Utrecht University

Artificial Intelligence

Department of Information and Computing Sciences

Finding Patterns in DNA Sequences For Prediction of Errors

Utrecht University

April 9, 2018

Author:
Leyi Au Yeung,
3826147

Supervisors:
Dr. A.J. Feelders
Dr. J.F.J Laros
R. H. de Leeuw
Second Examiner:
Prof. Dr. A.P.J.M. Siebes

Abstract

With the use of DNA analysis it is possible to determine whether or not a suspect can be placed at a crime scene. In the field of forensics it is often the case that the DNA sample obtained is of low quality, the sample found could be from a little bit of saliva or small hairs left behind on the crime scene. Therefore DNA amplification is necessary. After DNA amplification, it is possible to sequence the DNA. This is a process to determine the order of the DNA sequence, which is a string made up of As, Cs, Gs and Ts. These methods are however prone to errors, making it more difficult to analyze the DNA correctly. For instance, the DNA could be a mixed sample where more than one individual's DNA is found, then it will be harder to determine when a sequence is an erroneous sequence or a genuine sequence, belonging to an individual. We have used sequential pattern mining to determine whether it was possible to predict the reads that could be considered as non genuine sequences. This was done by taking segments of the DNA sequence before and after an error was made. Sequential pattern mining is a method in data mining to finding relevant patterns in a database of sequences. A pattern is considered relevant if it exceeds a threshold, which is user defined. This threshold is based on the number of times a pattern appears in the sequences in the database. We have found several patterns that were associated with a certain position on the DNA sequence. This certain position showed a bias towards one type of error. With the use of the pattern and the position associated with the pattern it might be possible to determine the number of sequences that are non genuine in a new sample. However, many other factors might contribute to whether or not a pattern shows this type of error, for instance, read orientation, error ratio and sequence length.

Acknowledgements

First, I would like to thank my head supervisor at the LUMC, although he had a busy schedule, he always tried to have a weekly meeting during the period that I worked there. Knowing that I still needed to learn a lot about Python and programming in general, he appointed one of his colleagues whom I could go to for advice about Python. I was even allowed to follow a course in Python for bioinformatics, which gave me many insights. Secondly, I would like to offer my thanks to my other supervisor at the LUMC whom I could ask for advice on a daily basis. He patiently explained all the biological processes using many visual representations. Even when these processes seemed difficult he patiently explained these in the simplest terms possible. I have noticed that I have learned many things about DNA and the whole process behind it. I also wish to thank my supervisor at the UU for always replying to my e-mails quickly and always having the time to schedule in a meeting shortly after I had asked for one. Last but not least, I would like to express my thanks to the head of the forensic lab at the LUMC for giving me feedback as well. Throughout the project each of my supervisors gave me the feeling that I was always allowed to come to them for advice, I appreciated this very much.

Table of Contents

Background

DNA (short for deoxyribonucleic acid) is the building block of any organism. It is a molecule that contains information about the type of proteins the cell has to make. DNA consists of two strands that are coiled around each other called a double helix. The strands consists of small units called nucleotides. These nucleotides come in four types Adenine (A), Thymine (T), Cytosine (C) and Guanine (G), also known as a base. The double helix is connected in the middle through these nucleotides, for each A there is a T connected, for each C there is a G connected to it and vice versa, illustrated in Figure 1. Thus the double helix are complements of each other. In Figure 1 it can be seen that the DNA sequence is labeled with 5' and 3', these numbers give the DNA sequence a sense of direction, the 5' to 3' direction refers to the forward direction, while the 3' to 5' direction refers to the reverse.

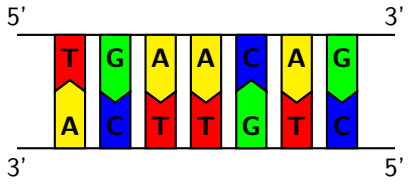

Figure 1: Example of double stranded DNA

Humans have two sets of 23 chromosomes containing DNA where each set was passed on by the parents. Positions on the chromosomes are called alleles. Due to the fact that these chromosomes come from two individuals, the alleles can have variations. The alleles can either be identical or not. When the alleles are identical, that part of the DNA will be considered homozygote. If the pair is not identical then that part will be considered heterozygote, as illustrated in Figure 2. These positions will be referred to as markers. Thus a marker can be either homozygous or heterozygous depending on the two alleles.

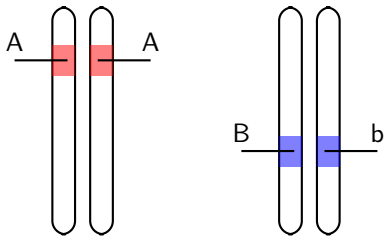

Figure 2: Position on the chromosomes, homozygote (left) and heterozygote (right)

In molecular biology PCR is a method that amplifies specific locations on the DNA which are called markers (additional information on PCR in Section B.1 of the Appendix). After the PCR there is enough DNA to perform the necessary steps for DNA sequencing with a sequencing machine. Sequencing is the process of determining the order of the nucleotides. The Illumina

MiSeq is a sequencing machine that reads millions of sequences parallel to each other, this process is called massive parallel sequencing (MPS). First the DNA is attached to a glass plate, called a flow cell, afterwards the DNA will be amplified using bridge amplification, illustrated in Figure 36 in Appendix A. With the amplification small clusters of DNA are formed on the flow cell. Single nucleotides with fluorescent labels bind to the DNA strands. A high quality camera is able to read the fluorescent labels and determine the nucleotide, illustrated in Figure 37 in Appendix A.

During sequencing the DNA is processed single stranded, thus for each single stranded DNA there will always be a complementary strand that matches. The Illumina MiSeq outputs the DNA sequences into two separate reads: Read 1 and Read 2. For each sequence that is read in one direction (forward or reverse) for Read 1, the complementary sequence will be found in Read 2. For instance, if Read 1 contains the forward strand of the sequence, then Read 2 will contain the reverse strand of that specific sequence.

1 Introduction

The establishment of the innocence or guilt of a suspect plays a vital role in criminal justice. With the use of DNA analysis one can determine whether or not a suspect can be placed at the crime scene (Butler, 2015). In the field of forensics it is often the case that the DNA sample is of low quality, the sample found could be from a little bit of saliva on a cigarette butt or small hairs left behind on the floor. Therefore amplification of DNA is necessary (Bright et al., 2012). Polymerase Chain Reaction (PCR) is one of the most used methods in molecular biology. Through temperature cycles it activates processes of melting, annealing and elongation. Afterwards the DNA can be sequenced, which is a process that determines the order of the nucleotides (A, C, G and T).

In forensic research, short tandem repeats (STRs) have proven to be successful for human identification (van der Gaag et al., 2016). STRs are small segments of DNA that are repeated varying number of times, one segment will be referred to as a repeat unit. This variation in number of repeat units differs per individual. By using many markers containing STRs a profile can be made that is almost unique for each individual. Therefore PCR can be used to amplify regions (markers) that have these STRs.

Although PCR is one of the most used tool in molecular biology, it is prone to errors. One of these errors is PCR stutter, which is an error where the repeat units of a sequence are less or more than the repeat units of the original sequence. Several factors that can contribute to stutter are DNA overamplification, analysis of imbalanced mixtures, certain PCR conditions and the chosen primers, which are short segments of the DNA used for binding to the part of the DNA that is going to be amplified (Brookes et al., 2012). Other errors that appear with PCR are deletions of bases, insertions of bases and substitutions, where one base is substituted by another (e.g. an A is replaced by a T) in a DNA sequence.

On top of PCR errors, DNA sequencing is also subject to errors, such as insertions, deletions

and substitutions, with substitution errors being the most prominent ones. One possibility for these errors is due to the fluorescent labels that the machine uses to read in the sequence. These fluorescent labels share similarity in intensity of the colours, where the fluorescent label for A and C are closely correlated and the fluorescent label for G and T (Schirmer et al., 2015).

When we know that DNA is of one individual, it is easy to establish the genuine sequences from the non genuine sequences. The non genuine sequences that are present, which have errors, do not matter as much. However in the field of forensics it is often the case that the DNA samples are mixed where the balance between the contributors is often unequal, making it difficult to distinguish the DNA sequences of the differing contributors, especially for minor contributors (van der Gaag et al., 2016). Therefore it is more of a challenge to determine whether a certain sequence belongs to the suspect's DNA or if it is an error made by either DNA amplification or DNA sequencing.

Previous research suggests that the errors made during sequencing with the Illumina Miseq are not randomly occurring errors (Schirmer et al., 2015). This was done by analyzing 73 different DNA samples with five different preparations. To infer the type of errors made within one sample the authors used alignment of the DNA sequences against reference sequences. Sequence alignment is a method used in bioinformatics for finding similarities between DNA sequences. To align two sequences gaps are inserted at certain positions on the sequence to correct for insertions and deletions, these gaps come with a certain penalty score. The goal of alignment is to find the best possible similarity between sequences without using too many gaps. In Figure 3 an example of an alignment is shown, Alignment will be further explained in Section 2.4.

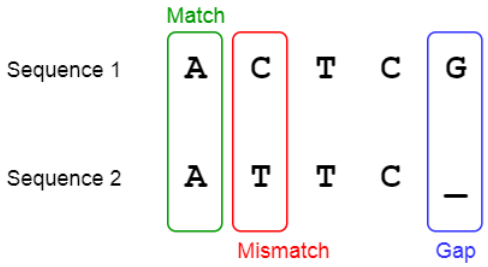

Figure 3: Example of pairwise alignment (Mallawaarachchi, 2017)

After sequence alignment, they used the Compact Idiosyncratic Gapped Alignment Report (CIGAR) to find the position of the error and report the substitution, insertion and deletion error rates. For each error position three bases were recorded before the errors, these were referred to as motifs. The top three motifs were then examined to measure the percentage errors they could explain. The results showed that the samples that were sequenced with the same preparations had similar top three motifs. Another driving factor were the forward and

reverse primer combinations, these had to do with which marker was sequenced. The samples that were sequenced with the same forward and reverse primer combinations, showed similar motifs that were more apparent before the errors. On average the three motifs were able to explain 34% of the substitution errors, 72% of the insertion errors and 48% of the deletion errors.

Previous research greatly supports the idea that using different preparations are more associated with certain motifs being present with errors. We are interested in whether motifs or patterns are also present in markers containing STRs. If there are certain motifs correlated to errors then we can use this information to disregard certain DNA sequences as non genuine sequences.

FDSTools is one of the tools designed to account for PCR stutter in forensic MPS data (Hoogenboom et al., 2017). It uses a reference database to correct for stutter errors, which were errors caused by DNA amplification. The reference database contains all alleles that were found from the samples that the authors have used. Each sample contained up to 24 markers where each marker had only one or two sequences that are considered as the genuine (allele). To be able to use this reference database, a noise profile was created for each allele. Noise referred to the sequences of the marker that were not categorized as genuine. To create such a noise profile the noise ratio

$$\text{Noise Ratio} = \frac{\text{Noise Reads}}{\text{Allele Reads}}$$

was calculated, where Noise Reads were the non genuine sequenes and Allele Reads were the genuine sequences.

To account for sequences that were not present in the reference database, a statistical model obtained from the reference database was used to predict possible noise ratios for these new alleles. The statistical model were polynomial functions fitted to the relationship between the length of the STRs and the stutter ratios. The genuine sequences, which varied in repeat length, were established. For each of the genuine sequences the stutter ratio was determined by counting the sequences with one repeat unit less or more from these genuine sequences. FDSTools is able to recover reads (sequences) that were categorized as noise with the correct genuine allele, this correction could help with identifying the minor contributors in mixed samples. FDSTools has proven to be a successful tool in determining PCR stutter and has been able to detect other errors such as substitution errors. It does not however look at patterns that could be present in the DNA sequences that might be related to the errors.

Therefore, we propose another approach. In this thesis we will address two research questions: What patterns are associated with errors made during DNA amplification and DNA sequencing? And how can these patterns be used for prediction?

To be able to determine which patterns are interesting, segments of sequences were taken before and after an error and used as input for sequential pattern mining. Sequential pattern mining is an approach for finding relevant patterns in a database consisting of sequences. A pattern is considered relevant if it appears in many sequences of the database, which is set to a user defined threshold. We apply sequential pattern mining to the DNA segments to find motifs

(patterns) that are appearing in these segments, a detailed explanation of sequential pattern mining will be discussed in Section 2.6.

In this thesis, the methods are explained in Section 2 with the tools used to acquire the DNA segments (subsequences) and the patterns, followed by the results discussed in Section 3. In Section 4 a brief summary of the results and discussion are given.

2 Methodology

From previous research, there is reason to believe that the errors made are not entirely random (Schirmer et al., 2015). To investigate this, subsequences were taken before and after each error on the erroneous sequences. Afterwards pattern mining was applied to these subsequences. To perform pattern mining on the DNA sequences the following tools were used to obtain the subsequences.

2.1 TSSV

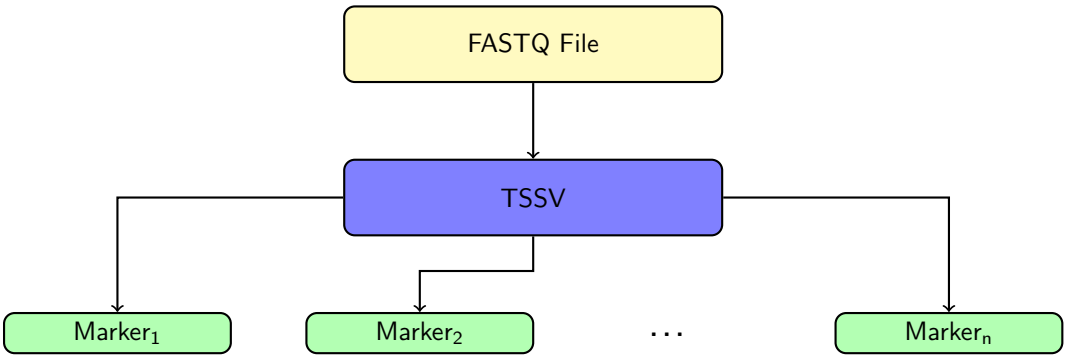

Figure 4: Diagram of the process after sequencing. With a FASTQ file as input, which is a file that contained all the DNA sequences, TSSV separated the sequences per marker

TSSV was used to go through the DNA sequences of each individual sample. A library containing the information was used to categorize each sequence (Anvar et al., 2014). This library consisted of the marker names, where each marker was defined with the flanking sequences and the target sequence. The target sequence consisted of a prefix and/or suffix and the STR, which could be found between the flanking sequences (Figure 5). TSSV separated the sequences per marker into two files, *knownalleles* and *newalleles*. The *knownalleles* file contained the sequences where both the flank sequences (Flank 1 and Flank 2) were found and where the sequence matched with the requirements of the target sequence. The *newalleles* file contained the sequences where both the flank sequences were present, but did not meet the criteria of the target sequence.

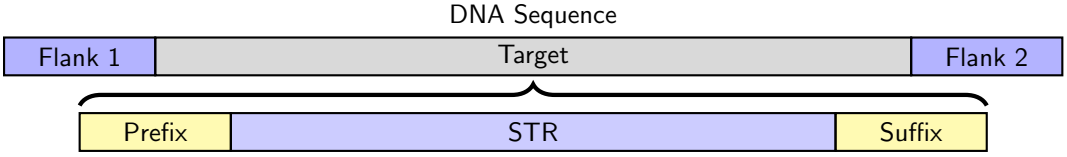

Figure 5: Example of the DNA sequence composition, between Flank 1 and Flank 2 the target sequence could be found, this target sequence consisted of a prefix, a suffix and the STR.

Each TSSV output contained one individual sample with the sequences categorized per marker into the *knownalleles* and *newalleles* file. In total it was run on 795 samples. Rather than having the sequences split among the two files, the library was modified that the requirement of the target sequence was never met which caused the sequences to be categorized in the *newalleles*. The formatting of the sequence was the raw sequence which differed from the *knownalleles* where the sequences were converted to a more human readable format (TSSV format). For instance, the TSSV format would be AATT(3)AAGT(2), where the three and the two are referred to as the repeat number, which indicated that AATT was repeated three times consecutively and AAGT was repeated two times consecutively. The raw sequence would be AATTAATTAATTAAGTAAGT.

After categorizing the sequences per marker. Each marker contained a csv file (*newalleles*) consisting of four columns, the sequence, labeled as 'allele', the total, forward and reverse reads, where the reads referred to the number of times the sequence was counted in the sample. The target sequence was considered as the forward orientation. If the sequence was found in the reverse orientation (reverse complement), then the sequence was converted to its forward orientation. TSSV sorted the sequences in descending order of the total reads, as illustrated in Table 1. From this point for each sample and each marker the following steps were applied to determine the errors and obtain the subsequences.

allele	total	forward	reverse
Seq_0	100	60	40
Seq_1	80	40	40
Seq_2	10	8	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Seq_n	1	0	1

Table 1: Example of the *newalleles* file

2.2 Homozygous and Heterozygous markers

For each marker in a sample, it was possible to infer the genuine sequence(s) from the non genuine sequences, which were the erroneous sequences. A marker was considered heterozygous if the second most abundant sequence exceeded a threshold, which was defined as a percentage of the most abundant read count. For example, in Table 1 and given a threshold of thirty percent, the marker would be heterozygous, where the total of Seq_1 was exceeding the threshold minimum of thirty percent from the total of Seq_0. The genuine sequences would then be Seq_0 and Seq_1 while the remaining sequences were the non genuine sequences, $Seq_2, \ldots, Seq_n$.

2.3 Sequence Selection

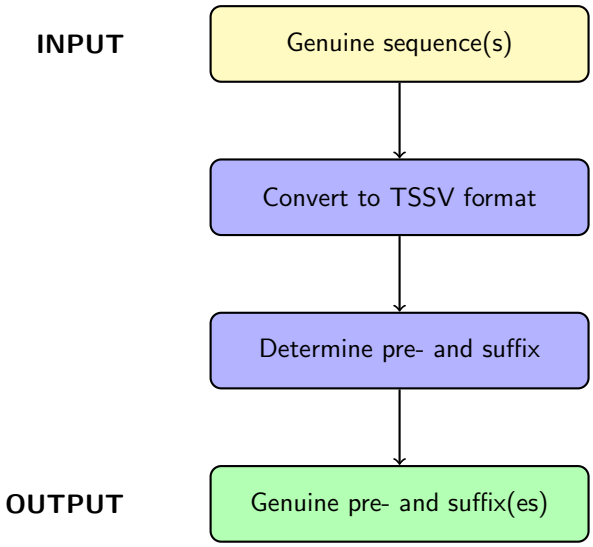

Figure 6: After obtaining the genuine sequences, the sequences are converted to TSSV format to determine the pre- and suffixes

Instead of using the complete sequence, we used the parts of the sequence without the STRs, which were the prefix and the suffix. This was due to the large variations in repeat lengths of the STRs, which made it difficult to pinpoint the exact position of the error on the genuine sequence. These repeat variations were not present in the prefix and suffix. Another complication had to do with categorization of the non genuine sequences with the correct genuine sequence in heterozygous markers. At first a pattern growth algorithm, based on Ye et al. (2009), was implemented for this purpose, but after further examination many markers were excluded from the data set due to the lack of distinguishing features between the two genuine sequences. Further explanation on the pattern growth algorithm can be found in Section B.4 in the Appendix.

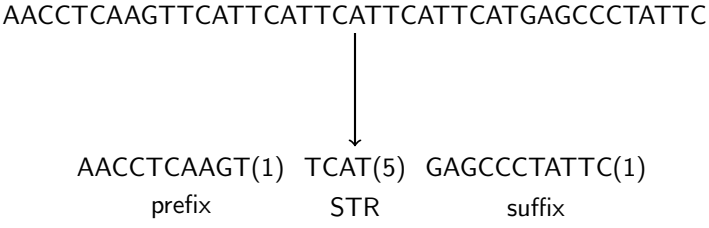

Figure 7: Conversion of a raw sequence to TSSV format

To determine the length of the pre- and suffix, the genuine sequences of the marker were converted to the TSSV format, which separated the sequence into segments followed by the

repeat number, illustrated in Figure 7. The prefix was found at the start of the STR, while the suffix was found at the end of the STR. To determine the prefix of a marker, the first segment was taken from the converted sequence. If the segment was not followed by a number higher than one, this would mean that the segment occurred only once in the sequence, it would then be labeled as the prefix. The same principle was used for the suffix, where the last segment of the converted sequence was used.

