CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	DNA base-calling from Nanopore sequencing data
Student:	Bc. Tomáš Horák
Supervisor:	Mgr. Petr Daněček, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2018/19

Instructions

The nanopore sequencing is a rapidly developing field DNA sequencing technology, where an ionic current is measured as a DNA molecule passes through a nanopore. Changes in the ionic current are indicative of nucleotide composition of the molecule.

Get familiar with basic concepts of DNA sequencing technologies. Study in detail the emerging technology of Nanopore sequencing. Discuss, design, and implement an algorithmic method for processing the ionic currents data and determining of the DNA sequence composition. The method should be implemented as an open source software. Compare its accuracy with other software implementations of nanopore sequencing methods.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.	prof. Ing. Pavel Tvrdík, CSc.
Head of Department	Dean

Prague March 2, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of Knowledge Engineering

Master's thesis

DNA base-calling from Nanopore sequencing data

Bc. Tomáš Horák

Supervisor: Mgr. Petr Daněček, Ph.D.

27th June 2017

Acknowledgements

I would like to thank my supervisor Petr Daněček for support needed to finish this thesis. The credit also goes to everyone involved in providing the real world testing data.

Declaration

In Prague on 27th June 2017 . . . .

Citation of this thesis

Horák, Tomáš. *DNA base-calling from Nanopore sequencing data.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Určování DNA sekvencí z Nanopore dat se v součastnosti velmi rychle vyvíjí. Od představení zařízení MinION™ je možné se soustředit na část zvanou určování bází. Tato práce implementuje snadno použitelný nástroj, který k tomuto účelu využíva Viterbiho a Forward–Backward algoritmy. Přesnost daného řešení je porovnatelná s již existujícími nástroji jako Nanocall a DeepNano. Data použitá k testování byla získána pomocí R9 chemie z E.coli DNA molekul.

Klíčová slova DNA, Určování bází DNA, Algoritmus Forward–Backward, Skrytý Markovův Model, Nanopore, Zarovnání sekvencí, Viterbiho algoritmus

Abstract

MinION Nanopore sequencing technology is a novel approach to DNA sequencing which allows to sequence individual molecules in real time and can generate continuous sequences 100 $\times$ longer than the existing short-read technologies. One of the downsides of this emerging technology is much less accurate base-calling, the process of determining the actual DNA sequence from raw data. In this work an open-source implementation of a base-caller is presented. It is based on Hidden Markov Models (HMM) and implements Viterbi and Forward-Backward algorithms. The accuracy is evaluated on E.Coli data and compared to the existing programs.

Keywords DNA, DNA Base-calling, Forward–Backward Algorithm, Hidden Markov Model, Nanopore, Sequence Alignment, Viterbi Algorithm

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

In the recent years a significant progress has been made in DNA sequencing. The first practical method of sequencing invented by Frederick Sanger in 1977 was in the past 15 years replaced by the second-generation high-throughput methods. The most successful and most widely used today became sequencing by synthesis (Illumina) which can read billions DNA bases with high accuracy in the matter of days. The main disadvantage is that the generated sequence comes in short reads (typically around 150 bases) that originate from random places of the genome. Because biological sequence can be highly repetitive (mammalian genome is 40 % repetitive, plant genomes 80 %), the assembly of the genome from short reads is a very difficult problem. Also the massively parallel approaches sequence ensemble of sequences, rather than individual molecules.

The MinION Nanopore technology allows to sequence tens of thousands bases of a single molecule in real time with a relatively low cost. However, the accuracy of base-calling is much lower than for Illumina data. This is mostly due to the low resolution caused by the rapid movement of the strand through the Nanopore.

The main idea of this method is based on an ability to drive a single DNA strand through a biological pore. It is possible to measure ionic current which is released by the molecules in the actual part of the DNA sequence present in the pore. So, the input data for base-calling is a relatively simple sequence of numbers which depend mostly on the current context of the Nanopore. A hindrance to the process of base-calling is however the fact, that the current is measured periodically but the move of DNA strand seems rather stochastic as it is determined by the chemistry involved. This is addressed by measuring the current with higher frequency, which leads to the need of segmentation of the raw data. Another problem is caused by the unrepeatable nature of the measuring. Even the same DNA sequence measured by the exactly same device will not yield the same results as some conditions changed.

One of the first methods used to perform base-calling using Nanopore data utilized Hidden Markov Models and specifically Viterbi algorithm. HMM uses a set of hidden states, in this case pore contexts, and visible observations, here the measured ionic currents, to predict the probability of parts of the measured sequence. Viterbi Algorithm is designated to recreate the whole sequence, so it is an obvious choice for base-calling. Another widely used algorithm operating with HMM is a Forward–Backward Algorithm, which has a different purpose however.

Results of the base-calling are then compared with the reference data. This is accomplished by designated alignment algorithms. Used alignments include Smith–Waterman Algorithm, Needleman–Wunsch Algorithm and a modified computation of Levenshtein distance.

In summary, the aim of the thesis is to write a lightweight open source tool for base calling from MinION Nanopore data. Other necessary modules will be implemented as well, which among others include the alignment module and a script which can scale the input parameters for the actual base-calling. As there already exist some tools for base-calling like Metrichor or Nanocall, the goal is to provide an alternative open source, easy to use and low demand tool, which is usable on any machine.

To accomplish this goal it is necessary to thoroughly analyze given problem which is presented in the chapter 1. Chapter 2 discusses the implemented solution and its modifications, and also provides an in depth analysis of measured results with an included comparison to the State-of-the-Art solutions.

Chapter 1

Methods

The focus of this chapter is on the theoretical foundations which will then be used in the realization and evaluation of achieved results in later chapters. The chapter is structured as follows: First a general information about DNA 1.1 and DNA sequencing 1.2 with an introduction to the problem of base-calling, followed by the HMM algorithms 1.3 and algorithms used to compare the resulting sequence 1.4.

1.1 DNA

Deoxyribonucleic acid, hereafter called by a commonly used abbreviation DNA, is a vital part of every organism. It keeps its genetic information. DNA is in a form of double helix, where each strand is a sequence of four bases known also as nucleotides. These bases are Adenine (A), Guanine (G), Cytosine (C) and Thymine (T) [12]. An illustration of a double helix can be seen in Figure 1.1, where each color represents a different base. For the purpose of base-calling, it is sufficient to know that DNA is composed of four bases (called nucleotides). The genetic information is encoded in the order of the nucleotides and this sequence is unique to every organism. The process of determining the order is known as DNA sequencing.

Two strands of the double helix are held together by hydrogen bond interactions between the nucleotides. Adenosine pairs with Thymine and Guanine pairs with Cytosine. Therefore, if the sequence of one strand is known, the sequence of the complementary strand can be determined as well. As the pairs are fixed the sequences have varied CG to AT ratio specific to the given organism.

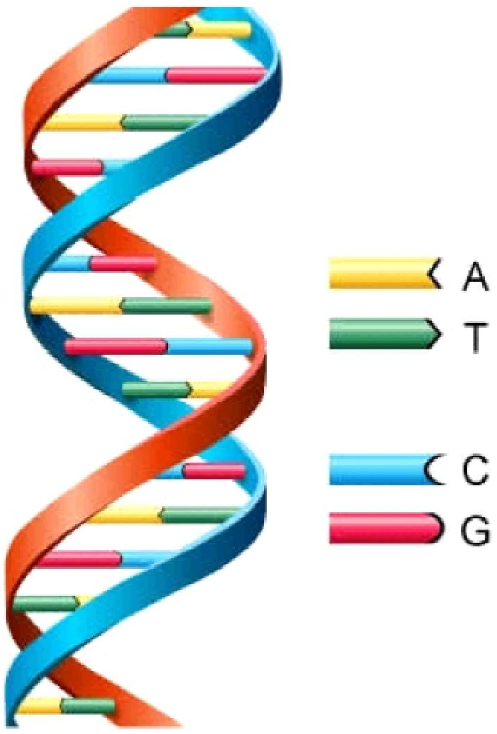

Figure 1.1: DNA double helix [1].

1.2 DNA Sequencing

The task of recognizing the sequence of bases in given DNA strand is called DNA sequencing. It remains a very difficult to perform accurately on long sequences.

Precise sequencing is desired in many fields. One area is particularly focused on it and even if it was the only area where it could be used, which is by far not true, the motivation to create an accurate sequencer would be immense. It is of course Medicine. Not only it could be used to more accurately diagnose genetic diseases, but also it would be possible to improve the drug design to target specific genes and much more. Other existing field interested in this research is for example agriculture. Understanding of crops genome could lead to creating more resilient and more producing individuals. Another important applications are for example in DNA forensics and detecting bacteria responsible for pollution. Actually, the possibilities are innumerable, any field which works with any kind of organic material probably can use results of these researches in some way.

Following sections will discuss some of the most popular or promising methods separated into three categories. Although the DNA was first isolated already in 19th century, its structure was not known until 1950 and the first widely used methods for DNA sequencing were proposed in 1970s [13]. These are presented in section 1.2.1. In section 1.2.2 some of the high-throughput methods are discussed, which were developed mainly to reduce the high cost of previous methods. Finally, section 1.2.3 is focused on Nanopore sequencing. It is one of the most recent methods, which tries to address some of the disadvantages of the used high-throughput methods. It is also the focus of this work, so it will be discussed more thoroughly.

1.2.1 First Methods

As was already mentioned the first sequencing methods were proposed in the second half of the 1970s. Two similar approaches are presented as they were proposed almost at the same time and both use electrophoresis to create electropherogram traces for base-calling.

1.2.1.1 Maxam–Gilbert Method

First discussed method was published in 1976 by Maxam and Gilbert [14]. The method chemically modifies specific nucleotides of a chemically labeled DNA molecule, leaving the backbone susceptible to cleavage at the modified bases. Partial cleavage at each base produces fragments extending from labeled end to each of the positions of that base. Then these single-stranded fragments are resolved by electrophoresis of polyacrylamide gel.

The process required only three different chemical cleavage reactions to identify every base. Each with single-base specificity. The fourth base would be recognized using the information from all three negative results.

As can be seen from the description this method involves a relatively complex chemical process. At the time of proposition, it was possible to sequence both double-stranded and single-stranded DNA molecules which were labeled at one end. Usual length of the sequenced molecule was about 100 bases, however with high dependency on resolving power of the used gel.

1.2.1.2 Sanger Method

Next method was proposed in 1977 and is still popular and widely used today. The method was proposed by [15] and is usually called Chain Termination method or even Sanger method. The Chain Termination method involves the Polymerase Chain Reaction (PCR), which creates many copies of the single stranded DNA template fragment. The template is then used in four sequencing reactions, where each is specific for one nucleotide. To each reaction all four ordinary nucleotides are added with one of the dideoxynucleotide, which terminates DNA strand elongation when bound to the strand. The resulting

DNA fragments are heat denatured and electrophoresis is performed to separate the fragments based on their length, resulting in an electropherogram trace.

An example of human DNA electropherogram is shown in Figure 1.2. The graph is composed by four separate curves, each from different chemical reaction. If overlapped, the peaks then show occurrences of respective nucleotides based on the curve. The figure shows a relatively clean part of the DNA sequence, which may tempt to think it would be a simple process to use a very simple base-calling method. Here and in any sequencing method, most of the data contain some noise.

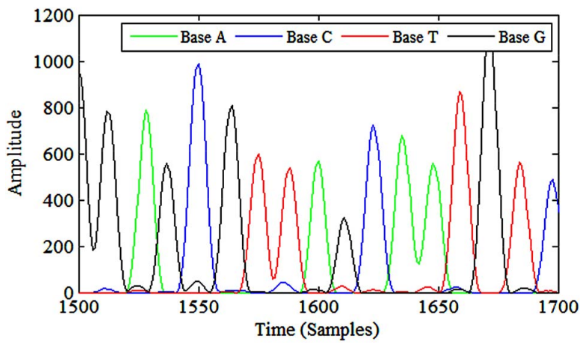

Figure 1.2: Segment of an electropherogram trace [2].

Chain Termination is also a complex chemical process which could sequence tens to hundreds of bases in single DNA strand. However, as it is still a popular method used usually for targeted validation, it has been improved over the time.

1.2.1.3 Electropherogram Trace Base-calling

As the Sanger method is widely used for several decades, a large number of heterogenous base-callers were developed to precisely determine nucleotides using the electropherogram traces. They vary from heuristic methods [16, 17], through statistical base-callers [18] up to machine learning methods [19, 20] and others [21, 22, 23]. Each method obviously has some advantages and disadvantages and is better suited for specific data.

1.2.2 High-throughput Methods

High-throughput methods is a category which describes a wide variety of approaches. The main reason for development was to increase throughput and reduce high cost of previously presented methods. This is done mostly by parallelization of the sequencing process allowing to produce thousands or more sequences simultaneously.

Many methods were developed over the time, however three of them were considered as the most promising and thus will be presented here. The methods are pyrosequencing 1.2.2.1, 2 base encoding also known as SOLiD sequencing 1.2.2.2 and Illumina dye sequencing 1.2.2.3.

All of the mentioned approaches share some features. First, all platforms require a library obtained either by amplification or ligation with custom adapter sequences. Each fragment is amplified on solid surface with linkers. This creates clusters of DNA for each library fragment. A sequencing reaction will be performed with each cluster. This reaction is specific for each technology. The raw output data is then a collection of DNA sequences generated for each cluster.

1.2.2.1 Pyrosequencing

First presented method is Pyrosequencing. The idea of the process is to recreate complementary strand to the template DNA. Two strategies are possible. First a solid state phase pyrosequencing uses immobilized DNA in the three-enzyme system. Second is a liquid phase pyrosequencing which introduces apyrase to make a four-enzyme system. It relies on the detection of released pyrophosphate (PPi). When a nucleotide is bound to the DNA backbone, a pyrophosphate group is released to form the inherent phosphodiester bond. This reaction is driven by the large negative free energy change associated with the release of pyrophosphate.

As the process is similar in both strategies, only the liquid phase pyrosequencing will be presented. One iteration of the process is shown in Figure 1.3. First step is polymerase which adds a solution of one of the nucleotides to the sample. A pyrophosphate is released only if a first unpaired base is complemented. PPi is then converted into ATP by sulfurylase. It then acts as a substrate for luciferase which produces light detectable by a camera. A pyrogram is used for analysis. The last step is "washing" performed by apyrase which degrades unused nucleotides and ATP [3].

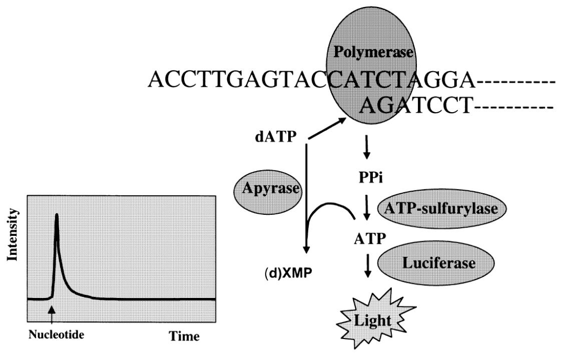

Figure 1.3: One iteration of Liquid Phase Pyrosequencing process [3].

The most known use of this process was by 454 Life Sciences company founded in 2000. The 454 sequencer was the first high-throughput technology available and was used to determine the first million base pairs of the Neanderthal genome. Although the reads produced by the platform were longer than its competitors (500bp), it became non-competitive due to high reagent cost, lower reliability, and higher error rate, especially in homopolymers [24].

1.2.2.2 SOLiD Sequencing

Next method is called 2 base encoding or sequencing by oligonucleotide ligation and detection (SOLiD) [25]. It does not rely on DNA polymerase as the other technologies, it is based on ligation sequencing. It uses 8-mer probes, each with one of four fluorescent dyes attached. Each 8-mer consist of two probe specific bases and six degenerate bases, which are able to pair with any nucleotide.

The process of the sequencing starts with binding of the primer to the single stranded sequence and after that the correct probe is ligated to the sequence and the rest are washed away. The fluorescent signal is recorded and along with the last three bases of the oligonucleotide is cleaved. The next cycle is performed the same way. After a few cycles the DNA strand is denatured and another primer, which is offset by one base, is attached. Five primers are used in total.

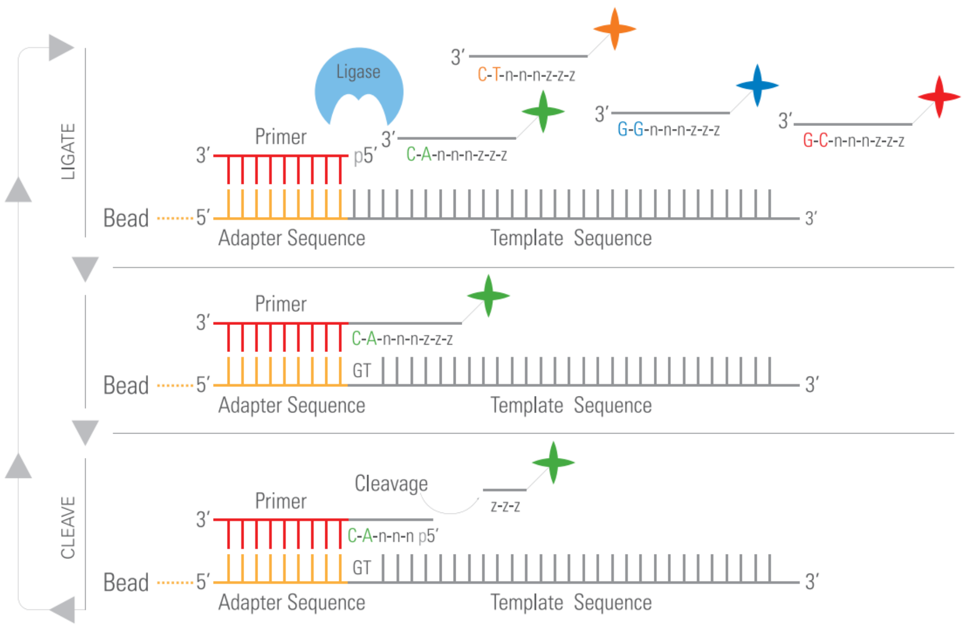

Figure 1.4: Illustration of sequencing by ligation in SOLiD System [4].

