UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGENEERING AND COMPUTING

MASTER THESIS no. 1005

Long Read RNA-seq Mapper

Josip Marić

Zagreb, February 2015.

Table of Contents

List of Tables

List of Figures

1. Introduction

Bioinformatics is an interdisciplinary field of science that combines computer science, statistics and biology to process, analyze and study biological data. Its primary goal is to develop methods and software tools that are used in various areas of biology such as molecular biology, genomics and genetics. Annotating genomes and analyzing gene and protein expression and regulation are just some of the fields where bioinformatics plays great role.

A more specific part of bioinformatics is sequence analysis. DNA and RNA sequencing is the process of determining the precise order of nucleotides within a DNA or RNA molecule. Since the first sequencing tool was invented, thousands of organisms have been decoded and stored in the databases. This information is analyzed to determine genes, regulatory sequences motifs etc. The sequencing methods have greatly improved in the past few years accelerating biological and medical research and discovery. This also caused a great need for bioinformatics tools and methods which are also rapidly developing in the past few years.

One type of biological sequences analysis is RNA sequencing that analyzes sequences of RNA molecules obtained by different sequencing tools. This paper will be concentrated on analysis of an RNA-seq tool BBMap and its usage with long sequences of RNA molecules.

Chapter 2 will give a short summary on the RNA sequencing method, tools and problems that occur with RNA sequences which are not found when manipulating DNA sequences. Chapter 3 presents the method that BBMap tool uses for RNA data which was reimplemented for the purposes of further analysis. Chapter 4 presents some improvements to the method that were implemented and tested. Finally in the chapter 5 analysis and evaluation of the method and suggested improvements are given.

2. RNA Sequencing

RNA sequencing (RNA-seq) which is also called *whole transcriptome shotgun sequencing* is the application of any of a variety of next-generation sequencing techniques to study RNA [1]. Transcriptome is the set of all RNA molecules, including mRNA, rRNA, tRNA and other non-coding RNA transcribed in a single cell or a population of cells. RNA-seq reveals a snapshot of RNA presence and quantity from a genome at a given moment in time. Since the transcriptome of the cell continually changes, RNA transcripts in a cell, produced by RNA-seq, provide the ability to detect changes in gene expression which, for example, can be used in cancer studies [3]. Some of the other RNA-seq uses are: detection of alternative gene splicing, detection of SNPs, determination of exon/intron regions and other [2].

RNA molecules in a cell are transcribed from the DNA molecule. By knowing the structure of the DNA molecule, reads, sequenced from RNA molecules, can be mapped to the position in the genome where they were transcribed from. Sequencers that sequence RNA molecules produce sequences of cDNA molecules synthesized from mature mRNA or spliced mRNA. These sequences are reverse complements of RNA molecules, but with DNA nucleotides. Spliced mRNA is RNA transcript that has already been spliced and processed and is ready for translation, while pre-mRNA is a single strand of mRNA synthesized from the DNA template. Splicing is the modification of pre-mRNA in which some regions of the pre-mRNA molecule are removed from the molecule. The remaining regions form spliced mRNA. The removed regions are called introns, and the regions that remain are called exons, as shown in Figure 2.1. Synthesized cDNA reads consist exclusively of exon regions, because they are synthesized from spliced mRNA. cDNA molecules have the same four bases as the DNA molecule, marked A, C, G and T (short for adenine, cytosine, guanine and thymine). The problem of mapping cDNA reads to the reference genome is similar to the DNA sequencing problem, but since cDNA molecules have only exon regions, some of them can span over two exons and contain big gaps. These types of reads are called spliced reads and they make RNA sequencing more difficult than DNA sequencing.

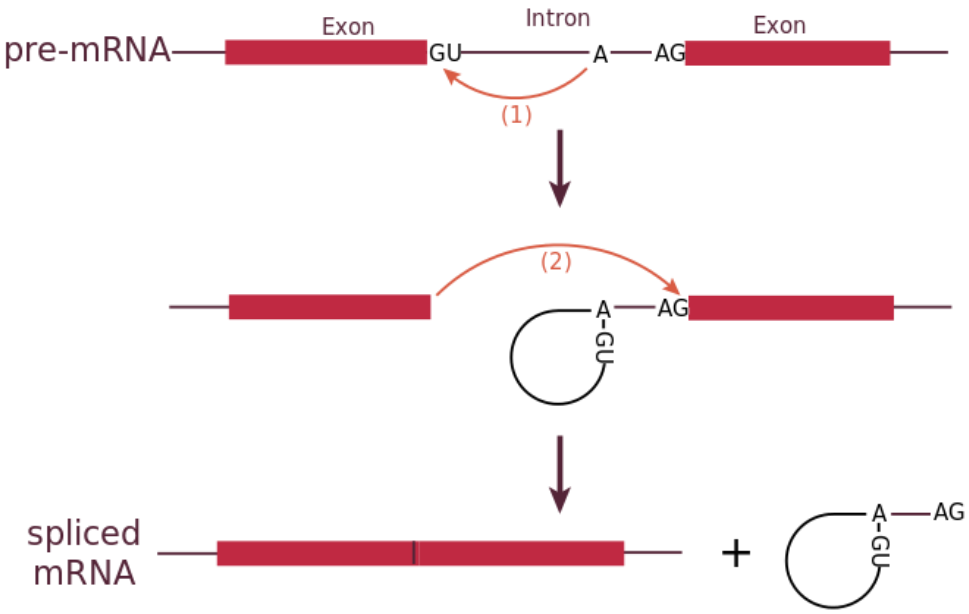

Figure 2.1: Process of cutting intron region from pre-mRNA

When talking about RNA sequencing bioinformatics tools, there are several characteristics they can be categorized by: the type of the reads they use and the type of the mapping method they use. By the type of the reads they use, most of the aligning tools are shot reads aligners. They use reads produced by second generation sequencing platforms, such as Ilumnia, which sequence reads short in length (50-700 bp), but have low error rate (0.5 – 5%). There are some mapping tools that use reads from the third generation sequencing platforms such as Pacific Biosystems RS. These reads can have average read length up to 10000 bp, but their average error rate can be over 15%. While short reads have a much lower error rate, their mapping to the genome is prone to error because of the repeating regions in the genome. Long reads have a smaller chance of being mapped to many repeated regions in the genome. Also, accurate transcriptome assembly, produced by short reads, is computationally more challenging. There have been some attempts of using both short and long aligners to make transcriptome assembly. In [6] an idea of narrowing down the regions in genome where reads can be mapped by long reads and then error correcting these regions by short reads is presented. The analysis of using long reads in RNA-seq is

debated in [5], where they state that long PacBio reads used for RNA-seq have lengths ranging from 500-6000 bp with average length 2500 bp.

RNA-seq bioinformatics tools can also be categorized by the aligning method they use in two main categories:

- short unspliced aligners – short aligners can align continuous reads, with no gaps or splices, to the reference genome. They are based on the Burrows-Wheeler method, such as Bowtie and BWA, or the Needleman-Wunsch or Smith-Waterman algorithms. While the first group tends to be much faster, second group aligns reads with more precision.
- spliced aligners – these aligners can align reads that are not continuous, which have gaps and splices. They can use special methods to do this or use unspliced aligners first and then a different strategy to align spliced reads. These types of aligners can be divided in two groups considering whether they are annotation-guided or not.
    - annotation guided spliced aligners – these aligners detect splice junctions on data available in databases about known junctions. Some of the representing tools are RUM and SpliceSeq. However, these tools cannot identify new splice junctions.
    - de novo spliced aligners – this type of aligners can detect new splice junctions and they are of great interest in this paper. Some of the mostly used tools of this type are: BBMap, GSNAP, TopHat and other [7] [8].

A research about de novo spliced aligners is given in [4]. In this paper BBMap, GSNAP and TopHat are compared to the new developed RNA-seq tool by their time and memory requirements and precision. All these tools are short read aligners, although BBMap can also use PacBio reads. In [4] BBMap showed best results in almost every test, and, while TopHat and GSNAP accuracy extremely dropped with longer reads and higher average error rate, BBMap continued to show acceptable results. A newly developed tool used in [4] used unspliced short aligners to map unspliced reads, while rest of the spliced reads were mapped using a different algorithm and heuristic knowledge about intron and exion regions.

It showed promising results, but its performance was still limited by short unspliced tools it was using. This is why the motivation for this paper was to explore the method for mapping RNA reads to the reference genome used by BBMap and to analyze its performance using long reads sequenced by the third generation sequencing platforms. In this paper, the BBMap aligning method was reimplemented and tested in C++ programming language. The discoveries made by this reimplementation were then used to explore possibilities of improving this method even further.

3. BBMap Alignment Method

BBMap is a tool that offers various options for RNA and DNA alignment. The sequence alignment method used by this tool is specific because it is splice-aware and is therefore suitable for RNA-seq data and transcriptome assembly. This tool uses short k-mers to align reads directly to the genome. It is highly tolerant of errors and indels, and faster and more sensitive than TopHat. It supports short reads, such as Illumnia, as well as long reads such as PacBio reads, paired or single-ended. It doesn't use any splice-site-finding heuristics, which is ideal for studying new organisms with no annotation. This chapter will present a brief introduction of the sequence aligning method that BBMap uses for aligning reads to the reference genome. While the method is suitable for RNA-seq data, it can also work with DNA data without any modification.

BBMap was implemented in the Java programming language with various options for RNA and DNA mapping. The sequence alignment method used in BBMap will be reimplemented in the C++ language, and will be called core method, later referenced as CM. The whole process of aligning reads to the reference genome will be called the alignment process.

3.1. Alignment Method Overview

The basic idea behind this method is the usage of short k-mers to map the reads to the reference genome. Before reads are processed, the reference genome is specifically indexed in the way that makes the locations of certain k-mers easy to found. Also, before processing, every read is cut to a certain number of small k-mers whose locations in the genome are found using this genome index. These locations are used to determine the best alignment of the read, using two special arrays: location array and gap array, which are explained later in this chapter.

3.2. Reference Genome Indexing

The first step in the alignment process is the creation of the index of the reference genome. The reference genome is indexed by small k-mers. BBMap

uses the default value 13 for the length of these k-mers, but this value can vary. Shorter k-mers show more sensitivity, while longer kmers show much faster execution time of the alignment process. The process of genome indexing can be pictured as the process of scrolling the sliding window of length KEYLEN along the genome for every base of the genome, as shown in Figure 3.1. For every k-mer, that is, for every array of KEYLEN bases that are in the current sliding window, a key k is generated. Key generation from KEYLEN bases is done by coding every nucleotide base with two bits. For example: base A can be coded with code 00, base C with code 01, base G with code 10 and base T with code 11. This way, every different sequence of bases of length KEYLEN can be expressed with 2*KEYLEN bits. That bit sequence can further be expressed with integer values from the interval [0-2^(2*KEYLEN)], which means that every different k-mer has a corresponding integer from that interval. Figure 3.1 shows the example of calculation of key values for four k-mers of length 13.

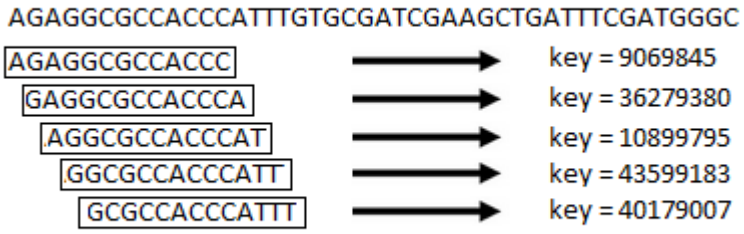

Figure 3.1: Process of key creation

3.2.1. Index Creation

The genome index consists of arrays: *sizes* and *sites,* whose creation will be explained in this subchapter. Index creation could be divided into three small steps.

The first step of index creation starts by counting the number of occurrences of all keys created from the genome. Keys are created as explained in subchapter above. For this purpose temporary array *temp* of size 2^(2*KEYLEN) is allocated and filled with zeroes. Every occurrence of key k is counted by adding 1 to the element at the position k in the array.

The second step of index creation builds the first array that is the index, array *sizes*. This array holds accumulated sums of elements of the temporary created array *temp* in previous phase. First element of *sizes* array holds the value 0, second element holds the value of the first element of *temp* array; third element of *sizes* is the sum of first and second element of *temp* and etc. For example: array *temp* with values {1,3,5,0,4} will produce array *sizes* with values {0,1,4,9,9}. In this example the genome has been indexed with 13 keys whose occurrences were counted in *temp* array.

Third step of index creation builds the second array of genome index. This array is named *sites* and its size is equal to the sum of the elements of *temp* array. For the array *temp* from the example above, array *sites* would have size of 13. An array *sites* has the locations of the keys in the reference genome (generated in the first phase of index creation). Since there are 13 keys in this example, the array *sites* will have 13 elements, where the value of each element is the location of the corresponding key in the reference genome. At which position in *sites* will the location of key *k* is put, is decided by next procedure:

For every key *k* whose starting location in the genome is *loc:*

- take the value *v* from *sizes* at position *k*
- put the value *loc* in the *sites* array at position *v*
- in array *sizes* increment the value at position *k* by 1

After this procedure is done, the array *sizes* has to be restored to the state it had before the procedure, by shifting the content by one to the right and setting the first element to zero. The array *sites* now holds the locations of every key at certain positions.

3.2.2. Index Usage

The purpose of the genome index is to enable quick lookup of all locations of a kmer *k* in the reference genome. With these two arrays, that can be done by fetching the value *v1* from the *sizes* array at position *k* and value *v2* at position *k+1*. Value *(v2 – v1)* is the number of occurrences of k-mer *k* in the reference genome. Value *v1* is the position in the *sites* array where the first location of k-mer

k in the reference genome is stored. All other locations of k-mer *k* in the reference genome can be fetched by traversing through sites from position *v1* to position *v2-1*.

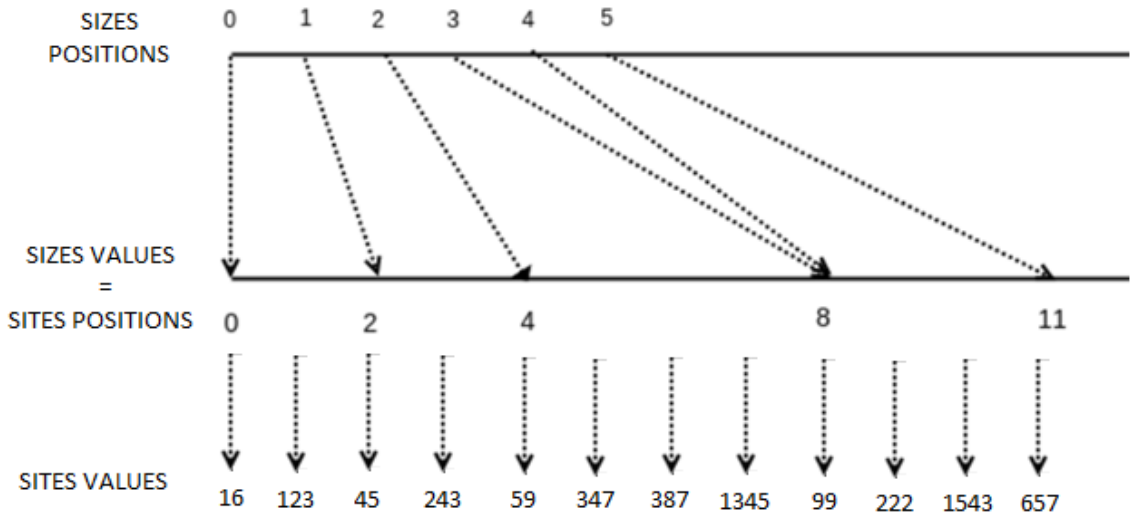

Figure 3.2: Usage of genome index

Figure 3.2 shows the idea behind the genome indexing. The positions in the *sizes* array are values of keys generated from the genome as explained in subchapter above. For every key value *k*, at position *k*, in the *sizes* array, the position in the *sites* array where the first occurrence of that key *k* is in the genome, is stored. In example in Figure 3.2, key 0 has two hits in the reference genome, whose locations are stored in the array *sites* at positions 0 and 1. The first occurrence of key 0 is at location 16 in the reference genome, and the second occurrence is at location 123. Key 1 also has two hits, while key 2 has four hits with locations stored in array *sites* from positions 4 to 7. Key 4 has no hits, which can also be seen in the Figure 3.2. It is clear now that by knowing the value of the key *k* generated from k-mer *K*, locations of k-mer *K* in the genome can be found by fetching the value *v* from *sizes* at position *k*, and value *v2* at position *k+1*, and by traversing through array *sites* from position *v* to position *v2-1.*

3.3. Reads Keys Generation

The reference genome index is something that the alignment method uses while mapping reads to the reference. Before the mapping process starts, every read that is being mapped also has to be preprocessed. Figure 3.3 shows one read with length 100 that is being aligned to the reference genome.