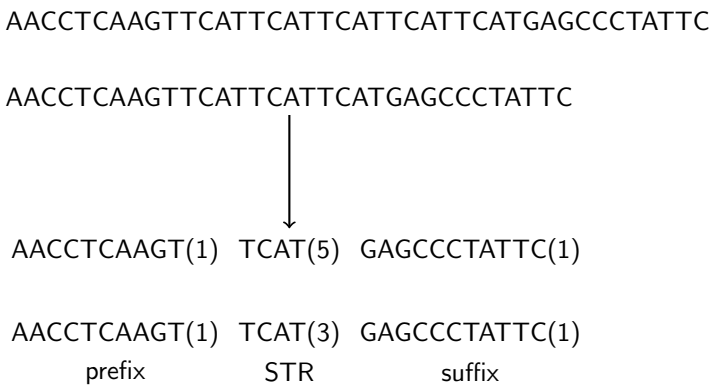

Figure 8: Example of a heterozygous marker with identical pre- and suffix, where the sequences shown are the genuine sequences. This marker would be included in the data set

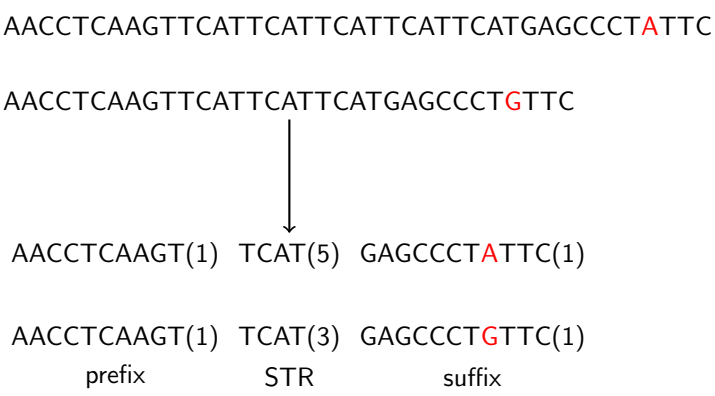

Figure 9: Example of a heterozygous marker with varying suffix, where the two sequences shown are the genuine sequences, this marker would be exclude from the data set

The pre- and suffix of the genuine sequences needed to be compared in heterozygous markers due to small variations. If the pre- or suffix were not equal to each other (Figure 9), the markers were excluded due to categorization issues of the pre- and suffix of the non genuine sequences of heterozygous marker. If the pre- and suffix of the genuine sequences were equal then the pre- and suffix of the non genuine sequences could be determined by either one of the genuine sequences, illustrated in Figure 8.

After determining the prefix and suffix of the genuine sequences, the lengths were used to determine the prefixes and suffixes of the non genuine sequences. Four bases after the prefix and before the suffix were taken from the STR part to account for insertion and deletion errors, illustrated in Figure 10. The prefix and suffix of the non genuine sequences were then compared to the prefix and suffix of the genuine sequence, this was done with sequence alignment.

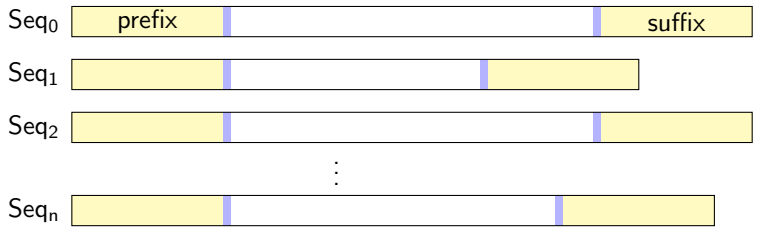

Figure 10: After determining the prefix and suffix for the genuine sequence(s), the lengths of the pre- and suffix were used to take that part of the sequence from the non genuine sequences. Few extra bases are taken after and before the pre- and suffix to correct for insertion and deletion errors, which is colored in blue.

2.4 Needleman-Wunsch Algorithm

The Needleman Wunsch algorithm was used for sequence alignment. The Needleman Wunsch algorithm works with a scoring system and based on this scoring system tries to find the highest score possible. First the scoring system needs to be determined. This scoring system is based on a match, a mismatch or a gap. The gap refers to an insertion or deletion. While the match and mismatch are dependent on the comparison between two characters, denoted as $S(a,\ b)$.

Given two sequences, $A = A_iA_{i+1}\dots A_{n-1}A_n$ and $B = B_jB_{j+1}\dots B_{m-1}B_m$, the score $S(A_i, B_j)$ would either be $+1$ for a match and -1 for a mismatch.

$$S(x,y) = \begin{cases} 1 \text{ iff } x = y \\ \\ \text{-1 otherwise} \end{cases}$$

For this example sequence $A = ACTCC$ and sequence $B = GCTC$. From the sequences a two dimensional matrix is created, where each column represents a character in sequence A and each row represents a character in sequence B, as illustrated in Figure 11. This matrix is denoted as F, where the cell value of row i and column j is denoted as F_{ij}.

		A	C	T	C	C
G						
C						
T						
C						

Figure 11: Two dimensional matrix of two sequences

Each cell value F_{ij}, where $i = 0, \ldots, n$ characters in sequence A and $j = 0, \ldots, m$ characters in sequence B, is calculated by finding the highest score possible based on three cell values, the left ($F_{i-1,j}$), the left diagonal ($F_{i-1,j-1}$) and the top cell ($F_{i,j-1}$). After defining the scoring system, it is possible to calculate each value in the matrix F with Algorithm 1, as illustrated in Figure 12. For this example a gap penalty of -1 is used.

$$F_{ij} = \max \begin{cases} F_{i-1,j-1} + S(A_i, B_j) \\ F_{i-1,j} + \text{gap} \\ F_{i,j-1} + \text{gap} \end{cases}$$

Algorithm 1 Compute F Matrix

```
1: procedure COMPUTE F MATRIX(A, B, gap)
2:     for i = 0 to length(A) do
3:         F(i, 0) ← gap * i
4:     end for
5:     for j = 0 to length(B) do
6:         F(0, j) ← gap * j
7:     end for
8:     for i = 1 to length(A) do
9:         for j = 1 to length(B) do
10:            Match ← F(i − 1, j − 1) + S(A_i, B_j)
11:            Delete ← F(i − 1, j) + gap
12:            Insert ← F(i, j − 1) + gap
13:            F(i, j) ← max(Match, Delete, Insert)
14:        end for
15:    end for
16:    return F
17: end procedure
```

		A	C	T	C	C
	0	-1	-2	-3	-4	-5
G	-1	-1	-2	-3	-4	-5
C	-2	-2	0	-1	0	1
T	-3	-3	-1	1	0	0
C	-4	-4	0	0	2	3

Figure 12: The scores of the Needleman-Wunsch alignment with a gap penalty of -1

To find out what the best alignment is, we start from F_{nm}, this is the highest score possible from the alignment of sequence A and sequence B. Figure 13 shows the trace back of Algorithm 2. For example, from the bottom right corner $F_{54} = 3$, the algorithm will add a gap to $AlignmentB$. In the end the algorithm returns the alignment where $AlignmentB$ has a gap at the end of the sequence. Based on Algorithm 2, $AlignmentA = ACTCC$ and $AlignmentB = GCTC–$.

Algorithm 2 Alignment

```
1: procedure ALIGNMENT(A, B, gap)
2:     F ← Compute F Matrix(A, B)
3:     AlignmentA ← ""
4:     AlignmentB ← ""
5:     i ← length(A)
6:     j ← length(B)
7:     while (i > 0 or j > 0) do
8:         if (i > 0 and j > 0 and F(i, j) == F(i − 1, j − 1) + S(A_i, B_i)) then
9:             AlignmentA ← A_i + AlignmentA
10:            AlignmentB ← B_i + AlignmentB
11:            i ← i − 1
12:            j ← j − 1
13:        else if (i > 0 and F(i, j) == F(i − 1, j) + gap) then
14:            AlignmentA ← A_i + AlignmentA
15:            AlignmentB ← "–" + AlignmentB
16:            i ← i − 1
17:        else
18:            AlignmentA ← "–" + AlignmentA
19:            AlignmentB ← B_j + AlignmentB
20:            j ← j − 1
21:        end if
22:    end while
23:    return AlignmentA
24:    return AlignmentB
25: end procedure
```

		A	C	T	C	C
	0	-1	-2	-3	-4	-5
G	-1	-1	-2	-3	-4	-5
C	-2	-2	0	-1	0	1
T	-3	-3	-1	1	0	0
C	-4	-4	0	0	2 ←	3

Figure 13: Trace back of the scores starting from F_{nm}

If sequence A was considered the genuine sequence and sequence B was considered the non genuine sequence the differences were inferred as errors on positions with respect to the genuine

sequence. For instance, the first position of sequence A and the first position of sequence B would return a description of 1A>G, where 'A' was substituted by a 'G' in sequence B. The gap that was inserted in $AlignmentB$ would return a description of 5C>–, where it would indicate a deletion of a 'C' on position 5 of the genuine sequence.

2.5 Subsequences Before and After Error Position

The non genuine pre- and suffix lengths were corrected for insertion and deletion errors that were found with alignment, afterwards for each non genuine pre- and suffix the positions of the errors were inferred using the same alignment method described in Subsection 2.4, but then with respect to the non genuine pre- and suffix. Ten bases before and after the position of the error were used as subsequences for sequential pattern mining. One mismatch was allowed within these ten bases due to co-occurrences of errors (additional information is described in Appendix B.5). The subsequences before and after the error were then mined for patterns separately.

2.6 Sequential Pattern Mining

Sequential pattern mining is performed on a list of sequences in a sequence database $SDB = \langle S_1, S_2, \ldots, S_i \rangle$. Each sequence is an ordered list of letters made up from a fixed alphabet Σ, where $S = s_1 s_2 \ldots s_n$, such that $s_i \in \Sigma$, where i denotes the position in the sequence. A sequence $A = a_1 a_2 \ldots a_n$ is considered a subsequence of another sequence $B = b_1 b_2 \ldots b_m$ iff there exists integers $1 \leq i_1 < i_2 < \ldots < i_n < i_m$ such that $a_1 = b_{i1} < a_2 = b_{i2} < \ldots < a_n = b_{in}$ (denoted as $A \subseteq B$). For instance, 'ACT' is a subsequence of 'GGACTG', but also a subsequence of 'GACGGT'.

Sequential pattern mining counts the number of occurrences of a subsequence in a sequence database SDB, this is called support (sup). Thus given a sequence A, $sup(A) = |\{S_i \in SDB : A \subseteq S_i\}|$. Sequential pattern mining uses the support count and minimum support count ($minsup$) defined by a user to determine whether a subsequence is frequent. A subsequence S_i is considered frequent if $sup(S_i) \geq minsup$.

Sequential pattern mining generates subsequences, called candidates, based on the sequence database. First the subsequences containing one letter will be generated and counted in the sequence database (level 1). Afterwards the level 2 candidates will be generated from the level 1 candidates. This is done by combining the candidates from one level with each possible combination, for instance from the level 1 candidates: A, C, G, T. The level 2 candidates will be AA, AC, AG, AT, .., TT. Afterwards, the level 2 candidates support will be calculated in the sequence database. After having generated all the possible candidates, the subsequences with support exceeding the $minsup$ are the frequent subsequences, called patterns.

Following this principle, candidates that are generated are not always subsequences that are found in the sequence database. Thus, to reduce the list of candidates, it is possible to only extend the candidates that are frequent in the database, which is the *a priori* approach. Algorithm 5 shows an example of a sequence mining algorithm with the *a priori* approach

embedded, where the search space is structured as a prefix search tree. The tree starts at level 0 containing an empty sequence with each letter in the alphabet as children. A node contains a sequence $s = s_1 s_2 \dots s_n$ at level k with children of the form $s' = s_1 s_2 \dots s_n s_{n+1}$ at level $k+1$. Thus, s is a prefix of each child s'.

Given a sequene database (SDB), an alphabet (Σ) and minimum support ($minsup$), GSP initially creates a node for each letter $x \in \Sigma$ as the level 1 candidates, $C^{(1)}$. For each $s_i \in SDB$, we check whether each candidate $r \in C^{(k)}$ is a subsequence of s_i. If that is the case, this is added to the support of r. Based on Algorithm 4 the next level candidates at level $k+1$ are generated from the previous candidates at level k. Each candidate r_a is extended with the last letter of the other candidates r_b sharing the same parent. If the new candidate r_{ab} is infrequent it is pruned.

Algorithm 3 ComputeSupport

```
1: procedure ComputeSupport(C^(k), SDB)
2:     for each s_i ∈ SDB do
3:         for each r ∈ C^(k) do
4:             if r ⊆ s_i then
5:                 sup(r) ← +1
6:             end if
7:         end for
8:     end for
9: end procedure
```

Algorithm 4 ExtendPrefixTree

```
1: procedure ExtendPrefixTree(C^(k))
2:     for each leaf r_a ∈ C^(k) do
3:         for each leaf r_b ∈ CHILDREN(PARENT(r_a)) do
4:             r_ab ← r_a + r_b[k]                ▷ extend r_a with last item of r_b
5:             if r_c ∈ C^(k), for all r_c ⊂ r_ab, such that |r_c| = |r_ab| − 1 then
6:                 Add r_ab as child of r_a with sup(r_ab) ← 0
7:             end if
8:         end for
9:         if no extension from r_a then
10:            remove r_a from C^(k)
11:        end if
12:    end for
13:    return C^(k)
14: end procedure
```

Algorithm 5 GSP

```
1: procedure GSP(SDB, Σ, minsup)
2:     F ← ∅
3:     C^(1) ← {∅}                          ▷ Initial prefix tree with single symbols
4:     for s ∈ Σ do
5:         add s as child of ∅ with sup(s) ← 0
6:     end for
7:     k ← 1
8:     while C^(k) ≠ ∅ do
9:         ComputeSupport(C^(k), SDB)
10:        for each leaf s ∈ C^(k) do
11:            if sup(r) ≥ minsup then
12:                F ← F ∪ {(r, sup(r)}
13:            else
14:                remove s from C^(k)
15:            end if
16:        end for
17:        C^(k+1) ← ExtendPrefixTree(C^(k))
18:        k ← k + 1
19:    end while
20:    return F^(k)
21: end procedure
```

2.7 CM-SPAM

The *a priori* principle is dependent on a horizontal representation of the sequence database, which is a database where each sequene is represented as is with a correpsonding sequence ID (SID), illustrated in Table 2a. CM-SPAM is a sequential pattern mining algorithm that is based on a vertical representation of the sequence database, illustrated in Table 2b, each letter in the sequence database contains a list of the SIDs with the positions indicated with an integer. For example, the letter A appears in SID a on position one and two of the sequence.

Vertical sequential pattern mining algorithms have been considered to be one of the fastest mining algorithms due to the fact that the sequence database does not need to be scanned multiple times. The disadvantage of vertical algorithms is that the candidates generated are not necessarily appearing in the sequence database or are infrequent (Fournier-Viger et al., 2014). Therefore the authors have considered pruning the candidates based on a new data structure called a Co-Occurrence Map (CMAP).

SID	Sequence
a	$AACGT$
b	$AACT$
c	$CGTT$
d	AA

(a) Horizontal representation of the sequence database

A		C		G		T	
SID	Position	SID	Position	SID	Position	SID	Position
a	1, 2	a	3	a	4	a	5
b	1, 2	b	3	b		b	4
c		c	1	c	2	c	3, 4
d	1, 2	d		d		d	

(b) Vertical representation of the sequence database

To get the vertical representation of the sequence database, CM-SPAM scans the database once to create the lists for each letter and the level 1 candidates, which are all the letters. Afterwards the candidates are generated by finding extensions of the previous candidates with the same prefix. A sequence $A = a_1, a_2, \ldots, a_n$ is considered a prefix of sequence $B = b_1, b_2, \ldots, b_m$, $\forall n < m$, iff $a_1 = b_1,\ a_2 = b_2,\ \ldots,\ a_{n-1} = b_{n-1}$.

Although the regular SPAM algorithm does not extend infrequent candidates this process can be reduced to a faster way. The CMAP is considered to be faster where each letter is represented with next candidates that exceed the $minsup$. The main difference being that prefixes that are infrequent are also pruned (Fournier-Viger et al., 2014). The authors concluded that a stored data structure (CMAP) increased the performance of the vertical algorithms due to removing unnecessary checks of infrequent candidates.

We used the CM-SPAM algorithm with minimal pattern length of three, maximal pattern length of ten and gapless patterns. A minimum support of five percent was used, because the input database consisted of subsequences before and after the errors on the non genuine sequences. Thus frequent patterns were also present on the genuine and non genuine sequences where no such error was made. Therefore the low support was used to get many frequent patterns that were going to be reevaluated, this will be described in Section 2.9.

2.8 Conversion

To use the CM-SPAM algorithm, the subsequences before and after the error needed to be translated into specific format. Each letter of the subsequence represented a different number separated by -1 indicating the end of an itemset and -2 indicating the end of a sequence. Itemsets were not of interest due to the fact that the data were strings, where only the order of the letters needed to be maintained. In Table 3 the conversion per position is shown, we used a different integer indicating the position of the letter on the subsequence, which was more convenient for determining the distance of the pattern from the error. With this conversion we were able to distinguish noise, which were errors that occurred on each position per default, from the more striking errors that appeared with the patterns.

	Position									
	1	2	3	4	5	6	7	8	9	10
A	0	1	2	3	4	5	6	7	8	9
C	10	11	12	13	14	15	16	17	18	19
G	20	21	22	23	24	25	26	27	28	29
T	30	31	32	33	34	35	36	37	38	39

Table 3: Conversion of the characters per position on the subsequence

2.9 Pattern Processing

Each frequent pattern was converted back to the original format with an integer at the end of the pattern, which indicated the distance from the pattern to the error. We used binomial tests to determine whether the support that we found for each pattern was to be expected based on the probability of the pattern occurring in the database. In this case the database, $SDB = \langle S_1, S_2, \ldots, S_n \rangle$, were the sequences for each sample within a marker.

The sequence in which the occurrences of a given pattern were counted was dependent on whether the pattern was found before or after the error. If a pattern was found before the error then the sequences of each sample for each marker consisted of the primer (flank1), the prefix, a part of the STR and the suffix, illustrated in Figure 14. If a pattern was found after the error then the sequences for each sample for each marker consisted of the prefix, a part of the STR, the suffix and the primer (flank2), illustrated in Figure 15.

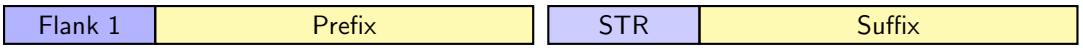

Figure 14: Sequence used for comparing patterns if pattern was found before the error, the small gap indicates the space between the prefix and suffix, where the STR indicates the end of the STR

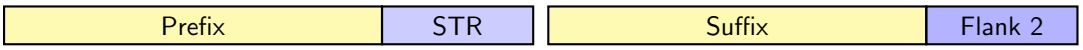

Figure 15: Sequence used for comparing patterns if patter was found after the error, the small gap indicates the space between the prefix and suffix, where the STR indicates the start of the STR

The probability of finding a pattern randomly in SDB was calculated by taking the average of the probability of finding a pattern in one sequence of the SDB, where the probability for finding a pattern was calculated for each sequence first, which was

$$P(pat,\ S) = \frac{count(pat,\ S)}{|S| - (|pat| - 1)},$$

where $S \in SDB$, $count(pat,\ S)$ were the occurrences of the pattern in a sequence and $|S|-|pat|$ were the number of possibilities the pattern would fit in the sequence. For example, given a sequence A of length 10 and a pattern x of length 3, then $|A| - (|x| - 1) = 8$. Afterwards the probability of finding the pattern within a window w was defined. Where w was the length of the subsequences, which was ten, used for CM-SPAM. This was used to determine the probability of finding the pattern at least once within ten bases in a sequence, which was

$$P(pat,\ S|w) = 1 - (1 - P(pat,\ S))^{w-(|pat|-1)}.$$

The average probability

$$P(pat,\ SDB) = \frac{\sum P(pat,\ S|w) \cdot |\{S : S \in SDB\}|}{\sum |\{S : S \in SDB\}|}$$

was calculated afterwards by accounting for the total number of reads.