An illustration of the process is shown in Figure 1.4. This process has an advantage of lower error rate, because each base is read twice. The main disadvantage is the very short sequencing reads generated.

1.2.2.3 Illumina Dye Sequencing

Last presented method relies on sequencing by synthesis [26]. It is currently the most popular and most widely used method for DNA sequencing as it offers fast and low-cost usage compared to the formerly used methods. The results reports robust performance and ensure uniform accuracy and coverage across any sequenced genome regions [27].

The sequencing reaction in this approach uses incorporation of cleavable fluorescent and terminated nucleotides. All four nucleotides are added and after one of them incorporates to the sequence, the others are washed away. The fluorescent signal is read from the cluster and after that the fluorescent molecule and the terminator group are washed away as well. For better illustration Figure 1.5 is provided, which shows one example iteration of the described process.

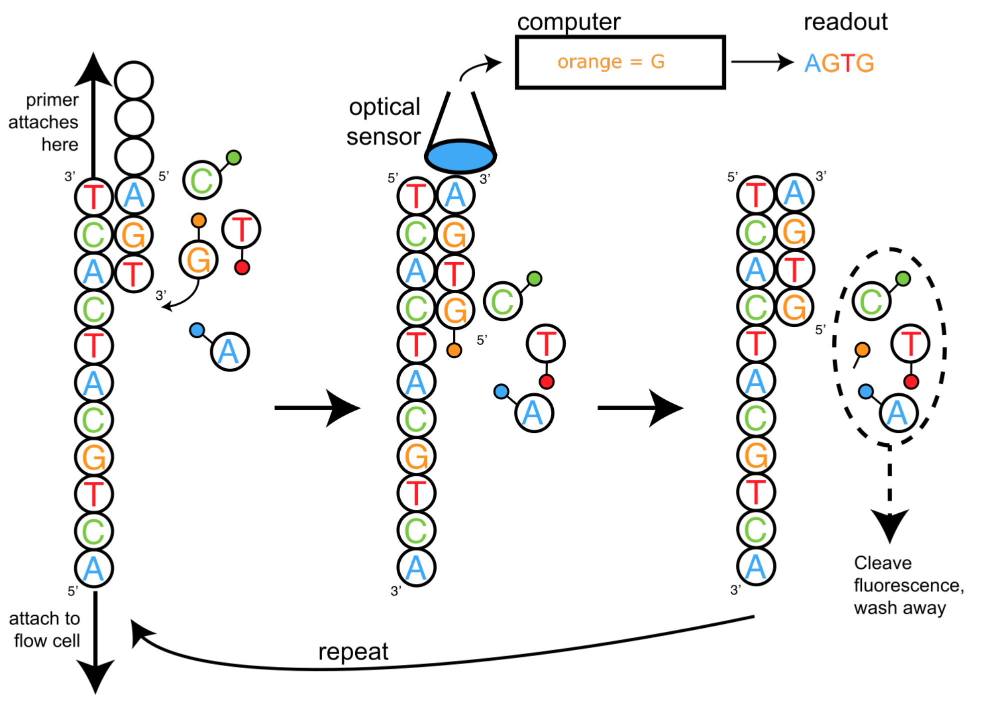

Figure 1.5: Illustration of sequencing by synthesis [5].

This process overcomes the issue of pyrosequencing by incorporating single nucleotides with the terminator group. However, the error of the sequencing increases over time due to the incomplete removal of the fluorescent signal, which leads to higher background noise levels, thus limiting the lengths of sequences to several hundreds base pairs.

1.2.3 Nanopore Sequencing

Although most of the sequencing produced in the world today comes from Illumina instruments, the technology still has severe limitations. Currently, the most used sequencing method performs very short reads and it is difficult to sequence single molecules. To address these two issues a new generation of sequencers is being actively developed. The concepts of nanopore sequencing have been proposed more than two decades ago, but with first instruments starting to be commercially available only a few years ago.

Usually the device consists of a protein with a gap which is approximately 1.4 nanometers wide, depending on the version, or it can be a solid state device with a drilled hole which is also maximally three nanometres wide to prevent secondary structures to flow through with the single strand. An illustration of the Nanopore is shown in Figure 1.6. During the transition of the DNA strand through the pore there is a recognizable decrease in ionic current. This ionic current can be translated, base-called, to a sequence of bases and thus finish the sequencing [6].

Already in 1996, Brantons group [28] has first shown a method to actually drive a single-stranded DNA through a pore protein. However, it took almost two more decades to unveil the MinION device. Oxford Nanopore Technologies created a Nanopore device which is able to read DNA strands with the length of several thousand bases.

This allowed researchers to focus on the actual base-calling which consist of translating the ionic current to the DNA bases. The main obstacle is, that while Nanopore is narrow enough to allow only one strand to pass at any given time, the measured current depends on the nucleotide context, instead of a single base. Typically five to six bases are taken into account. This is a problem for the following stages of the sequencing however. Thus it completes the experimental part of sequencing, which produces useful data from the chemical DNA sample.

1.2.3.1 Nanopore Base-calling

Each nanopore read is a sequence of currents measured in the pore, while a single strand of DNA flows through. And even though the sequencing can be performed almost anywhere and anytime it always yields data with similar attributes, therefore any base-calling technique should be adjustable to the currently measured dataset.

So, the usual process of base-calling consists of adjusting the parameters for current data (preprocessing) and after that the actual base-calling algorithm [29]. The first part of the preprocessing is **segmentation**. The raw data

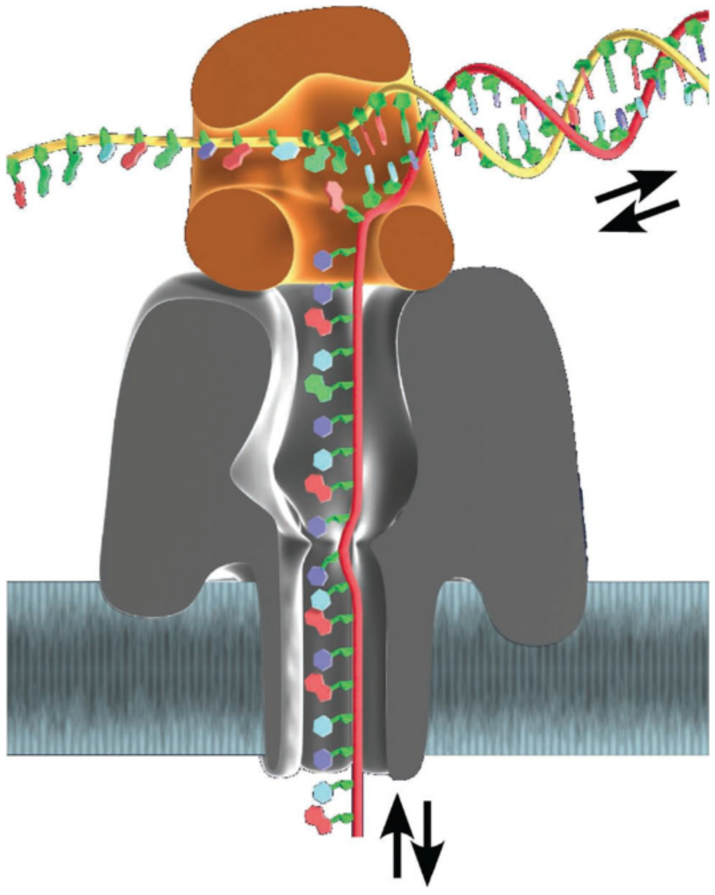

Figure 1.6: Nanopore illustration [6].

generated by MinION are usually much longer than the actual DNA sequence which passed through. The reason for this is, that it captures the current values periodically but usually with a much higher frequency than the bases pass through. The threading of the DNA strand through the Nanopore seems like a stochastic process, nevertheless it is controlled by biological enzymes so in other words the strand slows and speeds up, which makes it almost impossible to synchronize it with the measuring. Therefore, each context may be either measured more than once, or not at all. The frequency tends to be higher than seemingly necessary not to miss too many contexts. This requires a segmentation, which groups the currents of the same context in the Nanopore. Ideally, the result of this step is a sequence of events, where each event follows the previous event with a single move of the bases.

Two types of error can occur. First, called oversegmentation, is caused by the incorrect segmentation, where two following events describe the same context in the pore. The latter, called undersegmentation, is caused by the DNA molecule moving too fast through the pore and leaving none or significantly fewer data points in the signal.

Usual approach to segmentation detects all regions of ionic current longer than certain time threshold (e.g. 500 ms) which is in predefined interval (e.g. from 0 pA to 90 pA). After that, each event is recursively split at the point, which best splits a region into two Gaussian distributions until a threshold in probabilistic gain is reached, representing each time interval as a segment characterized by its mean current, standard deviation and duration [30].

Second step is **parameters scaling** with the goal to set or at least acknowledge the scaling of the measurement. Each device produces a slightly different values and even the same device used in different conditions does not yield the same values. Most prominent characteristics are *shift* and *scale*. Before any base-calling is possible, it is necessary to adjust these parameters to match the base-called sequence. Otherwise it would not only be inaccurate it may even fail to produce any meaningful result.

The last step is then the actual **base-calling**. The process always uses the events created by the segmentation, which may or may not be normalized by the scaling parameters. Other parameters mostly depend on the method used. However the usually needed information consist of the mean currents for each context in some form, e.g. table, and the way to get transition probabilities. The process itself may be done by many ways using a variety of algorithms. Some of those will be presented below.

1.2.3.1.1 Metrichor While presenting the most used base-callers, it is almost impossible not to start with Metrichor [31]. It is a platform created by Oxford Nanopore Company and it is probably the most accurate base-caller currently available. It used Hidden Markov Models for the base-calling and a complex preprocessing based on a thorough research.

However, it had several drawbacks, first and the most cast upon was, that it was an online service only. So, even though the MinION may have been used anywhere to sequence any DNA molecule obtained, the base-calling had to be then send via internet to the Metrichor, where it was processed. In some cases it may have limited otherwise astounding effectiveness of MinION device in practice.

Also the source code was accessible only under a restrictive proprietary license, which may have hindered the progress of the community in developing more and better base-callers. Because of that, other presented base-callers are easier to access, if not fully open source.

Recently a new version was released which now uses Artificial Neural Networks and is free to use, so most of the disadvantages were eliminated.

1.2.3.1.2 Nanocall One of the first available open source offline base-caller was proposed in late 2016 [29], so around the time the work started on this thesis. Authors as well noticed the lack of possibilities to perform local and private analysis of MinION data.

All three parts in Nanopore sequencing presented in section 1.2.3 are considered. However, as the Metrichor solution computes the segmentation locally it was not replaced by other method. The scaling on the other hand has been implemented and there are several options to pick from. The simplest method is Method of Moments which matches the first two moments of the distribution of pore model to the first two moments of the distribution of sequenced event means. Another option is an iterative Expectation–Maximization (EM) algorithm which uses several runs of Forward–Backward Algorithm to assess scaling parameters.

The final decoding is done by Viterbi algorithm, which computes the most likely observed event sequence and finally the base sequence is constructed by iteratively adding the minimum number of bases require to transition from one state to another. The whole project is written in C++11 and is available on github. Unfortunately for this work, their process is very similar to the planned outcome of this thesis. On the other hand, it was possible to overcome some difficulties based on the solutions proposed in this paper.

The Nanocall solution produces impressive results, where the mappability is comparable with the former Metrichor 1D reads with around 68 percent identity. As such and considering its similarity with this work it will be the main method with which the results will be compared.

1.2.3.1.3 DeepNano Another option to base-call offline is DeepNano [32]. This solution is implemented in Python and is accessible on bitbucket. It was presented around the same time as Nanocall for the same reasons. However, DeepNano uses Recurrent Neural Networks as a base-caller, which can potentially capture long-distance dependencies in the data. On the other hand, HMM works only with the fixed k-mers.

Used RNN is a type of Artificial Neural Network which generates sequence of output vector $\{\vec{y_1}, \vec{y_2}, \dots \vec{y_t}\}$ for given sequence of input vectors $\{\vec{x_1}, \vec{x_2}, \dots \vec{x_t}\}$. In this use, the inputs vectors consist of the mean, standard deviation and length of each event, and the output vectors give a probability distribution of called bases.

This method also uses already segmented data, so the first step is scaling of parameters of current reads. Considering the nature of the Neural Networks, it is done by iterative training, which uses alignment with a reference sequence using again EM algorithm. 1D base-calling is then simply done by applying the Recurrent Neural Network on other sequences. In this case, the RNN has three hidden layers and 100 hidden units. For 2D base-calling it is necessary to build an RNN which takes input from the two strands and combines them

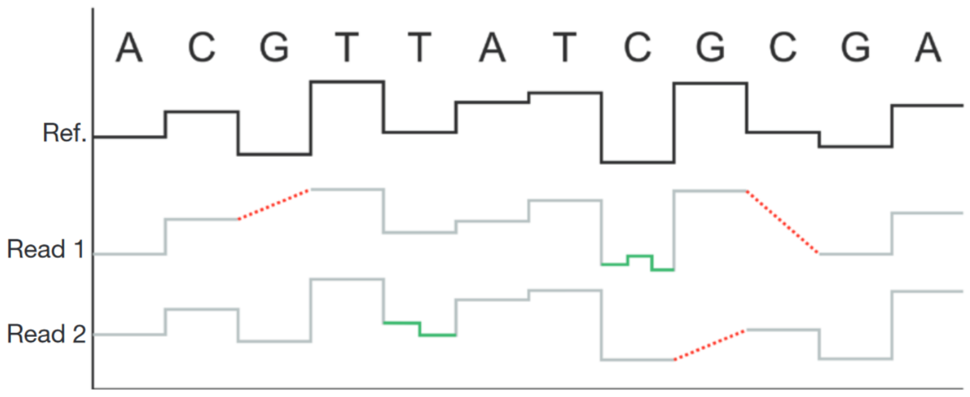

Figure 1.7: Sequence of events alignment in PoreSeq [7].

into a single prediction. This version of RNN uses four hidden layers and 250 hidden units.

1.2.3.1.4 PoreSeq Already in 2015 an algorithm called PoreSeq [7] which creates a consensus sequence from multiple sequencing reads of the same region of DNA. This algorithm iteratively optimizes the sequence of currents (events) to best fit the actual DNA sequence. An example of creating the best fit sequence is in Figure 1.7. It uses similar assumptions to the HMM version in terms of the k-mers present in Nanopore at any given time. So the measured current depends only on this short sequence of bases.

The model may even be extended to further use with the DNA sequences. And as it is written in Python and available on github, it is possible to use and modify by anyone. This type of method is called *de novo* genome assembly and is not exactly the base-calling performed in this work.

1.2.3.1.5 Nanopolish Around the same time Loman *et al.* published a similar approach as the PoreSeq. This method is called Nanopolish [33]. Their *de novo* genome assembly works in three stages. First stage detects overlaps between reads using multiple-alignment process. Next stage uses Celera Assembler and the last stage is polishing the result using a probabilistic model.

The main cause of errors was caused by an under representation of 5-mers consisting of single base, because it relies on change in electric current to detect base-to-base transitions, which may not be captured by the Nanopore as the measured current does stay in the same vicinity. The final version deals with this problem by polishing the assembly. However, any base-calling method has to deal with this fact.

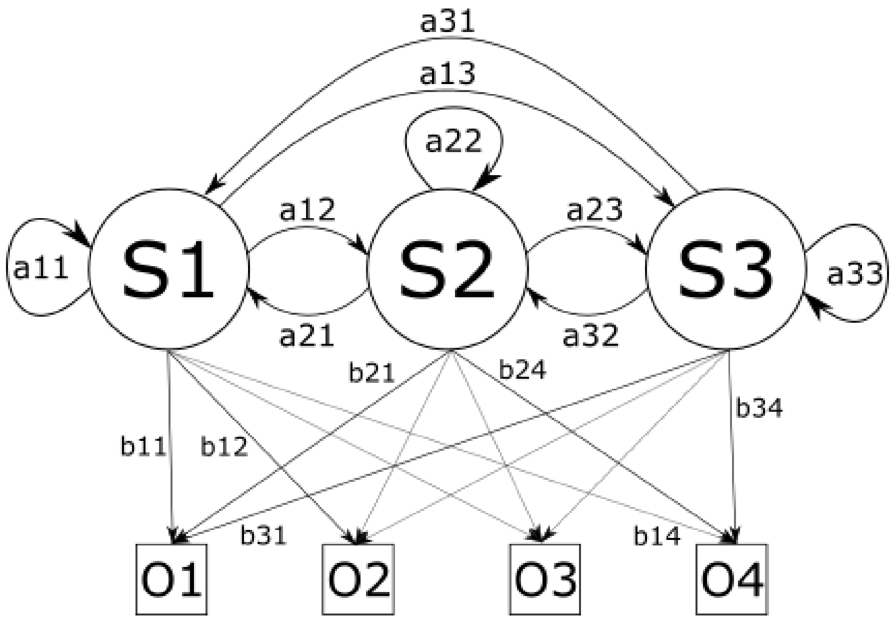

Figure 1.8: Illustration of Hidden Markov Model with three hidden states and four discrete observations.

1.3 Hidden Markov Model

This work will create a base-calling tool which uses commonly used Hidden Markov Model, so it is necessary to present its definition and properties.

Hidden Markov Model (HMM) is a finite model that describes a probability distribution over an infinite number of possible sequences [34]. It may be used for various purposes like speech recognition [35]. However here it will be described as a base-calling technique so all principles will be discussed using mostly DNA terminology.

The HMM is composed of a number of states, which are interconnected by transitions just like in basic Markov Model. In this case however, the current state is not visible. In other words, just by looking at the process it is not possible to determine in which state the model actually is in any given time. This is why it is called Hidden MM. However each state emits an observable value. The value emitted depends on emission probabilities of the current hidden state. The observations can be discrete values or, as in the case of DNA, continuous. If the emission probabilities are different for each hidden state it is then possible to recreate the order of the hidden states with certain probability. A general HMM is shown in Figure 1.8. Depicted HMM has these properties:

Hidden States: $S1$, $S2$, $S3$

Observations: $O1$, $O2$, $O3$, $O4$

Transition probabilities: $a11$, $a12$, $a13$, …, $a33$

Emission probabilities: $b11$, $b12$, $b13$, …, $b34$

In nanopore base-calling, the hidden states are the different contexts of the pore. Therefore, if using 3-mers, there are already 64 (4^3) hidden states. And thus, using longer context exponentially increases the complexity of the model. Observations are of course the recorded currents usually in the form of segmented events. The default transitions are then between contexts that are moved by one base, e.g. 'ACG' and 'CGT'. In theory, if the events were optimally segmented, it would be possible to have 25 percent probability between these contexts and 0 percent otherwise. However, as there are some segmentation errors it is necessary to adjust those probabilities accordingly. Lastly, the emission probabilities are continuous and have Gaussian distribution specific for each context present in the Nanopore.