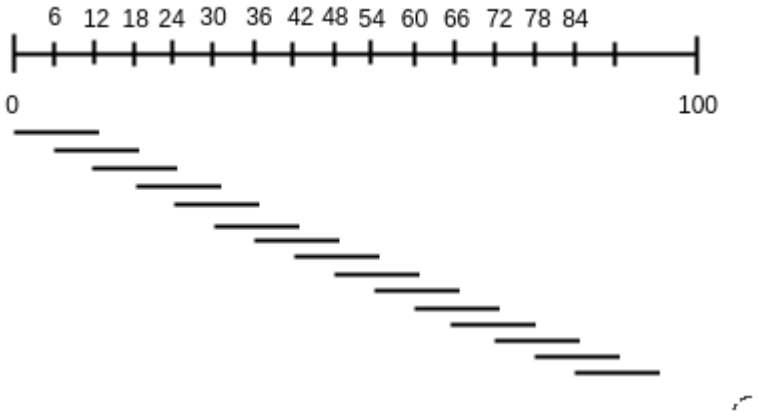

Figure 3.3: Read with kmers at certain positions

Similar as in genome indexing, keys will be generated from the read by coding corresponding k-mers to 2*KEYLEN long binary numbers, but not for every base of the read. The read in Figure 3.3 is being cut to k-mers of size 13 at every 6^{th} position. This generates 15 k-mers in total and also, 15 keys generated from those k-mers. In general, starting positions of k-mers used in generating keys from the read are called offsets. In Figure 3.3 the read has offset values { 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84 }. The number of keys generated from one read can vary, but the usual number of keys for read of size 100 is 15.

3.4. Important Terms Definitions

This subchapter presents the definitions of some important terms that are used in CM. CM uses genome index explained in the chapter 3.2 and read keys explained in the previous chapter. It is important to note that all steps of the CM are also done for the reversed complement of the read as well. For the reversed complement of the read, reversed offsets and keys are generated and the rest of

the method continues the same for both cases. Before explaining the algorithm itself, the notion of key hit, triplets, and one possible alignment will be presented.

3.4.1. Key Hits

Key hits are the occurrences of keys, generated from the read, in the reference genome. Once the keys are generated from the read, some of them will have hits in the genome, and some will not. For every key *k* of the read *r*, values *v1* and *v2* are looked up in array *sizes* at positions *k* and *k+1*. The number of hits of the key *k* is *v2-v1.* Position in *sites* array where the location of the first occurrence of key *k* in the genome is stored is *v1*. Values *v1* and *v2* are all that is needed to fetch all hits for key *k* in the genome. By doing this for every key *k* of the read *r,* the number of total key hits for one read can be counted.

3.4.2. Triplets

Triplets are structures which are a representation of key hits. They are used in CM and they have three values: *column*, *row* and *site*, which are sufficient to describe every key hit. For every key of the read that has hits in the genome, every key hit of that key can be represented by a corresponding triplet. Value of the *column* is the id of the key hit and it corresponds to the ordinal number of the key of that key hit. *Row* is the value *v1* of the key hit, as noted in the paragraph 3.4.1. *Row* is the position in *sites* array where the location of occurrence of the actual key hit of the corresponding key, in the genome, is stored. Third value, *site*, is the value at position *v1* in array *sites*, subtracted by offset value of the corresponding key.

Using the example shown in the Figure 3.3 it is clear that offset values of read r are { 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84 }. Key with offset value 24 will be denoted as key *k*. If that key has more than zero hits in the genome, array *sizes* will have value *v* at position *k* and array *sites* will have value *loc* at position *v*, which corresponds to the first hit of the key *k* in the genome. Triplet *t* that represents this key hit will have values: column = 3, as it is the ordinal number of the key *k*, row = v and site = loc -24, because the value of offset for key *k* is 24.

3.4.3. Possible Read Alignment

After describing how to generate keys from the read, how to find and count hits of those keys in the genome and how to represent those key hits, it is important to define what makes one possible alignment of the read or what makes the result aligning one read to the genome. If the currently aligning read is denoted as *r*, that read has multiple keys created from the read: { k_1 , k_2 , …, k_n }. Every key k_i, can have m_i hits in the genome. h^{th} hit of the key k_i is denoted as h_{ij} . The s et H_i can then be defined as set of h_{ij} $j \in \{1..m_i\}$, where m_i is the number of hits key k_i has. All elements of every Cartesian product of every combination of sets H_i create the set *S*. Set *S* contains elements of the Cartesian product of only one set H_i , as well as elements of Cartesian product of all sets H_i . For example, it contains elements of Cartesian products of sets: H_2 , $H_2 \, x \, H_3$, and $H_1 \, x \, H_2 \, x \, H_n$ and products of every other combination of sets H_i . Elements in *S* are actually all combinations of key hits of keys generated from read *r* but every element of set *S* does not contain two key hits of the same key. Every element of set *S* is one possible result of CM and from that set, a resulting alignment of read *r* can be created. But, not every element of set *S* is an equally good alignment. There is a need to define how the elements of set *S* will be scored, which properties of key hits in one possible alignment make a good alignment and how will that set of key hits produce read alignment. It is also necessary to define an algorithm for searching the best alignment, since the space search of all elements of set S would be computationally too complex. This algorithm is defined in the next subchapter, while two more arrays: *location* array and *gap* array are presented in two subchapters after that, because they are crucial in alignment scoring and in producing the resulting alignment.

3.5. Algorithm

The algorithm of the CM will be presented in this chapter. The algorithm starts by creating triplets for every first hit of every key that has at least one hit in the genome. This set of key hits is an element from set S that is used as the starting element of the algorithm for the currently processed read. Every created triplet has a column that says which key it was built from, *row* value which holds the position

in the *sites* array where the location of the hit in the genome is saved, and *site* value which is the location of the hit in the genome subtracted by the offset of the corresponding key. The lowest site value of currently built triplets is denoted *low_site*. The subset of triplets with site values higher or same as *low_site* and lower than *low_site*+MAX_INDEL is one possible alignment of the currently processed read. These triplets are scored by the scoring function and if they have a higher score than some CUTOFF value, a resulting alignment is built from these triplets. The MAX_INDEL value is usually set to 32000 and it represents the maximum length of the currently processed read's alignment. Because of that, key hits should span at the interval of MAX_INDEL. That is why only a subset of triplets from that interval is considered as a possible alignment. This part of the algorithm is called Triplet Scoring.

The next step in the algorithm is the transition from the current set of triplets to the next set of triplets, or in other words, from one element of the set *S* to the next element of set *S*. This is done by incrementing the *row* value of every triplet in the current set by 1, and fetching the value from array *sites* at position of the new *row* value of the triplet, subtracting it by the offset value of the corresponding key, and storing that value as a new *site* value of the triplet, for every triplet whose *site* value is equal to *low_site*. In this way, the *site* value of the changed triplets will have the value of next location of the hit in the genome of corresponding key. If some triplet's corresponding key does not have any hits left, it is removed from the set. This step of the algorithm is called Triplet Translation. After Triplet Translation, algorithm repeats the Triplet Scoring step again. The algorithm ends when all triplets are removed from the set, which means that all key hits were processed.

This algorithm can find more possible solutions, but the best scoring solution is considered as the resulting alignment. Using these steps, the alignment method processes subsets of key hits looking for the best alignment. The way the subset of key hits is scored and the way an alignment solution is built from that set of key hits is explained in the next two chapters, where a brief summary of *location* and *gap* array usage is presented.

3.6. Alignment Scoring

This chapter presents the scoring function of one possible alignment which is presented by a set of key hits. Theoretically, every set of keys can make a possible alignment, but sets with few key hits, or sets with key hits that are not close to each other in the genome, should be valued with low scores. Sets of key hits with many key hits close to each other should be valued with high scores. The process of scoring the set of key hits is done in two phases: creation of the *location* array and scoring of the created location array. Before these steps are presented, different characteristic types of sets of key hits will be discussed.

3.6.1. Possible Alignment Types

3.6.1.1. Perfect unspliced alignment

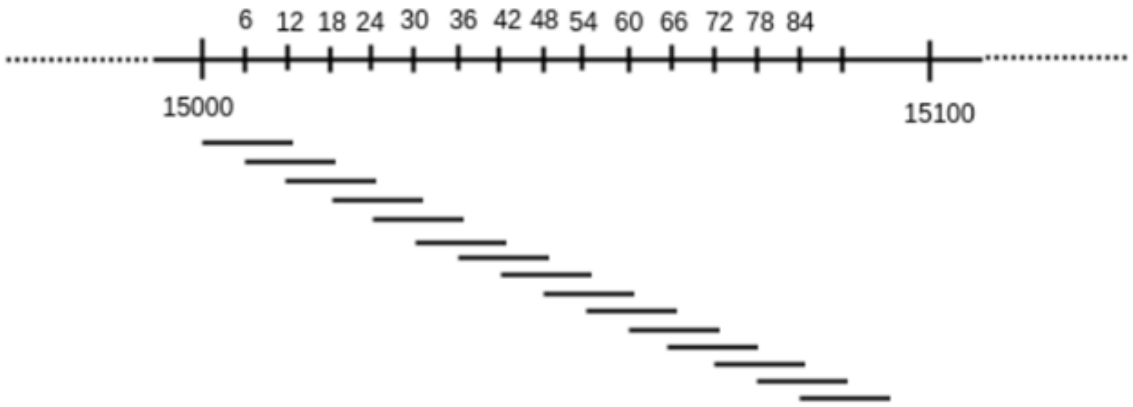

Figure 3.4: Example of perfect alignment

Figure 3.4 shows the set of key hits for the read shown in Figure 3.3. This read is not spliced, and in this figure, all 15 keys have hit the genome at the location they were actually sequenced from. This alignment is the example of a perfect read alignment. All triplets that were created form these key hits have the same *site* value which is in this case the value of the start of the read in the reference genome.

3.6.1.2. Unspliced alignment with errors

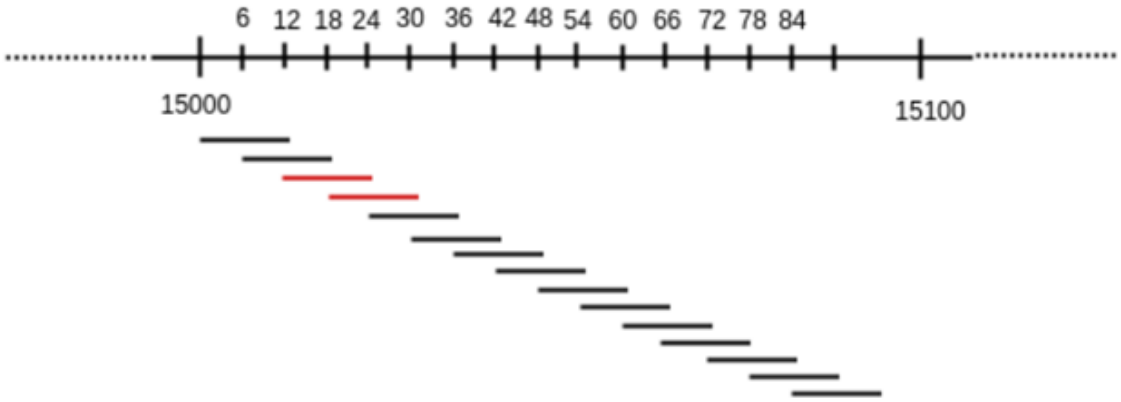

Figure 3.5: Example of alignment with single error

An example of alignment when substitution, deletion or insertion occurs in the read is shown in Figure 3.5. This time the processed read has an error somewhere between the 18th and 24th nucleotide as shown in Figure 3.5, with two key hits marked with red color. These key hits would exist if there was no error in the read, but since the read has an error these two keys will not hit the genome at those locations. Two red keys have hits with site value not equal 15000, their site values are somewhere on the right in the genome, greater than 15100. In case of substitution all black key hits will have *site* value equal 15000. In case of deletion first two black key hits will have site value equal 15000 while other black key hits will have value 14999. In case of insertion first two key hits will have site value 15000 and other will have 15001. This is the simplest example of alignment with one error. More errors make an alignment more complicated, but the principle of key hit locations is the same.

3.6.1.3. Spliced alignment with no errors

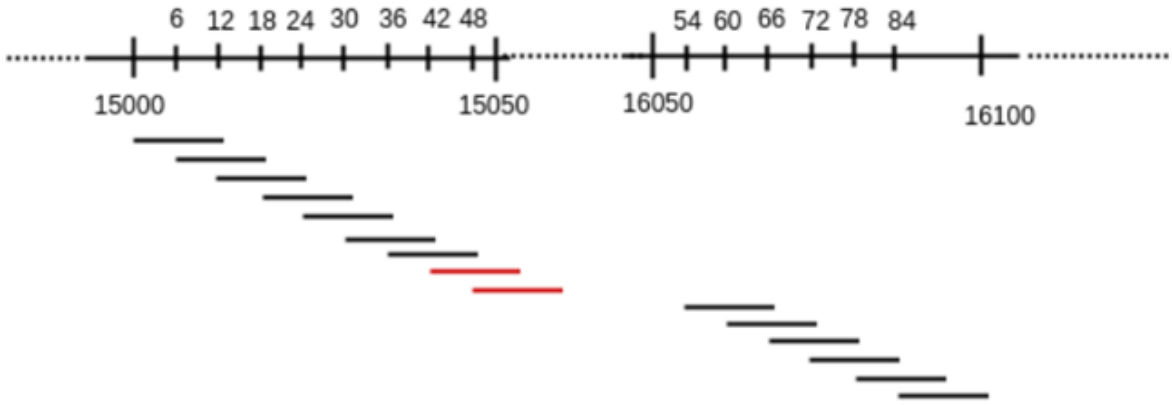

Figure 3.6: Example of spliced alignment

Figure 3.6 shows the third type of read alignment with spliced read. In this case, the first part of the read was transcribed from the reference genome from location 15000 to location 15050, and the second part of the read was transcribed from location 16050 to location 16100. This read has spanned over two exon regions. Triplets built from first seven key hits have site value 15000. Two key hits marked with red, did not hit genome at locations they are drown in the Figure 3.6. The rests of the black hits have their corresponding triplet sites values 16000. This is a common type of alignment with one big gap between several close key hits. This type of alignment is not scored with low score value as it will be explained in next subchapter, but this alignment can also have substitutions, deletions and insertions which will further lower its score value. The building and scoring of *location* array which will be explained in next subchapter is the reason why this method can detect gaps and align spliced reads as well as unspliced reads.