The binomial test was performed with a null hypothesis (H_0) stating that the observed value of the pattern, $sup(pat)$, was equal or less than what we expect to find in the database. The alternative hypothesis (H_a) was that the observed value of the pattern was greater than what we expected to find, based on probability of finding the pattern randomly in the database. Where the expected value was $P(pat,\ SDB) \cdot n$ reads, where n was the total number of reads in SDB, denoted as $E(pat,\ SDB)$. If the p-value of the binomial test was lower than 0.05, the pattern was considered significant.

H_0: $sup(pat) \leq E(pat,\ SDB)$

H_1: $sup(pat) > E(pat,\ SDB)$

The significant patterns were many patterns that were overlapping. When these patterns were overlapping on the same subsequence that the patterns were found in, the pattern with the highest support count was included in the final pattern database. Afterwards the rest of the patterns that were also found in the same subsequence were included in the final pattern database. Thus the final pattern database consisted of patterns that were not overlapping within one subsequence.

2.10 Exceptions

Between the general procedure of determining the frequent patterns, some sequences needed to be removed from the database. The first step was removing the sequences containing the letter 'N', which were sequences where one nucleotide was not recognized by the sequencing machine. Afterwards non genuine sequences were also removed if the length of the pre- and suffix were below the threshold, which was ten percent of the length of the pre- and suffix of the genuine sequence. The same was done for non genuine sequences exceeding the treshhold. For example, if the length pre- and suffix of the genuine sequence was 110, then the length of the non genuine pre- and suffix needed to be between 99 and 121.

During alignment non genuine sequences that contained too many errors were also excluded from the input sequence database for CM-SPAM. This was done based on two criteria. The first criteria was that the error description was not allowed to be more than two bases, thus if a non genuine sequence had an error description of AAA>–, it would be excluded. The second criteria was the number of errors that the alignment returned, if the number of errors was larger than ten percent of the pre- or suffix that was aligned then the non genuine sequence would also be excluded.

Subsequences included in the input sequence database for CM-SPAM were compared with the genuine sequences if the subsequence was part of the STR. If the marker was heterozygotic then there was a possibility that the STRs differed in the genuine sequences, therefore the non genuine subsequence needed to be matched with the correct genuine sequence to account for mismatches. The correct genuine sequence was found by obtaining the subsequences on the genuine sequences on the position of the error, afterwards the genuine subsequences were both aligned with the non genuine subsequence. The alignment with the least errors was analyzed for mismatches, if more than one error (mismatch) was present, then the non genuine sequence was excluded from the input sequence database.

3 Results

3.1 Error ratio

Seven of the 24 markers had significant patterns in the forward read in Read 1, while fourteen of the 24 markers had significant patterns in Read 2. In Figure 16 the error ratios are shown per marker where an asterisk indicated the markers with significant patterns. The error ratios were calculated by taking the fraction of the reads that had errors in either the pre- or suffix compared to the total read numbers in in a marker,

$$\text{Error ratio} = \frac{\text{Non genuine reads}}{\text{Total reads}}.$$

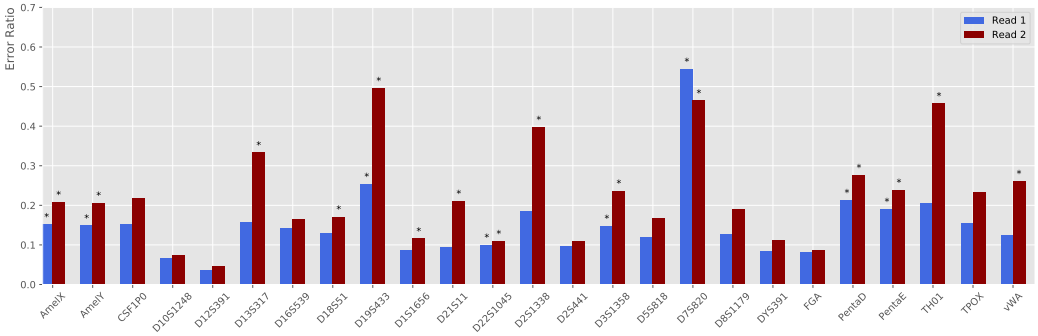

Figure 16: Read 1 and Read 2 error ratios per marker in the forward direction, each marker indicated with an asterisk (*) had significant patterns present

The error ratios for each marker in Read 1 ($\mu = 0.1523$) were on average lower than in Read 2 ($\mu = 0.2238$). The error ratios between the markers that had significant patterns were compared to the markers without significant patterns, illustrated in Figure 17. Markers with significant patterns had on average a higher error ratio ($\mu = 0.2492$) than markers without significant patterns ($\mu = 0.1307$), these were calculated by combining the error ratios of the markers with and without significant patterns for Read 1 and Read 2.

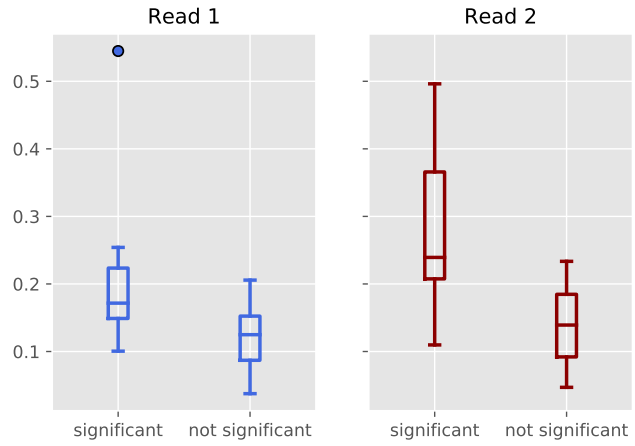

Figure 17: Read 1 and Read 2 error ratios for markers with and without significant patterns

To determine where the patterns were found the pre- and suffix of each marker were plotted on on the x-axis against the frequency of errors, which were the total errors in all the samples on each position, using a logarithmic scale. For the pre- and suffix the primers and a small segment of the STR were also plotted. To determine the contribution of the errors per position, the relative frequencies

$$\text{Relative Frequency}(pos) = \frac{\text{Frequency}(pos)}{\text{Total reads}}$$

were calculated. Where pos referred to the position on either the pre- or suffix. Due to varying pre- and suffixes between the samples in a marker, the patterns were plotted pre- and suffix having the most samples.

To account for background noise, which were errors that were expected to be present on each position per default, a baseline was calculated. Separate baselines were calculated for the pre- and suffix, because of the sequencing machine being more prone to errors at the end of the sequence. Based on the length of the pre- or suffix n frequencies were removed from the highest and the lowest relative frequencies, where $n = \text{length(prefix or suffix)}/10$. Afterwards the average of the relative frequencies was taken as the baseline for background noise.

3.2 Amel

The marker Amel, short for Amelogenin, is used to determine the sex of an unknown sample. If the marker was heterozygote then two genuine sequences were present, AmelX and AmelY, this would mean that the sample was male. Due to the fact that Amel had no STRs, it was possible to categorize the non genuine sequences with the correct genuine allele (either AmelX or AmelY). Significant patterns were found in AmelX and in AmelY, illustrated in Figure 18 and

Figure 19. The error ratios for AmelX and AmelY in Read 1 were 0.15, thus 15% of the total forward reads were reads with errors. Read 2 statistics and figures are illustrated in Appendix C and D.

For AmelX there was one significant pattern before the error and one after the error, both patterns were associated with one position, which was position 184 on the suffix. The most common error on this position was a substitution error from A to C, which contributed to 65.60% of the errors on this position, the second most common error was a substitution error from A to T (21.97%). 1.06% of the total forward reads were associated with an error on position 184. After correcting for the background noise (0.22%), position 184 contributed to 0.84% of the total forward reads.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CCAGCTTCCC_0	184	A>C (65.60)	0.84

*After accounting for background noise (0.22%)

Table 4: Read 1 AmelX patterns before error

For AmelY there was one significant pattern after the error, which was also position 184 on the suffix. The most common error on this position was a substitution error from A to C (41.71%), whereas the second most common error was an insertion of a T (19.66%). After accounting for the background noise (0.22%), the pattern associated with the error accounted for 0.46% of the total forward reads (Table 5).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTTTAAGCTC_0	184	A>C (41.71)	0.46

*After accounting for background noise (0.22%)

Table 5: Read 1 AmelY patterns after error

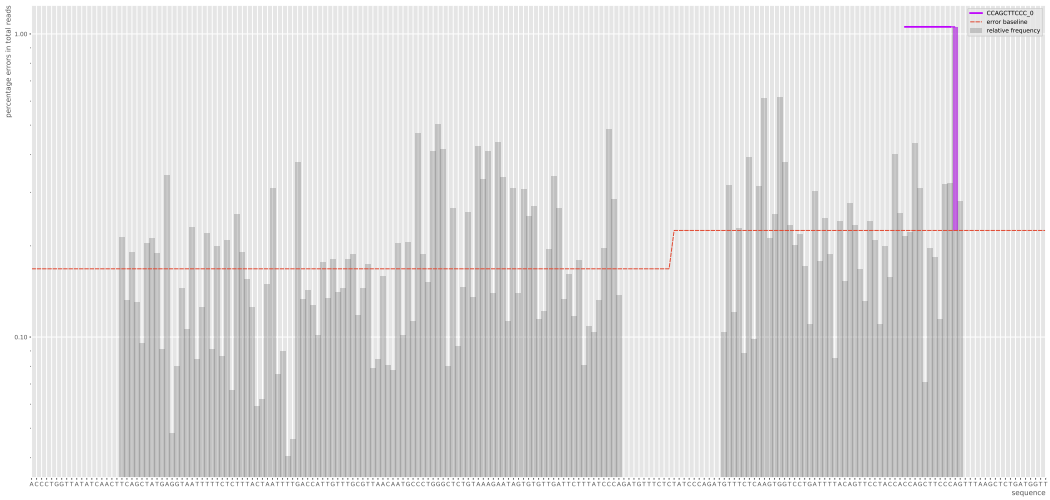

Figure 18: Read 1 AmelX patterns before error

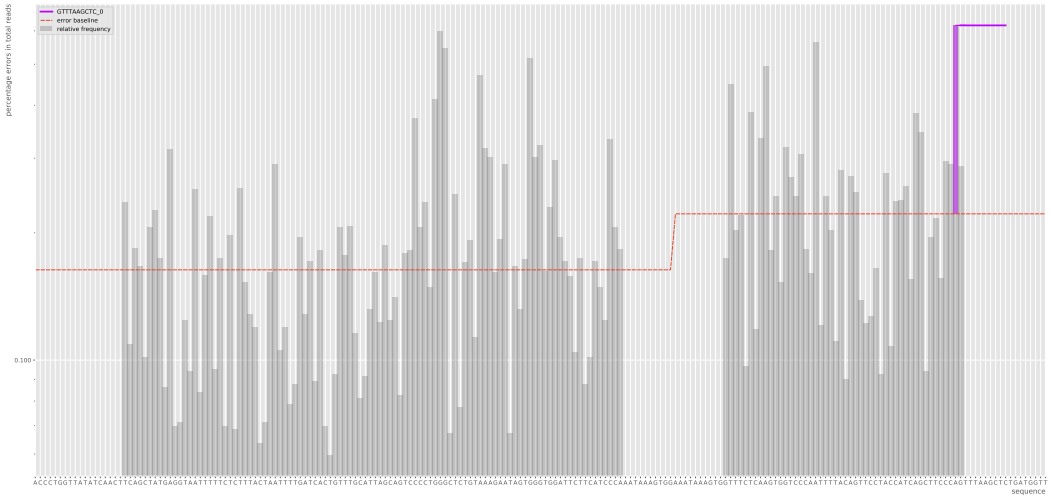

Figure 19: Read 1 AmelY patterns after error

3.3 D13S317

Marker D13S317 had significant patterns before and after the errors for Read 2 only. The error ratio was 0.33. Two patterns were before and two patterns were after the error. Two positions were associated with the patterns, which were positions 126 and 146 on the suffix, illustrated in Figure 20 and Figure 50 in Appendix D.

The most common error on position 126 was a substitution error from G to T (79.23%), the

second most common error was a substitution error from G to C (17.82%). 3.86% of the total forward reads had an error on this position, after accounting for background noise (0.62%).

For position 146, the most common error was a substitution error from A to C (87.91%) and the second most common error was a substitution error from A to T (7.41%). Of the total forward reads 3.28% had errors on this position, after account for background noise.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTTTTT_2	126	G>T (79.23)	3.86
CAACCCA_0	146	A>C (87.91)	3.28

*After accounting for background noise (0.62%)

Table 6: Read 1 D13S317 patterns before error

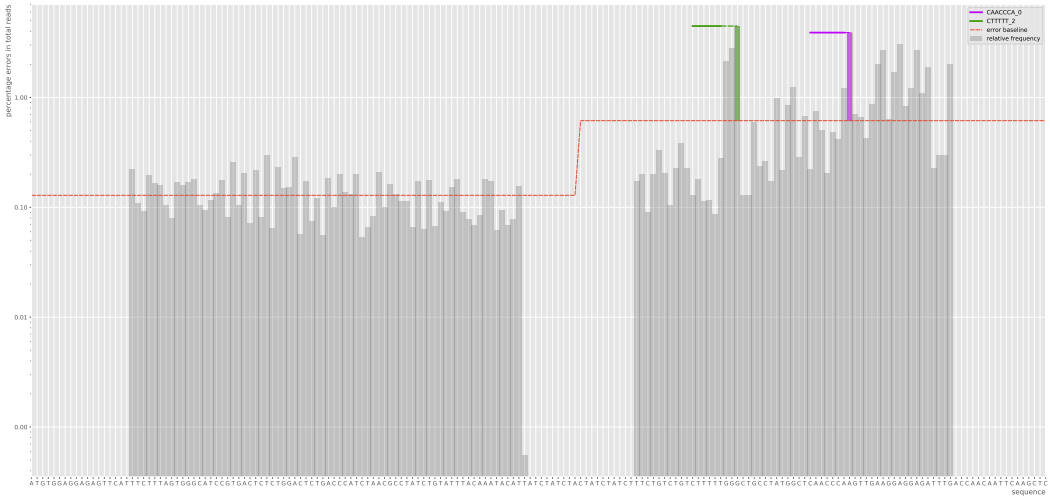

Figure 20: Read 2 D13S317 paterns before error

3.4 D18S51

Marker D18S51 had one significant pattern after the error in Read 2, illustrated in Figure 21. The error ratio was 0.17 for Read 2. The pattern was associated with one position on the suffix, which was position 145. The most common error on this position was a substitution error from T to A (54.26%) and the second most common error was a substitution error from T to C (22.22%). 1.39% of the total forward reads had an error on this position, after accounting for background noise (0.28%) the position accounted for 1.12% of the total forward reads (Table 7).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTAAGACATC_0	145	T>A (54.26)	1.12

*After accounting for background noise (0.28%)

Table 7: Read 1 D18S51 patterns after error

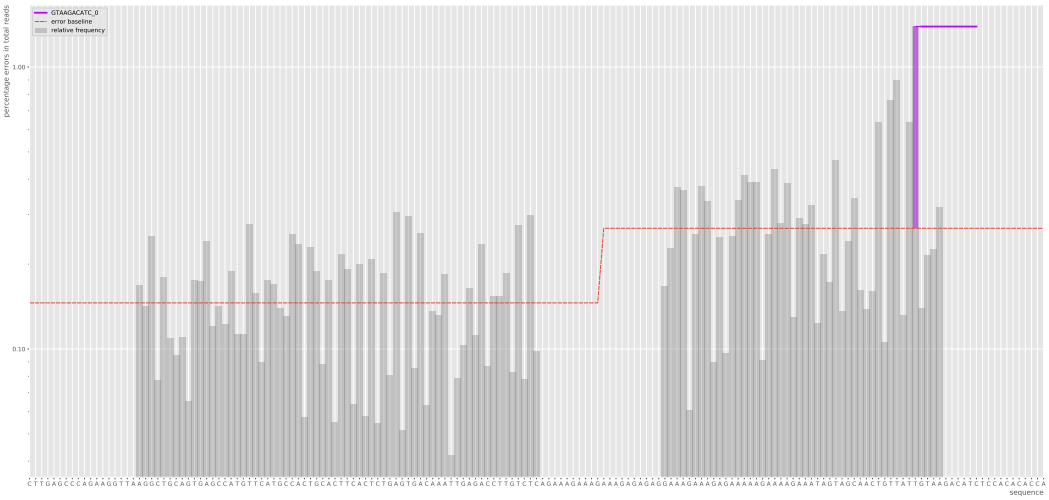

Figure 21: Read 2 D18S51 patterns after error

3.5 D19S433

Marker D19S433 had significant patterns in Read 1 and Read 2, illustrated in Figures 22, 23, 24 and 42. The error ratios were 0.25 for Read 1 and 0.50 for Read 2 with a prefix length of 76 and a suffix length of 38.

For Read 1 there were two patterns before the error, which were associated with two positions on the suffix, positions 122 and 146 (Table 8. For position 122 the most common error was a substitution error from A to T (57.34%) and the second most common error was a substitution error from A to C (38.02%). 2.69% of the total forward reads had an error on this position, after accounting for background noise (0.54%) it was 2.14%.

For position 146 the most common error was a substitution error from A to C (64.51%) and the second most common error was a substitution error from A to T (22.02%). 2.71% of the total forward reads had an error on this position, after accounting for background noise it was 2.17%.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TTCAACAGA_0	122	A>T (57.34)	2.14
CCAGGCTGG_0	146	A>C (64.51)	2.17

*After accounting for background noise (0.54%)

Table 8: Read 1 D19S433 patterns before error

These positions were also associated with two patterns after the error for Read 1. A third pattern was found after the errors, which was associated with position 151 on the suffix (Table 9. The most common error was a substitution error from A to C (31.51%), the second most common error was a substitution error from A to T (31.45%). 0.67% of the total forward reads have an error on this position, after accounting for the background noise it was 0.12%.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TCTTATTCT_0	122	A>T (57.34)	2.14
TGCAGTGGT_1	146	A>C (64.51)	2.17
TGGTACAAT_1	151	A>C (31.51)	0.12

*After accounting for background noise (1.52%)

Table 9: Read 1 D19S433 patterns after error

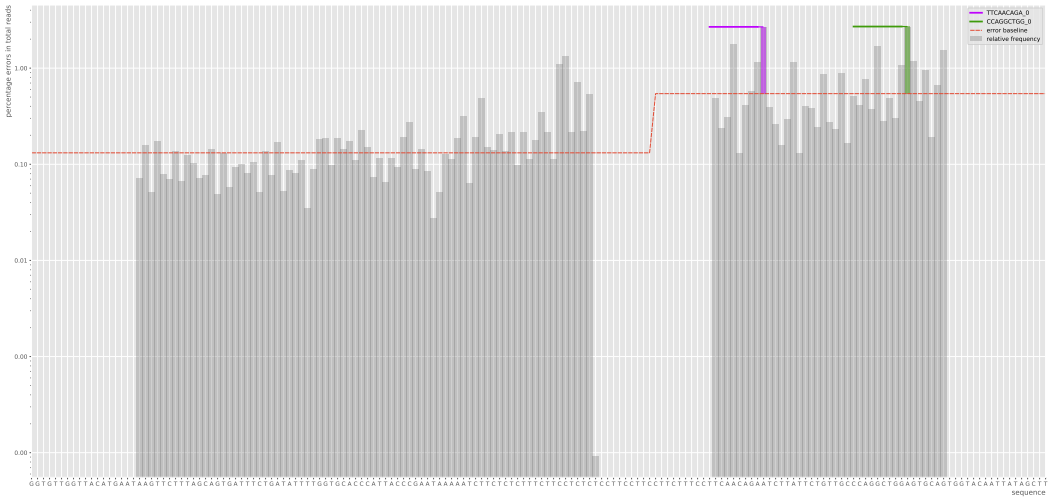

Figure 22: Read 1 D19S433 patterns before error

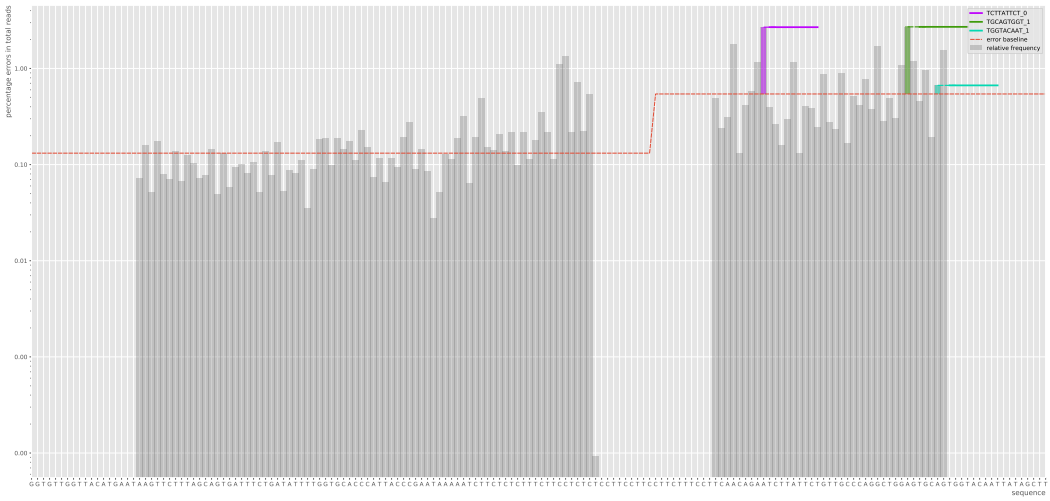

Figure 23: Read 1 D19S433 patterns after error

For Read 2 multiple patterns were associated with one position. The positions associated with the patterns were on the suffix, which were positions 117, 122, 141 and 146, illustrated in Figures 24 and 42. The positions were all associated with two patterns.