There exist several algorithms designated for HMMs. The three most used are Forward Algorithm, Forward–Backward Algorithm and Viterbi Algorithm, where all of them use dynamic programming to accomplish their goal. The purpose of Forward Algorithm is filtering, which means it finds distribution of probabilities for hidden states at the last observation. This by itself is not very useful for DNA sequencing, so the next algorithm called Forward–Backward adds a backward phase by which it is possible to compute a distribution of probabilities of hidden states at any observation. This algorithm will be presented in greater detail in the section below. Lastly, the Viterbi Algorithm computes joint probability of the entire sequence. As the last algorithm looks the most promising for the actual base-calling it will be presented next.

1.3.1 Viterbi Algorithm

Viterbi algorithm proposed already in 1966 by Viterbi [37] is a widely used algorithm. Its main idea is to use dynamic programming followed by a backtrack to assess the most probable order of hidden states in Hidden Markov Model. It creates two matrices with the dimensions of $n \times s$, where n is the number of observations and s is number of hidden states. The computation process is shown in Algorithm 1.

As can be seen from the pseudocode, the assumption is, the sequence of hidden states can lead to each given state with a certain probability and this algorithm picks the highest probable state from which the sequence moved to this particular state. The probability is dependent on transition probabilities and on the probability of the previous state. The resulted maximum is then also multiplied by emission probability for the current current.

During the process however, the values become quickly too small to be represented directly and in order to prevent arithmetic underflow, a normalization is required, often combined with computing in log space.

```
Data: Observations (O), States (S), Transition Probabilities (tp),
      Emission Means (em)
for each observation (i) in Observations do
    for each hidden state (s) in States do
        $prev[i][s] = \arg\max_{ps \in S}(prob[i-1][ps] \times tp[ps][s])$
        $prob[i][s] = \max_{ps \in S}(prob[i-1][ps] \times tp[ps][s]) \times \mathrm{P_E}(em[s], O[i])$
    end
end
Backtrack prev States for: $\arg\max_{s \in S}(prob[last][s])$
```

Algorithm 1: Pseudocode of Viterbi Algorithm [8].

1.3.2 Forward–Backward Algorithm

Second discussed algorithm is Forward–Backward Algorithm. It was first presented by Chang [40] and later used and discussed by Baum [41]. The algorithm has two main phases and then a smoothing phase. The phases are called Forward phase and Backward phase, hence the name Forward–Backward.

The Forward phase is actually identical to the mentioned Forward algorithm. Its goal is to compute the most probable state for certain event. It uses the history of observations to asses conditional probability for each hidden state. This process is called filtering. However, if a Backward phase is added, the process is referred to as smoothing. The Backward phase, as the name suggests, starts at the last event of the input data and computes probability of observing the remaining observations. Last phase uses results of both previous computations and uses its products for each event to assess the probability for all hidden states. The whole process is shown in Algorithm 2. As can be seen, both of the main phases are very similar to the Viterbi main phase. The difference is however, the probabilities of previous, next respectively, states are summed instead of finding just the maximal value.

Theoretically, if only the states with the highest probability are considered, it would be possible to reconstruct whole sequence, however it is not the purpose of this algorithm. The neighbouring events would not consider the transition probabilities. As an example, transition probability from ‘ACG’ to ‘GTA’ is very low because it would mean that at least one event, which describes the connecting context, is missing in the input data. These segmentation errors happen, although rarely, and two or more missing events in a row are even rarer. However, using Forward–Backward algorithm, these “skips” would occur commonly because the continuity is not taken into account. Therefore, this algorithm is commonly used to assess the quality of other base-calling methods by computing the probability of error for each base.

```
Data: Observations (O), States (S), Transition Probabilities (tp),
      Emission Means (em)
Forward Phase:
for each observation (i) in Observations do
    for each hidden state (s) in States do
        $fwd[i][s] = \mathrm{P_E}(em[s], O[i]) * \sum_{ps \in S}(tp[ps][s] * fwd[i-1][ps])$
    end
end
Backward Phase:
for each observation (i) in Observations reversed do
    for each hidden state (s) in States do
        $bkw[i][s] = \sum_{ns \in S}(\mathrm{P_E}(em[s], O[i]) * tp[s][ns] * bkw[i+1][ns])$
    end
end
Consolidation Phase:
for each observation (i) in Observations do
    Estimated hidden state$[i] = \arg\max_{s \in S}(fwd[i][s] * bkw[i][s])$
end
```

Algorithm 2: Pseudocode of Forward–Backward Algorithm [9].

Base algorithm works with an assumption that the starting and ending states are known. This is not the case in the presented DNA base-calling so instead of that, starting probabilities depend solely on the first current and means of all contexts. Probabilities of end state then depends on the last current and means as well. This may cause inaccurate predictions for the starting and ending events due to the seemingly random deviations of recorded currents. However, considering how long most of the DNA sequences are, it should not cause concern.

Same as in the Viterbi Algorithm, there is a possibility to use logarithmic values, because in both Forward and Backward phase, the probabilities cannot keep any precision in commonly used data types. However, in this case it is not that simple, since there are the sums of probabilities. Using logarithmic values changes multiplications to additions. Additions on the other hand cannot be done so easily. The solution to this is that the logarithms are used only for the normalization purposes and the values are kept in original form.

1.4 Comparing Sequences

After finishing the base-calling process using a base-caller in development, it is necessary to compare the predicted sequence with a reference one to assess its accuracy. That is obviously just an example, but one which applies for this thesis. To accomplish the comparison, it is possible to use many methods and means. One of the highly used methods is called alignment. It is because it can be relatively clearly visualized so even by looking at the alignment, it is recognizable if the given sequences are similar. For example, if sequence A is ‘ACGTGTCACCG’ and sequence B is ‘GTGACACGG’, the alignment can look like this:

```
ACGTGTCACCG-
--GTGACAC-GG
```
(1.1)

There is also this option:

```
ACGTGTCACCG
--GTGACACGG
```
(1.2)

In most cases, there are multiple possible alignments, however it is not easy to determine which one is the best. It also depends a lot on the preferences of the author. Sometimes mutations can not cause a big difference, however gap (insertion/deletion) is then highly penalized.

Hence, there exist more ways to do the actual alignment and each takes into account different preferences, but mainly the sequences can be aligned using two methods. First is local alignment and the other is global alignment. As the names suggest, the local alignment focuses on smaller portions of sequences and finds local similarities. Because of it, it is best suited for finding out if a smaller sequence is a subsequence of a larger one.

This work will however focus on computing whole sequences, so the global alignment method, which is designated to figuring out if the two given sequences are parts of the same DNA, should provide better comparison of the results. Only advantage of the local alignment in this work should be, that it does not penalize different starting and ending dissimilarities as the global alignment does. It can prove to be useful as well. That is why there will be both local and global alignment method implemented.

A third way of alignment is presented as well. It is a common edit distance with the addition of remembering the taken steps. This solution may provide the best insight as it counts the sum of insertions, deletions and substitutions.

1.4.1 Smith–Waterman Algorithm

Local alignment is usually done by Smith–Waterman algorithm [42]. The main idea of the algorithm is to use dynamic programming so the computation is relatively fast, considering the number of different alignments. The speed is compensated by higher space requirements.

Implementation uses matrix of semi-results F with the dimensions of $(m+1) \times (n+1)$, where m is the length of one compared sequence and n is the length of the other sequence. Then another matrix D of the same dimensions, which contains direction of the current step (left, top, top-left). This matrix is used for backtracking phase, where the actual alignment is completed.

The algorithm is relatively simple as shown in Algorithm 3.

```
Data: Sequence $A$, Sequence $B$
for each row in $F$ do
    for each element in row do
        $match = F_{i-1,j-1} + \mathrm{S}(A_i, B_j)$
        $delete = \max_{k \in \{1,i-1\}}(F_{i-k,j} - w_k)$
        $insert = \max_{l \in \{1,i-1\}}(F_{i,j-l} - w_l)$
        $F_{i,j} = \max(delete, match, insert)$
        $D_{i,j} = \mathrm{direction}(delete, match, insert)$
    end
end
$Result = \max_{i \in 1 \ldots m, j \in 1 \ldots n}(F_{i,n}, F_{m,j})$
Backtrack in $D$ from $D_{i,j}$, ($i$ and $j$ are same as in the result)
```

Algorithm 3: Pseudocode of Smith–Waterman Algorithm [10].

S in this context returns comparison value of the given bases. In most cases it is represented as a substitution matrix, where the values are usually:

$$S_{i,j} \begin{cases} > 0, & \text{if } i = j \\ \leq 0, & \text{if } i \neq j \end{cases} \tag{1.3}$$

The matrix can take into account, that Adenine and Guanine are purines, while Cytosine and Thymine are pyrimidines, so these pair may be almost interchangeable. Or the symbol comparison can be very simple as shown in the example Table 1.1.

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

Table 1.1: Substitution matrix.

Another important thing is the Gap Penalty denoted by w_i. It comes into play when an insertion or deletion is made, which creates a gap (denoted by '–' in alignment). Usually the Gap Penalty is computed like this:

$$w_i = u + vk,$$

in this context u is the initial penalty for starting the gap, v is a penalty for expanding the gap and k is the length of the gap minus 1. So with the gap of length 3, starting penalty of 12 and continuing penalty of 5, the whole gap would cost $12 + 5 * 2 = 22$.

1.4.1.1 Gotoh Improvement

The algorithm in the explained version still has relatively high time complexity. It is $O(m^2n)$. This may not be a problem for shorter sequences, however in this work some of the sequences (either reference or predicted) can be tens or even hundreds of thousands bases long, so it would make this algorithm almost unusable.

The space complexity can be also considered too high especially for personal computers. The space complexity is $O(mn)$, because it uses $m \times n$ matrix of semi-results and $m \times n$ matrix of two bit (three values) directions. The actual Gotoh improvement does not consider the space complexity at all, however the improvement enables the algorithm not to use the semi-results matrix and instead can use only two m long vectors and one variable. The actual space complexity stays at $O(mn)$, however the matrix consist of only three bit values instead of whole integers.

The improvement is done in the step where the algorithm selects the best direction for current step i, j. The steps looks like this:

$$F_{i,j} = \max(delete, match, insert)$$

In base Smith–Waterman algorithm the time complexity for computing *delete* is $O(m)$ and *insert* is $O(n)$. Gotoh proposed [43] a way to compute both *delete* and *insert* in $O(1)$. Method uses the idea, that it is not necessary to look back through the whole row or column, but remembering partial results enables to use only one comparison.

delete is then computed [44]:

$$delete_{i,j} = \max(F_{i-1,j} - w_1, delete_{i-1,j} - u)$$

and insert:

$$insert_{i,j} = \max(F_{i,j-1} + w_1, insert_{i,j-1} - u)$$

However when backtracking, the algorithm has to know which value was taken as a new *delete*, *insert* respectively, and continue the same way instead of turning to the diagonal direction. Hence the third bit in directional matrix.

1.4.2 Needleman–Wunsch Algorithm

Needleman–Wunsch algorithm [45] has a similar structure to previously presented Smith–Waterman method, especially with the Gotoh improvement. The main difference is that the gap penalty is applied to every insertion/deletion so continuing the gap is penalized by the same amount as starting it. Another important difference is in initialization. In this case the first column and row are filled by using the higher index multiplied by gap penalty, so $F_{i,0} = i * \text{gap}()$, and $F_{0,j} = j * \text{gap}()$ respectively. Then the algorithm continues as shown in Algorithm 4.

```
Data: Sequence $A$, Sequence $B$
for each row in $F$ do
    for each element in row do
        $match = F_{i-1,j-1} + \mathrm{S}(A_i, B_j)$
        $delete = F_{i-1,j} - w$
        $insert = F_{i,j-1} - w$
        $F_{i,j} = \max(delete, match, insert)$
        $D_{i,j} = \mathrm{direction}(delete, match, insert)$
    end
end
$Result = F_{m,n}$
Backtrack in $D$ from $D_{m,n}$
```

Algorithm 4: Pseudocode of Needleman–Wunsch Algorithm [11].

Lastly as shown in the pseudocode, the result is always in the "bottom-right corner", so in $F_{m,n}$.

Otherwise, the properties are same as for the Smith–Waterman–Gotoh algorithm, so both the time and space complexity are $O(mn)$, and it uses one n long vector and $m \times n$ matrix of 2 bit directions.

1.4.3 Levenshtein Distance

Levenshtein distance, also ordinarily called Edit distance, is a string metric, which measures the distance of two strings by counting the number of necessary insertions, deletions and substitutions [46]. This is done very similarly as in both algorithms discussed before, however considering the definition, the lower the results the more similar are the compared strings. So, instead of maximums, the algorithm uses minimums.

The decision then looks like this:

$$F_{i,j} = \begin{cases} F_{i-1,j-1} & \text{if } A_i = B_i & \#match \\ 1 + min \begin{cases} F_{i-1,j} \\ F_{i,j-1} \\ F_{i-1,j-1} \end{cases} & & \begin{matrix} \#deletion \\ \#insertion \\ \#substitution \end{matrix} \end{cases} \tag{1.4}$$

Though defined only as a metric, it can also be used to remember the "movement" during comparison and the result can be then shown in the same manner as it was in alignment methods. Algorithm then looks very much like Needleman–Wunsch algorithm using specific substitution matrix and gap penalty.

Chapter 2

Results

First section 2.1 of this chapter presents the dataset, which was used to optimize implemented solutions that are presented in section 2.2. Following sections focuses on obtaining the best settings for both Viterbi and Forward–Backward Algorithms. Therefore several runs were executed and graphs and tables showing results are presented. First testing was performed on simulated data in section 2.3, after that on sample set 1 2.4 and lastly on sample set 2 2.5 with the adjusted table of means and standard deviations. Sample set 3 was used only in comparison with other solutions.

Each dataset was tested using 3, 4 and 5-mers. After all test were exhausted the best performing solution was selected 2.6 and compared with both Nanocall and DeepNano 2.7. The last section 2.8 briefly discusses the time and space complexity of the solution.

2.1 Dataset

Data provided by Nicholas Loman [47] were used for most of the testing and optimization purposes. These data contain 164,472 E.coli DNA sequences. The data were recorded during the 28th and 29th of July of 2016. The measuring was performed in six different locations using the same R9 chemistry. Even though the same type of device was used, each of the station will have a different parameters, so the scaling will be necessary.

But first, the dataset as a whole will be presented. The dataset can be summarized by these numbers using base pairs as a unit:

Minimal Length:	177
Average Length:	9,009
Maximal Length:	131,969

The histogram 2.1 shows that the lengths have almost geometric distribution if grouped by thousands. Except the very start, where there are not as many sequences shorter than thousand base pairs.

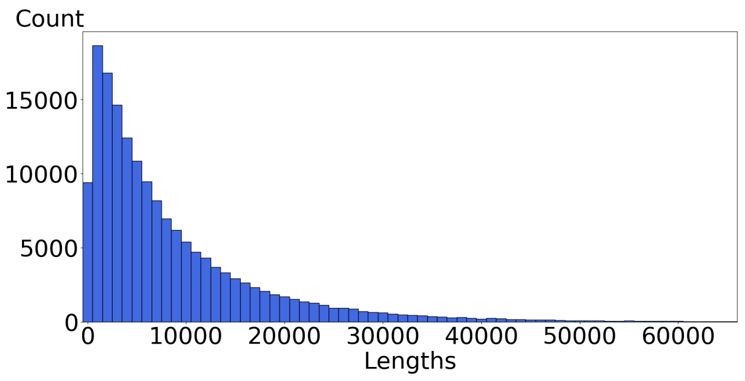

Figure 2.1: Histogram of sequence lengths in used dataset.

The distribution of individual bases is almost uniform, where both Cytosine and Guanine bases are represented by more than 26 percent, while Adenine is represented a little under 24 percent. This is shown in Figure 2.2. As such the data are slightly CG rich.

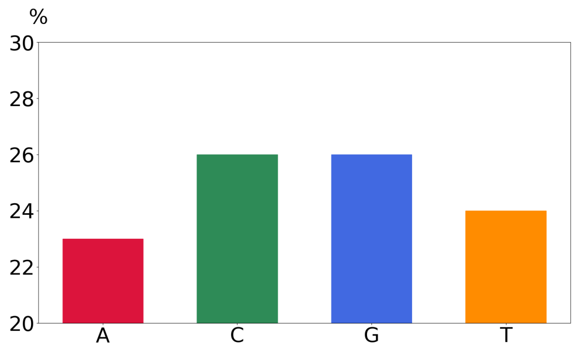

Figure 2.2: Nucleotide distribution in the datatset.

In subsections below an example sequence will be shown in detail. After that, two sample sets, taken from the introduced dataset, are presented and their properties discussed. Last section shows another sample set, which is on the other hand a measurement of one specific human gene, which should be harder to base-call because of its repetitiveness and different CG to AT ratio.

2.1.1 Example Sequence

Each sequence is stored in a *fast5* file. Whole tree of the content is shown in Appendix C. One of the sequences will be shown in detail, others have similar properties. The presented sequence is from sample 1 discussed below. It is a sequence recorded on 29th of July 2016 in the lab with ID of 26075 on channel 115 read 849. It contains 5,671 bases.

One of the most important parts for this work will be the *Signal* dataset from *Raw* group and the *Events* dataset from *EventDetection* group. The raw data are shown in Figure 2.3. As can be seen the data points are mostly between 400 and 800 units. However most of the sequences start and some even end with the measuring highly above the 800 mark. These parts of the sequences will not be taken into account as they are the parts, where the DNA strand was not located inside the nanopore. A closer look in Figure 2.4 shows about 30 base shifts, because the whole dataset consist of 96,260 valid raw measurements so that leaves almost 17 data points per base move.

