'G', 'T' subsequently. We now obtained the manipulated level strings that are represented using the DNA sequences. Now, we can use the Minimap2 algorithm for finding our manipulated query level string in the manipulated reference level string. As the Minimap2 is most of the time used and also optimized to perform well on DNA sequences, we need to be careful of some catches that come with using it for this purpose. For example, we need to ignore the hits on the reverse strand as the reverse strand does not carry the same information as in the real DNA sequence. This also speeds up the whole algorithm considerably.

We now take the level string of the relatively big part of the squiggle. We take the fixed part of the squiggle from the 5 000th readout to the 60 000th. Of the 50 tested squiggles, all of them were correctly aligned to the correct contig and also correct position. This was a very promising result, so we decided to optimize a Minimap2

algorithm with adjusting some hyperparameters that this algorithm allows us to change to speed it up. For this solution, to be usable, we need to take only a shorter part from the start of the squiggle. With signal from the 5 000th readout to the 10 000th we have been able to find only the 31 out of 100 squiggles in around 1 minute.

3.3 Alignment Algorithms Inspiration

The second approach that we tried is similar to how the alignment algorithms like Minimap2 work. The first step of these algorithms is to find the pairs of the shorter exactly matching sequences. These are then used as the kernels of the alignment. We want to use kernels as the starting points, to see where are the places that could lead to the successful matching of the query level strings.

In order to try this approach we need to know what is the ideal length of this short sequence. We need to find the maximum length of the k-mer for the particular level such that the level strings of most of the squiggles will have at least one k-mer of this length common with their corresponding simulated reference level string. As an experiment, we will take 200 random reads from the sapIng dataset. We run the experiment on the data used in the previous experiments in Section 2.5.4. Now, we will again work with the smaller number of 2 000 readouts so we simulate the conditions during the real DNA sequencing. In the table 3.1 we can see the number of the squiggles with at least one shared k-mer with their reference level string. We bring the values for the multiple levels. Based on this table, we can expect what length of k-mer for the particular number of levels will be located in the reference at least one time. We want to find the good tradeof between number of squiggles that we will be able to find and the length of the k-mer which determines the number of hits in the index we have to investigate.

Now, when we have for all the levels the particular number of k that has the minimal number of hits for most of the reads we can proceed to use this information. We will focus on finding the pairs of level number - k-mer length such that the number of squiggles with at least one hit is around 175. This will not eliminate a lot of reads and it will give us an edge over the fake hits in the big reference sequence as we favor the biggest k-mer length possible. We now take the entire reference of the sapIng dataset and 200 reads from this reference. We will want to know, what is the mean and median number of hits of these squiggles in the entire reference and what is the number of the hits between the squiggle and its corresponding simulated reference signal. We need to push the number of k-mers for the particular number of levels as high as possible such that the large percentage of reads still share at least some k-mers with their simulated squiggle but the overal number of hits in the whole reference is smallest possible. What

Table 3.1: The number of squiggles with at least one shared k-mer with their corresponding reference for particular number of levels. The total number of tested reads is 200

level number	k-mer lengths									
4	16	17	18	19	20	21	22	23	24	25
	200	199	198	196	195	186	180	174	163	148
5	17	18	19	20	21	22	23	24	25	26
	200	199	199	197	190	184	176	163	158	148
7	15	16	17	18	19	20	21	22	23	24
	200	199	198	195	183	167	157	132	119	100
9	13	14	15	16	17	18	19	20	21	22
	200	199	191	184	160	134	108	86	71	49
11	12	13	14	15	16	17	18	19	20	21
	200	199	192	175	153	124	95	72	54	41
13	11	12	13	14	15	16	17	18	19	20
	200	196	190	176	152	122	82	63	44	28

happens if we do not push the k-mer length enough is that we receive a lot of false hits in the entire reference. If we push too much, we will lose even the good hits and the results will be some random hits in the entire reference that represent very similar signal but not our target signal.

We can see in Table 3.2 that the number of hits in the whole reference is still really high. What we can do is that we cut the reference into the smaller sequences and track the number of hits in these individual sequences. We can focus on the sequences with the high number of hits and then do there some post-processing to remove as many hits as possible.