For position 117, 122 and 146 the most common error was a substitution error from A to C (Table 10). For position 141 the most common error was a substitution error from G to T (51.46), the second most common error was a substitution error from A to C (46.43). 6.10% of the total forward reads had an error on this position, after accounting for the background noise it was 4.58%.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TTTCC_4	117	A>C (83.71)	5.60
TCA_0	117	A>C (83.71)	5.60
TCA_5	122	A>C (56.22)	7.77
AGA_0	122	A>C (56.22)	7.77
TGTT_6	141	G>T (51.46)	4.58
CCAG_0	141	G>T (51.46)	4.58
CCAG_5	146	A>C (75.33)	7.30
CTGG_0	146	A>C (75.33)	7.30

*After accounting for background noise (1.52%)

Table 10: Read 2 D19S433 patterns before error

The patterns found after the error were also associated with the same position. Another

pattern was associated with another position on the suffix, which was position 151. The most common error on this position was a substitution error from A to C (45.39%) and the second most common error was a substitution error from A to T (34.31%). 2.15% of the reads had an error on this position, after accounting for the background noise it was 0.63% (Table 35).

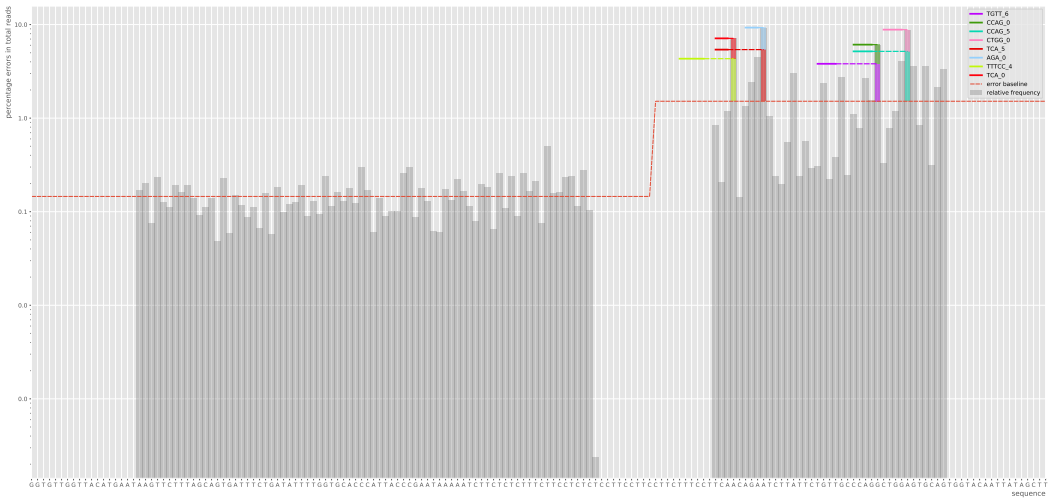

Figure 24: Read 2 D19S433 patterns before error

3.6 D22S1045

Marker D22S1045 had one significant pattern before the error and one after the error for Read 1 and Read 2, illustrated in Figure 25 and Figure 53 in Appendix D. The Read 2 Figures are shown in Appendix D. The error ratios for Read 1 and Read 2 were 0.10 and 0.11. The marker had a prefix of length 51 and a suffix of length 24.

The pattern was associated with one position on the prefix, which was position 68. The most common error on this position was a substitution error from C to T (99.30%). 1.39% of the total forward reads had an error on this position, after accounting for the background noise (0.12%) it was 1.27% (Table 11). For Read 2 the same position was associated with the patterns, shown in Appendix C.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATAGTAGTCT_0	68	C>T (99.30)	1.27

*After accounting for background noise (0.12%)

Table 11: Read 1 D22S1045 patterns before error

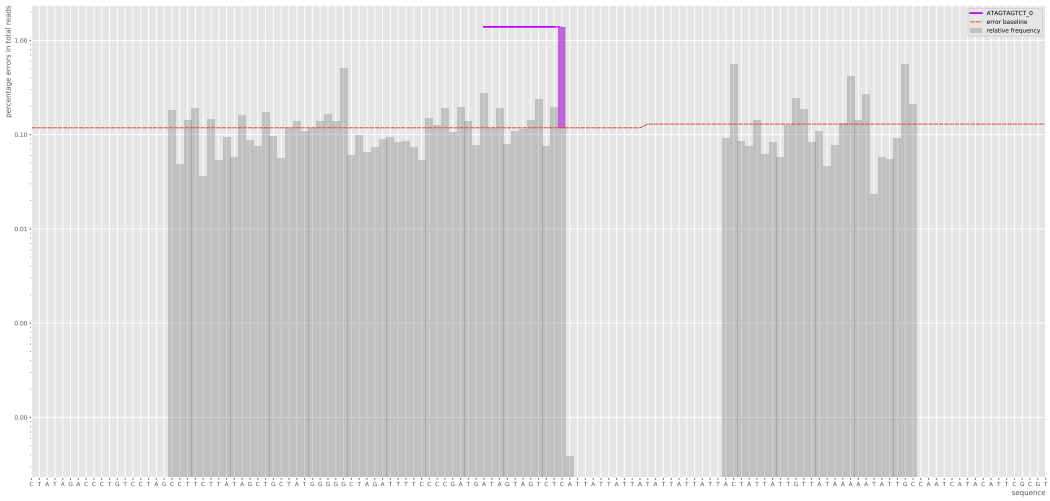

Figure 25: Read 1 D22S1045 patterns before error

3.7 D3S1358

Marker D3S1358 had significant patterns in Read 1 and Read 2. The error ratios for Read 1 and Read 2 were 0.15 and 0.24 with a prefix length of 100 and suffix length of 22. For Read 1 only one pattern was found after the errors, where the most common error was a substitution error from G to T (81.78%). After accounting for the background noise (0.25%) 0.75% of the total forward reads in Read 1 had an error on this position (Table 12).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTCTTGCTCT_0	146	G>T (81.79)	0.75

*After accounting for background noise (0.25%)

Table 12: Read 1 D18S51 patterns after error

For Read 2 there were three patterns before and three patterns after the errors, illustrated in Figures 26 and Figure 46 in Appendix D. These were associated with three different positions on the suffix. For the three positions the most common error was a substitution error from G to T (Table 13).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATGAGACAG_0	146	G>T (86.16)	1.69
TGAGACAGG_0	147	G>T (83.26)	1.34
TCTTGCTCT_0	157	G>T (75.98)	1.68

*After accounting for background noise (0.65%)

Table 13: Read 2 D3S1358 patterns before error

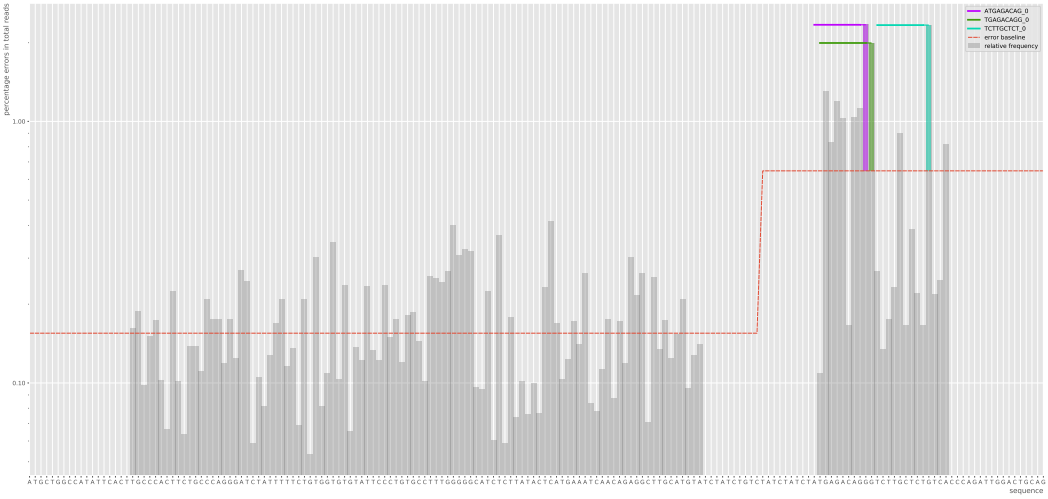

Figure 26: Read 2 D3S1358 patterns before error

3.8 D7S820

Marker D7S820 had significant patterns in Read 1 and Read 2, illustrated in Figure 27 and in Appendix D. There were eight significant patterns found before the errors that were associated with twelve different positions. One pattern was found on four positions, which were the positions 115, 119, 135 and 154 on the suffix. The most common error for these positions were a substitution to a T and substitution to an A (Table 14.

Two patterns were associated with three positions on the prefix, which were positions 80, 81 and 82. These positions showed deletion and insertion errors where after a consecutive number of As, the most common errors were either an insertion of an A, deletion of an A or multiple As.

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AAAAAAA_0	80	AA>–(79.38)	0.32
	82	–>A (82.00)	0.40
TAAAA_4	81	A>–(97.94)	6.87
AACTAT_0	86	C>T (78.61)	0.03
	135	G>A (73.04)	7.18
TCTA_6	115	G>T (87.05)	4.67
	119	G>T (87.22)	5.22
	135	G>A (73.04)	7.18
	154	C>A (57.72)	0.42
GTTA_4	123	G>T (84.98)	8.44
ACAA_0	140	G>T (68.18)	10.25
TATG_4	140	G>T (68.18)	10.25
ACAA_2	142	G>T (63.05)	6.53
TATG_6	142	G>T (63.05)	6.53
TAAAA_4	182	T>C (73.16)	0.51

After accounting for background noise (0.10% in the prefix 0.89% in the suffix)

Table 14: Read 1 D7S820 patterns before error

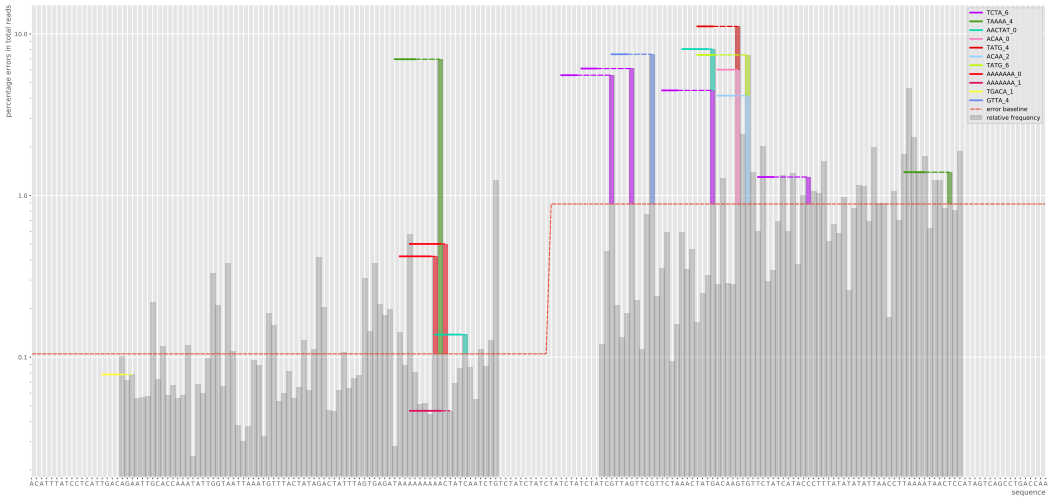

Figure 27: Read 1 D7S820 patterns before error

3.9 PentaD

Marker PentaD had a prefix (length of 72) and a suffix (length of 59) with error ratios 0.21 for Read 1 and 0.28 for Read 2. Significant patterns were found in Read 1 and Read 2. Two significant patterns were found before and after the errors for Read 1 and Read 2, illustrated in Figure 28 and Figure 56 in Appendix D. The Read 2 patterns are shown in Appendix D.

One pattern was associated with two positions, one in the prefix and one in the suffix, which were positions 88 and 111. The most common error on position 88 was a deletion of an A (A>–), which was 91.18% of the errors on this position. For position 111, the errors were below the background noise. 3.33% of the forward reads had an error on position 88, after accounting for background noise (0.13%) it was 3.20%.

Another pattern was associated with one position on the suffix, which was position 124. The most common error on this position was a deletion of an A (78.86%) and the second most common error was a substitution error from A to G (19.85%). 2.27% of the total forward reads had an error on this position, after accounting for background noise it was 2.1%. The same positions were associated with the patterns found after the errors (Table 15).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AAGAAAAAA_1	88	A>–(91.18)	3.20
GGAAAAAA_0	124	A>–(78.86)	2.10

*After accounting for background noise (0.13%)

Table 15: Read 1 PentaD patterns before error

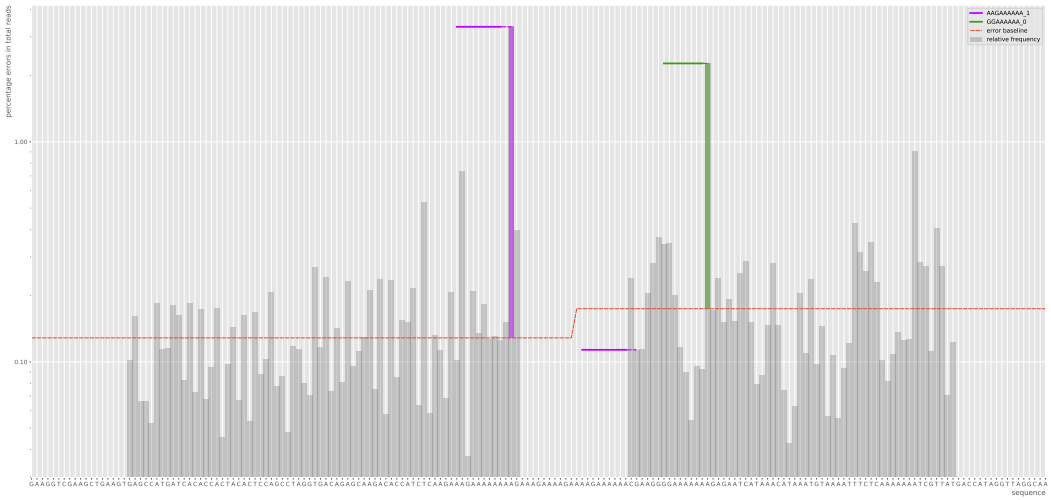

Figure 28: Read 1 PentaD patterns before error

3.10 PentaE

Marker PentaE had a prefix (length of 103) and suffix (length of 8) and error ratios of 0.19% for Read 1 and 0.24% for Read 2. Significant patterns were found for Read 1 and Read 2, illustrated in Figure 29 and Figure 59. Two patterns were found before the errors and two patterns were found after the errors. The patterns were associated with two positions on the suffix, where the most common error was a substitution error from G to T (Table 16).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTTTT_4	144	G>T (96.17)	5.95
TTTCTTTGA_0	146	G>T (93.14)	1.23

*After accounting for background noise (0.56%)

Table 16: Read 1 PentaE patterns before error

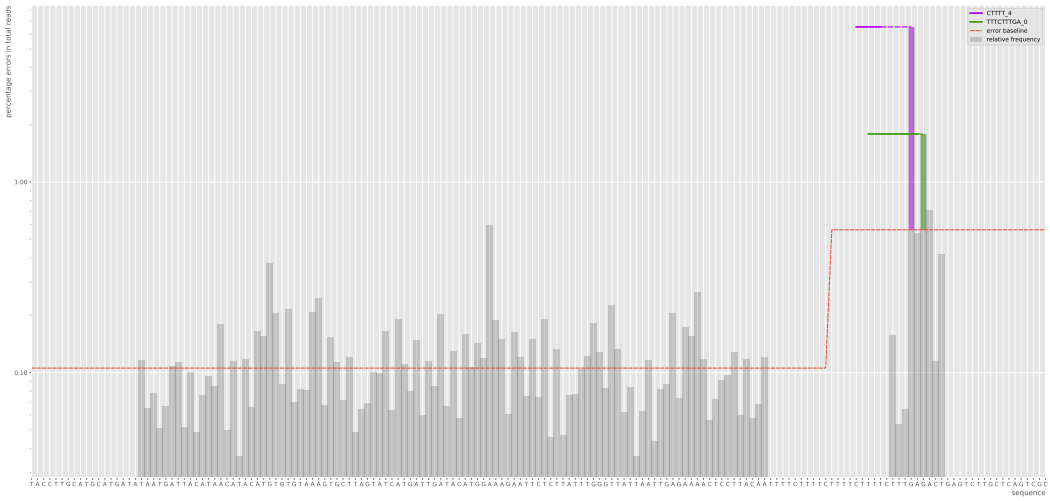

Figure 29: Read 1 PentaE patterns before error

For Read 2 the same positions were associated with the patterns found, a third pattern was found before the error and after the error which was associated with position 147 on the suffix, illustrated in Figure 30. The most common error on this position was a substitution error from A to T (74.88%). After accounting for background noise the position (1.34%) 0.73% of the total forward reads had an error on this position (Table 17).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TTTCTTT_0	144	G>T (93.47)	5.80
TTTGA_0	146	G>T (90.60)	3.17
TTCTTTGAG_0	147	A>T (74.88)	0.73

*After accounting for background noise (1.34%)

Table 17: Read 2 PentaE patterns before error

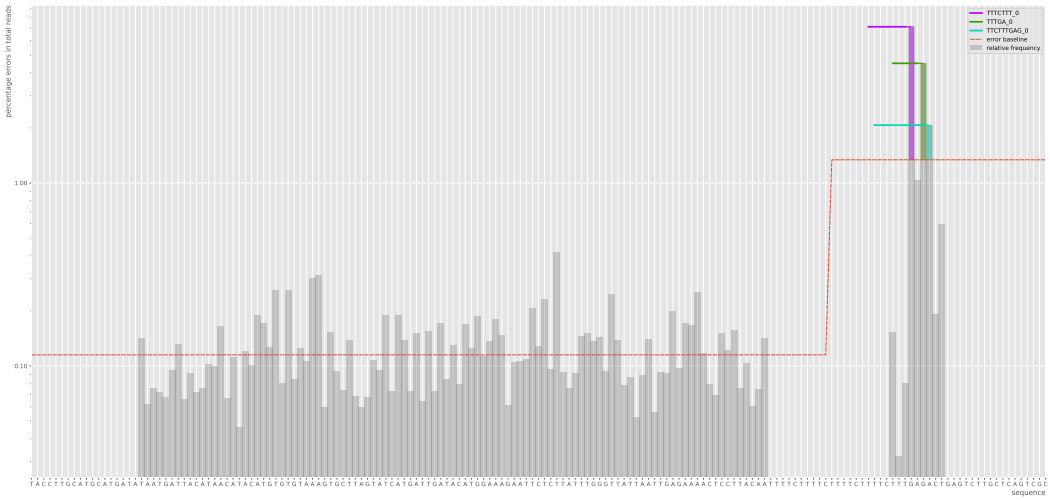

Figure 30: Read 2 PentaE patterns before error

3.11 TH01

Marker TH01 had a prefix (length of 8) and a suffix (length of 164) with error ratios of 0.21% for Read 1 and 0.46% for Read 2. Significant patterns were found for Read 2, illustrated in Figure 31 and Figure 61 in Appendix D. Six patterns were found before the error and three patterns were found after the errors. All the patterns were associated with positions on the suffix. The most common error for the positions were substitution errors to C (Table 18).