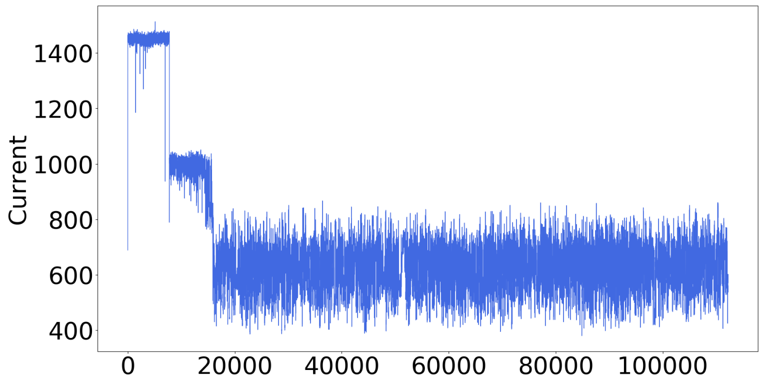

Figure 2.3: Example of raw currents in one *fast5* file.

The segmentation of raw data creates a sequence of events. In optimal case each event represents one step of the sequence reading, meaning that any single event would be shifted by exactly one base. So, for example if one event represents ‘TTCAC’, the next one can be ‘TCACG’. However, the segmentation is not always perfect and it is necessary to introduce so called stays and skips.

Stays are events which made no shift in context from the previous event, in other words, the Hidden Markov Model used a loop back to the same state. This may happen only due to a segmentation error, which wrongly assessed a shift. Skips, on the other hand, are events with context shifted by more than a single base. Skips may occur due to an error in segmentation as well or due to an irregularity in sequencer, where the strand of DNA moved too fast and the sequencer did not measure enough values for the skipped context.

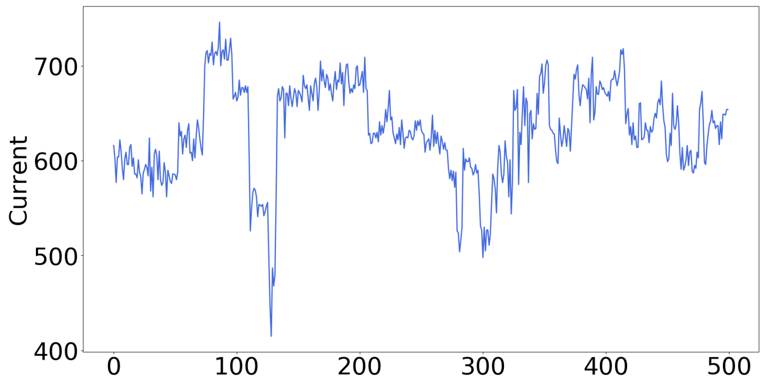

Figure 2.4: Closer look at 500 data points in one real data file.

The correct events will be called steps in future references. An example of segmentation using the same sequence is shown in Figure 2.5. Closer look at the segmentation results is in Figure 2.6. It shows the same part of the raw data in orange, with the segmented section shown in blue. All of those events are stored in the dataset *Events* in the *fast5* file.

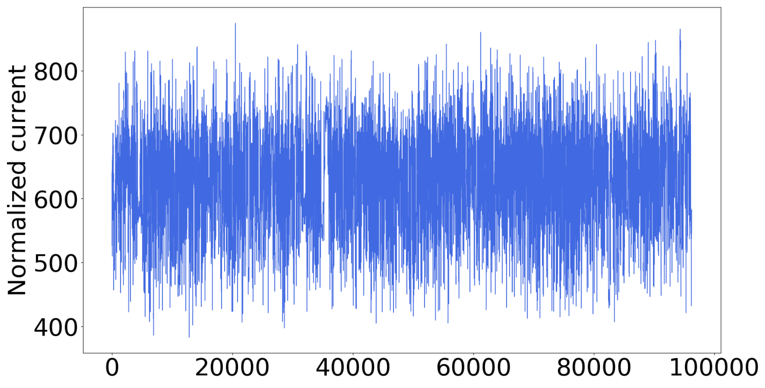

Figure 2.5: Example of segmented events in one data file.

2.1.2 Sample Set 1

Both selected samples contain exactly hundred sequences. The first of the two sample sets was measured by a station with ID of 26075 on 29th June 2016.

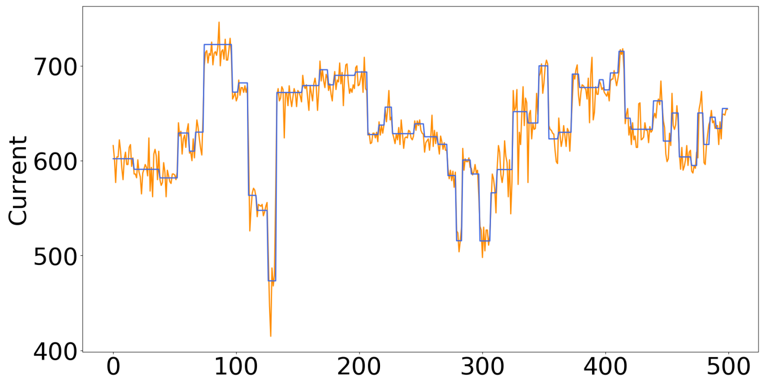

Figure 2.6: Detail of 50 events in the same data file.

Main properties are as follows:

Minimal Length:	606
Average Length:	4,784
Maximal Length:	11,982

Whereas the full dataset contain sequences over 130,000 base pairs long, in the sample the longest sequence is only 12,000 base pairs long. The main reason for choosing shorter sequences is the shorter run time. Closer comparison of the bases and event count is in Figure 2.7, where the ratio between event and base count is visible. There are up to twice as much events which means a high number of stays will have to be detected. However, it is necessary to take into account some skips as well. By the examination of the provided analysis, the stay probability should be around 0.5 and skip probability little over 0.002, although more probabilities will be tested using these as defaults.

2.1.3 Sample Set 2

The other sample set was measured by a station with ID of 40525 a day earlier on 28th June 2016. The properties are very similar to the previous set and are listed here:

Minimal Length:	528
Average Length:	6,197
Maximal Length:	16,208

As in previous sample, there are more events recorded than the actual base count (Figure 2.8). Stay and skip probabilities should be similar to the previous sample.

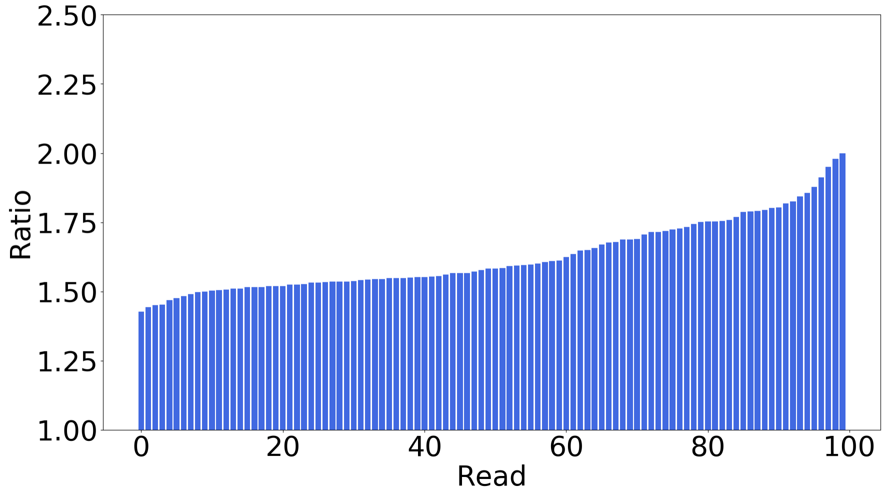

Figure 2.7: Event count to Base count ratio for each read in sample 1.

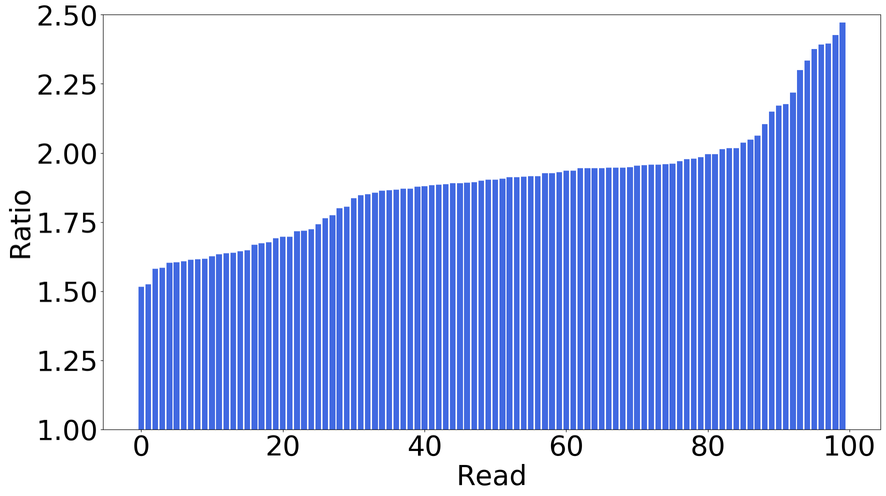

Figure 2.8: Event count to Base count ratio for each read in sample 2.

2.1.4 Sample Set 3

As was already mentioned, the third sample is a measurement of human gene with a distribution of all nucleotides presented in Figure 2.9. As can be seen, the distribution is very different from the previous dataset. It has lower CG content, although as it describes only single strands without their complements, the number of C and G bases is different.

On the other hand, some properties are very similar due to the sequencing technology used. The sample contains 100 sequences and the lengths of used strands are as follows:

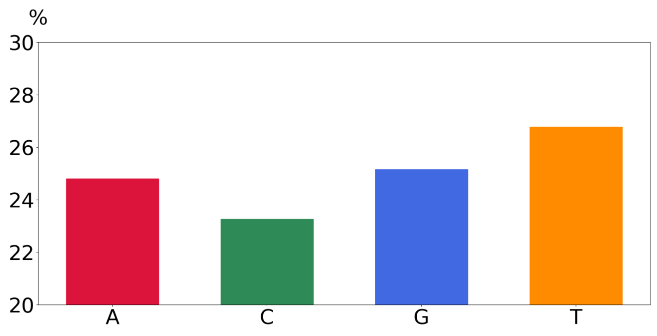

Figure 2.9: Nucleotide distribution in sample 3.

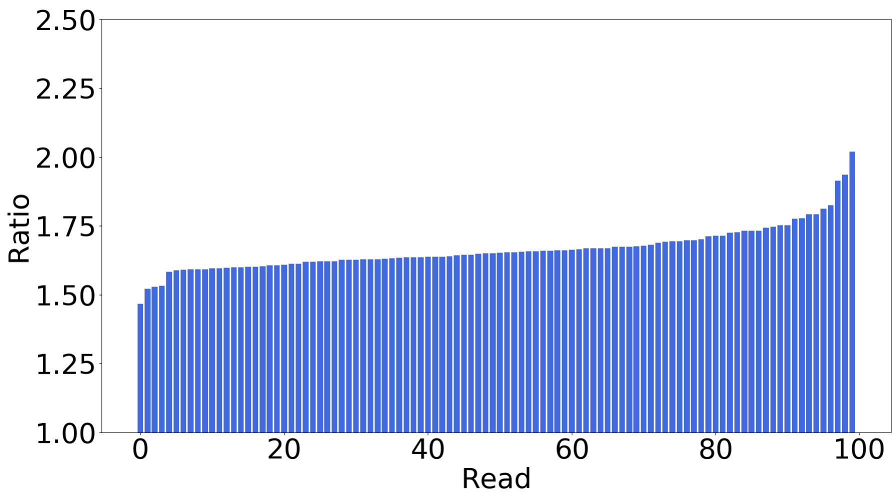

Figure 2.10: Event count to Base count ratio for each read in sample 3.

Minimal Length:	472
Average Length:	8,043
Maximal Length:	11,968

Ratios between event and base pair counts are also similar to the previous samples, so the skip and stay probabilities should be very similar. The specific values are shown in Figure 2.10 and it is again sorted by the shown value.

2.2 Implemented Methods

This section describes the proposed approach to the DNA Base-calling. Introduction to algorithms and the pseudocodes were described in methods 1, the focus of this chapter is on the practical implementation and usage of both Viterbi and Foward–Backward algorithms as well as the supportive scripts, which were used for processing the data and the outputs. In the first two sections 2.2.1 and 2.2.2 both HMM algorithms are discussed. Next section 2.2.3 will show data simulation for the initial test and the last section 2.2.4 will discuss mainly the implementation of the alignment algorithm.

It might be helpful to first introduce a flexible toolkit *Poretools* [48], which was created by Loman and Quinlan in 2014 in order to easily handle *fast5* files generated by the MinION. Currently the device generates single *fast5* file for each read, which results in a large number of files. Thus the *Poretools* can be used on whole directories or single files. It is written in Python and is easy to use. It is possible to convert *fast5* files to either *fastq* files or to *fasta* files which are expected as an input in the proposed application.

However, considering that the data were filtered only once and it was more convenient to do so using a script, instead a *h5py* [49] library was used to not only extract both events and reference sequences from the *fast5* files, but also to analyse all features of the dataset. This is the source of the information given in section 2.1.

2.2.1 Viterbi Algorithm

Viterbi algorithm was chosen due to its proven usefulness in this area. Timp *et al.* in 2012 [50] proposed using Viterbi algorithm as a high accuracy base-caller. It was not tested on real sequences, but on simulated data, which should be generated using the same parameters the actual data are distributed. However, one very important and not very accurate assumption was made. The generated data used optimally segmented events, so the algorithm did not have to consider stays and skips and instead could take into account only steps by exactly one base. The algorithm performs well despite the simplification, which was later confirmed by others like [29]. The algorithm just needs to take into account stays and skips by adjusting transition probabilities.

In this approach only the scaling of parameters and base-calling steps are implemented as the segmentation is not only a rather difficult process to perform accurately, but the given dataset provided already segmented data as well. Also the Metrichor platform segments the data locally so it is not necessary to create a new segmentation tool. On the other hand, both parts implemented are usable for computation with 3-mers, 4-mers and 5-mers as contexts. Those three versions were tested, however other sizes should be viable without further modifying the solution.

2.2.1.1 Scaling of Parameters

The default parameters for the algorithm were based on means and standard deviations taken from the sample 1 presented in section 2.1. The dataset Events contains means, standard deviations and matching 5-mers, so using this information it is possible to reconstruct every value for up to 5-mers.

When reconstructing means and standard deviations for currents for smaller k-mers, the last nucleotides were taken into account as they should be those which entered the pore last and thus influenced the ion current values.

These means and standard deviations were used in testing on sample set 1. However for the sample set 2, those values would not be optimal. Therefore, a simple adjustment using a normalization similar to Min-Max is used. The assumption is that both compared datasets will have the same distribution of currents. Then the k-mer with lowest and the highest mean current will have the value at the similar percentile of the data. Thus it is possible to get new values for those two k-mers and use their values as new Min and Max. After that the new value of mean current for each k-mer is computed like this:

$$\mu_{new} = (max_{new} - min_{new})\frac{\mu_{old} - min_{old}}{max_{old} - min_{old}} + min_{new}$$

Measured percentiles and matching k-mers are shown in Table 2.1. This is a relatively simple solution and it should work for sample 2 mostly because both datasets are measured using E.coli DNA strands. When this scaling is used on DNA from different organism it may yield skewed results, because the ratio of CG to AT nucleotides may not be the same and thus the distributions of currents will differ as well.

	Min		Max	
k	k-mer	perc.	k-mer	perc.
3	GTT	2.7179	TCA	96.9545
4	GGAT	0.5402	ATCA	98.9582
5	GGGAT	0.3041	GATCC	99.2965

Table 2.1: Percentiles of contexts with minimal and maximal current in sample 1.

To confirm the assumption about the distribution of currents a histogram 2.11 is presented. It is visible that the values are slightly shifted and are wider in sample 2 compared to sample 1, otherwise they both have two similar peaks. The decrease in the middle is caused by steeper increase in currents between corresponding contexts in the middle of the table.

Also, as was predicted, sample 3 follows a different curve, so the scaling method may not be as viable as for the sample 2. The last curve represents the distribution of currents in the whole E.coli dataset. As the dataset contains measurements from several laboratories with differently shifted values, the decrease in the middle of the curve is smoothed out and almost perfect Gaussian curve is created. As can be seen, sample 1 is shifted more from the average sequence, although it can still be considered common.

The scaling was done by a Python 2.7 script *adjustTable.py*, which requires the size of k-mers used, the default table of means and standard deviations of

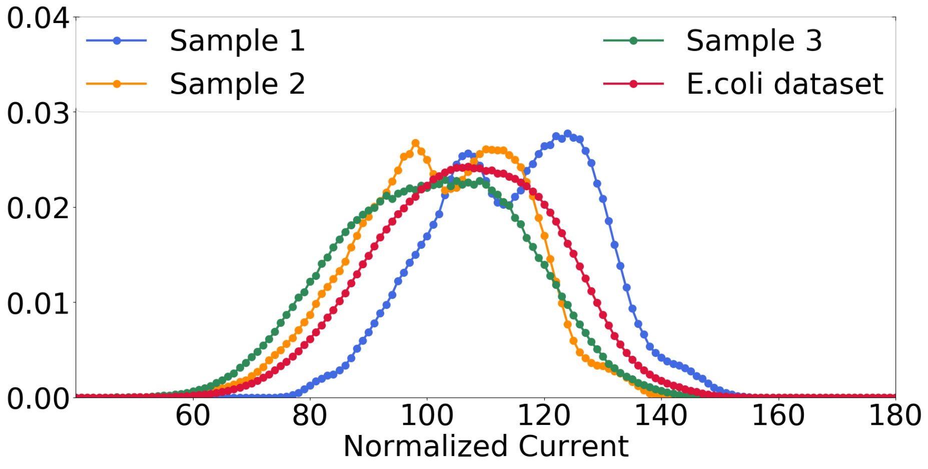

Figure 2.11: Comparison of current representations in used data samples.

currents for each context, *fasta* file of the sequences to be base-called and the name of output table.