3.6.1.4. Spliced alignment with false gap

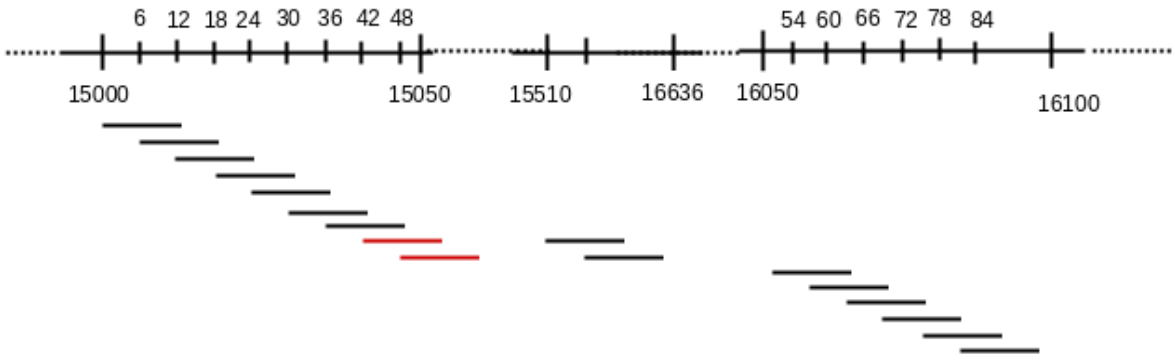

Figure 3.7: Example of alignment with false gap

Figure 3.7 shows another example of alignment which is similar to the example above. The read in the example is spliced and has one gap, but in this example, two keys marked red have hit the genome further on the right at positions 15510 and 15523, because they have some errors. This alignment will be interpreted as alignment with two gaps and three exon regions, which is false. This is an example of wrong alignment discovery because of the errors in the read. The probability of this happening is higher the shorter the keys are and this is why shorter keys will not always be more precise than longer keys and they tend to find more exon regions than there really are.

3.6.1.5. Spliced alignment with missing exon

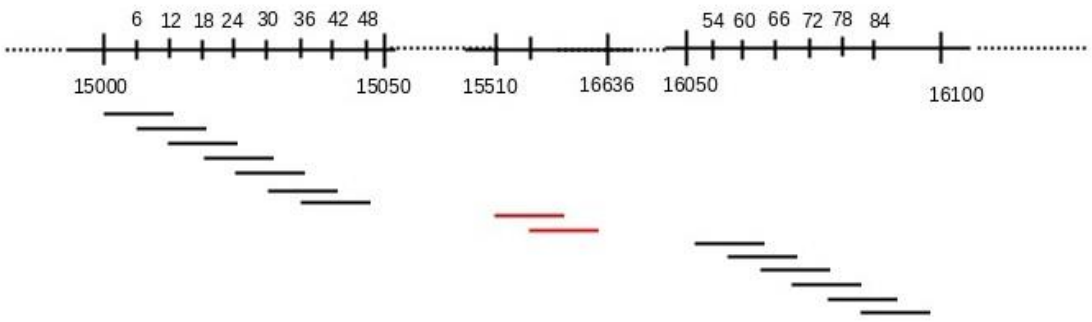

Figure 3.8: Example of alignment with missing exon

The last example of common alignment that will be explained in this work is shown in Figure 3.8. This example shows an alignment where opposite situation occurred than in the example above, an exon region was skipped. The read in this example spans over three exon regions, but two red keys have not hit the genome at those positions because of errors. This is why this exon region will not be recognized and will be skipped. This will happen more often with longer keys because there is a higher probability that more keys will be affected with one error. This shows that smaller and longer keys both have their weaknesses and advantages and that the density of keys also has a high effect on how accurate alignment will be.

3.6.2. Location Array

Location array is the main structure used in the whole alignment process. It is built out of the set of key hits and it is used in scoring the set of key hits. It is also used in building the resulting alignment. *Location array* has the size equal to the size of the read, which will be denoted as READ_LENGTH, and it is first filled with value -1 at all positions.

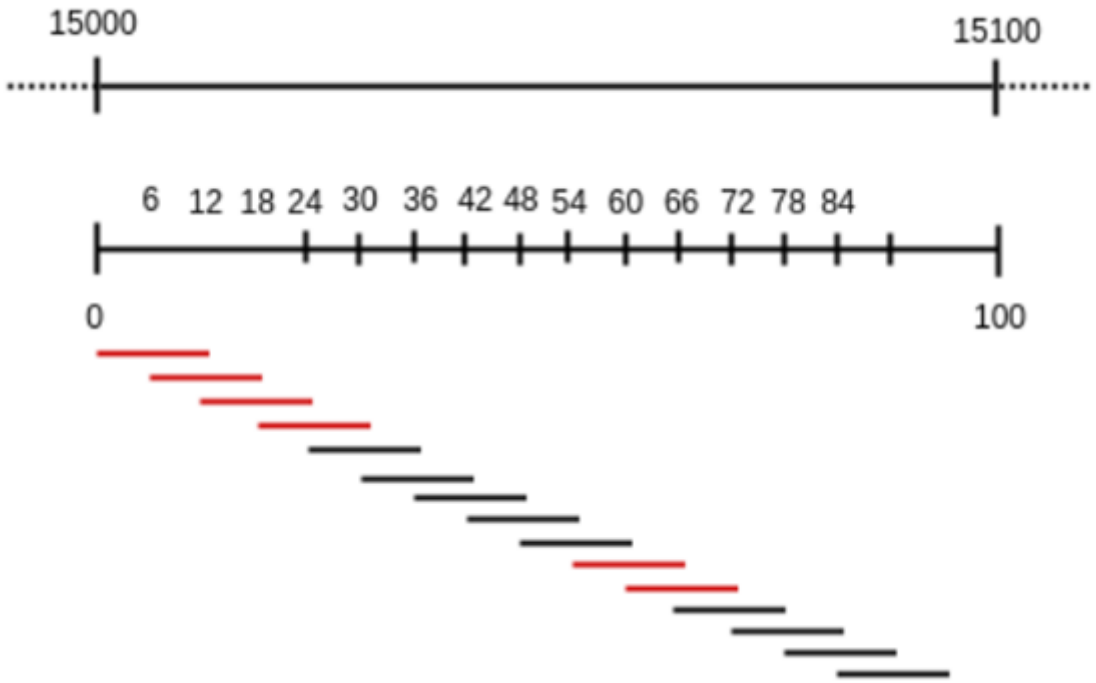

Figure 3.9: Read with 9 hits being aligned to the genome

Figure 3.9 shows an example of possible alignment where first 4 keys did not make a hit because of substitution or deletion or insertion. Keys with offset value 54 and 60 hit the reference genome, but at some location in the genome more further on the right, because a substitution occurred on nucleotide 63. This example will be used to show how the location array is built from this set of key hits and how the score of alignment is calculated. All key hits in black color have *site* value equal 15000 and this is also the *low_site* value for this set of key hits. Keys that are marked red might have no hits in the reference genome at all, or they might have hit the genome at some location that is further on the right from locations 15000 and 15100. For the purpose of this example it will be assumed that the first four keys have no hits. The filling process starts with the first key hit, which in this example is the key hit with offset 24 (this doesn't have to be the key hit with *low_site* value). For this key hit, all bases from the interval [24 + KEYLEN - 1, 0] in the read, and bases from the interval [15000 + 24 + KEYLEN – 1, 15000] in the genome are compared (since 15000 is the site value of triplet created from key hit with offset 24). In this case all bases in the read are equal to the bases in the reference genome at all corresponding locations from those intervals. For every matching base at one position in the read, a *site* value of current key hit's triplet is put in the same position in *location array*. In this case *site* value of key hit with offset 24 is 15000 and that value is put in the location array at positions [24 + KEYLEN -1, 0]. After processing the key hit with offset 24, *location array* has a value of 15000 at locations [0,24+KEYLEN-1] and values -1 at locations [24+KEYLEN, READ_LENGTH]. For the first processed key hit, comparison of bases is always done all the way to the beginning of the read regardless of the possible bases mismatch. In case of mismatch, value -1 is left in the location array.

This process is repeated for the next key hit with offset 30; comparison of bases in read from the interval [30 + KEYLEN -1, 0] and bases in the genome from the interval [15000 + 30 + KEYLEN -1, 15000]. *Site* value of triplet of this key hit is also 15000 and that value will be filled in the *location* array where matches of bases occurred. Filling of location array will start at position (30 + KEYLEN – 1) and continue descending until position (24 + KEYLEN - 1) is reached. Array the

filling stops at that location because there is already value that is not -1. After second key hit is processed, the *location* array is filled with values 15000 at positions [0, 30 + KEYLEN – 1] and with values -1 at positions [30 + KEYLEN, READ_LENGTH]. Using this explanation, the first general rule of location array filling can be presented:

- For every key *k* with offset *o*, that has hit *h* in the genome, whose triplet *t* has site value *s*, the substring of read *r* from interval [o + KEYLEN – 1, 0] and the substring of genome from interval [s + o + KEYLEN – 1] are compared base by base. For every match at location *loc*, location array is filled with value *s* at position *loc* if the current value at location *loc* is -1. For mismatch, the first processed key the value value -1 in the location array, and continues the process to the beginning of read. Every other key stops the process at first mismatch or if the location array is already filled with the same site value.

Continuing the analysis of this example, a situation where an error is reached occurs. Red keys with offsets 54 and 60 have key hits whose triplets have site values 17000. For these key hits the first rule of location array filling is done, where the *location* array is filled with value 17000 at all positions where there is no error. The *location* array is filled with value 17000 from position (54 + KEYLEN - 1) to position (48 + KEYLEN - 1) and from position (60 + KEYLEN - 1) to position (54 + KEYLEN - 1). The key hit with offset 66 will fill the *location* array with value 15000 at positions [66 + KEYLEN -1, 64]. The process will stop at position 63 because that is where substitutions occurred. All remaining key hits will fill the rest of *location* array with 15000.

This phase of the location array filling is called reverse location array filling. The location array is fully filled after the second phase of location array filling is done and it is called forward location array filling. Forward location array filling is similar to reverse filling, but in forward filling read and genome bases comparison is done in ascending order. Second general rule of location array filling is:

- The location array is filled forward and reverse. In reverse location array filling, for every key hit *h* with offset *o* and site *s*, comparison between

read at positions [o + KEYLEN – 1, 0] and genome at positions [s + o + KEYLEN -1, s] is done. Both forward and reverse location array filling have the same first general rule.

In this example, a specific situation occurs when the key hit with offset 48 is processed in forward location array filling. When applying the first rule of location array filling, this key hit should fill the location array with value 15000, which is *low_site* value, from position (48 + KEYLEN - 1). This time the location array will not have value -1 at those position, but value 17000. Since this key hit has site value equal to *low_site*, it will overwrite values 17000 in location array and this makes the third general rule of location array filling:

- If the triplet of current key hit has site value that is equal to the *low_site* value, when filling the location array by applying the first general rule of location array filling on this key hit, all other values in location array that are not equal to *low_site*, are overwritten by *low_site*.

After finishing reverse and forward location filling, the location array will be filled with value 15000 at positions [0, 62] and [64, 100], and the value at position 63 will be value -1.

When applying this process for alignment with a big gap, as shown in Figure 3.6 the location array will be filled with values 15000 at locations [0, 49] and with value 16000 at locations [50, 100]. It can be seen from these examples that for alignments with key hits that are close to each other and that have a small amount of errors, the location array will be filled with similar site values. Alignments that have key hits with lots of errors and which are dispersed will have a location array with many different site values and with values -1. Alignments with a small number of big gaps between key hits will have few sequences of same site values in the location array. The structure of the location array suggests the structure of the alignment. The score function could be developed so that it values location arrays with uniform values at some intervals with high score. This scoring function will be presented in next chapter.

3.6.3. Scoring function

Filled location array is the most important structure of the CM and it is used to score an alignment and later to make an alignment representation. Calculating score of the location array is done by traversing through the array and processing every element of location array. At the beginning the score is set to zero and every processed element of location array adds or subtracts from the current score value. For every element:

- If the element is less than zero a substitution has occurred and some SUB penalty is reduced from the score.
- If the element is greater or equal to zero and if its value is equal to the previously processed element, this means that match has occurred and some positive MATCH value is added to the score.
- If the element is greater or equal to zero and if its value is greater than previously processed element, that means that an insertion has occurred and some INS penalty is reduced from the score. INS penalty is modeled so that its value is higher the greater the difference between the previous and current element is, because big insertions should not happen.
- If the element is greater or equal to zero and if the value is lower than the previously processed element, this means that deletion occurred and some DEL penalty is reduced from the score. Contrary to the insertion case, big deletions should be common in spliced reads and should only be slightly higher than singular deletions. This is why penalty should be adjusted in the way that big gaps are acceptable as single deletions are.

Modeling scores of penalties of insertions and deletions is a topic that could be further discussed but that would exceed the limits of this thesis. That is why these values will be taken from the current BBMap implementation.

3.7. Alignment presentation

So far one possible alignment was presented by set of key hits and with a filled location array. A function was constructed for scoring a filled location array and the best scored location array was chosen as the resulting alignment of the processed

read. Now this final alignment presented by location array will be presented by the final form which is consists of:

- starting position of alignment
- stopping position of alignment
- score of alignment
- gap array
- match string

The starting position of an alignment is equal to the *low_site* value. Stopping position of an alignment is equal to the maximum value in the location array increased by the size of the read. The score of the alignment is the score of the3 location array presented in the chapter above. Gap array and match string are two more elements of alignment presentation which will be explained in the next chapter.

3.7.1. Gap Array

The gap array is a structure that is the part of alignment representation, which has more than zero elements if the alignment contains splices. It is created from the filled location array. It is considered that an alignment has splices, or big gaps, if the difference between two nonnegative adjacent elements in the location array is greater than the MINGAP value. This value is usually set to 256 because that is the minimal size of intron regions. Before the gap array is built, the location array is transformed in the way that every element not equal to -1 is increased by the value of its position in the array. For example, if the location array has values [15000, 15000, 15000, -1, 15000, 16000, 16000, 16000], then after the transformation, it will have values [15000, 15001, 15002, -1, 15004, 16005, 16006, 16007]. Every element that has value -1 is replaced by value of the previous element increased by 1. For the previous example, the location array will now have values [15000, 15001, 15002, 15003, 15004, 16005, 16006, 16007].

In this example, only one gap will be detected because elements at position 4 and 5 have difference 1001, which is greater than MINGAP. If the number of gaps is g, then the gap array has 2*g+2 elements. The first element of the gap array is

the value of the first element in the location array, which is 15000 for this example. The last element of the gap array is the value of the last element in the location array, which is 16007 for this example. Other elements are values at positions in the location array, where big gaps appeared. In this example there is only one big gap at positions 4 and 5 with values 15004 and 16005. Finally, the gap array for this example is [15000, 15004, 16005, 16007]. The gap array is the fourth part of the resulting representation of the alignment. All exon regions and big gaps that were recognized by the alignment method are stored in the gap array and can be used in analyzing the solution and in creation of string representation of the alignment.