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTGG_0	162	G>C (52.17)	0.94
	201	T>C (60.21)	7.12
GCAG_2	194	A>C (86.62)	4.19
	198	T>C (59.59)	4.19
GCAGG_4	201	T>C (60.21)	7.12
GCAGG_5	202	T>C (56.71)	3.33

*After accounting for background noise (0.44%)

Table 18: Read 2 TH01 patterns before error

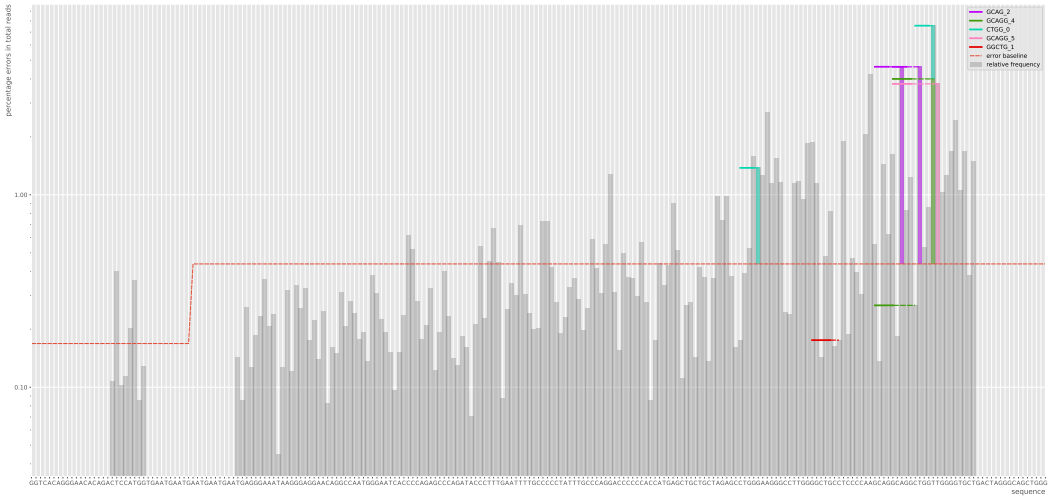

Figure 31: Read 2 TH01 patterns before error

3.12 Sequence length

To determine whether other factors contributed to significant patterns showing in markers, the lengths of the sequences per marker were compared. The average length of the complete sequence (prefix, STR and suffix) for the genuine sequences were compared between the markers with and without significant markers (Table 19). We found that the markers without significant patterns had a lower average length ($\mu = 148.94$) than markers with significant patterns ($\mu = 170.66$).

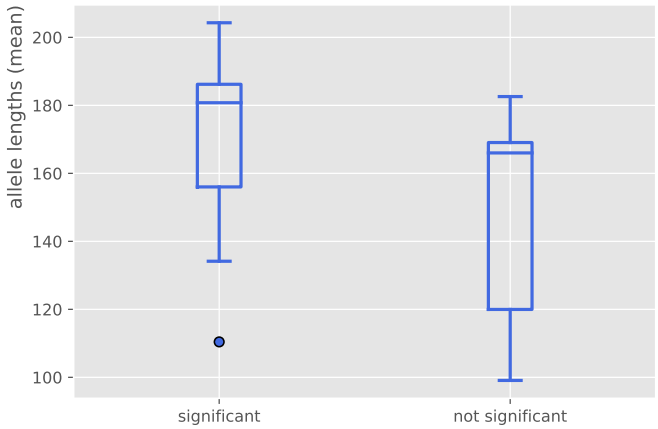

Figure 32: Boxplot of the mean allele lengths of markers with and without significant patterns

	n	Mean (σ)
Significant	15	170.66 (26.47)
Not significant	10	148.94 (31.12)

Table 19: Mean allele lengths and standard deviations of markers with and without significant patterns

3.13 Error ratio and pattern count

The relationship between the error ratios for markers with significant patterns and the number of significant patterns are illustrated in Figure 33. The calculated pearson's r was 0.78 ($p = 1.0.4\mathrm{e}^{-0.5}$), which is a positive correlation, markers with lower error ratios were related to having less patterns than markers with higher error ratios.

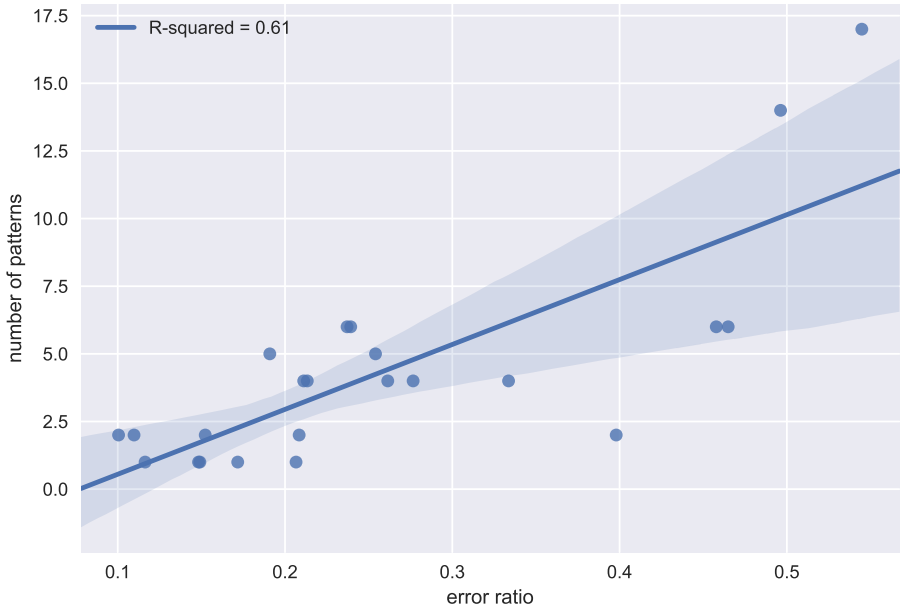

Figure 33: The error ratio plotted against the number of patterns found for each marker with significant patterns

4 Conclusion and Discussion

Sequential pattern mining was used to determine whether patterns were more apparent before or after errors were made. This was done by taking subsequences before and after all errors for each marker. After determining the patterns using a low support, the patterns were evaluated against the complete sequence. So as to determine whether the patterns were more apparent with errors than without errors. After this process seven markers for Read 1 remained with significant patterns and fourteen for Read 2. One possibility that Read 2 had more markers with significant patterns than Read 1 was that Read 2 had higher error ratios in general than Read 1. If the error ratios were high then more errors were made during the PCR and sequencing, which could lead to patterns reaching significance faster.

Within each marker the patterns showed a bias towards certain errors being made. For instance, the patterns for marker D19S433 were associated with positions where the majority of the errors were substitution errors to C. This might make it more straightforward to use these patterns for recognizing sequences as non genuine sequences. If we know that a pattern associated with a position predominantly makes one type of error, then this type of error will most likely be prominent in newly sequenced samples as well. This bias towards one type of error within a marker is found in Read 1 and in Read 2, this suggests that these errors are already present before sequencing, which means that they were made during the PCR.

Some markers showed patterns that were associated with more than one position. When analyzing the percentage of the total reads that had these errors, these percentages differed per position. For instance, with the marker PentaD, the pattern ('AAGAAAAAA_1') was associated with two positions on the sequence, where one position associated with the pattern was below the background noise. To look further into this, we found that after n consecutive As (A-stretch), a peak in errors occurs on the last A. This was found on three different positions on PentaD. The occurrence of an A-stretch was also seen in marker D7S820 in the prefix. The majority of the errors in these peaks are a deletion of an A. One might easily infer from this that after a stretch of As there is a likelihood that an error will appear on the last A. These patterns are however more difficult to determine with only sequential pattern mining due to the variance of the length of the A-stretches, some might be six bases long others might be eight bases long.

The patterns that were significant within the markers in the forward orientation were mostly found at the end of the sequence, which was the suffix in this case. This could be due to sequencing being more prone to errors near the end of a sequence. This suggests that for the reverse reads it could be the other way around, where the patterns would be more emergent in the prefix of the sequence, which in this case is the end for the reverse strand. If the same patterns are found associated with the same position for the forward and reverse orientation that are associated, we could infer that there might be a systematic error occurring on this position, which is less dependent on whether the sequencing machine was nearing the end of the sequence.

With the patterns that we have obtained, it was possible to determine certain sequences as non genuine sequences. For instance, if there is prior knowledge that a given pattern x was associated with a specific position having one type of error which occurs in two percent of the

total reads. Given a sample of 1000 reads, we expect to find twenty reads to have this error on that specific position after finding pattern x. In the example illustrated in Figure 34, it is possible to consider Seq_3 as a non genuine sequence. Ultimately, this would be the goal of using patterns to predict the amount of reads that could be considered as non genuine sequences.

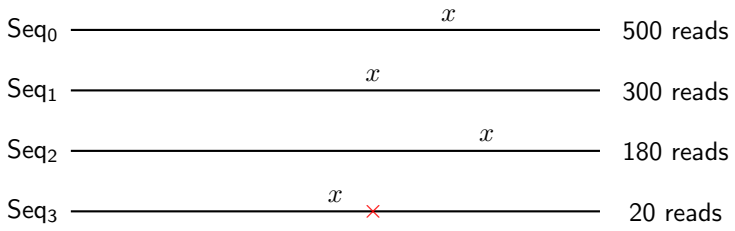

Figure 34: Example of a sequenced unknown sample, x is where the pattern is found and the cross indicates an error on Seq_3

To summarize the answer to the first research question: what patterns were associated with DNA amplification and DNA sequencing? We have found patterns in markers that show certain associations with positions in the sequence. Most of these patterns appear to be at the end of the sequence due to the read orientation (forward). Although not all markers showed significant patterns which could be related to the error ratio or sequence length of the marker. To answer the second research question: how can we use these patterns for prediction? The patterns found to be associated with certain positions made it possible for predicting the expected reads that contain a specific type of error before or after a given pattern. Although the accuracy of the prediction was not tested in new samples yet.

4.1 Limitations

We have found that the patterns solely are not able to explain the behaviour of the errors in a sequence. Many other factors need to be taken into account too, such as read direction, sequence length and the complexity of the sequence, for instance sequences containing A-stretches. Furthermore, we limited the sequences to the pre- and suffix, whereas errors are also made in the STRs. The errors in the STRs are more difficult to label with a correct error description due to variations in repeat lengths and stutters.

4.2 Future Research

The way that the sequences were converted allowed for patterns being mined for positions specific to a distance from the pattern. Although this deviates from the general notion of sequential pattern mining, this made it easier to determine patterns associated with the peaks in errors seen within markers. Due to the fact that we used subsequences, these position related patterns could be found on different places on the complete sequence. This conversion made the sequential

pattern mining to be more relevant for our purpose. A prospective research would be to analyze the nucleotide stretches and the associated error rates with the goal to develop a more DNA specific sequential pattern mining algorithm for error analysis.

Another matter to consider are the co-occurring errors within one sequence. We have only calculated the co-occurrences of errors within a specific range, although it might be possible that errors show a co-occurrence on the entire sequence. For instance, it could be that during sequencing the same type of error is made on different positions on the entire sequence. If this is the case, we might be able to show that the pattern which is related to one position is also occurring with the same type of error on another position. When both errors occur, this will show a stronger case towards suggesting that the sequence might be non genuine.

FDSTools uses a default baseline to indicate the percentage background noise. With the use of the relative frequencies per position based on the total reads, we calculated a baseline to account for background noise in each marker. This baseline indicates that certain positions are not as prone to errors as other positions. The patterns are mostly found on the positions with the highest peaks. By using the baseline which was calculated by taking the average, many positions are still found to be above the baseline. By defining this baseline correctly, for instance by doubling the average, we are able to eliminate more of the background noise. Using the baseline together with the patterns that we have found, we are able to better account for non genuine sequences for unknown samples.

Lastly, future research could be done by sequencing a few markers on different sequencing machines. If the errors produce the same patterns, then we know that these patterns are not only present in one sequencer, but other sequencing machines as well. This will give insight to what we might expect from sequencing machines.

References

S. Y. Anvar, K. J. van der Gaag, J. W. van der Heijden, M. H. Veltrop, R. H. Vossen, R. H. de Leeuw, C. Breukel, H. P. Buermans, J. S. Verbeek, P. de Knijff, et al. Tssv: a tool for characterization of complex allelic variants in pure and mixed genomes. *Bioinformatics*, 30 (12):1651–1659, 2014.

J.-A. Bright, P. Gill, and J. Buckleton. Composite profiles in dna analysis. *Forensic Science International: Genetics*, 6(3):317–321, 2012.

C. Brookes, J.-A. Bright, S. Harbison, and J. Buckleton. Characterising stutter in forensic str multiplexes. *Forensic Science International: Genetics*, 6(1):58–63, 2012.

J. M. Butler. The future of forensic dna analysis. *Phil. Trans. R. Soc. B*, 370(1674):20140252, 2015.

CeGaT. Next-generation sequencing. `https://www.cegat.de/en/services/next-generation-sequencing/`. Last Checked: 06/02/2018.

P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 40–52. Springer, 2014.

J. Hoogenboom, K. J. van der Gaag, R. H. de Leeuw, T. Sijen, P. de Knijff, and J. F. Laros. Fdstools: A software package for analysis of massively parallel sequencing data with the ability to recognise and correct str stutter and other pcr or sequencing noise. *Forensic Science International: Genetics*, 27:27–40, 2017.

V. Mallawaarachchi. Pairwise sequence alignment using biopython, 2017. Last Checked: 07/02/2018.

M. Schirmer, U. Z. Ijaz, R. D'Amore, N. Hall, W. T. Sloan, and C. Quince. Insight into biases and sequencing errors for amplicon sequencing with the illumina miseq platform. *Nucleic acids research*, 43(6):e37–e37, 2015.

K. J. van der Gaag, R. H. de Leeuw, J. Hoogenboom, J. Patel, D. R. Storts, J. F. Laros, and P. de Knijff. Massively parallel sequencing of short tandem repeats—population data and mixture analysis results for the powerseq^{TM} system. *Forensic Science International: Genetics*, 24:86–96, 2016.

Wikipedia. Polymerase chain reaction. `https://en.wikipedia.org/wiki/Polymerase_chain_reaction`. Last Checked: 06/02/2018.

K. Ye, M. H. Schulz, Q. Long, R. Apweiler, and Z. Ning. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871, 2009.

Appendices

A Explanatory Figures

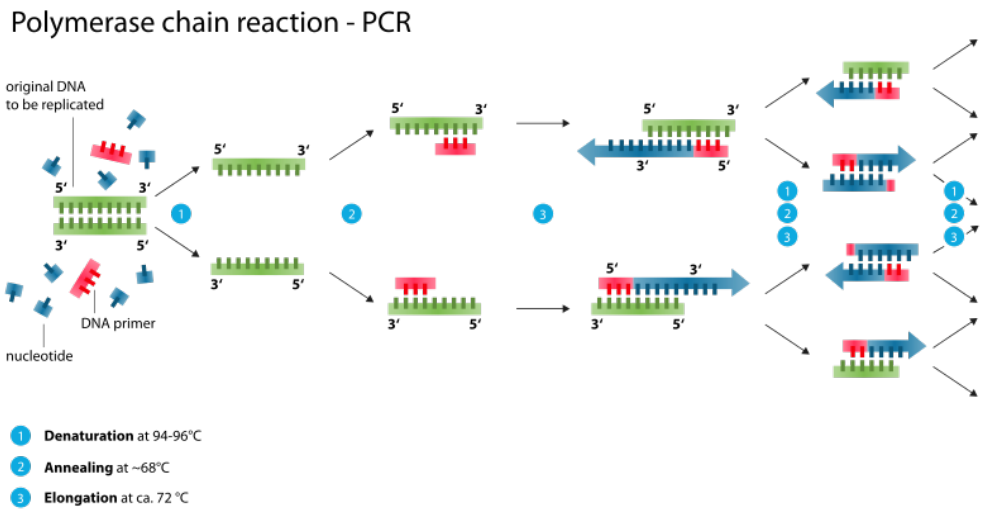

Figure 35: Example of a PCR cycle (Wikipedia)

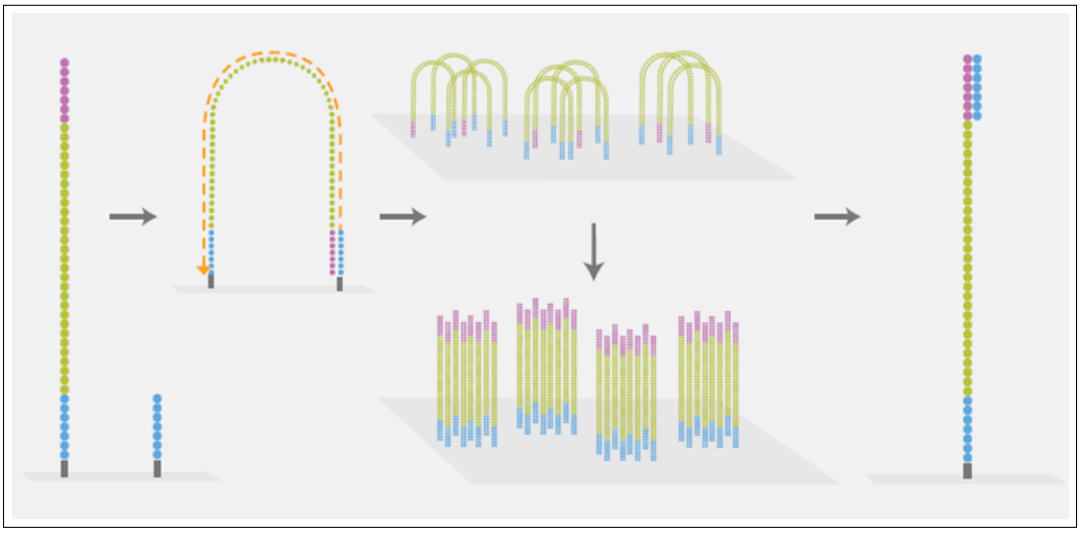

Figure 36: Bridge Amplification in Next-Generation Sequencing (CeGaT)

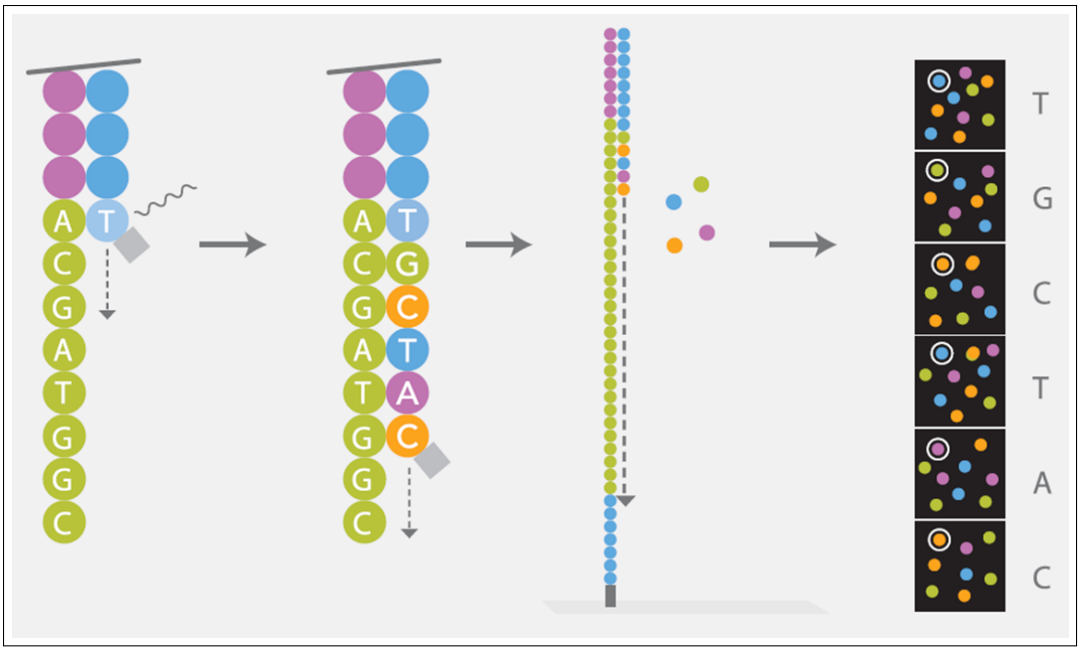

Figure 37: Sequencing the DNA in Next-Generation Sequencing (CeGaT)

B Additional Information

B.1 Polymerase Chain Reaction

With PCR one creates many copies of the DNA. Some basic materials are needed for a PCR, which are primers, the DNA template and nucleotides (the bases). These are all mixed together in a tube and through different temperature cycles will activate certain processes. The first step is heating of the tube, this causes the DNA strands to separate into two single DNA strands, this process is called denaturation. After this process, the tube gets cooled, which causes the primers to bind to the single stranded DNA, this process is called annealing. Primers are DNA sequences that are chosen depending on markers that the researcher wants to amplify. Primers are used as a starting point for the extension of DNA. After this step the tube gets heated again and the floating nucleotides in the tube bind to the single strand of DNA creating a complementary DNA strand. This process is called elongation. This is one cycle of the PCR and this cycle gets repeated a number of times which is user dependent, as illustrated in 35.