2.2.1.2 Base-calling

The process of base-calling is then implemented both in Python 2.7 (file *viterbi.py*) and in C++11 (file *viterbi.cpp*). The C++ version can be compiled by attached *Makefile*. All algorithms also implemented in C++ are included in the *Makefile* as well.

The run of base-calling using Viterbi Algorithm requires an input file with segmented currents, a table with means and standard deviations of currents, name of the output file, size of k-mers, stay probability and skip probability. The results will be measured using the size of k-mers from 3 to 5 and will iterate over both stay and skip probability with the default values set to the one determined in the section 2.1.

The procedure of Viterbi algorithm was previously discussed, here we show the probability computations. The probabilities of the first k-mer was uniformly distributed as no part of the sequence is known.

After that the transition probabilities were computed this way:

$$P_{(A \to B)} = p_{stay} * \delta_{A=B} + p_{step} * \delta_{\text{follows}(A,B,1)} + p_{skip} * \delta_{\text{follows}(A,B,2)},$$

where p_{step} is the complement of $p_{stay} + p_{skip}$ to 1 and the follows(A, B, i) function returns whether k-mer B is reachable from k-mer A using exactly i steps. This is an approximation in terms of not taking into account skips with the length of three and more. The adjustment was made mostly due to the acceleration of the computation by a significant amount. The simplification is possible because it should not affect accuracy much. The p_{skip} value is already really small and thus the longer skips would have even lower probability, which means that long skips are extremely rare and as such would be very hard to predict even if the algorithm expected skips of arbitrary length.

Other parts of the algorithm were not modified, so after finishing the core computing, backtrack is performed and then the base sequence is reconstructed using the smallest number of base shifts possible between the neighbouring predicted contexts.

2.2.2 Forward–Backward Algorithm

Forward–Backward algorithm was implemented twice, each with a different purpose. First uses this algorithm for its intended purpose. It uses an output from the base-caller and assesses the quality of each base using Phred quality score. The score is computed using this formula:

$$Q = -10 \log_{10} P,$$

where P is probability of an error. The probability is computed using an average value for all k-mers, which contain specific base. An output of this algorithm is the default *fastq* file, where the quality is in Phred+33 format. This means it is representable in ASCII characters from '!' for the lowest probability to 'I' for the highest probability.

The implementation is in *fwd_bkw_quality.cpp* script and is also included in the *Makefile*. The other use of the Forward–Backward algorithm is more experimental and thus will be discussed more thoroughly in the next section.

2.2.2.1 Experimental Approach

The Forward–Backward algorithm is not suitable for base-calling by itself due to the fact that its purpose is only to compute the hidden state with highest probability for each event. One could consolidate k-mers into a base sequence, but attempting to do this mostly results in a much longer sequence than expected, because the transitions are not taken into account during the Consolidation phase. It is not uncommon to see four or five bases long skips.

However, the issue can be addressed by modifying the Consolidation phase. The idea is to go through all events and find the state with the highest probability as usual, however if a skip of length three or more is encountered, the algorithm looks at other probabilities in descending order and if it finds a k-mer which follows the previous k-mer with shorter skip or even as a stay/step, this k-mer is used instead. However, it stops going through other states if the probability is too low. The boundary is set by a new input parameter called *smoothing*, which is used as a multiplier.

Another improvement is expected if the consolidation which ignores some events is used. The idea behind it is that many times the consolidation phase encounters two long skips immediately after each other. This may be caused by the nature of Forward–Backward algorithm, where the sequence of predicted *k*-mers is for example 'TCAAA', 'GGGGG' and 'AAATG'. There the third *k*-mer follows the first one using two steps (consolidated as 'TCAAATG'), but the second *k*-mer does not fit between them, which may be caused by some sequencing anomaly. The modified consolidation removes the 'GGGGG' event, because it neither follows the previous event, nor is followed by the next event.

Both of these experimental improvements are optional using input parameters and as such will be tested if they show any potential to be of any use. Again this algorithm is implemented in both Python and C++.

2.2.3 Simulated Data Generation

The algorithm was optimized and tested using simulated data. For this purpose there is a Python script *generateGrouped.py*. All random sampling was performed using *Random* library provided by Python distribution [52]. The input parameters of the script are as follows:

minLength maxLength count size stay_prob skip_prob table.csv

Min and *Max lengths* are the base count limitations for each sequence, *count* is the number of sequences to generate, *size* is the size of *k*-mers for generating data points, *stay* and *skip probabilities* are the probabilities discussed above and *table* contains the means and standard deviations of data points for each *k*-mer.

The actual simulation is relatively simple and uses several simplifications. First, bases are sampled from uniform distribution, so the CG to AT ratio should be around 0.5. Currents are then generated using the *k*-mers and their mean and standard deviation of data points. The stay and skip probabilities are taken into account as well. The table obtained from sample 1 was used for the final simulated dataset, so the data should be relatively similar to the segmented real data.

A sample of 20 sequences with the length between 5,000 and 15,000 base pairs was simulated. Stay probability was set to 0.5 and skip to 0.002 as the real sample set 1 suggested. An example of one sequence is shown in Figure 2.12 and the detail of 50 events is shown in Figure 2.13.

2.2.4 Alignment Implementations

Last required step that needs to be implemented is the alignment algorithm, which was used to measure accuracy of base-calling using the predicted and

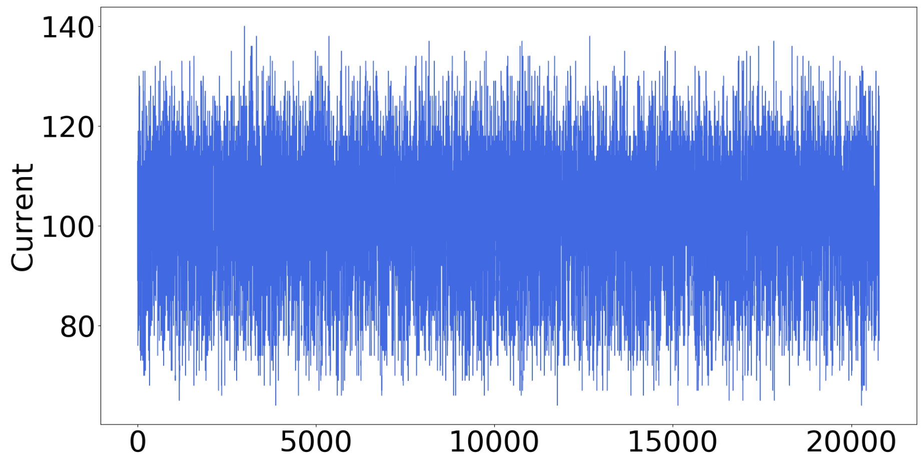

Figure 2.12: All Events of Generated Sequence.

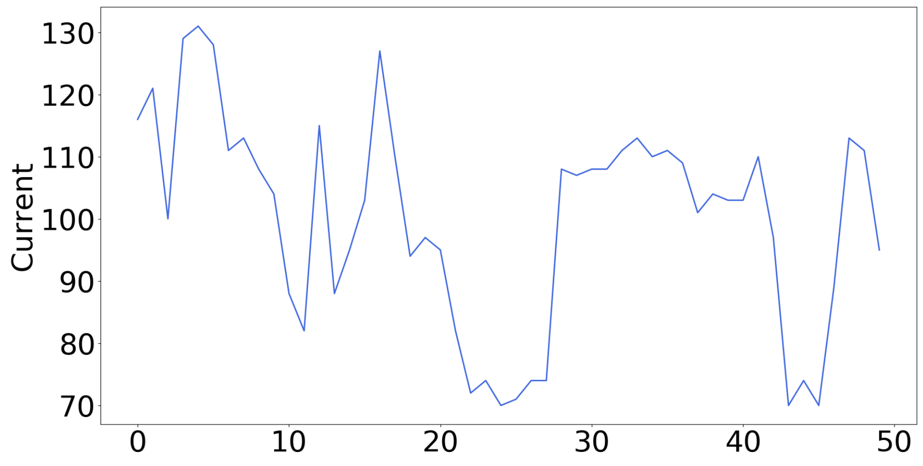

Figure 2.13: 50 Events of Generated Sequence.

reference sequences. For this purpose, three alignment techniques presented in Methods were implemented. Smith–Waterman algorithm as local alignment 1.4.1, Needleman–Wunsch algorithm as global alignment 1.4.2 and modified edit distance 1.4.3. As these methods have similar structure and purpose, they are all in one script called *align.cpp* (*align.py* for Python).

The absolute values returned by the alignment algorithms depend on the sequence length, therefore a normalized value is necessary for better comparison of results. Thus, the alignment value for local and global methods is computed like this:

$$result = \frac{2value}{Reference.length + Predicted.length}$$

and for edit distance:

$$result = \frac{Reference.length + Predicted.length - 2value}{Reference.length + Predicted.length}$$

The difference in formulas is due to the fact, that local and global alignment returns higher values for better alignment. On the other hand edit distances uses a sum of insertions, deletions and substitutions, so the lower the value the better the result. Using these formulas, predictions of different sizes are penalized and the maximum value is 1.

After that a simple script called *checkAcc.py* goes through the resulting alignments and finds minimal, maximal, mean and standard deviation values for each alignment. This will be used in presenting results in the following sections.

2.3 Simulated Data

The simulated data are generated according to the base-calling model presented above. This is a simplification, in real data we expect to observe more complex dependencies. Therefore, higher accuracy was expected. Nevertheless, the simulated data still contain noise, even if pseudo random, so the results should atleast indicate possible accuracy on real data. Data were simulated separately for each context size, so the biggest difference in accuracy in comparison to real data is expected in the 3-mer version.

2.3.1 Results of Viterbi Algorithm

For each algorithm the result is presented in set of figures. Each set shows results computed using local and global alignment and the modified edit distance in rows. Each column is then designated for each tested context size. Standard deviations were also recorded, however as all of the sequences were simulated using the same mechanisms their values were not discernible in the figures. Therefore Figure 2.14 shows only mean values.

Figures show values only for skip probability of 0.002, because changing its value had almost no effect and this value was the default. Results for all tested k-mers are shown in Appendices D, E and F for 3, 4 and 5-mers respectively. The x axis represents the stay probability.

The results improved significantly when increasing the k-mer size from 3 to 4. Already from these simulations it can be concluded that context size of atleast 4 should be taken into account. Difference between results for sizes 4 and 5 is minimal however, and curiously edit distance accuracy even decreases from 4 to 5. As explained later, the computational time depends exponentialy

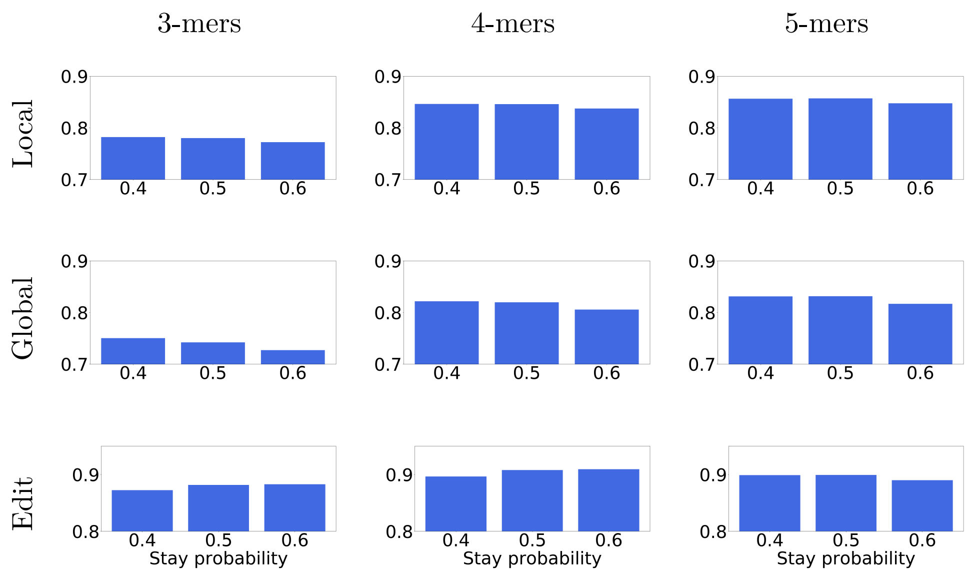

Figure 2.14: Generated Data, Viterbi Algorithm, Skip probability = 0.002

on the size of context. This suggests that using 4-mers may be viable in certain applications.

Also modifying the stay probability parameter changed the accuracy of the prediction and it was not necessarily most accurate for the value that corresponded to the value that the data were generated with. Edit distance showed different trend from local and global alignment. The purpose of edit distance metric is different from other used metrics which caused this disagreement. The edit distance probably uses more technical point of view focusing on frequency of base-call errors.

The main goal of this experiment was to confirm the viability of the implemented Viterbi algorithm and to outline the usability of the three context sizes. Using 3-mers proved significantly worse than the other two sizes, nevertheless it is used in the following tests as well.

2.3.2 Results of Forward–Backward Algorithm

As the Forward–Backward algorithm is usually not used for base-calling purpose, the main goal of this experiment was to show the improvement caused by the two modifications made. Figure 2.15 shows average results for different context sizes. Similarly to Figure 2.14, the three evaluation methods are displayed in rows and the smoothing parameter is shown on the x axis. Blue bars represent results with standard consolidation phase and orange bars represent the usage of the improved version, which removes predicted contexts that does not fit into the sequence as was discussed above. The results are presented in more detail in Appendices G, H and I again for different context sizes. The default values are 0.5 for stay probability and 0.002 for skip probability. Also the rest of the experiments for Forward–Backward algorithm use the same values.

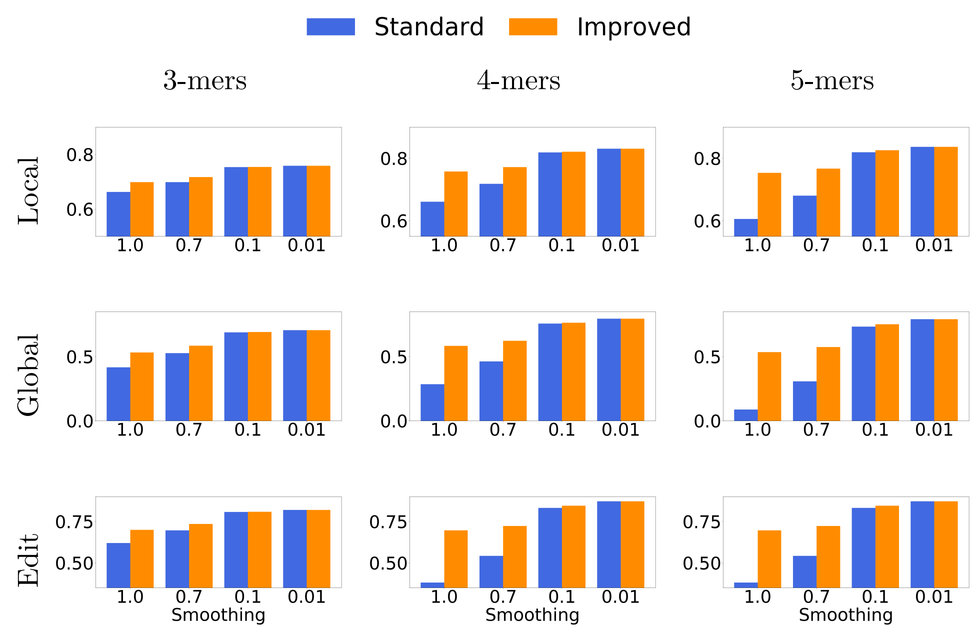

Figure 2.15: Generated Data, Forward–Backward Algorithm

Figure shows the reason this algorithm is not suitable for base-calling as the results without any modification, mainly for 5-mers, are absolutely unusable, with generated sequences much longer than the reference sequence. However, both modifications drastically improve accuracy in every measured situation. Also both modifications improve results by reducing the number of skips. Therefore, using low smoothing value combined with improved consolidation phase does not increase the accuracy over basic consolidation phase.

Even with these modifications, the Forward–Backward algorithm is less accurate than Viterbi algorithm. It also seems that the modifications has reached its limits, because the values for the last smoothing value does not change at all, whether using the improved consolidation or not.

On the other hand, this algorithm was not as sensitive to the context size as was Viterbi algorithm. Even when using 3-mers, the accuracy decrease was not as significant as in Viterbi experiment. Even though the Forward–Backward algorithm is a two pass process, using smaller context still reduces the time complexity of computation significantly.

In summary, the viability of both modifications was confirmed. But even the modified Forward–Backward algorithm does not achieve the accuracy of Viterbi algorithm.

Also the three different metrics for evaluating the accuracy were tested. All three methods can be used for evaluating the results and edit distance seems to use the most appropriate approach. Both local and global alignment search for matching sub-sequences split by insertions or deletions. In this case, compared sequences are expected to have the same length with occasional mismatch. Therefore, only edit distance will be presented in following sections.

2.4 Sample Set 1

In this experiment, predicted sequences were compared with the results of state-of-the-art base-caller. Those reference sequences were provided in *fast5* files and they showed very accurate results. Therefore, if the results are similar to the reference sequences, the base-calls are considered accurate.

A decrease in accuracy was expected, because the data are not just pseudo-randomly generated, but represent real E.coli DNA and also contain unpredictable noise from various sources. However, the means and standard deviations for each pore context should be very precise considering they were recreated specifically from the sequenced data. Other properties of the results should remain similar however, thus the Viterbi algorithm is presented first, because of its higher potential to show good results.

2.4.1 Results of Viterbi Algorithm

Individual average accuracies are again shown in Figure 2.16. As in previous section, only results for skip probability of 0.002 are shown, as the differences were minimal. Detailed results are presented in Appendix J. Again all three context sizes were tested, but only edit distance was used to evaluate the results. The figure also shows standard deviation error bars as in this case its values were high enough to be discernible. This applies for all following figures.