3.7.2. String Representation

For the purpose of making the string representation of the resulting alignment one local alignment method was implemented in this work. The idea of string representing the alignment was used in BBMap where the Needleman-Wunsch algorithm with affine gaps was used to make the match string from the resulting alignment. This method is suitable for spliced reads and it is also an example of a local alignment method that uses results from CM and then produces an alignment that is more precise. The idea of using local alignment for further improvement of results is presented in a later chapter. The analysis of this method was not done in this work because it would exceed the scope of this research.

4. Alignment Method Improvements

In the previous chapter a CM that BBMap uses was presented. This method was reimplemented in this work. The inputs of the method are: reference genome and a set of reads that are mapped to the reference genome. Output of the method is a set of resulting alignments of the reads that the aligner was able to map. Starting and stopping position of alignment, score of alignment and gap array make a resulting presentation of an alignment. In this chapter various options of improvement of CM are considered. Some of these improvements are implemented and analyzed in the next chapter, and some of them are left for future analysis.

4.1. Coverage Calculation

The first novelty in the alignment process is the introduction of coverage computation. In RNA sequencing, coverage refers to the number of times a nucleotide is read during the sequencing process. Coverage is the average number of reads representing a given nucleotide in the reconstructed sequence [4]. Data from the alignments of the mapped reads can be used to calculate the coverage of bases in the reference genome. For every resulting alignment a_i $i \in \{1..M\}$, a gap array has N_i exon regions $e_{ij}, j \in \{1, N_i\}$, where every exon region e_{ij} has K_{ij} number of bases b_{ijk} where every base b_{ijk} has position p_k in the genome. If the whole genome is denoted as G and one base of genome at position p is g_p, then total coverage C of the genome is an array whose length is the same as the length of the genome and whose content can be computed by the following expression:

$$\forall a_i \in A, \forall e_{ij} \in E_i, \forall b_{ijk} \in e_{ij}\ c_k = c_k + 1\ ,\ c_k \in C$$

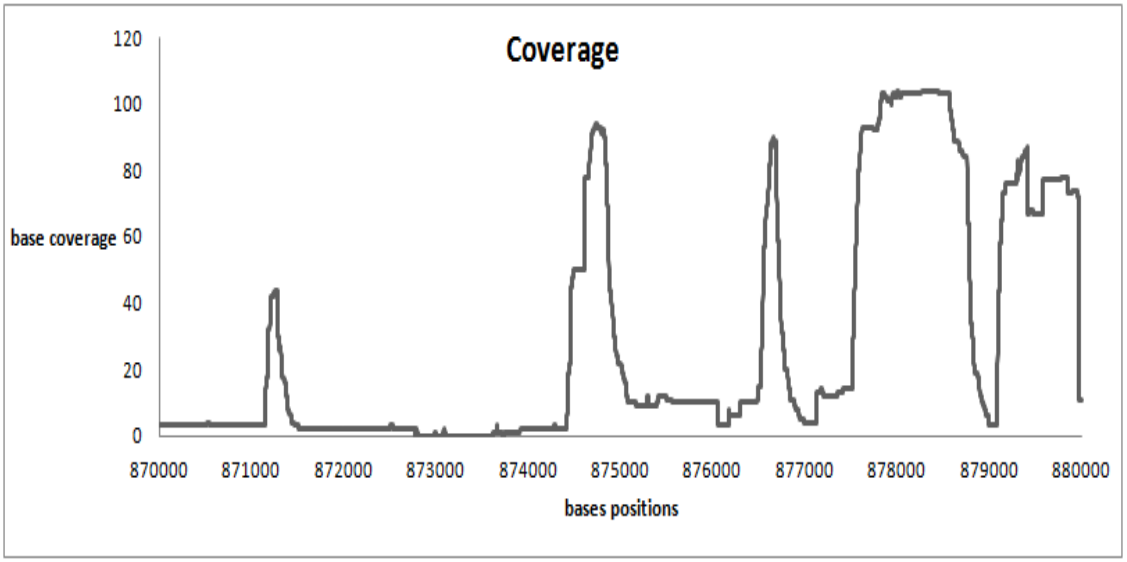

Figure 4.1: Coverage information for small part of genome

Figure 4.1 shows the computed coverage for bases of the genome from position 870000 to position 880000. It can be seen that some regions of this part of the genome have high coverage up to value 100. Regions around bases 875000, 876500 and regions from position 877500 to 878800 and from position 879100 to 880000 are candidates for exon regions. Some of the regions have coverage value around 20, which suggests possible errors in certain number of reads. This coverage information could be used to reduce the search space from the whole genome to some subpart of the genome. The main idea behind this improvement technique is to use the coverage information to make a new reference genome which would include only high covered regions of the whole reference genome and then use other improvement methods like the two phase method or the read realignment method which will be presented in next two chapters.

Coverage computation offers many possibilities and this alone only could be a topic of a whole new research, but in this work, only a small research of coverage information will be given and used to analyze other method improvement possibilities. In this work, new reference production from coverage information was done by cutting out the reference genome regions with low coverage. The problem in creation of new reference is to determine what the regions with low coverage are and what the regions with high coverage are. Three parameters were introduced to achieve that: COV_THRESH, COV_PADDING and COV_GAPLEN.

COV_THRESH is the minimal value of coverage one base has to have to be considered a base with high coverage. COV_GAPLEN is the minimal gap size between two bases with high coverage such that the region between them is considered a region with low coverage information. This region is then cut from the genome but the endings of the region with size COV_PADDING are included in the new reference genome because of the possible errors of coverage computation. The values of these parameters determine how of the genome will be included; higher COV_THRESH value cuts more of the genome including only exon regions in the new reference genome. Low COV_THRESH value includes more areas around the exon regions in the new reference genome. Low COV_GAPLEN value excludes small islands of low coverage information, while high COV_GAPLEN value only excludes regions that have low coverage information over greater area of genome, probably only excluding regions with no gene activity and including intron regions.

4.2. Two Phase Method

The two phase method is the idea of improving CM by executing the CM two times where every execution of CM is one phase of a method which is called the Two Phase Method (TPM). This is the simplest idea of improvement of CM where reads unaligned in the first phase of TMP are aligned in the second phase of the TPM. Second phase of the TMP is same as the CM, but executed under different conditions and with new additional information gained from the first phase of TPM. New conditions and information in the second phase should result in alignment of the reads that were not aligned in the first phase of TMP. In this work an implementation of TMP was done where the second phase of TMP is executed using new reference genome built from the coverage information presented in the chapter above mapping the reads that were not mapped in the first phase of TMP. A detailed analysis of this improvement method is given in chapter 5.

4.3. Read Realignment

Another possible improvement of CM is read realignment. This could be considered as the further improvement of the TPM or as another variant of second

phase of TMP. Read realignment is an improvement idea where some of the reads that are mapped in the first phase of TMP are realigned in the second phase of TMP. This is done because some of the reads that had low score alignment with low precision and quality could produce better alignment with higher score after being aligned with additional information and new conditions. This improvement could be achieved with realigning reads with low scored alignment to the new reference genome produced from coverage information. If the realignment of the read produced an alignment with better score, that read's resulting alignment from the CM or TFM is replaced with the resulting alignment from the realignment method. This improvement method raises questions about the THRESH_SCORE parameter, which is the score that determines low and high quality alignments. Reads with alignments scored below the THRESH_SCORE value are considered low quality reads and are included in the realignment process. The coverage computation that produces a new reference and conditions under which realignment is executed both have effect on the realignment method, but it will be interesting to analyze how the THRESH_SCORE parameter improves results of the whole alignment process. An analysis of a percentage of reads that have better score, a percentage with worse score and the ones that cannot be realigned in the realignment method will be given in chapter 5. Also, the realignment method can be used as a substitution for the second phase in TPM, but it can also be executed alongside TPM to produce possibly even better results. This possibility will also be analyzed in chapter 5.

4.4. Local Alignment Improvement

Local alignment is conceptually different than other improving methods presented above. Methods above can use other heuristic information to improve their performance such as donor-acceptor statistic as presented in [4], but their main concept is to use additional information and different conditions to execute CM on some new subset of reads. Local alignment is an improvement idea that uses the results of core BBMap method or of some of the improvement methods presented above and then executes another method to improve resulting alignments. It does not use CM and does not produce new alignments, but it

improves already produced alignments. The idea of local alignment improvement is based on the assumption that the preceding method has found the starting and stopping points of an alignment with great accuracy. If that is the case, then some local alignment could use that region of the genome to align the read more precisely to the genome and especially to detect the gaps and exon regions of the alignment more accurately. One example of local alignment that could be used to improve the resulting alignments is Needleman-Wunsch with affine gaps. This algorithm is suitable since it can detect big gaps in the alignment. An implementation of this algorithm was done in this work for the purposes of finding the matching string of the resulting alignment. An analysis of variations of this algorithm, as well as analysis wheter some other local algorithm would produce better results are a topic for further research.

5. Evaluation and Results

In this chapter an analysis of performances of the BBMap aligning method and an improvement methods is given. First, some metrics of quality of alignment methods are defined, after which parameters which will be evaluated are presented. After that, detailed evaluation of performance and accuracy of both phases of TFM are given followed by an evaluation of the read realignment method. Reimplemented core BBMap method is compared with the original BBMap tool and the best developed improvement method is also compared with BBMap tool. The developed tool can map DNA and RNA data, both stranded, not depending on the read length. Short reads can be mapped with high precision, but the tool was tuned so it can map longer reads with much higher error rate.

5.1. Evaluation Metrics

Before diving further into evaluation of different mapping methods, three evaluation metrics will be presented: hit rate, start-stop accuracy and coverage accuracy. These are different metrics for measuring precision and accuracy of the alignment method. For an aligning method, which maps R reads to the reference genome, the following metrics are defined:

- hit rate – the number of reads for which the alignments were found, divided by the total number of reads
- start-stop accuracy – this metric measures how precise a method can map starts and stops of the read to the right location. A start/stop is considered to be mapped precisely if the absolute difference between mapped start/stop and correct start/stop location is less than 15. There are four parts of the metric:
    - start accuracy – number of reads with precisely mapped starts, divided by the number of aligned reads.
    - stop accuracy – number of reads with precisely mapped stops, divided by the number of aligned reads
    - start-or-stop accuracy – number of reads with precisely mapped either starts or stops, divided by total number of aligned reads

    - start-and-stop accuracy – number of reads with precisely mapped both starts and stops, divided by total number of aligned reads.
- coverage accuracy – metric that counts the percentage of bases in the resulting alignment whose locations in the genome are correct. The resulting alignment's gap array holds the found exon regions. The total number of bases in these exon regions is the number of found bases. By comparing these regions with the correct exon regions, the total number of overlapping bases can be found. By knowing the total number of all bases being mapped and the total number of basis that were aligned, two parts of this metric can be expressed:
    - coverage amount – the number of overlapping bases divided by the total number of bases of all aligned reads
    - coverage precision – the number of overlapping bases divided by the total number of found bases of all aligned reads.

These three metrics show different properties of some alignment method. Some methods will have a big number of hits and high hit rate, but low coverage and start-stop accuracy. Other methods will have high coverage or start-stop accuracy, but will have too few hits and thus low hit rate. Great alignment methods will have high hit rate, but with high start-stop and coverage accuracy. Start-stop accuracy is important to evaluate because it shows how good is the method in finding starts and stops of the read in the genome. Methods with high start-stop accuracy could have potential for further improvement with local aligners. Coverage accuracy shows how good the method is in producing highly precise coverage information. Methods with high coverage accuracy will produce quality coverage information used to build new reference genomes for TFM and the realignment method.

5.2. Evaluation Parameters

Alignment methods evaluation will not only measure time and memory performance and accuracy of the methods, but will also give analysis of certain parameters whose value can highly affect the results of the method. Most of the parameters for the CM were taken from the BBMap implementation, but since BBMap has two modes of work, one for short reads and one for PacBio reads,

there are some parameters that had to be further analyzed. List of the parameters that will be further analyzed and tested is given in Table 5.1.

Name	Description	Usage	Domain
KEY_DENSITY	Set of constants that determine the number of keys created from the read	Core method, TFM, realignment method	{low density, high density}
SCORE_CUTOFF	Alignments with score below this value are discarded	Core method, TFM, realignment method	{low cutoff, high cutoff}
KEYLEN	Length of the key	Core method, TFM, realignment method	{9,10,11,12,13}
READLEN	Desired size of the read. Every read is cut in pieces with maximum size of READLEN	Core method, TFM, realignment method	{500, 2000}
THRESH_SCORE	Reads with alignments scored below THRESH_SCORE value are considered low quality reads and are included in realignment process	Realignment method	[0..1]

Table 5.1: List of analyzed parameters

KEY_DENSITY and SCORE_CUTOFF are sets of parameters that determine how many keys will be generated for one read and how high the score of an alignment has to have to even be considered as a result. These parameters are taken from the BBMap alignment method. Low density and high score cutoff were used in alignment of short reads and high density and low cutoff were used in alignment of long reads with more error rate. Tools implemented in this work will be evaluated with all combinations of values of these parameters to check for which parameters it works best. High density should give higher hit rate, but would also have higher execution time. High score cutoff would produce less alignments, thus lower hit rate, but the produced alignments would be more precise. Which combination of these parameters is the best will be evaluated in the following chapters.

KEYLEN is the key parameter for every method because it highly affects the precision and performance of the method. Methods using longer keys have lower execution time and have fewer exon regions found. Methods with shorter keys have longer execution time, but find more exon regions and tend to be more sensitive.

READLEN is another key parameter because CM shows different results for different lengths of the reads. Longer reads have less probability of not being mapped, while shorter reads have a more precise resulting alignment. In this work two read lengths will be analyzed: reads with length 500, which is the maximum value of the read length BBMap uses, and reads with length 2000, which is the average read length of reads produced by PacBio RNA-seq sequencers.

5.3. Simulated Datasets

All the tests used for evaluation of alignment methods were executed with the simulated reads that were generated by the BEERS simulator [9]. The BEERS simulator or Benchmarker for Evaluating the Effectiveness of RNA-Seq Software, uses different sets of annotations such as: AceView, Ensebl, Geneid, Genscan, NSCAN, RefSeq, SGP, Transcriptome, UCSC and Vega, to avoid biased dataset towards any particular annotation set. The simulator starts a with large number of gene models taken from different annotation sets and then chooses a fixed number of genes at random and introduces substitutions, indels, alternate splice forms, sequencing errors and intron signals. It can simulate mouse and human data, but in this work human data was used, more specifically part of the human chromosome 1. For the purposes of comparison of reimplemented CM with BBMap CM, short reads of length 250 and 500 were generated with error rate 5% and 10%, because BBMap was previously tested with short reads in [4] and the goal was to reconstruct the tool with similar performances. CM and improvement methods were also tested with reads of length 2000 and error rate 18% to simulate PacBio sequencers.

Simulated reads are saved in .cig format as well as .fa format. The developed tool can read .fa formats, but it can also read .cig format where true alignments are stored with starting and stopping positions of the true alignment, as well as the

information about gaps that happened in every read. Using this information, the output of the developed tool, with various method implementations, can be tested to calculate its coverage accuracy and start-stop accuracy. This is important for detailed research of method improvements introduced above.

5.4. First Phase Evaluation

In this chapter, reimplemented CM will be evaluated. This evaluation will be done in two parts. In the first part of the evaluation, the method itself will be analyzed to see which parameters give the best results by measuring hit rate, start-stop and coverage accuracy. This evaluation will be done with long reads of length 2000. In the second part, the reimplemented method will be compared to the original BBMap tool by its time execution and hit rate to see basic similarities and differences between the two tools. This comparison will be done with short reads of length 250 and 500.