B.2 Forward and Reverse Reads

DNA consists of two strands that are complementary to each other, thus the forward refers to one direction of the DNA strand while the reverse is the reverse complement. Given a DNA sequence 'ACTT', the reverse complement of that sequence is 'AAGT', it is complementary and

in reversed order of the other sequence.

B.3 Allelefinder

Allelefinder is a tool from FDSTools that finds all the genuine alleles in a given sample (Hoogenboom et al., 2017). Given a certain threshold it will look at all the markers in the sample and determine which are the genuine alleles. *Allelefinder* uses two thresholds, one is the allele threshold and the other is the noise threshold. The allele threshold is used to define whether a marker is heterozygous. And the allele threshold is a percentage of the highest read, thus the sequence that is most abundant, in the sample. The noise threshold is a threshold that is below the allele threshold. Which indicates that non genuine sequences should not be above this threshold. For a marker to return genuine alleles it needs to have at most two sequences above the allele threshold, if there is a third sequence that exceeds this threshold then no genuine alleles are returned. Another criteria is that there should not be a sequence in the marker that has a sequence below the allele threshold, but above the noise threshold. If this is the case then no genuine alleles are returned either. If two markers do not return genuine alleles then the sample is considered to be of low quality or contaminated. *Alellefinder* returns a detailed report of the sample for each marker with the reasons as to why there were no genuine alleles found.

B.4 Pattern Growth Algorithm

The pattern growth algorithm described in Ye et al. (2009) was used to determine whether a marker that was heterozygotic had distinguishing subsequences that could be used for categorizing the erroneous sequences with the correct genuine sequence.

Given two sequences 'ACGTCT' and ' ACGTTT', the algorithm would find the smallest distinguishing subsequence. First, the two sequences would be concatenated and a break point was determined on the point where the other sequence started. Afterwards a list was created containing each position with the corresponding base (e.g. 0: A, 1: C, ... 11: T). After determining the initial list, the pattern could be extended with the next base and searched for distinguishing patterns. All the patterns that were overlapping with the break point were not taken into account. Thus the next list consisted of the positions with the next base extended (e.g. 0: AC, 1: CG, ..., 10: TT). Whenever the algorithm extended the pattern, it would search for whether there were any distinguishing patterns. If at least one distinguishing pattern was found before and one after the break point the algorithm would stop extending and return the patterns for each sequence. The number of extensions that the algorithm would perform was user defined. For this data set the number of extensions was set to 16.

The distinguishing patterns were then used to split between the error sequences. If one of the patterns appeared in the erroneous sequence it would be categorized to the genuine sequence corresponding to that pattern. If however both of the patterns appeared in the sequence another approach was used, this approach would then extend both the patterns to see whether one of the patterns matched after extension. The match of both patterns could be an instance of an

error causing a complete match, therefore extension of the pattern was occasionally necessary. Another possibility could be that the erroneous sequence is a chimera. A chimera is a PCR artefact where the sequence is a fuse between two sequences. These chimeras do not belong to any of the two genuine alleles and were left out of the data set. Only if the patterns were not sharing overlap on the genuine sequences was it possible to determine chimeras, if this was not the case then the erroneous sequence containing both the patterns would be left out as well if none of the extended patterns matched.

B.5 Co-occurrences

One mismatch was allowed due to the co-occurrence of errors. An error was considered co-occurring when within ten bases another error was also present. To analyze whether this occurred more than expected in the database, binomial tests were used. The co-occurrences were counted within for all the samples within a marker (SDB), and were position specific. If this was not the case, the sub sequences would have been different around each co-occurrence, which would mean that the sub sequence did not have an influence on whether the error co-occurred or not. To test the significance the probability of finding the two co-occurring errors in SDB, $P(error_1,\ error_2,\ SDB)$ was compared to the observed probability of the co-occurrence in SDB, denoted as $P(error_1\ \wedge\ error_2,\ SDB)$. The null hypothesis ($H_0$) stated that the probability of the co-occurrence will be less than or equal to the probability of finding both errors per chance, while the alternative hyptohesis (H_a) stated that it would be greater.

H_0: $P(error_1,\ error_2,\ SDB) \leq P(error_1\ \wedge\ error_2, SDB)$

H_a: $P(error_1,\ error_2,\ SDB) > P(error_1\ \wedge\ error_2, SDB)$

Given two errors a and b, the probability of finding both errors in one sequence was calculated

$$P(a, b,\ SDB) = \frac{count(a,\ SDB)}{n \text{ reads}} \cdot \frac{count(b,\ SDB)}{n \text{ reads}},$$

where $count(error,\ SDB)$ referred to the number of times the error was found in SDB and where n reads were the total number of reads in the SDB.

To determine the probability of the co-occurrence of error a and error b, the number of times error a and error b occurred together were counted and accounted for the total number of reads (n reads),

$$P(a \wedge b,\ SDB) = \frac{count(a \wedge b,\ SDB)}{n \text{ reads}}.$$

The results showed a majority of the co-occurrences to be significant, which was why a mismatch of one was allowed within the subsequences.

C Tables

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTTTAAGCTC_0	184	A>C (65.60)	0.84

*After accounting for background noise (0.22%)

Table 20: Read 1 AmelX patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CCAGCTTCCC_0	184	A>C (65.60)	0.98

*After accounting for background noise (0.26%)

Table 21: Read 2 AmelX patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTTTAAGCTC_0	184	A>C (65.60)	0.98

*After accounting for background noise (0.26%)

Table 22: Read 2 AmelX patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTTTAAGCTC_0	184	A>C (61.11)	0.48

*After accounting for background noise (0.26%)

Table 23: Read 2 AmelY patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AGCAACACAG_0	104	G>T (63.02)	0.91

After accounting for background noise (0.22%)

Table 24: Read 2 D1S1656 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTGCCT_0	126	G>T (79.23)	3.86
GTTGAAG_0	146	A>C (87.91)	3.28

*After accounting for background noise (0.62%)

Table 25: Read 2 D13S317 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATTATTATTA_0	68	C>T (99.30)	1.27

*After accounting for background noise (0.12%)

Table 26: Read 1 D22S1045 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATTATTATTA_0	68	C>T (98.06)	1.29

*After accounting for background noise (0.14%)

Table 27: Read 2 D22S1045 patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATTATTATTA_0	68	C>T (98.06)	1.29

*After accounting for background noise (0.14%)

Table 28: Read 2 D22S1045 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AGCCAT_0	115	T>G (66.11)	2.88

*After accounting for background noise (0.89%)

Table 29: Read 2 D2S1338 patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TCTGTTT_0	115	T>G (66.11)	2.88

*After accounting for background noise (0.89%)

Table 30: Read 2 D2S1338 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTCTTGCTC_0	146	G>T (86.16)	1.69
TCTTGCTCT_0	147	G>T (83.26)	1.34
CACCCAGAT_1	157	G>T (75.98)	1.68

*After accounting for background noise (0.56%)

Table 31: Read 2 D3S1358 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
ATCA_2	81	A>–(97.94)	6.87
TTC_0	119	G>T (87.22)	5.22
	123	G>T (84.98)	8.44
	142	G>T (63.05)	6.53
TAAACTA_3	123	G>T (84.98)	8.44
ACA_0	135	G>A (73.04)	7.18
TTCT_2	140	G>T (68.18)	10.25
ATCA_6	140	G>T (68.18)	10.25
TCAT_5	142	G>T (63.05)	6.53

After accounting for background noise (0.10% in the prefix 0.89% in the suffix)

Table 32: Read 1 D7S820 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AAAAAAA_0	80	AA>–(71.27)	0.27
	81	A>–(97.55)	6.02
	82	–>A (89.76)	1.29
ACAA_0	140	G>T (81.81)	6.35
TATG_4	140	G>T (81.81)	6.35

After accounting for background noise (0.12% in the prefix 0.81% in the suffix)

Table 33: Read 2 D7S820 patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
TATCA_1	80	A>–(71.27)	0.27
TTCTAT_2	140	G>T (81.81)	6.35
TTCTAT_0	142	G>T (72.52)	4.45
AGTC_3	183	C>A (81.81)	6.35

After accounting for background noise (0.12% in the prefix 0.81% in the suffix)

Table 34: Read 2 D7S820 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AGA_1	117	A>C (83.71)	5.60
CTTA_6	117	A>C (83.71)	5.60
ATTC_4	122	A>C (56.22)	7.77
CTGGA_0	141	G>T (51.46)	4.58
GTGG_5	146	A>C (75.33)	7.30
TACAA_4	151	A>C (45.39)	0.63

*After accounting for background noise (1.52%)

Table 35: Read 2 D19S433 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GAAAGAAAA_0	88	A>–(91.18)	3.20
GAGAATCA_0	124	A>–(78.86)	2.10

*After accounting for background noise (0.13%)

Table 36: Read 1 PentaD patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
AAGAAAAAA_1	88	A>–(91.66)	3.19
GGAAAAAA_0	124	A>–(76.73)	2.09

*After accounting for background noise (0.15%)

Table 37: Read 2 PentaD patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GAAAGAAAA_0	88	A>–(91.66)	3.19
GAGAATCA_0	124	A>–(76.73)	2.09

*After accounting for background noise (0.15%)

Table 38: Read 2 PentaD patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GAC_1	144	G>T (96.17)	5.95
TGC_7	144	G>T (96.17)	5.95
ACTGAGTCT_0	146	G>T (93.14)	1.23

*After accounting for background noise (0.56%)

Table 39: Read 1 PentaE patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTG_3	144	G>T (93.47)	5.80
CTGAGT_1	146	G>T (90.60)	3.17
TGAGTCTTG_1	147	A>T (74.88)	0.73

*After accounting for background noise (1.34%)

Table 40: Read 2 PentaE patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GGGG_1	170	T>C (60.21)	7.12
	201	T>C (50.93)	0.70
TGCTG_5	201	T>C (60.21)	7.12

*After accounting for background noise (0.44%)

Table 41: Read 2 TH01 patterns after error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
CTTCTTC_3	110	A>C (66.90)	4.35
TTCTTCTG_2	111	A>C (79.06)	3.37

*After accounting for background noise (0.31%)

Table 42: Read 2 vWA patterns before error

Patterns	Position(s)	Most Common Errror (%)	% Reads Covered*
GTGGGAAC_2	110	A>C (66.90)	4.35
GTGGGAAC_1	111	A>C (79.06)	3.37

*After accounting for background noise (0.31%)

Table 43: Read 2 vWA patterns after error

Marker	Total Reads	Forward Reads	Forward Error Reads*	Forward Total Errors	Mean Error per Read	Forward Error Ratio	Significant Patterns?
AmelX	191672	91439	13928	18477	1.33	0.15	Yes
AmelY	162375	78634	11721	14859	1.27	0.15	Yes
CSF1P0	1874276	1000306	152403	223799	1.47	0.15	No
D10S1248	2465537	1316895	88273	101943	1.15	0.07	No
D12S391	2269906	958280	36008	37610	1.04	0.04	No
D13S317	2116481	981152	155769	207719	1.33	0.16	No
D16S539	1128629	466502	66336	79506	1.20	0.14	No
D18S51	2632844	1243886	160294	189074	1.18	0.13	No
D19S433	2101883	1077688	273827	441635	1.61	0.25	Yes
D1S1656	1540089	806659	70185	80203	1.14	0.09	No
D21S11	1700245	967623	92270	117323	1.27	0.10	No
D22S1045	5032099	2575621	258877	300955	1.16	0.10	Yes
D2S1338	784492	370345	68851	101680	1.48	0.19	No
D2S441	1946940	881777	84964	99307	1.17	0.10	No
D3S1358	1622444	798531	118430	147199	1.24	0.15	Yes
D5S818	1121301	506979	61255	82548	1.35	0.12	No
D7S820	921729	386320	210457	471828	2.24	0.54	Yes
D8S1179	1580077	781308	99495	142492	1.43	0.13	No
DYS391	1834252	1044304	88604	116005	1.31	0.08	No
FGA	1811340	976924	80977	91468	1.13	0.08	No
PentaD	2754727	1366052	291379	369112	1.27	0.21	Yes
PentaE	676055	372899	71168	83221	1.17	0.19	Yes
TH01	1481690	689707	141901	223268	1.57	0.21	No
TPOX	2268531	1150069	179928	242555	1.35	0.16	No
vWA	1130624	557012	69577	89770	1.29	0.12	No

*These error reads were the reads containing errors in the pre- and suffix that were found with alignment

Table 44: Read 1 forward summary descriptives for each marker

Marker	Total Reads	Forward Reads	Forward Error Reads*	Forward Total Errors	Mean Error per Read	Forward Error Ratio	Significant Patterns?
AmelX	169015	78078	16275	28187	1.73	0.21	Yes
AmelY	144464	66096	13655	23152	1.70	0.21	Yes
CSF1P0	1819380	836912	183451	336774	1.84	0.22	No
D10S1248	2341435	1133991	83809	105246	1.26	0.07	No
D12S391	2222977	1284599	60329	65847	1.09	0.07	No
D13S317	2031547	1086065	362341	637519	1.76	0.33	Yes
D16S539	1076354	643606	106148	151499	1.43	0.16	No
D18S51	2527937	1377983	236635	336044	1.42	0.17	Yes
D19S433	1923149	834158	413935	777860	1.88	0.50	Yes
D1S1656	1495787	703175	81823	100098	1.22	0.12	Yes
D21S11	1563566	644607	136119	191036	1.40	0.21	Yes
D22S1045	4956708	2415425	265080	326541	1.23	0.11	Yes
D2S1338	608309	360936	143645	284602	1.98	0.40	Yes
D2S441	1892389	1045388	114443	151168	1.32	0.11	No
D3S1358	1529322	759192	180000	256390	1.42	0.24	Yes
D5S818	1091119	600102	100393	155703	1.55	0.17	No
D7S820	905123	471286	219104	455121	2.08	0.46	Yes
D8S1179	1524178	759909	144717	259581	1.79	0.19	No
DYS391	1789922	767507	87043	124666	1.43	0.11	No
FGA	1703406	838953	72300	89409	1.24	0.09	No
PentaD	2519731	1485700	410909	603939	1.47	0.28	Yes
PentaE	640227	267189	63932	76446	1.20	0.24	Yes
TH01	1270143	681388	311931	736057	2.36	0.46	Yes
TPOX	2135142	1069774	249743	413191	1.65	0.23	No
vWA	1080216	532584	139224	236878	1.70	0.26	Yes

*These error reads were the reads containing errors in the pre- and suffix that were found with alignment