Table 3.2: The number of hits in the whole reference vs hits in the simulated squiggle corresponding to the real squiggle

levels	k-mer	mean hits in ref	median hits in ref	mean hits in simulated	median hits in simulated
4	22	223446.62	193751.0	16.28	12.0
4	23	153540.78	132338.0	13.1	8.0
5	21	401133.12	363887.5	23.32	19.5
5	22	275380.16	241922.5	18.26	14.5
5	23	192186.58	158792.5	14.54	9.0
7	20	271556.42	265182.0	17.06	14.5
7	21	175926.02	169163.0	12.88	10.5
9	18	213360.74	191186.0	12.54	10.5
9	19	127401.82	107961.5	8.58	7.0
9	20	76321.34	58724.0	5.96	5.0
11	16	243431.82	231441.5	11.16	8.5
11	17	132439.22	126377.0	6.56	4.0
11	18	72565.3	63293.0	3.78	2.0
13	14	371994.38	326821.0	16.14	14.0
13	15	194185.4	157827.5	10.18	8.0

Conclusion and future work

The goal of this work was to build the reliable index that will satisfy the needs of the selective sequencing. We wanted to build this index on top of the discretizing algorithm that would help us to work with the raw squiggles more easily and also very fast.

In the Second Chapter we come up with a discretizing algorithm that have been able to build representation that reliably matches the underlying raw squiggle. We proved many good properties of this representation in the Results section.

We have been unable to build the reliable yet fast enough index but we obtained some hints on how the problems that arised could be tackled in the future. Most notably, we have been able to tweak Minimap2 algorithm and use it to align our discretized form of the signal to the correct position in the reference. This leads us to the conclusion that it is possible to use this discretizing algorithm for furher progress in this area.

Bibliography

[1] *Genetics*, Retrieved May 30, 2020. `https://en.wikipedia.org/wiki/Genetics`.

[2] *MinION*, Retrieved May 30, 2020. `https://product.statnano.com/product/11661/minion-sequencer`.

[3] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[4] Hannah Ashworth. *How long is your DNA?*, Retrieved May 30, 2020. `https://www.sciencefocus.com/the-human-body/how-long-is-your-dna/`.

[5] Eduard Batmedijn. Nanopore data variant caller, 2015.

[6] Broňa Brejová, Hana Lichancová, Viktória Hodorová, Martina Neboháčová, L'ubomír Tomáška, Tomáš Vinař, and Jozef Nosek. Genome sequence of an arthroconidial yeast, saprochaete fungicola cbs 625.85. *Microbiol Resour Announc*, 8(15):e00092–19, 2019.

[7] Viktória Hodorová, Hana Lichancová, Stanislav Zubenko, Karolina Sienkiewicz, Sarah Mae U Penir, Philipp Afanasyev, Dominic Boceck, Sarah Bonnin, Siras Hakobyan, Urszula Smyczynska, et al. Genome sequence of the yeast saprochaete ingens cbs 517.90. *Microbiology Resource Announcements*, 8(50), 2019.

[8] National Human Genome Research Institute. *ACGT*, Retrieved May 30, 2020. `https://www.genome.gov/genetics-glossary/acgt`.

[9] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[10] Yu Li, Renmin Han, Chongwei Bi, Mo Li, Sheng Wang, and Xin Gao. Deepsimulator: a deep simulator for nanopore sequencing. *Bioinformatics*, 34(17):2899–2908, 2018.

[11] Nicholas J Loman and Aaron R Quinlan. Poretools: a toolkit for analyzing nanopore sequence data. *Bioinformatics*, 30(23):3399–3401, 2014.

[12] Matthew Loose, Sunir Malla, and Michael Stout. Real-time selective sequencing using nanopore technology. *Nature methods*, 13(9):751, 2016.

[13] Hengyun Lu, Francesca Giordano, and Zemin Ning. Oxford nanopore minion sequencing and genome assembly. *Genomics, proteomics & bioinformatics*, 14(5):265–279, 2016.

[14] John R Tyson, Nigel J O'Neil, Miten Jain, Hugh E Olsen, Philip Hieter, and Terrance P Snutch. Minion-based long-read sequencing and assembly extends the caenorhabditis elegans reference genome. *Genome research*, 28(2):266–274, 2018.

[15] Ryan R Wick, Louise M Judd, and Kathryn E Holt. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome biology*, 20(1):129, 2019.

[16] Marketa J. Zvelebil and Jeremy O. Baum. *Understanding bioinformatics*. Garland Science, 2008.

Appendix A: Implementation of the Experiments

All the source codes for the individual experiments as well as the most important part of the test data are available in the included storage media. The code repository for this work is hosted on GitHub and can be accessed through this link.