5.4.1. Evaluation Results

The evaluation of reimplemented CM was done for parameters: KEY_DENSITY (KD), SCORE_CUTOFF (SC), KEYLEN (KL) and READLEN (RL). For every combination of parameters the program was executed and all evaluation metrics were measured: hit rate (HR), start accuracy (SA), stop accuracy (SO), start-or-stop accuracy (SoS), start-and-stop accuracy (SaS), coverage amount (CA) and coverage precision (CP), as well as the time and memory efficiency. All tests were executed using the simulated set of 23447 reads with length 2000 and error rate of 18%. Size of the reference genome was 83083510 bases. The KEYLEN parameter was tested for values 13, 12 and 11 because methods that were run with shorter key values showed extremely high execution time and were not considered in the evaluation of the first phase. Detailed results of the evaluation tests are given in Table 5.2.

KD	SC	KL(bp)	RL(bp)	HR(%)	SA(%)	SO(%)	SaS(%)	SoS(%)	CA(%)	CP(%)
high	low	13	500	95,52	79,27	79,56	0	79,41	83,38	79,33
high	low	13	2000	93,11	73,46	69,94	51,31	92,09	79,73	74,63
high	low	12	500	94,29	80,92	79,82	0	80,98	85,16	80,19
high	low	12	2000	91,29	74,95	70,84	53,33	92,44	82,11	75,65
high	low	11	500	92,1	81,92	78,16	0	80	86,72	80,23
high	low	11	2000	87,2	75,52	71,18	54,02	92,68	83,9	74,38
low	high	13	500	41,15	92,72	89,71	0	91,22	89,13	86,52
low	high	13	2000	59,73	80,06	72,23	55,97	96,32	83,69	79,96
low	high	12	500	40,47	94,27	89,56	0	91,91	90,36	87,34
low	high	12	2000	56,17	81,94	74,23	59,55	96,61	85,83	81,22
low	high	11	500	44,88	91,58	86,68	0	89,14	89,91	85,69
low	high	11	2000	60,45	80,08	73,31	57,87	95,52	86,21	79,18
high	high	13	500	46,64	91,21	88,88	0	90,05	89,45	85,65
high	high	13	2000	69,38	80,49	74,32	58,78	96,03	84,63	79,28
high	high	12	500	43,69	93,79	89,33	0	91,56	90,68	86,33
high	high	12	2000	64,45	81,79	75,39	60,94	96,23	86,85	80,23
high	high	11	500	47,56	91,64	84,25	0	87,95	90,63	84,75
high	high	11	2000	66,87	80,72	74,86	59,89	95,69	87,68	77,66
low	low	13	500	93,12	76,7	77,89	0	77,29	81,13	78,19
low	low	13	2000	91,54	70,52	67,47	46,5	91,48	76,47	72,61
low	low	12	500	93,92	78,31	78,05	0	78,18	82,54	78,77
low	low	12	2000	90,77	72,03	68,93	49,3	91,67	78,99	73,99
low	low	11	500	91,33	79,7	77,23	0	78,47	84,09	79,12
low	low	11	2000	87,67	73,3	69,65	50,86	91,09	81,34	74,24

Table 5.2: Results of first phase evaluation

Figure 5.1 shows the results for the first combination of parameters KEY_DENSITY and SCORE_CUTOFF: high key density and low score cutoff. The

first graph shows start and stop accuracy for different lengths of the reads and different key lengths. Results of mapping reads with length 2000 are colored with red and reads with length 500 are colored with blue. Start-stop metrics graph shows that reads with length 2000 are mapped more accurate than reads with length 500, with start-or-stop value exceeding 90% for all key lengths. For reads of length 2000 start-stop metrics are more accurate as the keys get shorter, while for reads of length 500 start-stop metric is best for key with length 12. Overall start-stop accuracy is high for low score cutoff and high density. The hit rate graph shows high percentage of hit rate for every read length and key length. The accuracy was highest for key length 13 and worst for key length 11. Reads of length 500 achieved higher hit rate than reads with length 2000. Also, reads of length 500 show higher coverage accuracy than reads of length 2000, with coverage amount being highest for keys of size 11 and lowest for keys of size 13, and coverage precision having similar values for all keys.

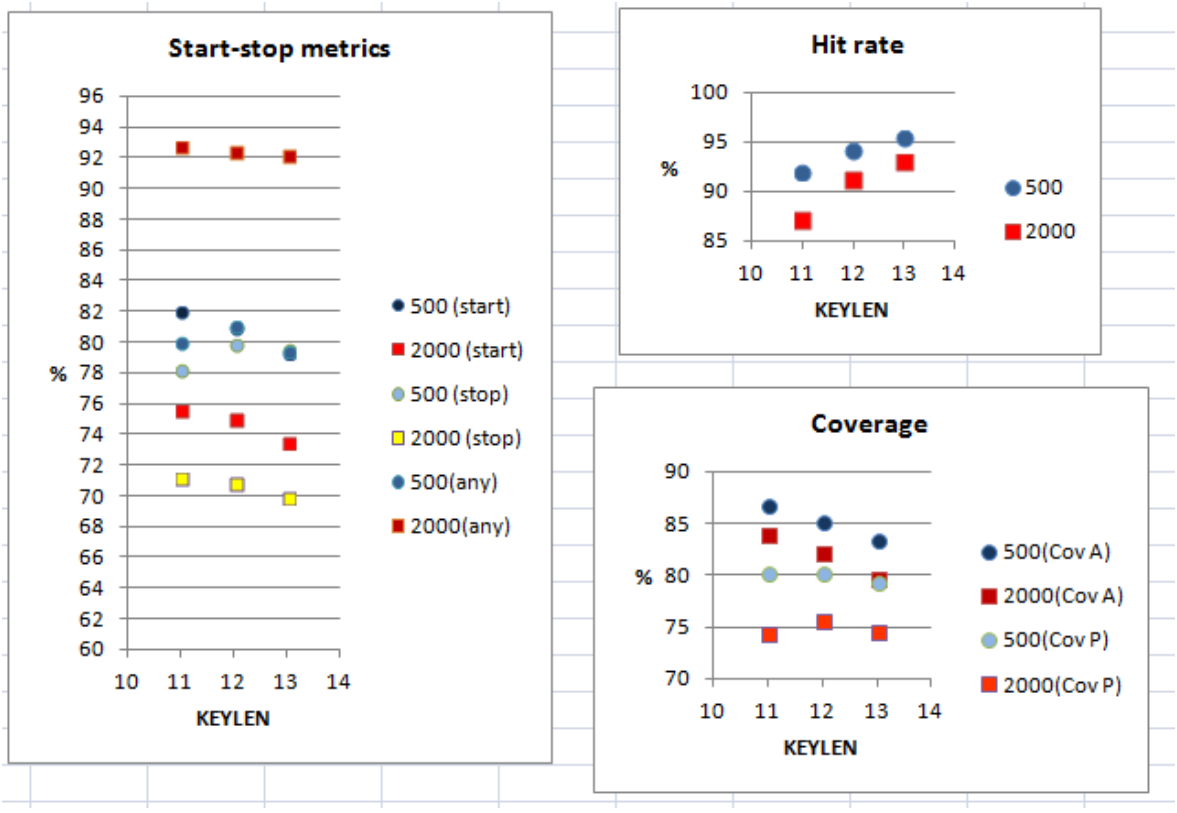

Figure 5.1: First phase metric results for high key density and low score cutoff

Results of analysis CM with second combination of key density and score cutoff are given in Figure 5.2. These are the results for low key density and high score

cutoff. Start-stop metrics are higher than for the previous test for reads of length 500, and they are most precise for key 12. For reads of length 2000 they are slightly more precise than in the previous test with also key 12 being the most precise. Coverage accuracy is also more precise than in previous example, where key 12 also shows best accuracy, and reads of length 500 showing more precise results. Hit rate is on the other hand much lower than in the previous example with reads of length 2000 having hit rate around 60% and reads of length 500 having hit rate around 40%. This is expected since high score cutoff lowers the number of resulting alignments while increasing start-stop accuracy and coverage accuracy.

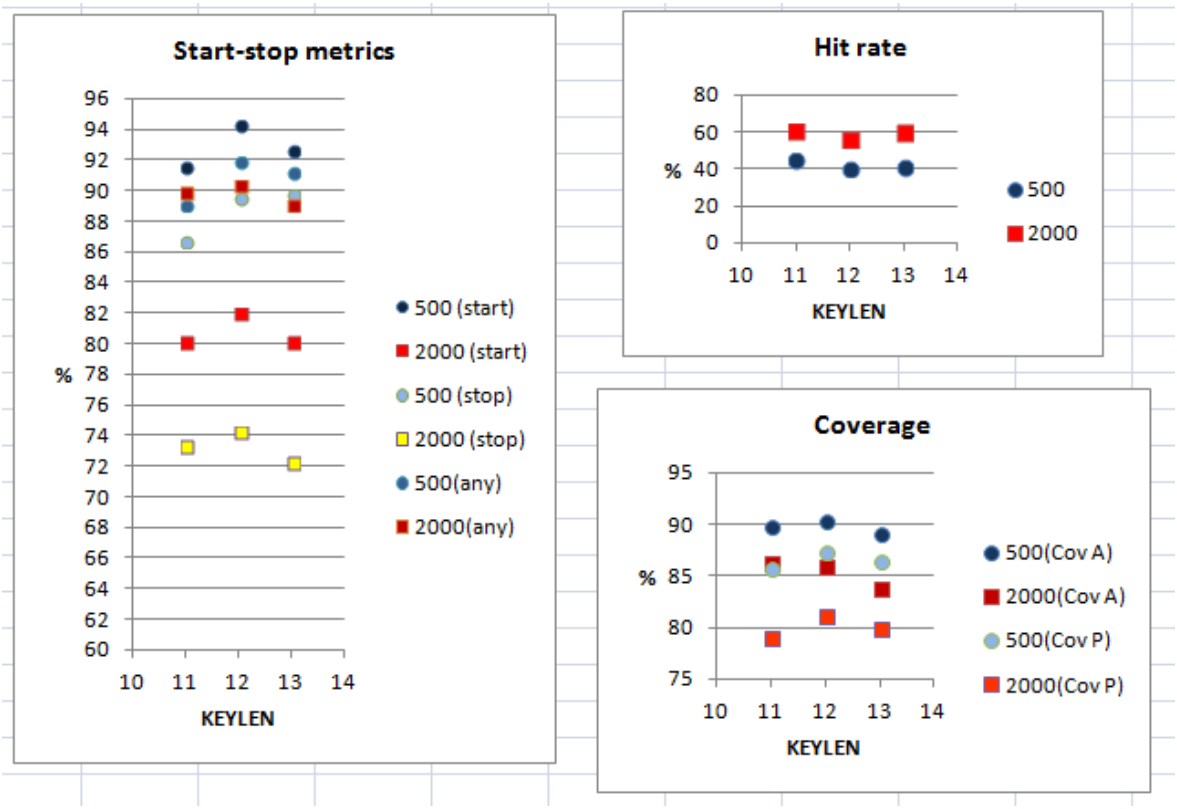

Figure 5.2: first phase metric results for high score cutoff and low key density

Results of analysis of the next combination of parameters are shown in Figure 5.3. In these graphs high score cutoff and high key density were tested. As expected hit rate is also low for this test as it was for previous test. High cutoff has high effect on the number of reads that will be aligned. Because of the high density, this test example has slightly higher hit rate, but the hit rate metrics are mostly determined by score cutoff. Also, this combination of parameters shows better precision than the previous combination, with start-or-stop metrics having

value of 96% for reads with length 2000. It can be concluded that higher key density has good effect on reads of bigger size such as 2000, while for short reads lower key density is equally good. Also for most of the test examples key length of 12 shows the most precise results thus showing that neither very short keys or very long keys are the answer for higher precision.

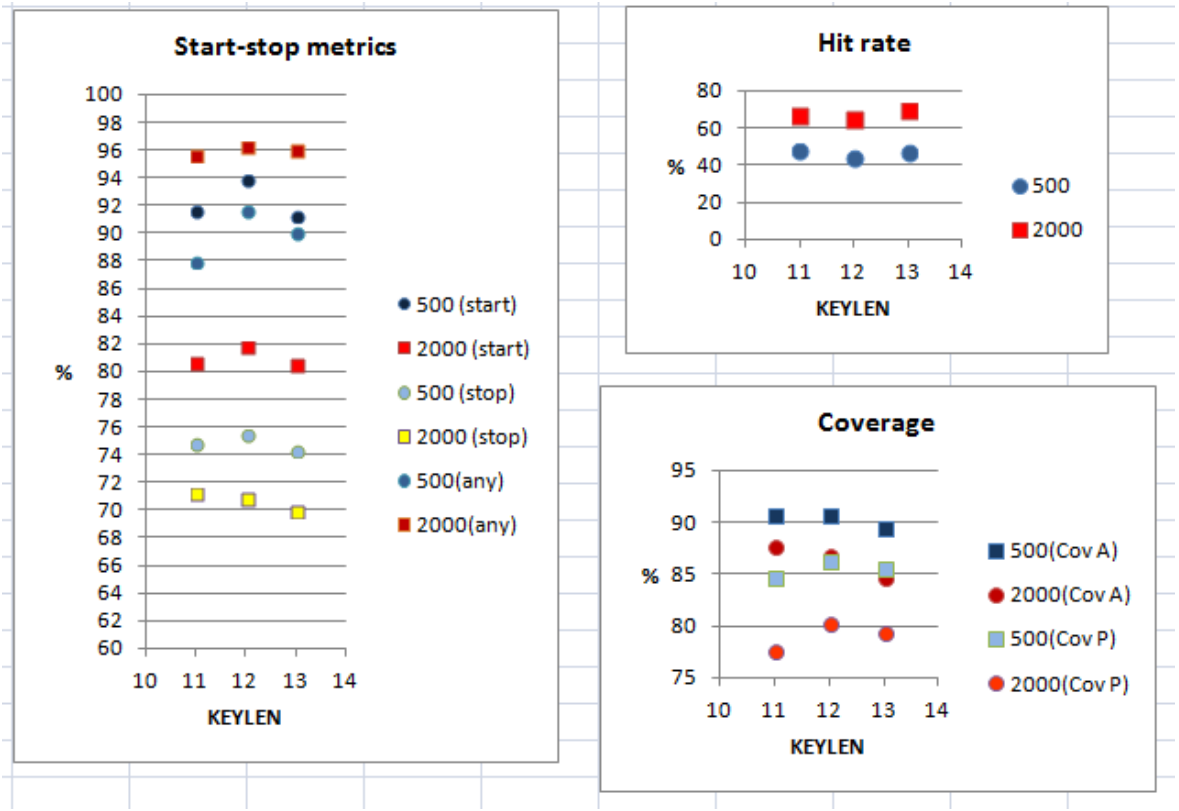

Figure 5.3: first phase metric results for high key density and high score cutoff

The last combination of parameters are low score cutoff and high key density. Results for this combination are showed in Figure 5.4. The results for this combination are similar to the one with combination of low score cutoff and high key density shown in Figure 5.1. Start-stop metrics are higher for reads with length 2000. Hit rate is very high with values around 90% for different keys. Reads with length 500 have higher hit rate than reads with length 2000, but overall hit rate is lower than for the first combination of parameters. Coverage and start-stop metrics are better for key 11, same as for the first combination of parameters.