Table 45: Read 2 forward summary descriptives for each marker

D Plots

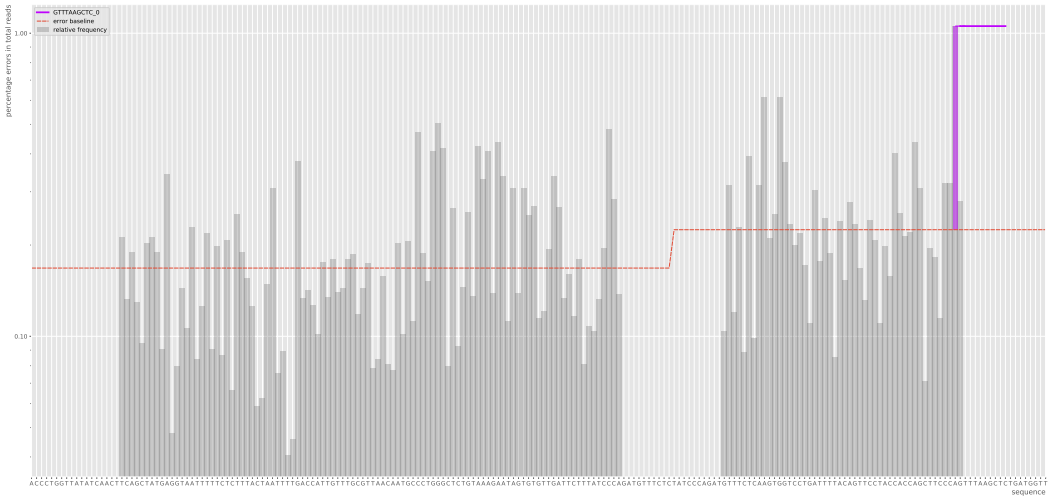

Figure 38: Read 1 AmelX patterns after error

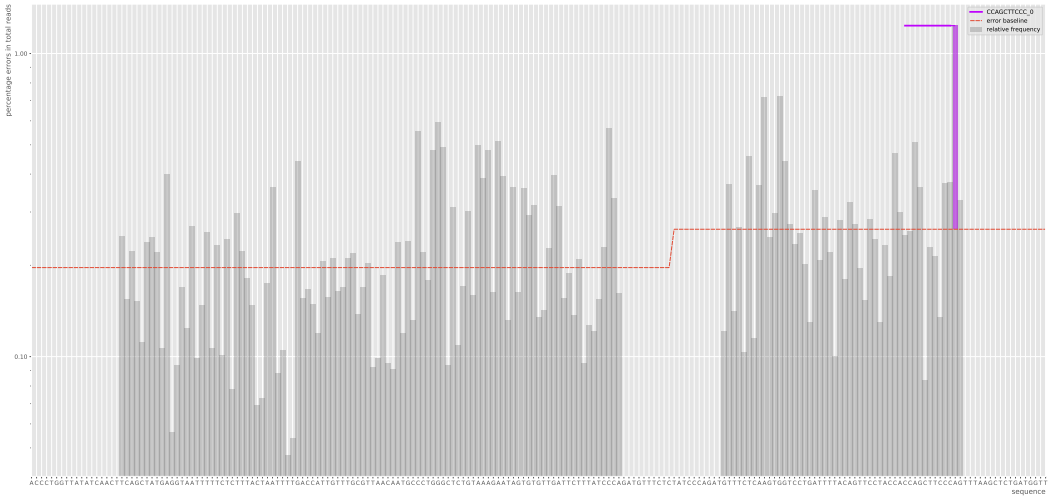

Figure 39: Read 2 AmelX before after error

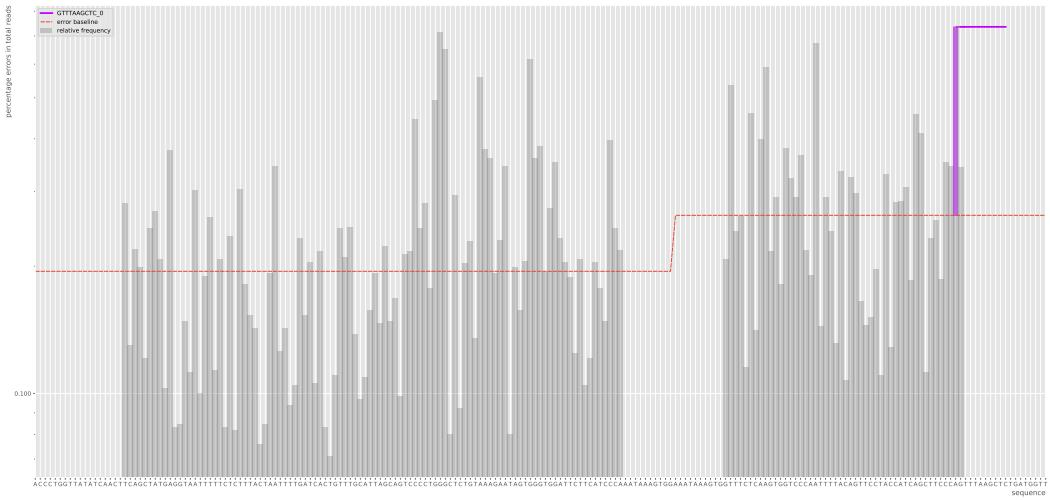

Figure 40: Read 2 AmelY patterns after error

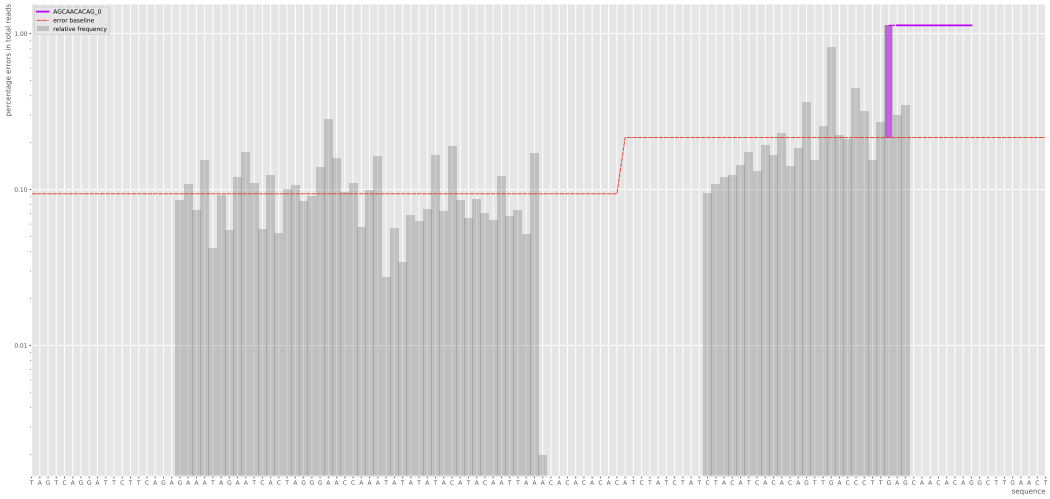

Figure 41: Read 2 D1S1656 patterns after error

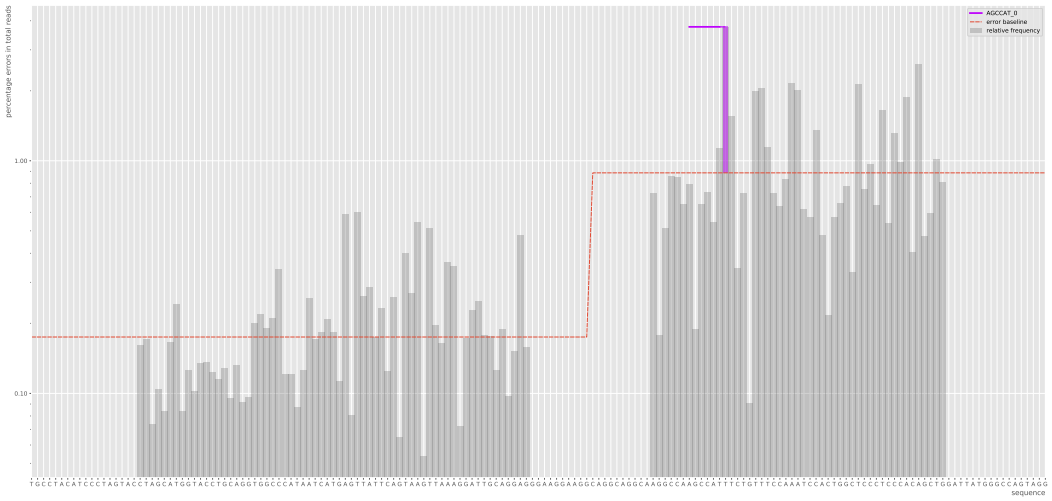

Figure 43: Read 2 D2S1338 patterns before error

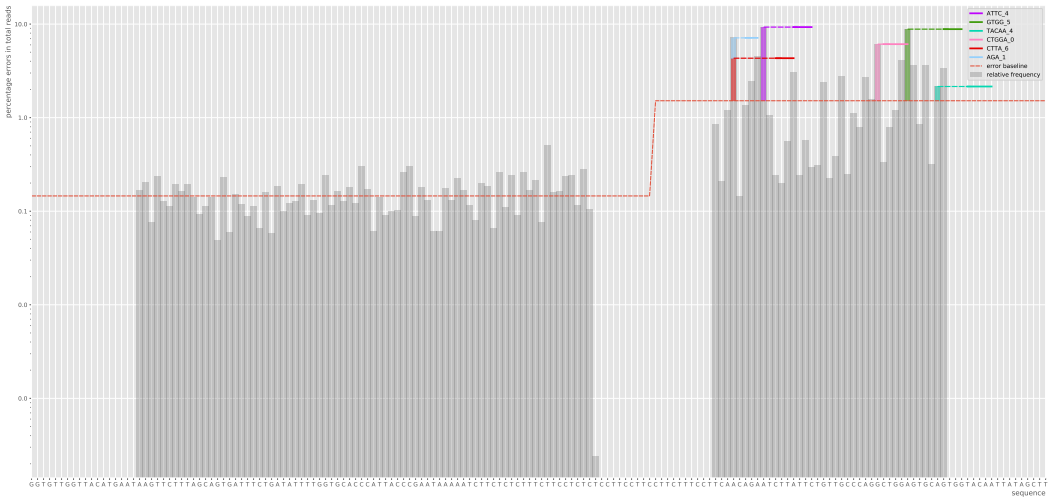

Figure 42: Read 2 D19S433 patterns after error

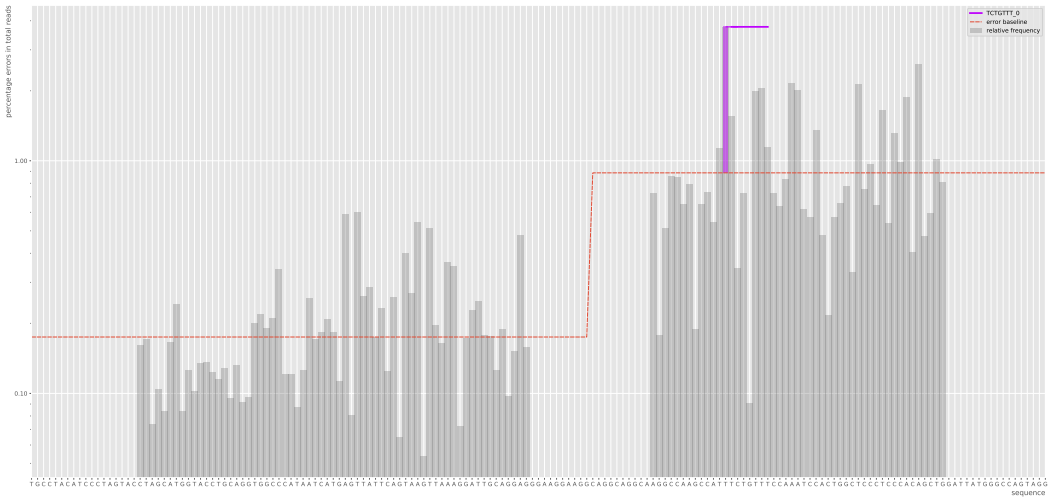

Figure 44: Read 2 D2S1338 patterns after error

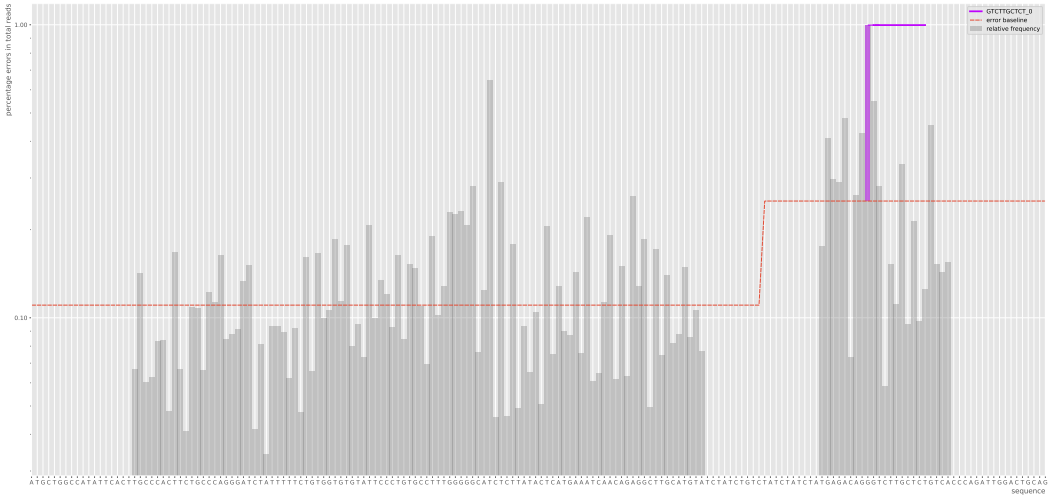

Figure 45: Read 1 D3S1358 patterns after error

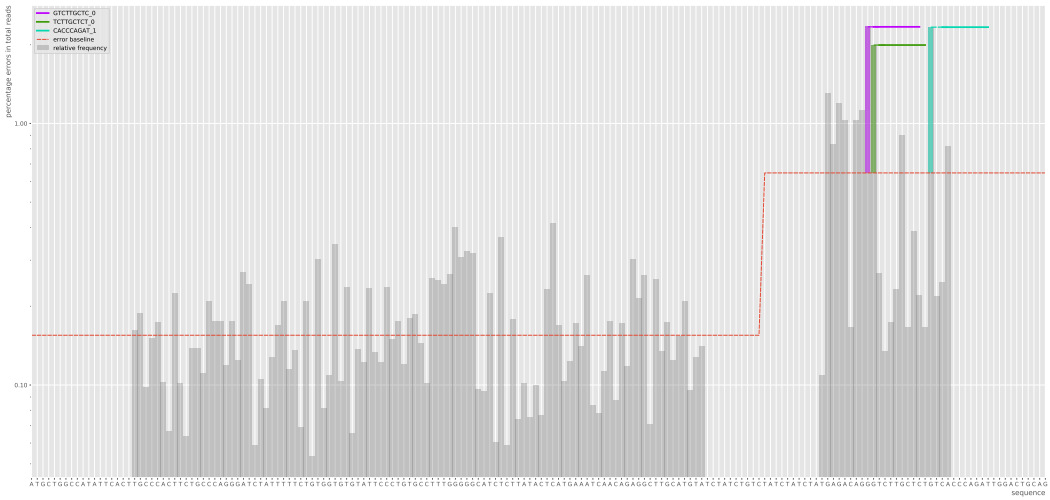

Figure 46: Read 2 D3S1358 patterns after error

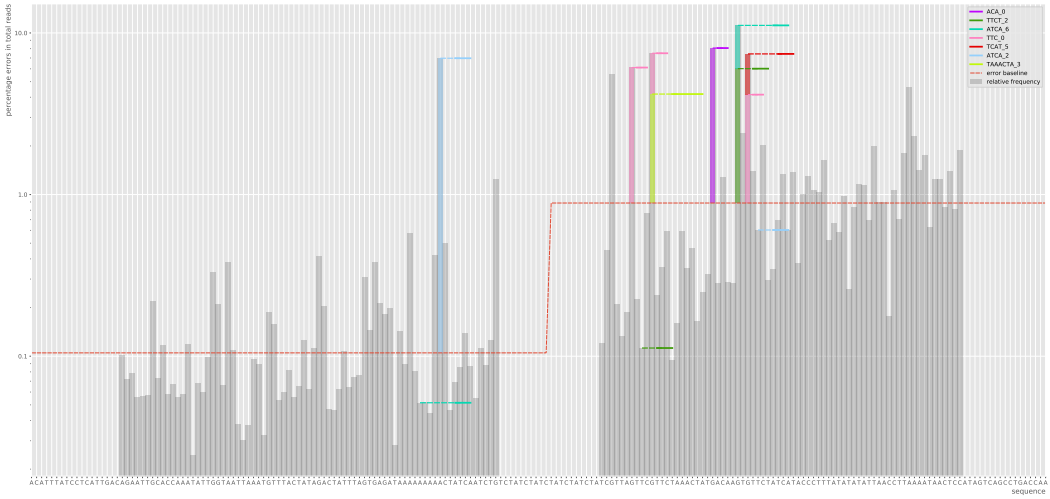

Figure 47: Read 1 D7S820 patterns after error

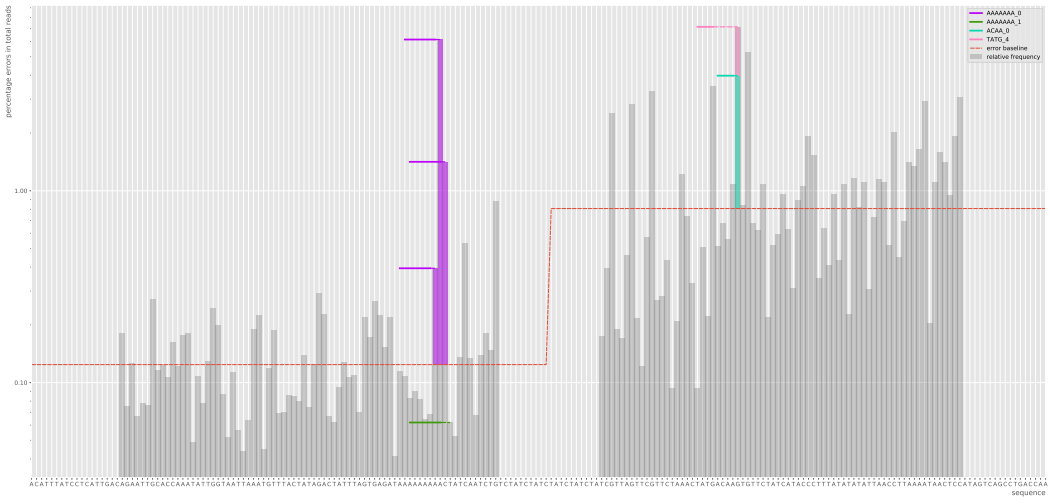

Figure 48: Read 2 D7S820 patterns before error

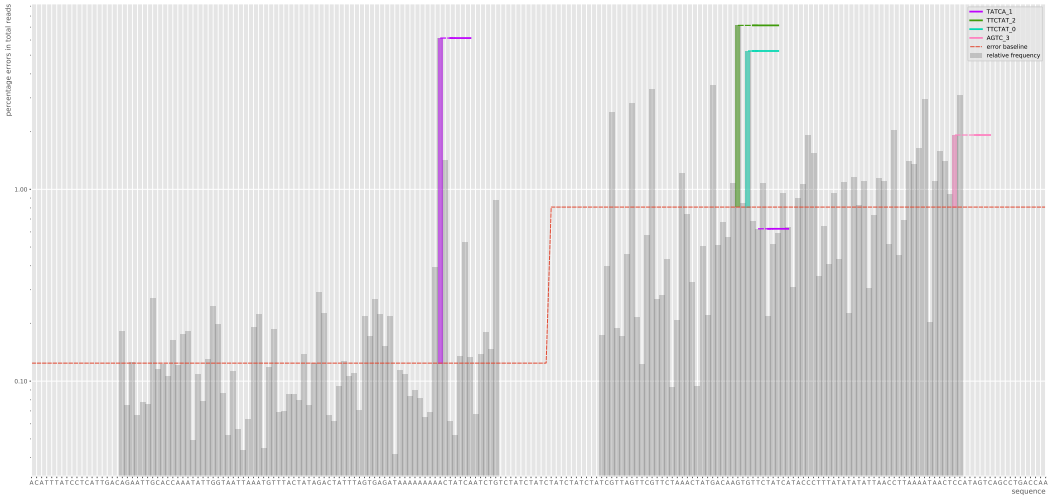

Figure 49: Read 2 D7S820 patterns after error

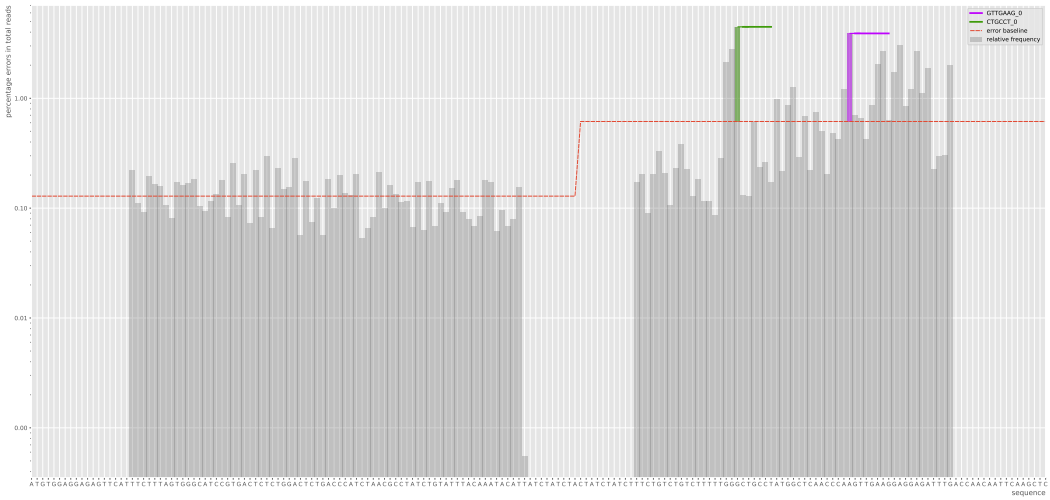