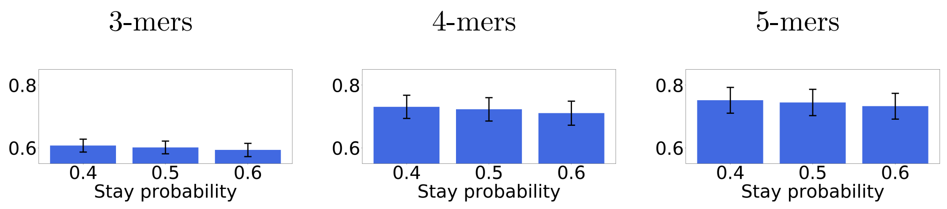

Figure 2.16: Sample 1, Viterbi Algorithm, Skip probability = 0.002

As expected, the usage of 3-mers proved to be least accurate, but the usage of 5-mers now has visibly higher accuracy than using 4-mers. Therefore, the conclusion drawn from simulated data has proven incorrect. In this case the lowest stay probability (0.4) yielded the best results, even though 0.5 corresponds better to the value computed from the reference data.

The overall accuracy decreased by more than 0.1. Most errors are in the form of short "detours", where the algorithm loses the correct way and returns to it in about five more steps. Example using local alignment of this is:

```
...GAATAGCGGG-------TGTG...
...GAAT------GGCAGATTGTG...          (2.1)
```

Edit distance aligns this section as:

```
...GAATAGCGGG-TGTG...
...GAATGGCAGATTGTG...                (2.2)
```

Note that in this example the edit distance uses more desired approach to the penalization of mismatches. It is a sign, that the selection of alignment algorithm was correct.

Overall, the results show that the Viterbi algorithm can achieve similar results to the reference sequence and if using context size of four or more, it can achieve over 0.8 score on some sequences.

2.4.2 Results of Forward–Backward Algorithm

This is the first real test of the improvements made in the last phase of Forward–Backward algorithm so it may prove not worth using and the Viterbi algorithm will stay more accurate as suggested the experiment on simulated data. Figure 2.17 again shows the acquired results using the same placement and the same settings. In this case, detailed results are presented in Appendix K.

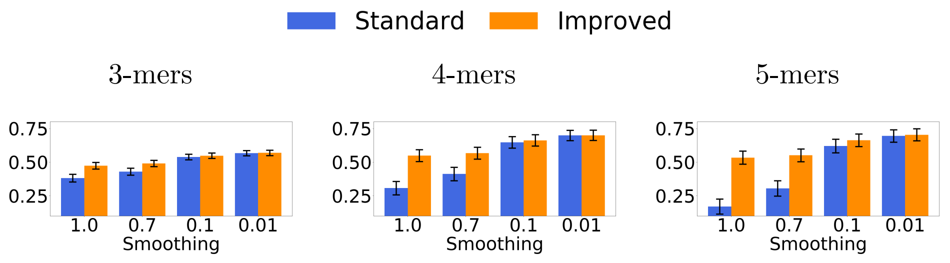

Figure 2.17: Sample 1, Forward–Backward Algorithm

In this experiment, using 3-mers proved significantly worse than 4 or 5-mers. However, other findings are very similar to the simulated dataset. The base algorithm is not worth using at all, but both modifications help significantly. The resulting accuracy is still lower than by using the Viterbi algorithm with the same settings.

As the previous results suggested, Forward–Backward algorithm is not suitable for base-calling. Therefore, here its used only for estimating base-calling quality.

2.5 Sample Set 2

The sample set 2 was used to test the scaling of parameters performed by Min-Max normalization. As in previous experiment, both Viterbi and Forward–Backward algorithms were used. Also, both modifications of Consolidation phase in Forward–Backward algorithms were tested. The emphasis is on Viterbi algorithm as it showed better results in previous experiments.

2.5.1 Results of Viterbi Algorithm

Again, Figure 2.18 is provided using the same settings for stay and skip probabilities. The table of means and standard deviations was normalized as discussed above. Detailed results are presented in Appendix L for 3, 4 and 5-mers.

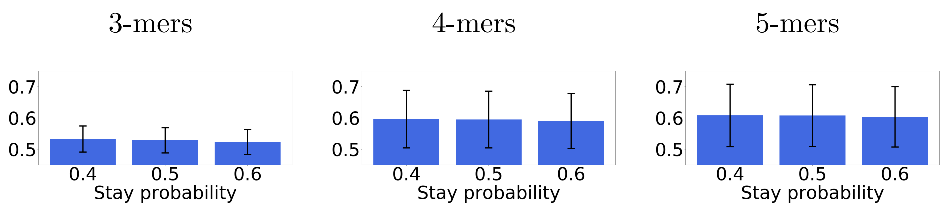

Figure 2.18: Sample 2, Viterbi Algorithm, Skip probability = 0.002

The overall conclusions remain the same. As expected, 5-mers provided best results. However, the mean accuracy decreased significantly. The standard deviation was much higher, because some of the sequences sequences in the set aligned poorly and some were aligned with approximately the same score as the best sequences in sample 1 (over 0.8). The reason for this will be discussed in section 2.7.5.

This test showed that the normalization method is not robust enough and can influence the results significantly even when the normalization was performed on E.coli dataset measured by the same R9 chemistry. Therefore, the normalization should be performed for each read separately as in sample 3.

2.5.2 Results of Forward–Backward Algorithm

Accuracy for Forward–Backward algorithm is shown in Figure 2.19. Detailed results are presented in Appendix M. The settings stayed the same as in the sample 1 experiment, so the only difference are the normalized means and standard deviations for the distributions in the sample set 2.

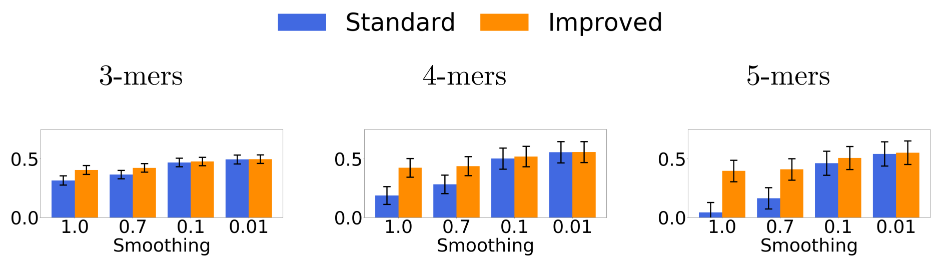

Figure 2.19: Sample 2, Forward–Backward Algorithm

Conclusions made from these results are the same as for Viterbi algorithm. The overall accuracy is lower with significantly higher standard deviations. As in Viterbi algorithm experiment the normalization gave reasonable results for only a part of the sequences and therefore it should be used with caution, especially when applied to data with significant difference in properties as in sample 3.

2.6 Method Selection

The best performing solution is the Viterbi algorithm using context size 5. For used datasets the stay probability of 0.4 and skip probability of 0.001 performs best if the edit distance alignment is taken as the decisive element. Forward–Backward algorithm proved to be too inaccurate even with modifications in the Consolidation phase. Still, both modifications increased the accuracy by a significant amount.

Using smaller context size also reduces the accuracy. With 4-mers the method can achieve decent results and if the computation time is a consideration, it may be a viable option. Using 3-mers is not advised at all, the decrease in accuracy is too big.

In the paragraphs below, this method will be referred to as Nanopore Basecaller Lite (NBL). In all future references, the default settings use contexts of 5 bases, stay probability of 0.4 and skip probability of 0.001.

2.7 Comparison with Existing Methods

This section will compare NBL, the method defined in previous section, with both Nanocall and DeepNano solutions. These two solutions were selected as they are both open source, thus easily obtainable, and claim to provide accurate results comparable to Metrichor. Nanocall method is similar to NBL, so the comparison is at hand. DeepNano on the other hand uses different approach so it serves as a complement to possible solutions. The comparison will be made on both sample set 1 and sample set 2 using edit distance alignment with the reference sequences.

On those samples, the evaluation was based on similarity with state-of-the-art base-caller. Instead, in sample 3 a BWA-MEM alignment of the base-called sequence to the reference (human) genome is performed, which allows to count the number of mismatches, insertions and deletions for all base-callers. Except for natural polymorphism, it should be possible to measure the real error rate of the base-calling.

Sample 1 and 2 came from two distinct experiments, but the sequences in sample 3 are a mixture of multiple experiments performed at different time, place and protocol chosen specifically by their alignment score using BWA-MEM algorithm.

2.7.1 Nanocall Overview

Nanocall is based on Viterbi algorithm as well, however it uses 6-mers as context lengths. It may provide the crucial advantage for Nanocall as the results has shown, that the longer the context the better the results. However, the dataset analysis used only 5-mers as a context so it may not prove as a significant advantage.

2.7.2 DeepNano Overview

DeepNano solution is based on Recurrent Neural Network. Its main advertised advantage is that it may capture dependencies longer than the length of a context. It is certainly true that in the DNA there are those kind of dependencies, however the impact on base-calling is hard to predict.

2.7.3 Burrows–Wheeler Aligner

Burrows–Wheeler Aligner (BWA) [53] is a software package for mapping base-called sequences against a large reference genome, such as the human genome. It consists of three algorithms, but only the MEM algorithm is recommended for this type of project. It works by seeding alignments with maximal exact matches (MEMs) and then extending seeds with the affine-gap Smith–Waterman algorithm.

BWA is widely used for the result assessment. Both Nanocall and DeepNano used it, so it should provide unbiased comparison between all used implementations.

2.7.4 Sample 1 Comparison

On sample 1, NBL has a significant advantage as it was specifically optimized for this dataset. Both Nanocall and DeepNano were used with their default settings, except for specifying R9 chemistry to DeepNano. Comparison of all three solutions is shown in Figure 2.20. The results are shown using violin plot, with mean value in the middle. As in previous figures, the alignments shown are performed by edit distance.

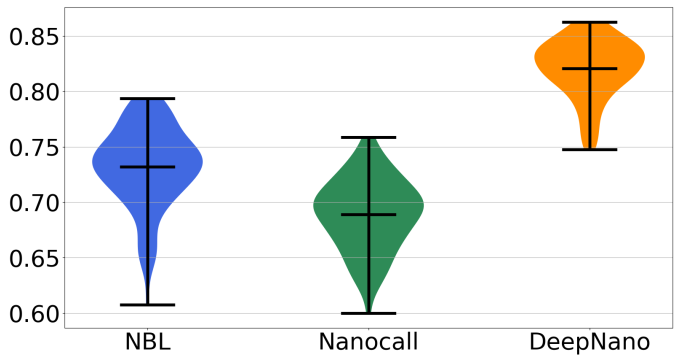

Figure 2.20: Sample 1, Comparison of Results

The results on a 100 randomly selected sequences show, that the implemented version of Viterbi algorithm performs better than the Nanocall solution, although Nanocall is disadvantaged by the need to adjust all settings by itself. DeepNano and its Recurrent Neural Network displays significantly better results, which indicates its advantage over both HMM based methods.

Overall, the RNN proved to be more accurate than HMM models in agreement with [32]. Nanocall has shown lower accuracy in comparison to NBL, however the cause for this is probably the need to use scaled parameters, which is both hard to perform precisely and very important for the resulting accuracy.

2.7.5 Sample 2 Comparison

On sample 2, the advantage of knowing the settings for implemented solution is not so prominent, because the scaling had to be used even in the NBL. Nevertheless, the dataset still contains similar E.coli sequences and was measured by the same chemistry.

Results are shown in the same manner as in previous section in Figure 2.21. Nanocall is again indicated by green color, DeepNano is orange and NBL is blue.

The results show similar performance for DeepNano with overall performance significantly better than both HMM solutions. Nanocall reports lower

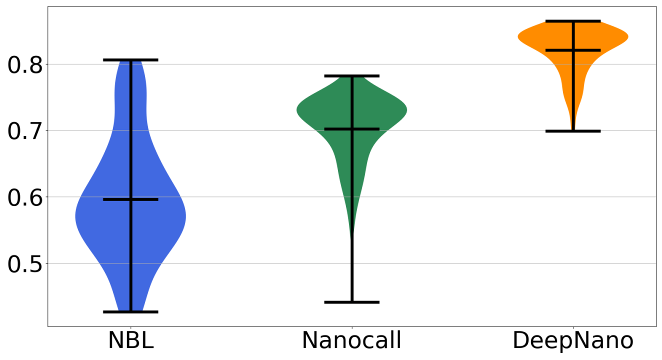

Figure 2.21: Sample 2, Comparison of Results

maximal accuracy than NBL, but in average case reaches higher values. The reason for the decrease in accuracy of NBL is probably due to incorrect use of scaling, where it should be applied for each read separately.

To sum both comparisons, the implemented solution performed well to certain extent, although it was designated exactly for the given dataset. DeepNano solution achieved the best results with the lowest variance. Nanocall is not as accurate, but on the other hand shows stable results on both samples. The numeric values of DeepNano and Nanocall results are shown in Appendix N.

2.7.6 Sample 3 Comparison

Last comparison is performed on sample 3. It does not use the edit distance alignment as was already mentioned. The comparison is made using BWA-MEM algorithm against the real human gene, which was base-called. To avoid the problem with scaling present in sample 2, which may be magnified in this sample because of its heterogeneity, all input parameters were adjusted for each read separately. Figure 2.22 shows error rate for each of the 100 sequences base-called. The error rate is computed from CIGAR string provided in the mapping result.

The CIGAR string is a sequence of base lengths and the associated operation. The main operations are hard clip (H), soft clip (S), insertion (I), deletion (D) and match (M). Clips can occur only at the start and the end of the sequence and signalize, that the aligned sequence had to be shortened to be successfully mapped, thus may be considered as insertions at the ends. An example CIGAR string looks like this:

$$\texttt{10S30M5I1D20M2S} \tag{2.3}$$

It shows a very short alignment with 10 bases clipped at the start and 2 at the end. 50 matches in total and 5 insertions and 1 deletion. The resulting

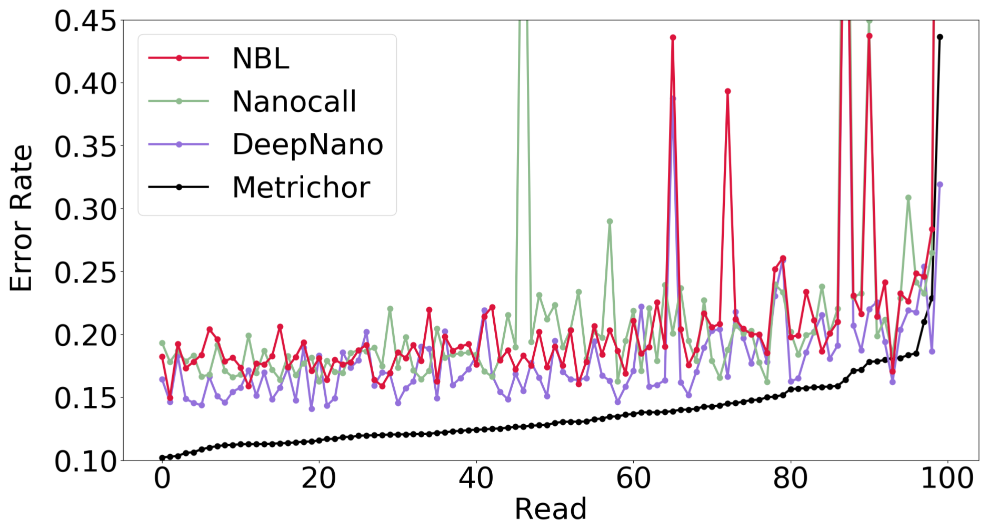

Figure 2.22: Error rate comparison on sample 3 using BWA-MEM.

error rate would be $\frac{10+2+5+1}{30+20} = \frac{18}{50} = 0.36$. Therefore, the error rate is an average number of errors per match.

From the figure, it is visible that Metrichor 1.2.3.1.1 achieves the best result on most sequences. There are several possible reasons. First is that the sequences were selected based on the achieved result by Metrichor, so other sequences could provide more balanced results. Also, Metrichor could utilize some information from complement strand whose measurement is sometimes concatenated to the template strand in the *fast5* file.

Otherwise, if only the three selected solutions are compared, DeepNano solution performs best on most reads. Average values per read are presented in Table 2.2. DeepNano achieves significantly lower error rate than both HMM based solutions. On the other hand, Nanocall creates longer mapped sequences with more than 100 bases difference in matches. The NBL solution does not outperform other solutions in any category, but the results are competitive.

	NBL	Nanocall	DeepNano
Clips	155	144	116
Insertions/Deletions	788	827	732
Matches	4712	4833	4711
Error Rate	0.205	0.209	0.183

Table 2.2: Average BWA-MEM alignment result comparison on sample 3.

The last read shows very high error rate even using Metrichor. All reads were selected using only clip, insertion and deletion count and this read was only several hundred base pairs long, therefore the error rate was affected. In this case both Nanocall and NBL generated a sequence which was not mapped at all. The figure shows value 1 so the other values were still recognizable. All numeric values of error rate are shown in Appendix O.

2.8 Time and Space Complexity

The last comparison between the three solutions is not about performance in base-calling, instead it focuses more on the hardware requirements. The first attribute is time complexity. The main disadvantage of NBL solution is the lack of parallelization. DeepNano uses four threads in the default settings, which is not adjustable by an input parameter. Nanocall can be run using arbitrary number of threads. The disadvantage can be partly evaded by running several instances of NBL, because the base-calling of one sequence does not affect other sequences, although the parallelization requires extra effort. The time complexity of NBL for one read can be interpreted as:

$$T_{NBL} = \mathcal{O}(N_e \times 4^k \times tr),$$

where N_e is number of events, k is the size of context and tr is the number of possible transitions from each state. As was already discussed, the value of tr is up to 21 in current implementation.

Table 2.3 shows performance using events per second (eps) which was achieved on the whole sample 1. All times are measured for the run of the whole process as it was used as a black box. Every run was measured on processor Intel Core i7-4700MQ CPU @ 2.40 GHz × 8.

Tool	Threads	Context	Eps
NBL	1	3-mers	20,287
NBL	1	4-mers	4,333
NBL	1	5-mers	975
Nanocall	1	6-mers	565
Nanocall	4	6-mers	1,513
DeepNano	1	N/A	836
DeepNano	4	N/A	2,824

Table 2.3: Events per second comparison in base-calling of sample 1.

From the table, it is visible that if run on single core without multi-threading, the NBL is the fastest. However, the parallelization speeds the computation by a significant amount. Also as expected, if smaller contexts are used, the number of events per second processed is significantly higher at the cost of lower accuracy.

Another attribute is space complexity. Table 2.4 shows the peak memory usage of each tool during the computation of sample 1 base-calling.