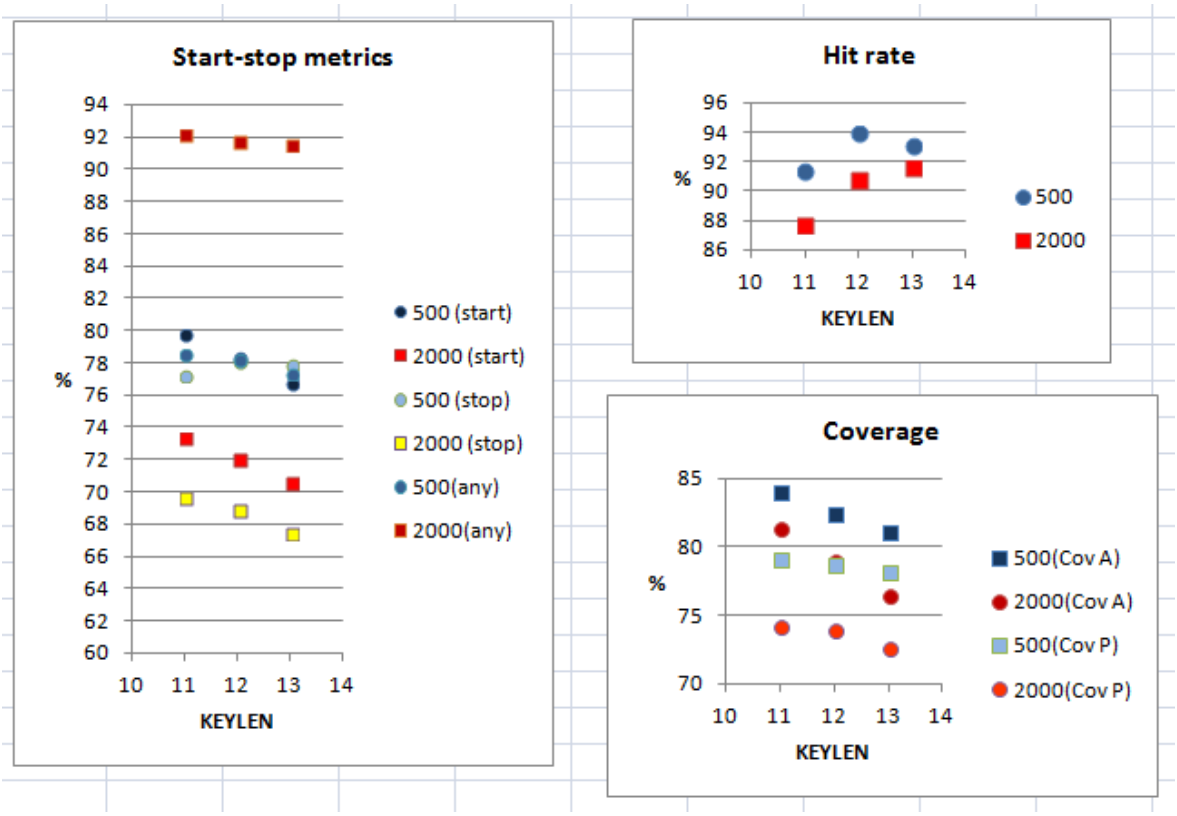

Figure 5.4: first phase metrics results for low key density and low score cutoff

It can be concluded that combinations 1 and 4 have higher hit rate because of the low cutoff score, and combinations 2 and 3 have higher start-stop accuracy as well as coverage accuracy because of the low score cutoff. High key density affects reads with length 2000, increasing hit rate as well as the precision. Reads with length 2000 have better results for combinations 2 and 3, while for combinations 1 and 4 reads with length 500 have higher accuracy and hit rate, but lower start-stop metrics. For combination 2 and 3 key with length 11 had better performance, suggesting that shorter reads could produce better results. For combination 1 and 4 key lengths had various effects on results. Key length 13 produced most hits, while key length 11 showed higher precision.

Figure 5.5 shows the results of time and space testing of CM for 4 different combinations of KEY_DENSITY and SCORE_CUTOFF parameters. First column shows time efficiency and second column shows memory usage. It can be seen that reads with length 500 are processed faster than reads with length 2000. Shorter keys have much greater time execution, but lower memory usage. High density and low cutoff produce a method with the slowest time execution, followed

by high density and high cutoff combination, suggesting that key density has higher influence on time execution than score cutoff. Low density and high cutoff, as expected, produced the fastest method. When considering accuracy evaluation presented above, it can be noticed that shorter keys have much greater execution time but do not produce proportional quality in accuracy and hit rate, especially key 11 which produces methods which are extremely slow.

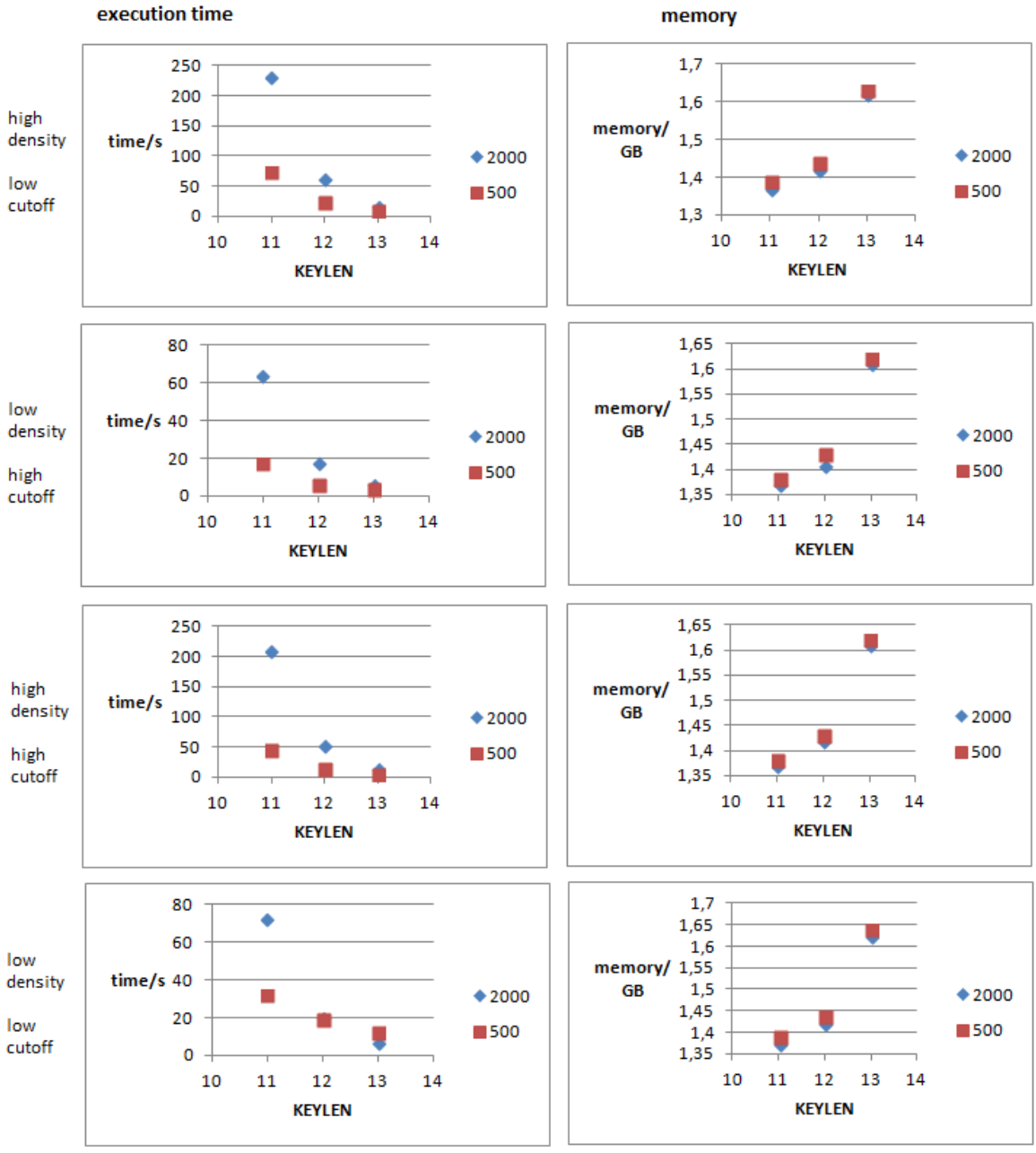

Figure 5.5: Time and memory results of first phase evaluation

Figure 5.6 shows only results of execution time for key 13. The combination of low key density and high score cutoff is the fastest, but that is the combination which produced a low hit rate with a not so high precision. Reads with length 500 are faster for all combinations.

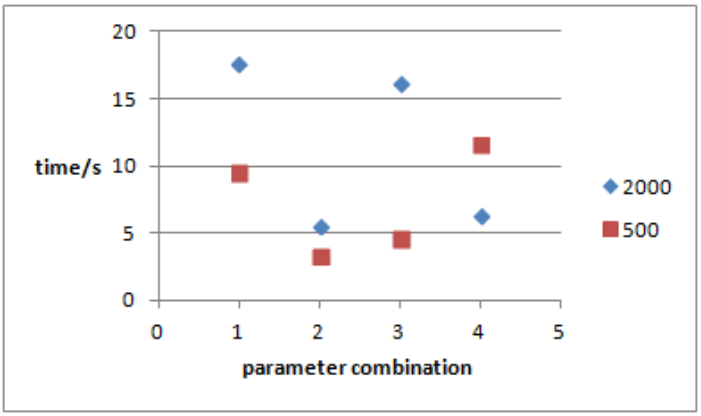

Figure 5.6: Execution time results for first phase evaluation for key 13

When taking all tests into consideration it can be concluded that combinations 2 and 3 for parameters KEY_DENSITY and SCORE_CUTOFF are not suitable for CM because they show very low hit rate despite lower execution time and slightly higher precision than other two combinations. Also, keys lower than 13 did not show dramatically more precise results and some had worse results than key 13, with key 13 showing best results in execution time tests. Combinations 1 and 4 do have lower accuracy than combinations 2 and 3 because they use low score cutoff, but low cutoff is also responsible for high hit rate. Also, combination 1 has slightly higher precision and hit rate, because it uses high density which produces more hits. Despite it being the slowest combination, combination 1 with high density and low score cutoff seems to be the best for CM because its high hit rate and coverage accuracy enable the computation of quality coverage information and enable usage of second phase method and realignment of reads. This is why the second phase and realignment read tests will be done using the first phase method with low score cutoff, high key density and key with length 13. Both read lengths will be tested because reads with length 2000 show better start-stop accuracy and offer the possibility to test both read lengths in second phase. If the reads are cut to length 500 in the first phase they will have the length 500 in the second phase.

5.4.2. Comparison with BBMap

The first phase method that was evaluated in the chapter above will be compared with BBMap. Execution time and hit rate were compared to check how similar the two tools are. Comparison was executed with four test sets generated with the BEERS simulator. Test results are given Table 5.3.

Test	Length of reads	Error rate	Number of reads	Hit rate BBMap	Execution time BBMap	Hit rate dev. tool	Execution time dev. tool
1	250bp	5%	48294	99,08	2,74 sec	98,56	2,29 sec
2	250bp	10%	49147	92,25	3,46 sec	96,73	2,88 sec
3	500bp	5%	48979	98,27	3,66 sec	97,66	6,18 sec
4	500bp	10%	49069	94,29	3,28 sec	96,35	4,41 sec

Table 5.3: Comparison of BBMap and developed tool

Four test samples with short read values and high error rate for short reads were aligned to the reference genome with BBMap and with developed tool. Both tools had very high hit rate, exceeding 90% for all tests. Analysis given in [4] showed that BBMap had much higher hit rate than any other tested tool including Tophat, but for reads shorter than 500 and with lower error rate. For longer reads with higher error rate BBMap stopped working. Figure 5.7 shows the values of execution time for different test sets. While the developed tool is better for reads of length 250 bp, BBMap is better for reads of length 500 bp. However, the execution times have quite similar values.

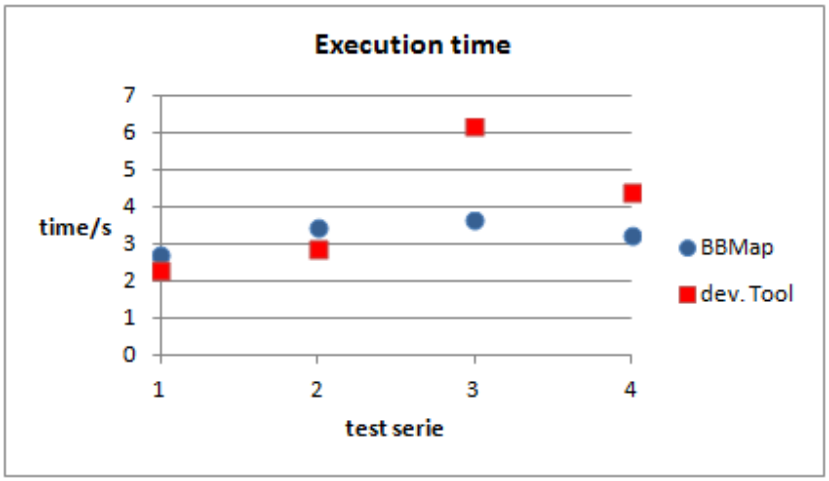

Figure 5.7: Execution time of BBMap and developed tool for 4 test sets

Figure 5.8 shows the values of hit rate for every test set. BBMap has slightly better hit rate for reads with error rate 5%, and the developed tool has better results for reads with error rate 10%. While the developed tool was a recreation of CM that BBMap uses, not every detail could be reimplemented correctly. Because of this, there are differences in resulting alignments, but this evaluation shows that the behavior of developed tool is very similar to the behavior of BBMap for short reads. This evaluation also shows that BBmap and the developed tool have excellent hit rate for short reads as it was shown in [4].

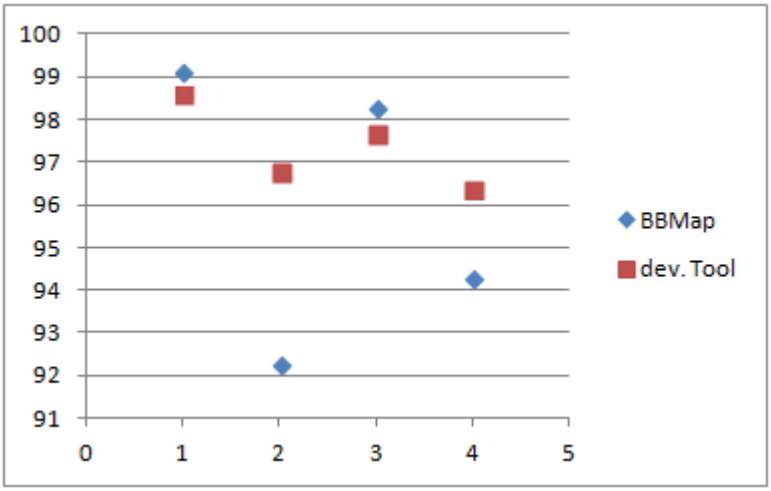

Figure 5.8: Hit rate of BBMap and developed tool for different test sets

5.4.3. First phase Evaluation Conclusion

There are a few important conclusions that the evaluation of CM or the first phase method revealed. Firstly, it was shown that the CM shows results very similar to BBMap. It was also shown that using parameters that BBMap uses for aligning short reads, such as low KEY_DENSITY and high SCORE_CUTOFF is not suitable for longer reads with higher error rate. Making keys shorter also did not produce better results, but made them less precise because shorter keys tend to produce more exon regions that there actually are. Splitting reads to shorter lengths, thus making them short reads, did produce good results, because BBMap is suitable for short read alignment, but it did not produce better results than aligning longer reads. This however could be improved by using the knowledge of shorter reads actually being the same read. The CM itself showed great results for short reads and solid results for longer reads with higher error rate. It has big

potential for using it as a foundation for acquiring starting results of long read alignment and then using different methods, techniques and heuristics to improve these results.