Figure 50: Read 2 D13S317 patterns after error

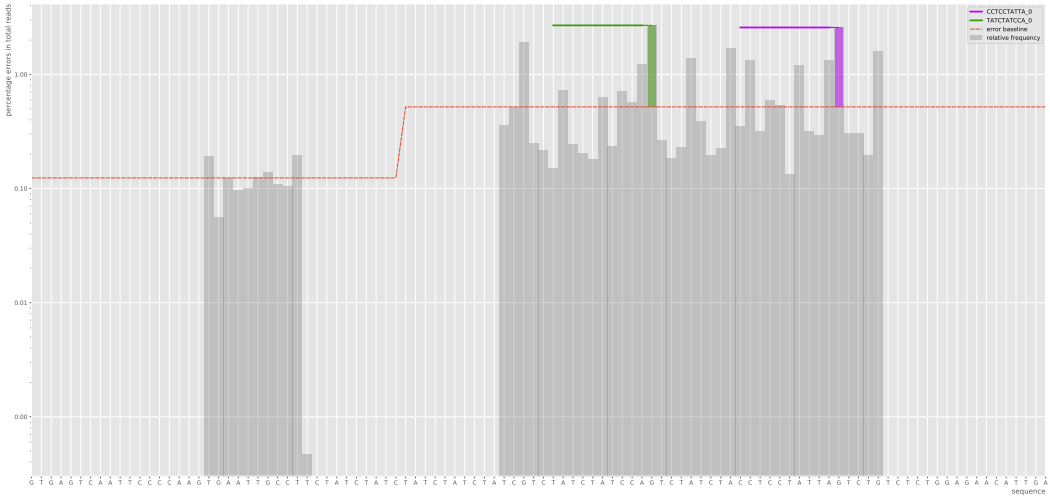

Figure 51: Read 2 D21S11 patterns before error

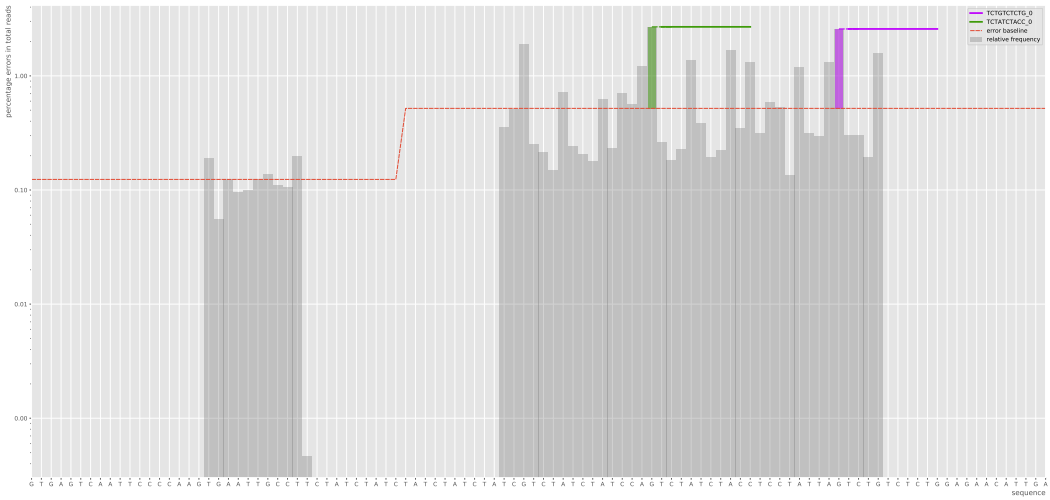

Figure 52: Read 2 D21S11 patterns after error

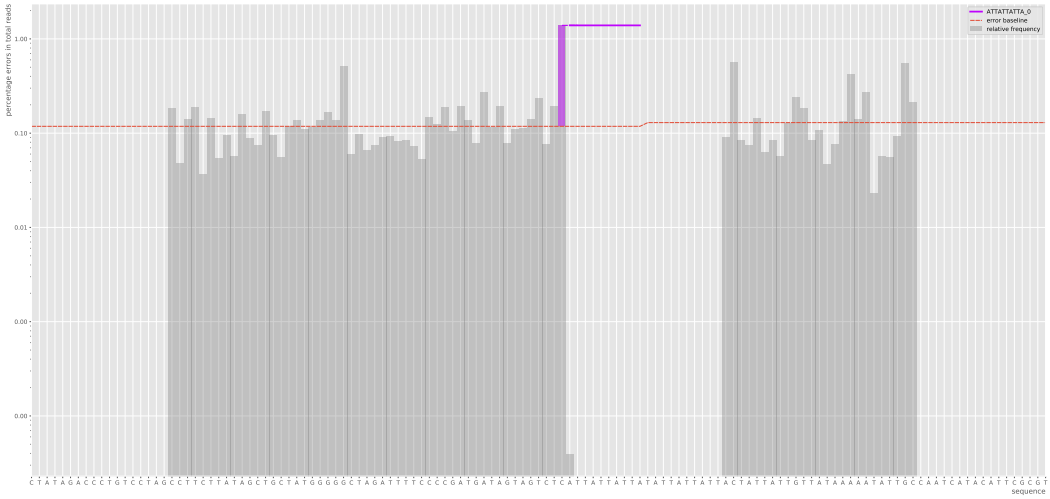

Figure 53: Read 1 D22S1045 patterns after error

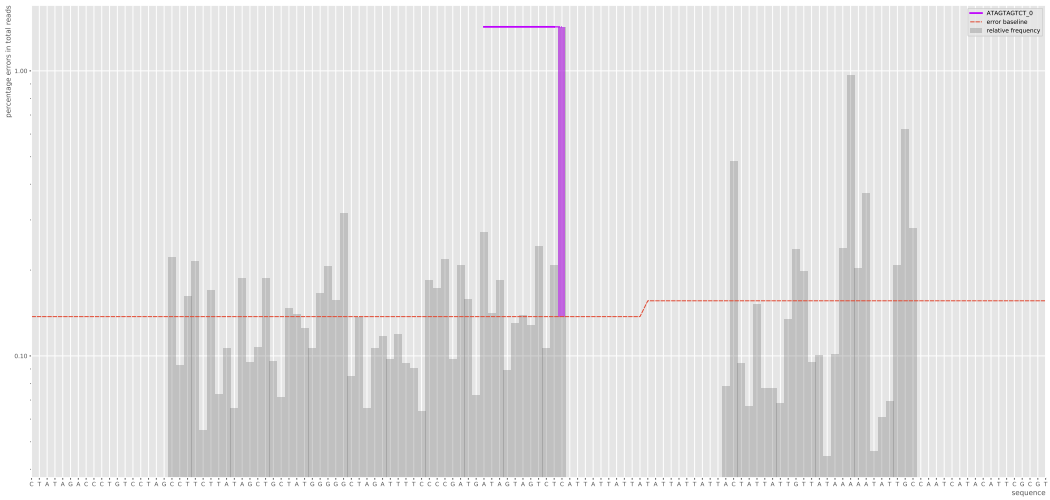

Figure 54: Read 2 D22S1045 patterns before error

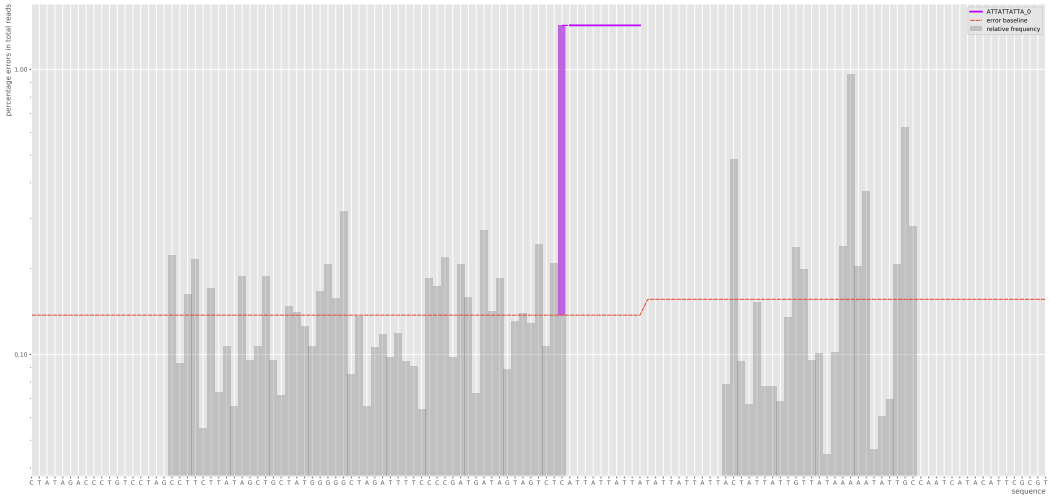

Figure 55: Read 2 D22S1045 patterns after error

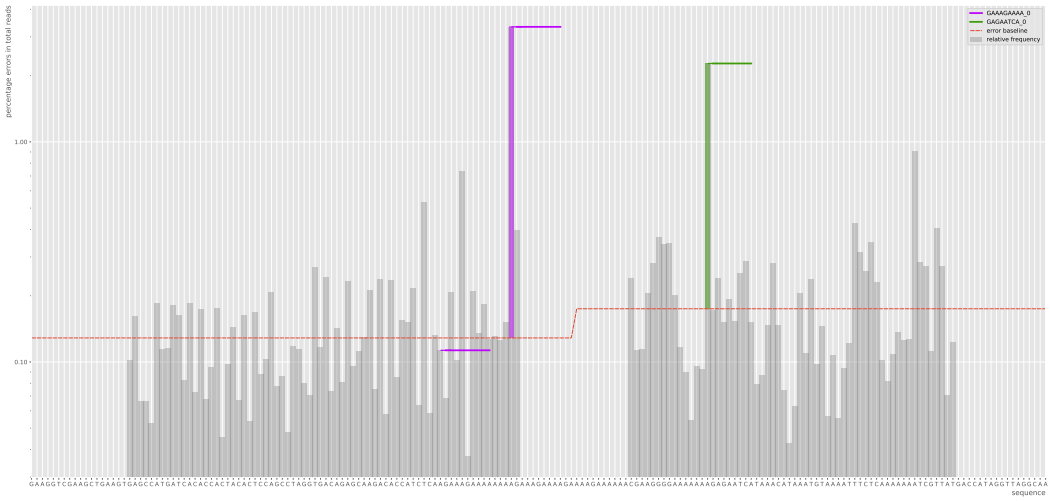

Figure 56: Read 1 PentaD patterns after error

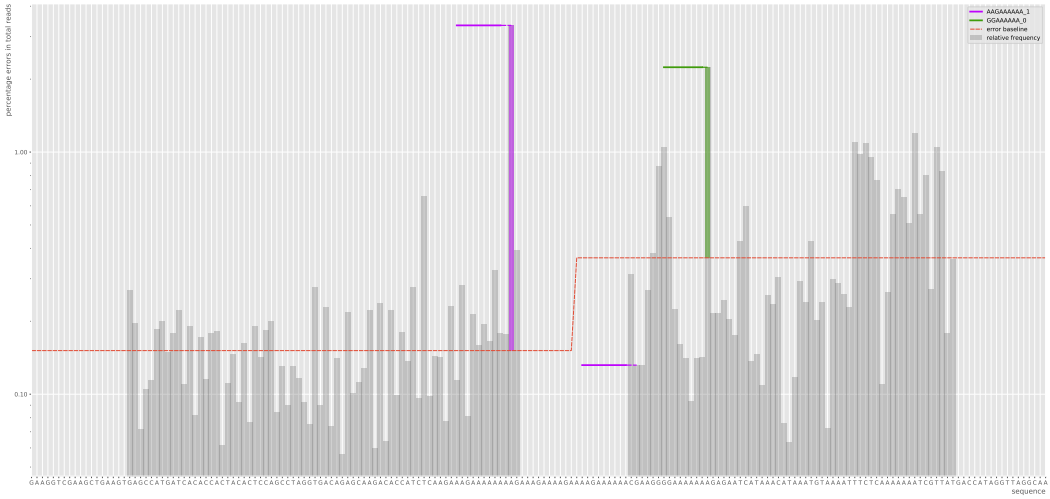

Figure 57: Read 2 PentaD patterns before error

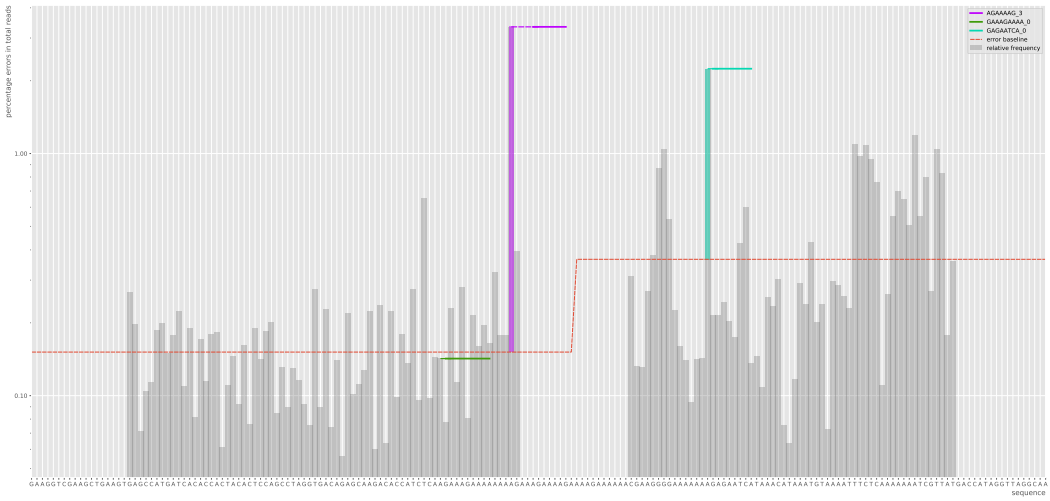

Figure 58: Read 2 PentaD patterns after error

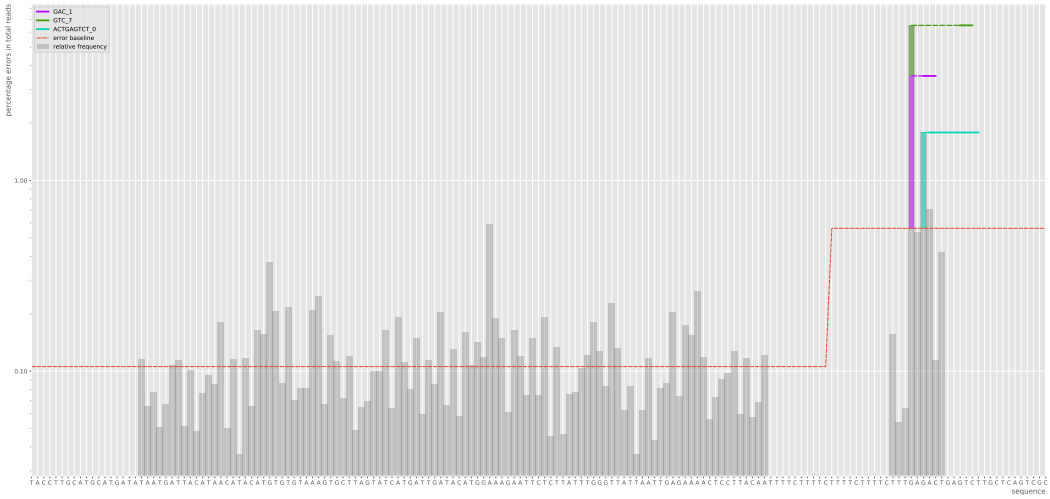

Figure 59: Read 1 PentaE patterns after error

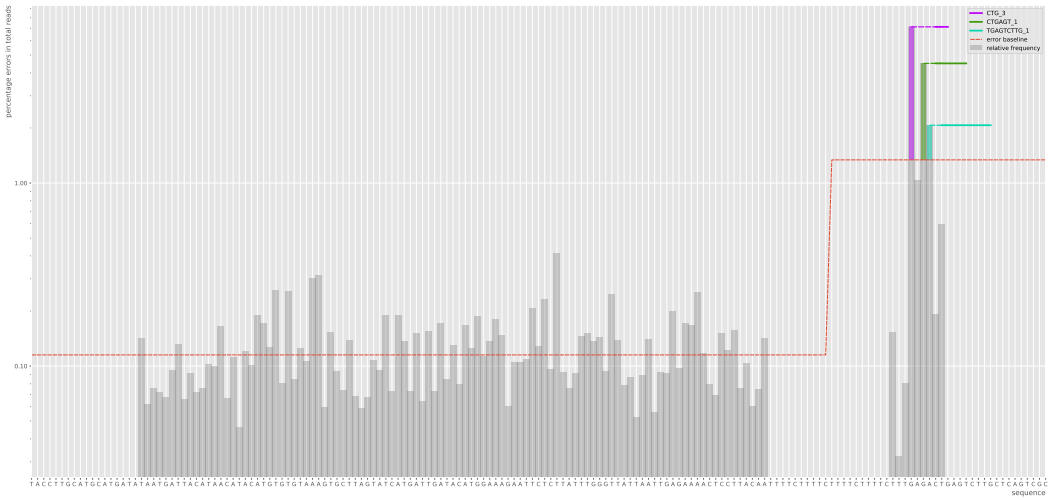

Figure 60: Read 2 PentaE patterns after error

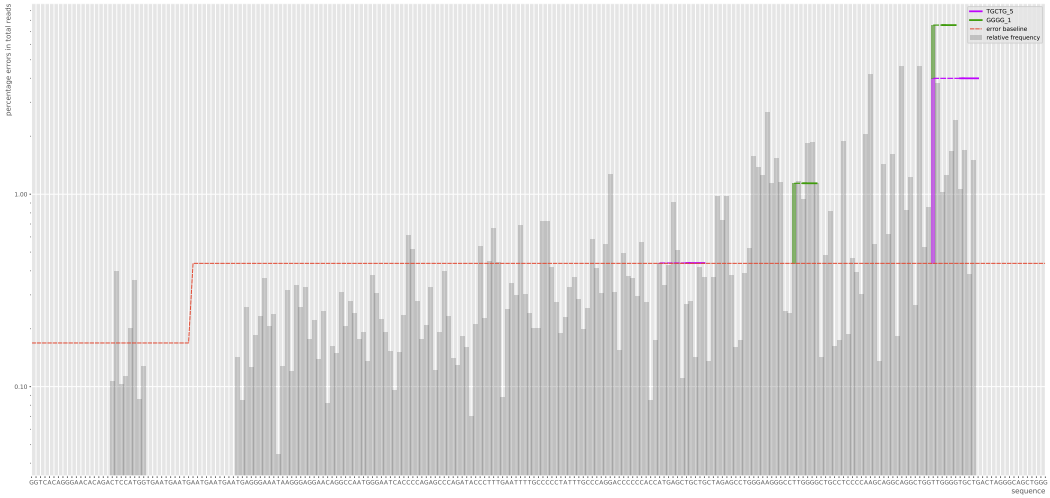

Figure 61: Read 2 TH01 patterns after error

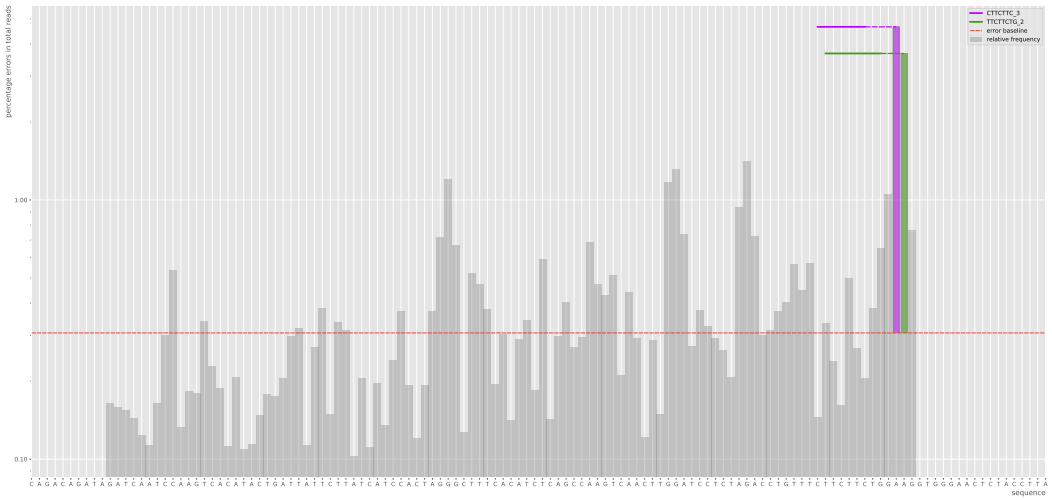

Figure 62: Read 2 vWA patterns before error

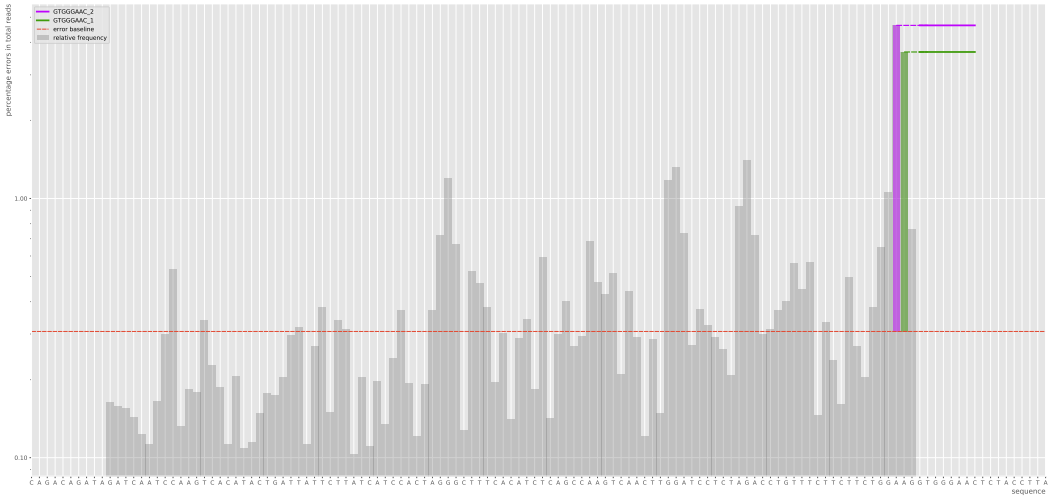

Figure 63: Read 2 vWA patterns after error