Tool	Context	Kilobytes
NBL	3-mers	18,992
NBL	4-mers	61,980
NBL	5-mers	250,792
Nanocall	6-mers	580,260
DeepNano	N/A	93,872

Table 2.4: Peak memory usage of base-callers on sample 1 in kilobytes.

Again as expected, the values strongly depend on the size of context in NBL solution. And as the DeepNano does not use HMM and dynamic programming, it requires smaller amount of memory unless using very short context size. The memory requirement for NBL can be expressed as:

$$M_{NBL} = \mathcal{O}(N_e \times 4^k),$$

using the same variables as in time complexity. The value 4^k in these formulas is the number of contexts and therefore the number of states of the Hidden Markov Model.

In summary, DeepNano excels in both presented comparisons, unless using very short contexts in NBL. However, the difference in accuracy for shorter contexts is so significant, it is not worth any consideration in most cases. Based only on these parameters, the RNN seems to have only advantages.

Conclusion

DNA sequencing is and probably will be in focus of many researchers, because of its wide variety of applications. With the release of MinION device which uses the nanopore technology and the constant improvements made in its chemistry, computer science oriented researchers may focus on optimizing the base-calling to improve its lacking accuracy, because the speed and lengths of reads achieved by this method provide a significant advantage over any other proposed sequencing method.

And now with the recent publication of Metrichor, other solutions and new methods may be easier to develop and optimize. A lightweight open source tool presented in this work may be useful for either comparison with new solutions or the practical usage in small scale projects. It may help, that it is implemented in both Python and C++ and does not require any specialized tools and only few common libraries so it can be used on almost any machine. Therefore, the goal to create a tool for base-calling sequences was accomplished.

The overall precision is comparable with other solutions like DeepNano or Nanocall, and although it is optimized specifically for R9 chemistry and E.coli sequences, the results were satisfactory even on presumably hard to base-call human gene. Although, the results show the reason why Metrichor stopped using HMMs and went over to the use of RNNs as the DeepNano solution performed better in almost every instance.

The NBL is also implemented as separate modules, therefore it is possible to use only some parts and for example replace the scaling of parameters with more sophisticated methods. This module may be a focus of the future work, as it is responsible for significant accuracy change if used on data measured using different chemistry or DNA.

Another important note is that the experimental use of Forward–Backward algorithm was not as successful as desired, even though both implemented modifications increased the accuracy by a large amount.

All of the implemented modules are provided on the enclosed flash drive or available at `https://bitbucket.org/horakto1/NBL`.

Bibliography

[1] Holub, J. 12. Pattern Matching in Bioinformatics. 2016, course MI-EVY, FIT CTU University Lecture.

[2] Mohammed, O. G.; Assaleh, K. T.; et al. Novel algorithms for accurate DNA base-calling. *Journal of Biomedical Science and Engineering*, volume 6, no. 2, 2013: pp. 165–174.

[3] Ronaghi, M. Pyrosequencing sheds light on DNA sequencing. *Genome research*, volume 11, no. 1, 2001: pp. 3–11.

[4] Applied Biosystems. SOLiD System Brochure. 2008, [Online; accessed 20-05-17]. Available from: `http://www.columbia.edu/cu/biology/courses/w3034/Dan/readings/SOLiD_System_Brochure.pdf`

[5] Snipcademy. 09. Illumina Sequencing-By-Synthesis (SBS) Technology. [Online; accessed 20-05-17]. Available from: `https://binf.snipcademy.com/lessons/ngs-techniques/illumina-solexa`

[6] Liang, F.; Zhang, P. Nanopore DNA sequencing: Are we there yet? *Science Bulletin*, volume 60, no. 3, 2015: pp. 296–303.

[7] Szalay, T.; Golovchenko, J. A. De novo sequencing and variant calling with nanopores using PoreSeq. *NATURE BIOTECHNOLOGY*, volume 33, no. 10, 2015: pp. 1087–1087.

[8] Lou, H. L. Implementing the Viterbi algorithm. *IEEE Signal Processing Magazine*, volume 12, no. 5, Sep 1995: pp. 42–52, ISSN 1053-5888, doi: 10.1109/79.410439.

[9] Khreich, W.; Granger, E.; et al. On the memory complexity of the forward–backward algorithm. *Pattern Recognition Letters*, volume 31, no. 2, 2010: pp. 91–99.

[10] Mott, R. Smith–Waterman Algorithm. *eLS*, 2005.

[11] Likic, V. The Needleman–Wunsch algorithm for sequence alignment. *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne*, 2008: pp. 1–46.

[12] Sinden, R. R. *DNA structure and function.* Elsevier, 2012.

[13] Pray, L. Discovery of DNA structure and function: Watson and Crick. *Nature Education*, volume 1, no. 1, 2008: p. 100.

[14] Maxam, A. M.; Gilbert, W. A New Method for Sequencing DNA. *Proceedings of the National Academy of Sciences of the United States of America*, volume 74, no. 2, 1977: pp. 560–564.

[15] Sanger, F.; Nicklen, S.; et al. DNA Sequencing with Chain-Terminating Inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, volume 74, no. 12, 1977: pp. 5463–5467.

[16] Giddings, M. C.; Brumley, R. L., Jr; et al. An adaptive, object oriented strategy for base calling in DNA sequence analysis. *Nucleic Acids Research*, volume 21, no. 19, 1993: p. 4530, doi:10.1093/nar/21.19.4530. Available from: `http://dx.doi.org/10.1093/nar/21.19.4530`

[17] Berno, A. J. A graph theoretic approach to the analysis of DNA sequencing data. *Genome Research*, volume 6, no. 2, 1996: pp. 80–91.

[18] Brady, D.; Kocic, M.; et al. A maximum–likelihood base caller for DNA sequencing. *IEEE transactions on biomedical engineering*, volume 47, no. 9, 2000: pp. 1271–1280.

[19] Manolakos, E. S. Clustering methods for accurate DNA base-calling. In *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*, volume 1, IEEE, 2002, ISBN 1058-6393, pp. 311–315.

[20] Khan, O. G. M.; Assaleh, K. T.; et al. DNA base-calling using artificial neural networks. In *Biomedical Engineering (MECBME), 2011 1st Middle East Conference on*, IEEE, 2011, pp. 96–99.

[21] Pereira, M. S.; Andrade, L.; et al. Statistical learning formulation of the DNA base-calling problem and its solution in a Bayesian EM framework. *Discrete Applied Mathematics*, volume 104, no. 1, 2000: pp. 229–258.

[22] Mohammed, O. G.; Assaleh, K. T.; et al. DNA base-calling using polynomial classifiers. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, IEEE, 2010, pp. 1–5.

[23] Thornley, D.; Petridis, S. Decoding Trace Peak Behaviour—A Neuro–Fuzzy Approach. In *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, IEEE, 2007, ISBN 1098-7584, pp. 1–6.

[24] Eltoukhy, H.; El Gamal, A. Modeling and base-calling for DNA sequencing-by-synthesis. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, IEEE, 2006, ISBN 1520-6149, pp. II–II.

[25] McKernan, K.; Blanchard, A.; et al. Reagents, methods, and libraries for bead-based sequencing. Apr. 30 2013, uS Patent 8,431,691.

[26] Ju, J.; Kim, D. H.; et al. Four-color DNA sequencing by synthesis using cleavable fluorescent nucleotide reversible terminators. *Proceedings of the National Academy of Sciences*, volume 103, no. 52, 2006: pp. 19635–19640.

[27] Illumina Inc. Sequencing by Synthesis (SBS) Technology. [Online; accessed 20-05-17]. Available from: `https://www.illumina.com/technology/next-generation-sequencing/sequencing-technology.html`

[28] Kasianowicz, J. J.; Brandin, E.; et al. Characterization of individual polynucleotide molecules using a membrane channel. *Proceedings of the National Academy of Sciences*, volume 93, no. 24, 1996: pp. 13770–13773.

[29] David, M.; Dursi, L. J.; et al. Nanocall: an open source basecaller for Oxford Nanopore sequencing data. *Bioinformatics*, volume 33, no. 1, 2017: p. 49, doi:10.1093/bioinformatics/btw569. Available from: `http://dx.doi.org/10.1093/bioinformatics/btw569`

[30] Schreiber, J.; Karplus, K. Analysis of nanopore data using hidden Markov models. *Bioinformatics*, 2015: p. btv046.

[31] An Oxford Nanopore Company. Metrichor. `https://metrichor.com`, [Online; accessed 29-04-17].

[32] Boža, V.; Brejová, B.; et al. DeepNano: deep recurrent neural networks for base calling in MinION nanopore reads. *arXiv preprint arXiv:1603.09195*, 2016.

[33] Loman, N. J.; Quick, J.; et al. A complete bacterial genome assembled de novo using only nanopore sequencing data. *NATURE METHODS*, volume 12, no. 8, 2015: pp. 733–U51.

[34] Eddy, S. R. Hidden markov models. *Current Opinion in Structural Biology*, volume 6, no. 3, 1996: pp. 361–365.

[35] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, volume 77, no. 2, Feb 1989: pp. 257–286, ISSN 0018-9219, doi:10.1109/5.18626.

[36] Rabiner, L. R.; Juang, B.-H. An introduction to hidden Markov models. *IEEE ASSP Magazine*, volume 3, no. 1, Jan 1986: pp. 4–16, ISSN 0740-7467, doi:10.1109/MASSP.1986.1165342.

[37] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, volume 13, no. 2, April 1967: pp. 260–269, ISSN 0018-9448, doi:10.1109/TIT.1967.1054010.

[38] Forney, G. D. The viterbi algorithm. *Proceedings of the IEEE*, volume 61, no. 3, March 1973: pp. 268–278, ISSN 0018-9219, doi:10.1109/PROC.1973.9030.

[39] Shinghal, R.; Toussaint, G. T. Experiments in Text Recognition with the Modified Viterbi Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume PAMI-1, no. 2, 1979;1978;: pp. 184–193.

[40] Chang, R.; Hancock, J. On receiver structures for channels having memory. *IEEE Transactions on Information Theory*, volume 12, no. 4, 1966: pp. 463–468.

[41] Baum, L. E.; Petrie, T.; et al. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, volume 41, no. 1, 1970: pp. 164–171.

[42] Smith, T. F.; Waterman, M. S. Identification of common molecular subsequences. *Journal of molecular biology*, volume 147, no. 1, 1981: pp. 195–197.

[43] Gotoh, O. An improved algorithm for matching biological sequences. *Journal of molecular biology*, volume 162, no. 3, 1982: pp. 705–708.

[44] Will, S. Sequence Alignment. 2011, course 18.417, MIT University Lecture. Available from: `http://math.mit.edu/classes/18.417/Slides/alignment.pdf`

[45] Needleman, S. B.; Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, volume 48, no. 3, 1970: pp. 443–453.

[46] Ukkonen, E. On approximate string matching. In *Foundations of Computation Theory*, Springer, 1983, pp. 487–495.

[47] Loman Labs. Nanopore R9 rapid run data release. 7 2016, [Online; accessed 29-04-17]. Available from: `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`

[48] Loman, N. J.; Quinlan, A. R. Poretools: a toolkit for analyzing nanopore sequence data. *Bioinformatics*, volume 30, no. 23, 2014: p. 3399, doi:10.1093/bioinformatics/btu555. Available from: `http://dx.doi.org/10.1093/bioinformatics/btu555`

[49] Collette, A. HDF5 for Python. 2008–, [Online; accessed 29-04-17]. Available from: `http://h5py.alfven.org`

[50] Timp, W.; Comer, J.; et al. DNA base-calling from a nanopore using a Viterbi algorithm. *Biophysical journal*, volume 102, no. 10, 2012: pp. L37–L39.

[51] Python Software Foundations. Python Laguage Reference, version 2.7. 2010–. Available from: `https://www.python.org/`

[52] Victor, S. Generate pseudo-random numbers. 1990–, [Online; accessed 29-04-17]. Available from: `https://docs.python.org/2/library/random.html`

[53] Li, H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

[54] Oliphant, T.; et al. NumPy. 2006–, [Online; accessed 29-04-17]. Available from: `http://www.numpy.org/`

[55] Jones, E.; Oliphant, T.; et al. SciPy: Open source scientific tools for Python. 2001–, [Online; accessed 29-04-17]. Available from: `http://www.scipy.org/`

[56] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, volume 9, no. 3, 2007: pp. 90–95, doi:10.1109/MCSE.2007.55.

[57] C++ Standards Committee; et al. iso/iec 14882: 2011, standard for programming language c++. Technical report, ISO/IEC, 2011. Available from: `http://www.open-std.org/jtc1/sc22/wg21`

Appendix A

Acronyms

A Adenine

ANN Artificial Neural Network

ATP Adenosine triphosphate

BWA Burrows–Wheeler Aligner

C Cytosine

CPU Central Processing Unit

DN DeepNano

DNA Deoxyribonucleic Acid

E.coli Escherichia coli

EM Expectation–Maximization Algorithm

eps Events per second

FB Forward–Backward Algorithm

FwdBkw Forward–Backward Algorithm

G Guanine

HMM Hidden Markov Model

MEM Maximal exact matches algorithm

MM Markov Model

NBL Nanopore Base-caller Lite

NC Nanocall

NW	Needleman–Wunsch Algorithm
PC	Polynomial Classifiers
PPi	Pyrophosphate
RNN	Recurrent Neural Network
SOLiD	Sequencing by oligonucleotide ligation and detection
SW	Smith–Waterman Algorithm
SWG	Smith–Waterman–Gotoh Algorithm
T	Thymine

Appendix **B**

Contents of Enclosed Drive

```
README.txt ................ the file with flash drive contents description
data ................................. the directory with sample data
  simulated
  sample1
  sample2
exe ............................ the directory with executables (C++)
srcCpp ............................ the directory of C++ source codes
srcPy ............................ the directory of Python source codes
srcThesis ............ the directory of LaTeX source codes of the thesis
thesis.pdf ........................... the thesis text in PDF format
```

Appendix C

Data File Structure

```
File
└── Analyses
    ├── Basecall_1D_000
    │   ├── BaseCalled_template
    │   │   ├── Events (Dataset)
    │   │   └── Fastq (Dataset)
    │   ├── Configuration
    │   │   ├── aggregator
    │   │   ├── basecall_1d
    │   │   ├── calibration_strand
    │   │   ├── components
    │   │   ├── event_detection
    │   │   ├── general
    │   │   └── split_hairpin
    │   ├── Log (Dataset)
    │   └── Summary
    │       └── basecall_1d_template
    ├── Calibration_Strand_000
    │   ├── Configuration
    │   │   ├── aggregator
    │   │   ├── basecall_1d
    │   │   ├── basecall_2d
    │   │   ├── calibration_strand
    │   │   ├── components
    │   │   ├── general
    │   │   ├── genome_mapping
    │   │   ├── hairpin_align
    │   │   ├── post_processing.3000Hz
    │   │   └── split_hairpin
    │   ├── Log (Dataset)
    │   └── Summary
    └──
└──
```

```
│   ├── EventDetection_000
│   │   ├── Configuration
│   │   │   ├── aggregator
│   │   │   ├── basecall_1d
│   │   │   ├── basecall_2d
│   │   │   ├── calibration_strand
│   │   │   ├── components
│   │   │   ├── event_detection
│   │   │   ├── general
│   │   │   ├── hairpin_align
│   │   │   ├── post_processing
│   │   │   ├── post_processing.4000Hz
│   │   │   └── split_hairpin
│   │   ├── Log (Dataset)
│   │   ├── Reads
│   │   │   └── Read_i
│   │   │       └── Events (Dataset)
│   │   └── Summary
│   │       └── event_detection
│   └── Segment_Linear_000
│       ├── Configuration
│       │   ├── aggregator
│       │   ├── basecall_1d
│       │   ├── calibration_strand
│       │   ├── components
│       │   ├── general
│       │   └── split_hairpin
│       ├── Log (Dataset)
│       └── Summary
│           └── split_hairpin
├── Raw
│   └── Reads
│       └── Read_i
│           └── Signal (Dataset)
└── UniqueGlobalKey
    ├── channel_id
    ├── context_tags
    └── tracking_id
```

Appendix D

Generated, Viterbi A., 3-mers

Alignment	Stay	Skip	Minimum	Average	Maximum
Local	0.4	0.001	0.763	**0.782**	0.793
		0.002	0.764	**0.782**	0.793
		0.004	0.764	**0.782**	**0.794**
	0.5	0.001	0.766	0.780	0.793
		0.002	**0.767**	0.780	0.793
		0.004	**0.767**	0.780	**0.794**
	0.6	0.001	0.760	0.772	0.781
		0.002	0.761	0.772	0.781
		0.004	0.761	0.772	0.782
Global	0.4	0.001	0.727	**0.750**	0.793
		0.002	**0.728**	**0.750**	0.793
		0.004	0.727	**0.750**	**0.794**
	0.5	0.001	0.722	0.742	0.793
		0.002	0.722	0.742	0.793
		0.004	0.723	0.742	**0.794**
	0.6	0.001	0.707	0.727	0.781
		0.002	0.708	0.727	0.781
		0.004	0.709	0.727	0.782
Edit	0.4	0.001	0.857	0.872	0.881
		0.002	0.858	0.872	0.881
		0.004	0.858	0.872	0.881
	0.5	0.001	0.870	0.881	**0.891**
		0.002	0.871	0.881	**0.891**
		0.004	0.871	0.881	**0.891**
	0.6	0.001	0.872	**0.883**	0.889
		0.002	**0.873**	**0.883**	0.889
		0.004	**0.873**	0.882	0.889