5.5. Second Phase Evaluation

This chapter covers the evaluation of the second phase of the TPM. For the first phase the following parameters were chosen: KEY_DENSITY was set to low, SCORE_CUTOFF was set to high and KEYLEN was set to 13. Two tests were processed, one with the read length of 500 in the first phase and one with the read length 2000 in the first phase. The process of evaluation was similar to the one for the first phase; same set of reads was processed for every combination of parameters: KEY_DENSITY, SCORE_CUTOFF, READLEN and KEYLEN. Hit rate, start-stop accuracy and coverage metrics were calculated. This analysis will be presented in two parts: evaluation with read length 2000 in the first phase and evaluation with read length 500 in the first phase.

5.5.1. Evaluation with read length 2000 in the first phase

Figure 5.9, Figure 5.10, Figure 5.11 and Figure 5.12 show the results of second phase method for the four combinations of parameters KEY_DENSITY and SCORE_CUTOFF. Figure 5.9 shows results for high KEY_DENSITY and low SCORE_CUTOFF. It can be seen that the coverage amount gets higher the shorter the key length gets. Coverage precision on the other hand tends to drop as the key length gets shorter. Reads with length 500 showed much better hit rate and coverage precision and accuracy than reads with length 2000. Best start-stop metrics has highest values with keys 10 and 11.

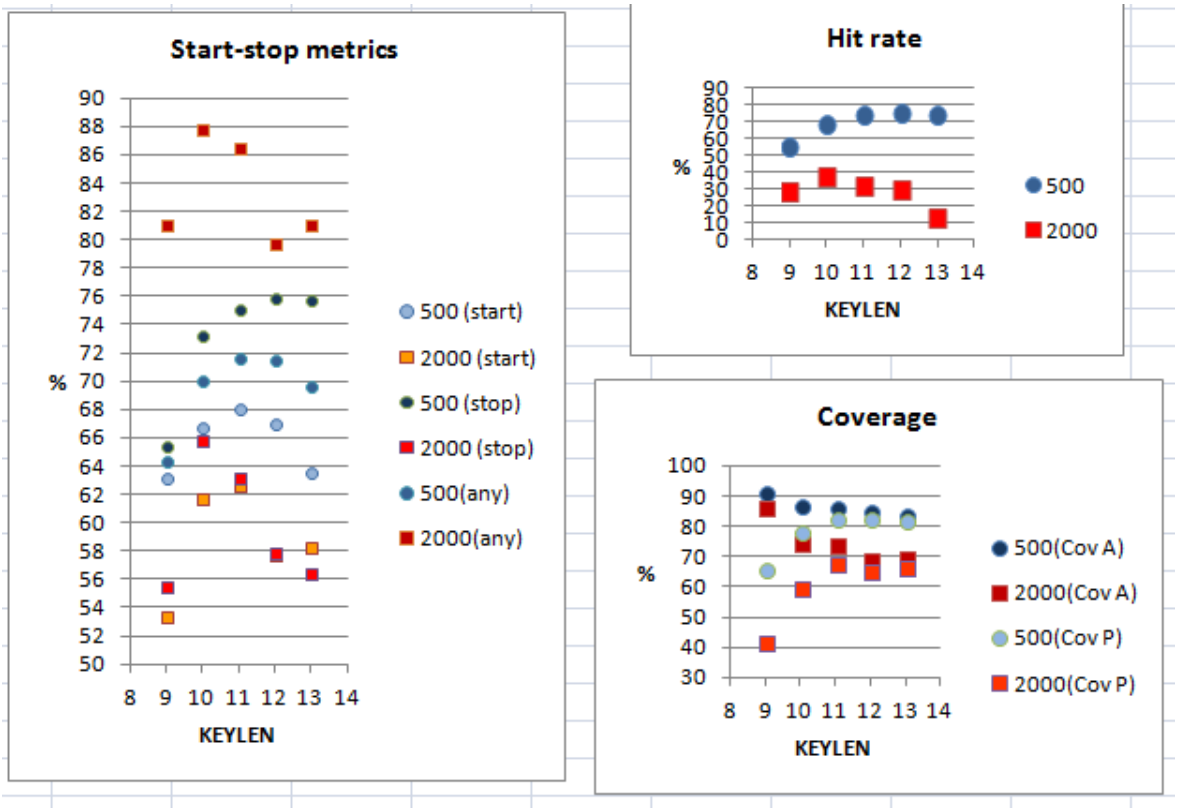

Figure 5.9: Second phase metric results for high key density and low score cutoff with uncut reads in first phase

Figure 5.10 shows the results for low KEY_DENSITY and high SCORE_CUTOFF. As expected this combination has low hit rate in this phase as well. Reads with length 500 produce more hit rate as in the previous combination. For this combination, key length 13 has better start-stop metrics which is probably because in the previous example same reads were aligned with same KEY_DENSITY and SCORE_CUTOFF values, while in this test different combination of parameters produces great results for key length 13. Coverage accuracy still shows the same properties as in the previous example, but it is higher than in the previous combination.

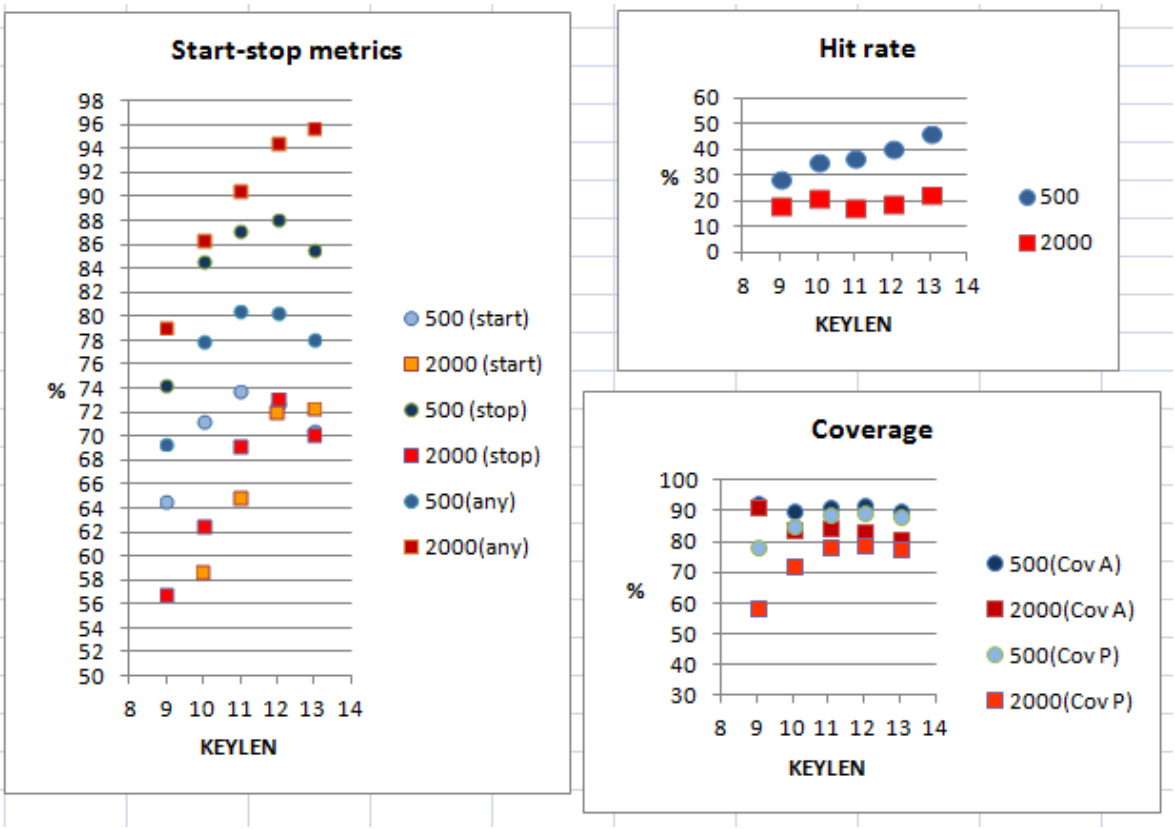

Figure 5.10: Second phase metric results for low key density and high score cutoff with uncut reads in first phase

Figure 5.11 shows results for combination of high KEY_DENSITY and high SCORE_CUTOFF. Results are very similar to the previous example with low hit score, better results for reads with length 500 and high coverage and start-stop precision and accuracy.

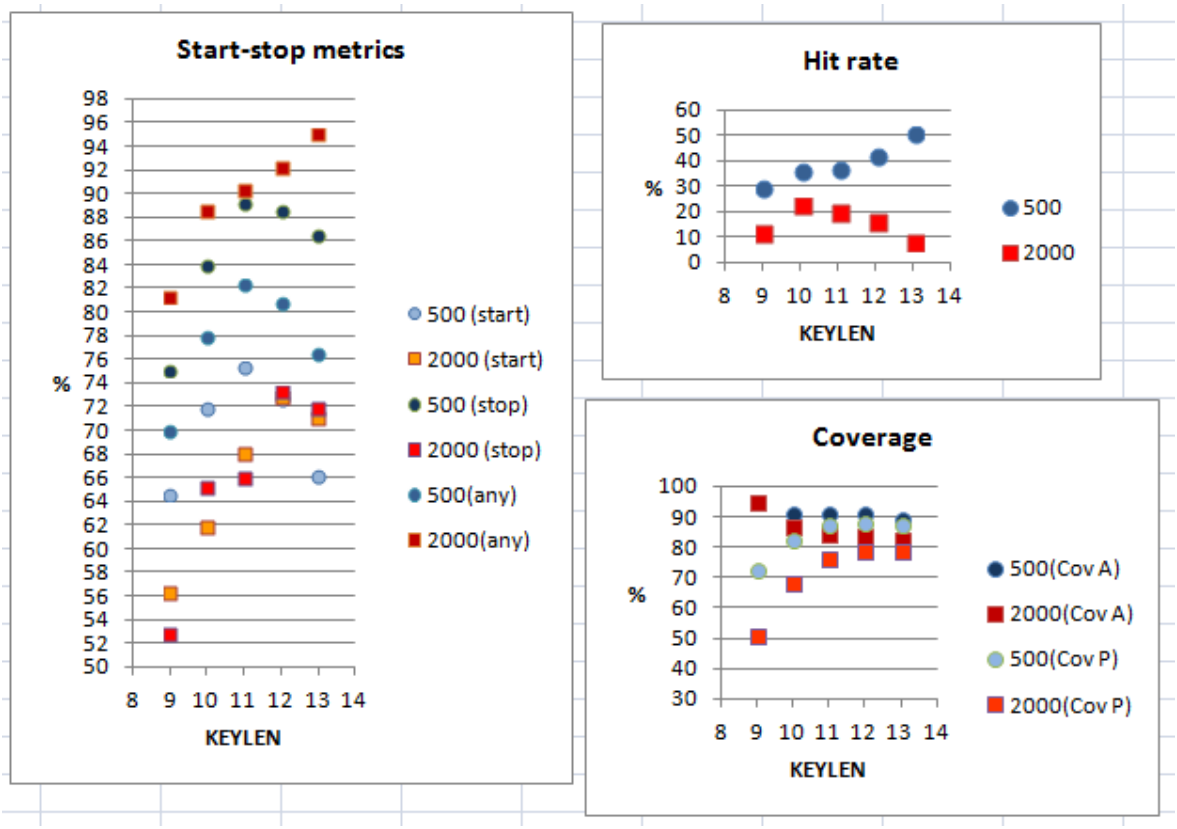

Figure 5.11: second phase metric results for high key density and high score cutoff with uncut reads in first phase

Figure 5.12 shows the last combination of low KEY_DENSITY and low SCORE_CUTOFF. This combination shows similar results as combination 1, but with the different results for read length 2000 and key length 13. In this combination, these parameter values produce high start-stop metric values. Again this is probably because different combinations of parameters affects different reads and produces more alignments. Still, the overall results are much better for reads with length 500.

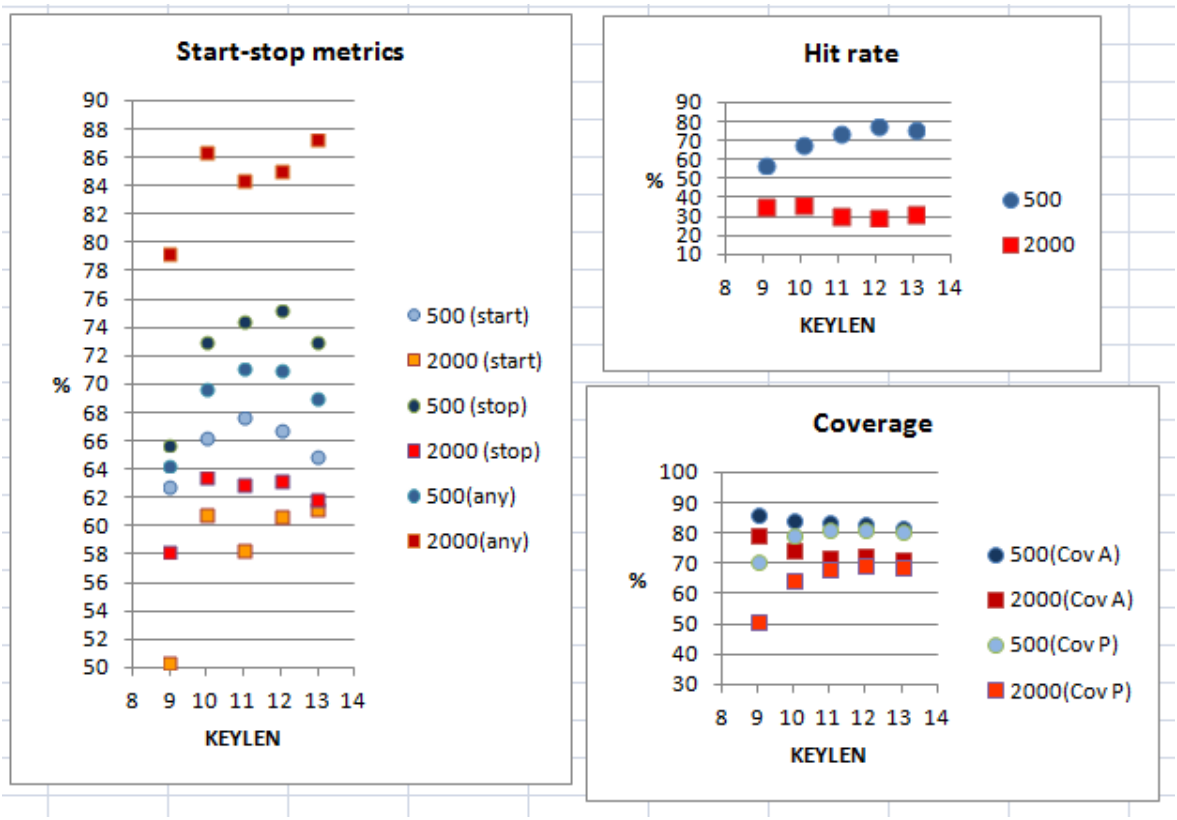

Figure 5.12: second phase metric results for low key density and low score cutoff with uncut reads in first phase

There are a few general conclusions that can be made from this analysis. Low value of SCORE_CUTOFF generally produces low hit rate and high coverage precision and amount. High precision is important but the hit rate is too low to be useful. Coverage precision and amount show the same property for all parameters; the amount rises with shorter keys and precision drops. This is because shorter keys with long reads and high error rate produce exon regions that don't exist. Tolerable ratio of coverage precision and accuracy is achieved for keys 11, 12 and 13. Reads with length 500 produced better hit rate and coverage precision and amount for all tests. This is because the reads with length 2000 which were unaligned, were cut to reads of length 500 in the second phase where some of these reads had low error rate or less splices. This is why they were mapped in the second phase. For low SCORE_CUTOFF, low KEY_DENSITY produces better start-stop accuracy for key 13, which is probably the result of different parameter combinations affecting different reads. Also for combination 4, coverage metrics and hit rate show slightly better results than for combination 1.