Appendix **E**

Generated, Viterbi A., 4-mers

Alignment	Stay	Skip	Minimum	Average	Maximum
Local	0.4	0.001	**0.831**	**0.846**	0.863
		0.002	**0.831**	**0.846**	0.863
		0.004	**0.831**	**0.846**	0.863
	0.5	0.001	0.827	**0.846**	**0.865**
		0.002	0.827	**0.846**	**0.865**
		0.004	0.828	**0.846**	0.864
	0.6	0.001	0.821	0.837	0.856
		0.002	0.823	0.837	0.856
		0.004	0.823	0.837	0.855
Global	0.4	0.001	**0.801**	0.821	0.863
		0.002	**0.801**	**0.822**	0.863
		0.004	**0.801**	0.821	0.863
	0.5	0.001	0.799	0.820	0.865
		0.002	0.799	0.820	**0.865**
		0.004	**0.801**	0.820	0.864
	0.6	0.001	0.788	0.806	0.856
		0.002	0.790	0.806	0.856
		0.004	0.791	0.805	0.855
Edit	0.4	0.001	0.884	0.896	0.912
		0.002	0.884	0.897	0.912
		0.004	0.884	0.896	0.911
	0.5	0.001	0.895	0.908	0.918
		0.002	0.895	0.908	0.918
		0.004	0.896	0.908	0.917
	0.6	0.001	0.899	**0.909**	**0.922**
		0.002	**0.900**	**0.909**	**0.922**
		0.004	0.899	**0.909**	0.921

Appendix F

Generated, Viterbi A., 5-mers

Alignment	Stay	Skip	Minimum	Average	Maximum
Local	0.4	0.001	**0.839**	0.856	0.869
		0.002	0.838	0.856	0.870
		0.004	0.838	0.856	0.869
	0.5	0.001	0.838	**0.857**	**0.872**
		0.002	0.838	**0.857**	0.871
		0.004	0.838	**0.857**	0.871
	0.6	0.001	0.829	0.848	0.860
		0.002	0.829	0.848	0.860
		0.004	0.828	0.847	0.860
Global	0.4	0.001	**0.810**	0.831	0.869
		0.002	0.809	0.831	0.870
		0.004	0.809	0.831	0.869
	0.5	0.001	0.807	**0.832**	**0.872**
		0.002	0.807	**0.832**	0.871
		0.004	0.806	0.831	0.871
	0.6	0.001	0.792	0.817	0.860
		0.002	0.793	0.817	0.860
		0.004	0.791	0.816	0.860
Edit	0.4	0.001	**0.887**	**0.899**	0.908
		0.002	0.886	**0.899**	0.908
		0.004	0.886	**0.899**	0.908
	0.5	0.001	0.885	**0.899**	**0.909**
		0.002	0.885	**0.899**	**0.909**
		0.004	0.885	**0.899**	0.908
	0.6	0.001	0.876	0.890	0.900
		0.002	0.876	0.890	0.900
		0.004	0.876	0.889	0.900

Appendix G

Generated, FwdBkw A., 3-mers

Alignment	Smoothing	Consolidate	Minimum	Average	Maximum
Local	1.00	yes	0.683	0.698	0.713
		no	0.648	0.662	0.675
	0.70	yes	0.697	0.717	0.730
		no	0.680	0.698	0.711
	0.10	yes	0.738	0.754	0.762
		no	0.738	0.753	0.762
	0.01	yes	**0.745**	**0.759**	**0.767**
		no	**0.745**	**0.759**	**0.767**
Global	1.00	yes	0.508	0.531	0.713
		no	0.396	0.416	0.675
	0.70	yes	0.548	0.584	0.730
		no	0.495	0.527	0.711
	0.10	yes	0.668	0.690	0.762
		no	0.666	0.688	0.762
	0.01	yes	**0.686**	**0.706**	**0.767**
		no	**0.686**	**0.706**	**0.767**
Edit	1.00	yes	0.684	0.699	0.711
		no	0.607	0.620	0.636
	0.70	yes	0.711	0.735	0.748
		no	0.675	0.696	0.712
	0.10	yes	0.795	0.809	0.819
		no	0.794	0.807	0.819
	0.01	yes	**0.808**	**0.820**	**0.830**
		no	**0.808**	**0.820**	**0.830**

Appendix **H**

Generated, FwdBkw A., 4-mers

Alignment	Smoothing	Consolidate	Minimum	Average	Maximum
Local	1.00	yes	0.744	0.757	0.771
		no	0.644	0.661	0.678
	0.70	yes	0.761	0.772	0.786
		no	0.704	0.718	0.735
	0.10	yes	0.808	0.821	0.840
		no	0.807	0.819	0.836
	0.01	yes	**0.818**	**0.831**	**0.847**
		no	**0.818**	**0.831**	**0.847**
Global	1.00	yes	0.562	0.583	0.771
		no	0.243	0.285	0.678
	0.70	yes	0.610	0.623	0.786
		no	0.432	0.462	0.735
	0.10	yes	0.745	0.763	0.840
		no	0.742	0.757	0.836
	0.01	yes	**0.777**	**0.794**	**0.847**
		no	**0.777**	**0.794**	**0.847**
Edit	1.00	yes	0.717	0.730	0.742
		no	0.500	0.527	0.552
	0.70	yes	0.749	0.757	0.771
		no	0.627	0.647	0.671
	0.10	yes	0.843	0.854	0.869
		no	0.841	0.849	0.863
	0.01	yes	**0.865**	**0.875**	**0.890**
		no	**0.865**	**0.875**	**0.890**

Appendix I

Generated, FwdBkw A., 5-mers

Alignment	Smoothing	Consolidate	Minimum	Average	Maximum
Local	1.00	yes	0.735	0.753	0.775
		no	0.577	0.605	0.634
	0.70	yes	0.749	0.767	0.788
		no	0.657	0.680	0.708
	0.10	yes	0.812	0.826	0.841
		no	0.805	0.819	0.835
	0.01	yes	**0.824**	**0.837**	**0.851**
		no	**0.824**	**0.837**	**0.851**
Global	1.00	yes	0.499	0.535	0.775
		no	0.067	0.087	0.634
	0.70	yes	0.546	0.574	0.788
		no	0.263	0.308	0.708
	0.10	yes	0.733	0.752	0.841
		no	0.715	0.733	0.835
	0.01	yes	**0.775**	**0.791**	**0.851**
		no	**0.775**	**0.791**	**0.851**
Edit	1.00	yes	0.673	0.696	0.727
		no	0.336	0.382	0.432
	0.70	yes	0.706	0.723	0.753
		no	0.514	0.542	0.586
	0.10	yes	0.833	0.845	0.859
		no	0.821	0.832	0.846
	0.01	yes	**0.862**	**0.872**	**0.882**
		no	**0.862**	**0.872**	**0.882**

Appendix J

Sample 1, Viterbi Algorithm

Context	Stay	Skip	Minimum	Average	Maximum
3-mers	0.4	0.001	**0.547**	**0.608**	**0.650**
		0.002	0.545	0.607	0.647
		0.004	0.541	0.605	0.642
	0.5	0.001	0.544	0.602	0.649
		0.002	0.543	0.601	0.638
		0.004	0.539	0.599	0.638
	0.6	0.001	0.538	0.593	0.645
		0.002	0.538	0.593	0.640
		0.004	0.536	0.592	0.637
4-mers	0.4	0.001	**0.607**	**0.731**	**0.793**
		0.002	0.597	0.730	0.791
		0.004	0.590	0.729	0.791
	0.5	0.001	0.604	0.723	0.787
		0.002	0.597	0.722	0.786
		0.004	0.591	0.721	0.785
	0.6	0.001	0.589	0.711	0.778
		0.002	0.588	0.710	0.777
		0.004	0.583	0.708	0.775
5-mers	0.4	0.001	**0.608**	**0.752**	**0.823**
		0.002	**0.608**	0.751	**0.823**
		0.004	0.600	0.750	**0.823**
	0.5	0.001	0.606	0.745	0.819
		0.002	0.599	0.744	0.820
		0.004	0.594	0.743	0.819
	0.6	0.001	0.597	0.733	0.809
		0.002	0.590	0.732	0.809
		0.004	0.585	0.730	0.807

Appendix K

Sample 1, FwdBkw Algorithm

Context	Smoothing	Consolidate	Minimum	Average	Maximum
3-mers	1.00	yes	0.375	0.473	0.511
		no	0.269	0.380	0.436
	0.70	yes	0.393	0.490	0.528
		no	0.324	0.429	0.476
	0.10	yes	0.465	0.547	0.578
		no	0.455	0.538	0.571
	0.01	yes	**0.492**	**0.568**	**0.600**
		no	0.489	0.567	**0.600**
4-mers	1.00	yes	0.385	0.549	0.629
		no	0.156	0.307	0.411
	0.70	yes	0.407	0.566	0.643
		no	0.254	0.412	0.520
	0.10	yes	0.498	0.661	0.748
		no	0.487	0.646	0.739
	0.01	yes	**0.554**	**0.699**	**0.764**
		no	**0.554**	0.698	**0.764**
5-mers	1.00	yes	0.363	0.533	0.641
		no	0.029	0.170	0.294
	0.70	yes	0.389	0.550	0.636
		no	0.103	0.305	0.417
	0.10	yes	0.490	0.662	0.740
		no	0.429	0.620	0.707
	0.01	yes	**0.533**	**0.703**	**0.782**
		no	0.521	0.694	0.773

Appendix L

Sample 2, Viterbi Algorithm

Context	Stay	Skip	Minimum	Average	Maximum
3-mers	0.4	0.001	0.470	0.532	0.633
		0.002	0.470	0.532	0.629
		0.004	0.469	0.531	0.626
	0.5	0.001	0.467	0.528	0.626
		0.002	0.470	0.529	0.624
		0.004	0.469	0.529	0.624
	0.6	0.001	0.459	0.523	0.620
		0.002	0.461	0.524	0.622
		0.004	0.464	0.524	0.620
4-mers	0.36	0.001	0.426	0.596	0.806
		0.002	0.424	0.594	0.804
		0.004	0.420	0.590	0.798
	0.46	0.001	0.437	0.595	0.807
		0.002	0.434	0.593	0.801
		0.004	0.434	0.590	0.798
	0.56	0.001	0.443	0.590	0.794
		0.002	0.441	0.588	0.795
		0.004	0.439	0.585	0.794
Edit	0.36	0.001	0.432	0.608	0.843
		0.002	0.431	0.604	0.842
		0.004	0.427	0.600	0.834
	0.46	0.001	0.442	0.607	0.847
		0.002	0.439	0.605	0.848
		0.004	0.436	0.601	0.845
	0.56	0.001	0.448	0.603	0.836
		0.002	0.446	0.600	0.837
		0.004	0.444	0.597	0.835

Appendix M

Sample 2, FwdBkw Algorithm

Context	Smoothing	Consolidate	Minimum	Average	Maximum
5-mers	1.00	yes	0.279	0.405	0.495
		no	0.183	0.316	0.401
	0.70	yes	0.318	0.423	0.509
		no	0.257	0.366	0.449
	0.10	yes	0.403	0.478	0.564
		no	0.383	0.469	0.555
	0.01	yes	**0.417**	**0.497**	**0.585**
		no	0.406	0.495	0.584
4-mers	1.00	yes	0.245	0.424	0.616
		no	0.010	0.187	0.378
	0.70	yes	0.265	0.438	0.633
		no	0.118	0.282	0.483
	0.10	yes	0.368	0.519	0.719
		no	0.342	0.502	0.710
	0.01	yes	**0.400**	**0.557**	**0.760**
		no	0.389	0.556	**0.760**
5-mers	1.00	yes	0.190	0.396	0.626
		no	0.162	0.043	0.276
	0.70	yes	0.218	0.410	0.646
		no	0.018	0.164	0.420
	0.10	yes	0.330	0.508	0.751
		no	0.274	0.462	0.712
	0.01	yes	**0.372**	**0.552**	**0.796**
		no	0.348	0.542	0.792

Appendix N

Nanocall and DeepNano, Sample 1 and 2 Results

Nanocall				
	Minimum	Average	Maximum	Standard Deviation
Sample 1	0.599	0.688	0.758	0.0323
Sample 2	0.441	0.702	0.781	0.0544

DeepNano				
	Minimum	Average	Maximum	Standard Deviation
Sample 1	0.747	0.820	0.862	0.0264
Sample 2	0.699	0.820	0.864	0.0357

Appendix O

Sample 3, Error Rate

Channel	Read	NBL	Nanocall	DeepNano	Metrichor
1	137	0.18191	0.16754	**0.14761**	0.11414
10	34	0.16398	0.17889	**0.14342**	0.11683
12	37	**0.15882**	0.19888	0.17136	0.11272
15	23	0.20556	**0.17910**	0.20288	0.14270
17	414	0.20065	0.20078	**0.18056**	0.15842
20	114	0.19372	**0.17709**	0.18946	0.11462
24	499	0.18470	**0.17109**	0.22191	0.13796
49	73	0.19018	0.23915	**0.16356**	0.13824
84	389	0.20404	0.16791	**0.16630**	0.11002
85	512	0.20443	0.20095	**0.19699**	0.14645
104	373	0.23078	0.22951	**0.20689**	0.17116
104	401	0.19799	0.20200	**0.16253**	0.15638
104	628	0.21605	0.23250	**0.18716**	0.17169
110	11	0.43600	**0.20083**	0.38763	0.13897
110	214	0.18143	0.16581	**0.15440**	0.11210
110	425	0.18535	**0.16244**	0.17827	0.14992
110	494	0.24131	0.21182	**0.19392**	0.17962
110	517	0.17374	0.16807	**0.15778**	0.11268
115	81	0.17864	0.17120	**0.14571**	0.11192
115	504	0.17231	0.19004	**0.16855**	0.12648
117	271	0.17077	0.18405	**0.16235**	0.18109
117	317	0.18790	0.17879	**0.17020**	0.14081
123	381	0.18726	0.18736	**0.17928**	0.11936
131	74	0.18994	0.22087	**0.15831**	0.13801
136	265	0.23393	0.19965	**0.18572**	0.15734
139	524	0.17307	0.17866	**0.14870**	0.10578

Channel	Read	NBL	Nanocall	DeepNano	Metrichor
141	121	0.24837	0.24138	**0.21770**	0.18492
144	391	0.21677	0.22697	**0.18956**	0.14266
146	130	**0.18642**	0.23814	0.21520	0.15820
151	80	0.19030	0.18479	**0.16536**	0.12322
157	401	0.28354	0.26462	**0.18632**	0.22860
163	104	0.17611	**0.16950**	0.18577	0.11838
172	28	0.18710	0.16250	**0.14632**	0.13477
173	306	0.20007	0.20293	**0.17673**	0.14771
177	152	0.16248	0.20434	**0.14930**	0.12180
177	159	**0.19037**	0.22333	0.19498	0.12947
186	136	**0.16052**	0.23372	0.16381	0.13060
195	221	0.17695	0.16917	**0.15132**	0.11273
195	429	**0.17599**	0.18420	0.18265	0.12418
198	105	0.17791	0.18514	**0.17371**	0.11847
198	345	0.18254	0.17207	**0.14836**	0.11301
199	186	0.17833	0.18053	**0.16534**	0.13067
213	140	0.17392	0.18291	**0.17309**	0.11354
215	13	0.17599	0.18694	**0.16968**	0.11279
218	70	0.18374	0.16643	**0.14363**	0.10863
218	334	0.21966	**0.17103**	0.18876	0.12087
219	185	0.19627	0.19146	**0.15088**	0.11114
227	431	0.22532	0.17893	**0.15970**	0.13803
227	433	0.17378	0.21213	**0.15091**	0.12808
230	177	0.17514	0.18984	**0.17035**	0.13047
231	190	0.18311	0.68662	**0.15494**	0.12669
236	13	N/A	N/A	**0.31904**	0.43636
247	371	0.21429	**0.17073**	0.21917	0.12437
247	496	0.18387	0.19750	**0.16723**	0.13307
248	492	0.20185	0.23140	**0.16568**	0.12773
252	460	**0.64618**	0.77397	0.66107	0.16394
261	519	**0.17525**	0.19401	0.17746	0.12734
262	212	0.43739	0.44922	**0.22008**	0.17821
274	177	0.25162	0.23972	**0.23034**	0.15044
285	11	0.18215	0.19321	**0.16417**	0.10216
285	298	0.18123	**0.16273**	0.18311	0.11567
291	1185	0.23245	0.22859	**0.20386**	0.18128
292	3	0.20007	**0.17775**	0.19949	0.14798

Channel	Read	NBL	Nanocall	DeepNano	Metrichor
292	164	0.19137	0.17162	**0.16249**	0.12069
292	396	0.18720	0.21531	**0.14858**	0.12575
293	197	0.17091	0.18162	**0.14093**	0.11484
296	562	0.19913	0.18386	**0.16497**	0.15680
298	237	0.17810	0.18318	**0.14557**	0.10625
301	13	0.20421	0.23654	**0.16200**	0.14003
303	230	0.19844	**0.18148**	0.20242	0.12200
304	498	0.20658	0.20238	**0.19458**	0.13252
306	419	0.20835	**0.16551**	0.20398	0.14338
309	2463	0.39327	0.18763	**0.16642**	0.14500
311	149	0.20336	0.28993	**0.16310**	0.13464
311	177	0.17977	0.17036	**0.14937**	0.11687
317	246	0.22179	**0.16638**	0.16756	0.12497
318	760	0.21398	**0.19857**	0.22549	0.17847
325	291	0.16907	0.19473	**0.15863**	0.13616
325	352	0.19225	0.18555	**0.17217**	0.12386
331	259	0.16934	0.22054	**0.16873**	0.12028
343	296	0.21090	0.21883	**0.17092**	0.13671
348	22	0.20341	0.20373	**0.16412**	0.13056
354	19	0.16393	0.18964	**0.15928**	0.11976
360	205	0.19162	**0.18633**	0.20200	0.11963
374	298	0.24584	**0.23243**	0.25402	0.20980
375	483	0.22620	0.30889	**0.21896**	0.18353
378	140	0.21185	**0.20691**	0.21782	0.14536
384	179	0.19240	0.18687	**0.18297**	0.10334
389	327	0.18582	0.17367	**0.14560**	0.12046
390	316	0.17908	**0.16449**	0.19013	0.12082
410	137	0.26052	**0.23348**	0.25912	0.15173
417	600	0.20622	0.16412	**0.15753**	0.11343
422	265	0.21130	**0.20145**	0.20186	0.15794
452	283	0.17930	0.18104	**0.15429**	0.12519
460	42	0.18112	0.19769	**0.15718**	0.12052
463	27	0.17550	0.19472	**0.15195**	0.14021
475	551	0.20988	0.22030	**0.19106**	0.15903
481	135	0.18713	0.18401	**0.15982**	0.12296
511	212	**0.15887**	0.17470	0.16975	0.12005
512	393	0.14968	0.17804	**0.14648**	0.10275