Since key length 11 shows highest coverage accuracy and precision and the number of unaligned reads is much lower than total number of reads, the execution time should not be of the highest priority, a best solution for second phase could be: low KEY_DENSITY, low SCORE_CUTOFF, read length 500 and key length 11.

5.5.2. Evaluation with read length 500 in the first phase

Figure 5.13, Figure 5.14, Figure 5.15 and Figure 5.16 show the results for evaluation of four combinations of parameters KEY_DENSITY and SCORE_CUTOFF for read length 500 used in first phase method. Coverage accuracy and precision show same property as in the previous tests with read length 2000 used in first phase method; precision drops with shorter keys and amount rises. Combinations 1 and 4 again show similar results to each other and combinations 2 and 3 show similar results to each other.

Figure 5.13 shows results for low SCORE_CUTOFF and high KEY_DENSITY. Hit rate for key 13 is very low because the same parameters were used in first phase. Overall hit rate is much lower than for the same combination of parameters used with read length 2000 in first phase.

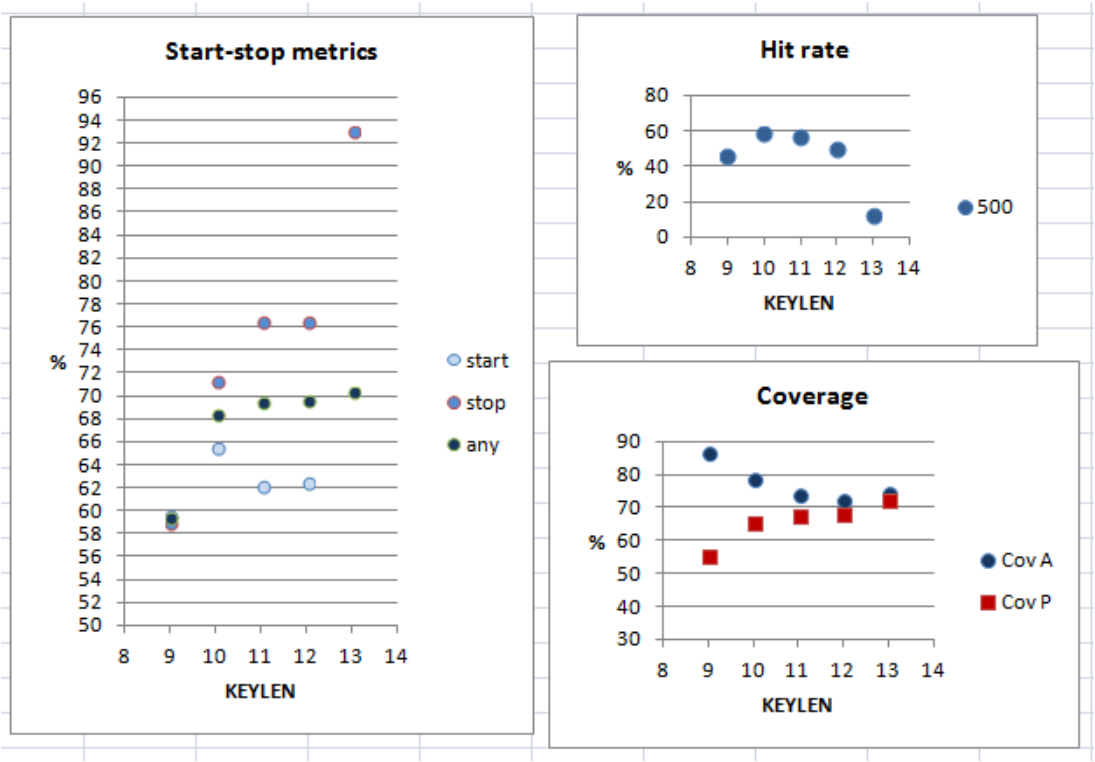

Figure 5.13: second phase metric results for high key density and low score cutoff with cut reads in first phase

Figure 5.15 and Figure 5.15 show results for combinations: low KEY_DENSITY, high SCORE_CUTOFF and high KEY_DENSITY and low SCORE_CUTOFF. These combinations produce very poor results with extremely low hit rate.

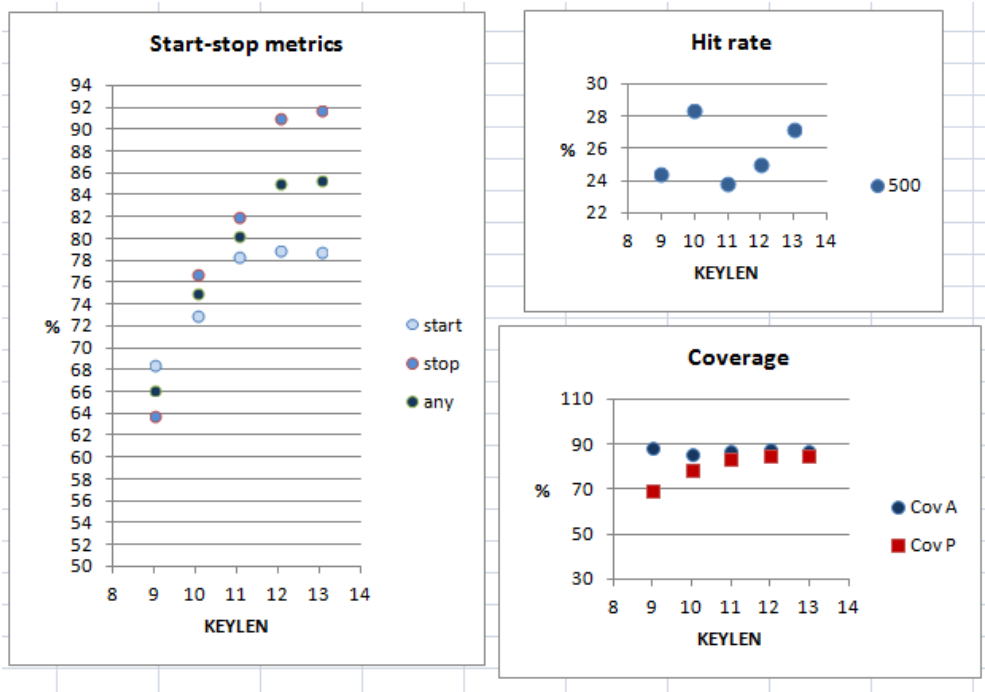

Figure 5.14: second phase metric results for low key density and high score cutoff with cut reads in first phase

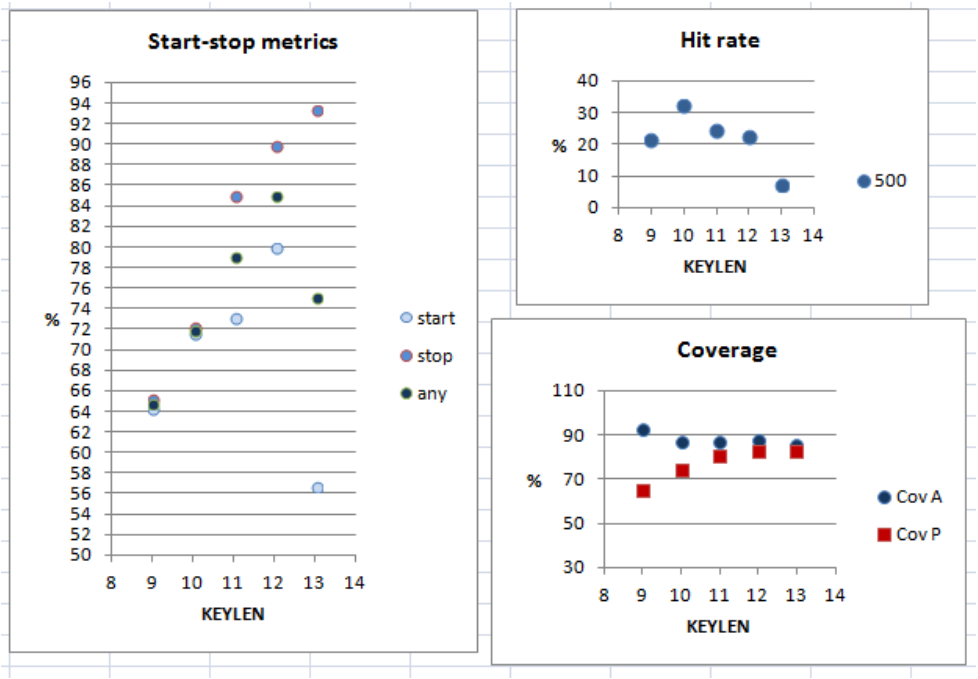

Figure 5.15: second phase metric results for low key density and low score cutoff with cut reads in first phase

Figure 5.16 shows last combination of low KEY_DENSITY and low SCORE_CUTOFF. This combination produces similar results as combination 1 with better results for key 13.

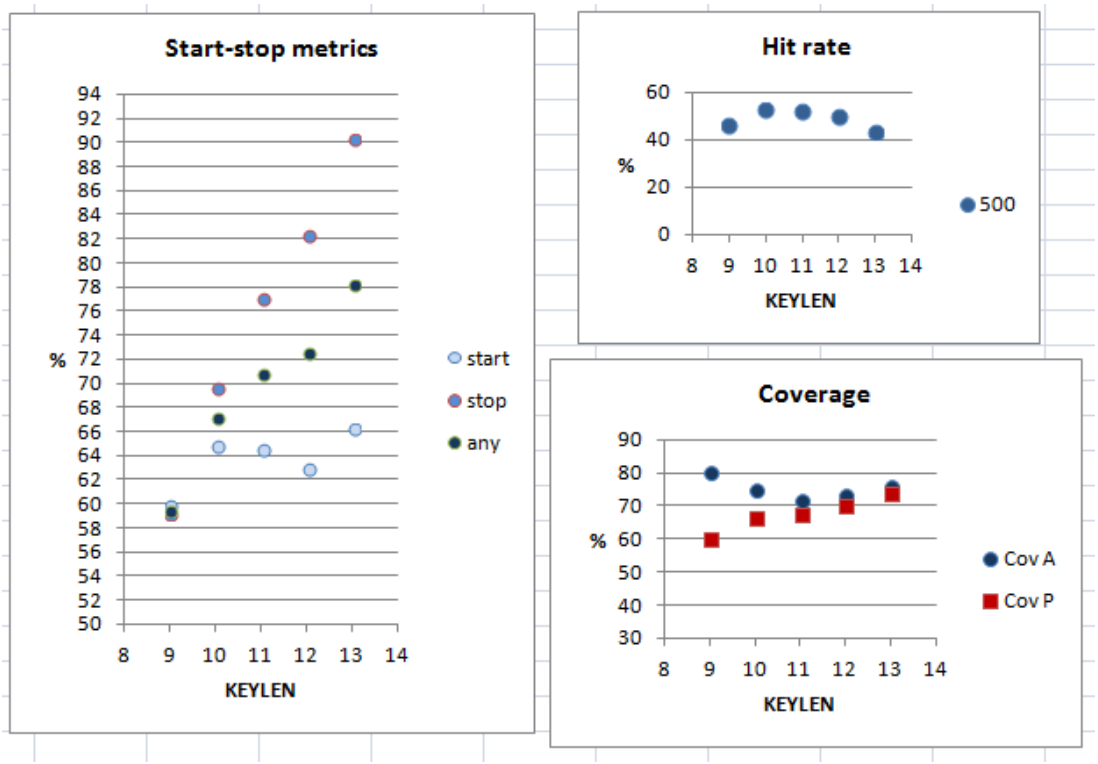

Figure 5.16: second phase metric results for low key density and low score cutoff with cut reads in first phase

There is an explanation for poor results when using read length 500 in the first phase. When the reads are being cut to length 500 in the first phase, pieces of reads with fewer errors and less splices are aligned in the first phase, thus leaving only pieces of reads with high error rate and more splices, to be aligned in the second phase. When using reads with length 2000 in the first phase, some of low quality reads of length 500 are included in the 2000 bases long reads that are mapped to the genome. In the second phase low quality reads with length 2000 are being cut to reads with length 500 which produces some reads of size 500 with low error rate and less splices. That is why using reads with length 2000 in the first phase and then cutting reads to length 500 in the second phase produces better results with very high hit rate. Therefore a conclusion that using reads with length 2000 in the first phase is a better option than using reads with length 500.

5.5.3. Second phase evaluation conclusions

Second phase evaluation did show moderate improvements in resulting alignments. It showed that coverage information does provide additional information that can be used in improving results of alignment. Changing the key and splitting the reads had a greater role in aligning the reads that were not aligned in the first phase. Since the execution time for the second phase is very short compared to the first phase, due to the shorter reference genome and smaller number of reads, it is useful to use the second phase in the alignment method. However, most of the reads were aligned in the first phase and thus the second phase cannot improve the results dramatically. It is expected that in most cases most reads will be aligned in the first phase because that is important to obtain the coverage information and to align unaligned reads in the second phase.

Cutting reads to shorter values in the first phase was not a good experiment because most of the low error shorter reads were aligned in the first phase and the reads that entered the second phase could not be aligned because they had too high error rate or to many splices. This is why longer reads should be used in the first phase and cut reads should be considered in further methods for improvement of results.

5.6. Read Realignment Evaluation

The center of this part of evaluation is the realignment method. Read realignment is an idea of mapping reads whose value of score alignment is below THRES_SCORE value. These reads are aligned to the new reference genome built from coverage information, as explained in chapter 4.3. In this chapter, this method will be evaluated for different values of THRES_SCORE. Since the first phase of the method is executed with reads of length 2000, key with length 13, high KEY_DENSITY and low SCORE_CUTOFF, realignment of reads will be executed with same parameters to check how SCORE_CUTOFF parameter affects the results.

Evaluation of the realignment method was done with the same dataset of 23447 reads with length 2000 and error rate of 18%. After executing CM, all the reads that have low score were queued for the realignment method. Low score of the

read was determined by THRESH_SCORE parameter that represents the percentage of maximum score that one read can achieve. Every read that has alignment score lower than THRESH_SCORE percentage of maximum score is queued for realignment method. The realignment method was executed with same parameter values as CM to analyze the behavior of read realignment depending on THRESH_SCORE. The tests were executed for different values of THRESH_SCORE { 0.5, 0.10, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7 } and the results are shown in Figure 5.17, Figure 5.18, Figure 5.19 and Figure 5.20.

For different THRESH_SCORE values different number of reads was set for realignment. Also, different number of reads had better alignment score, different number had worse alignment score and different number had the same alignment score. Some of the reads were unaligned in the realignment method. Figure 5.17 shows the number of reads with same score in both methods, number of reads with better score in realignment method, number of reads with worse score in realignment method and number of reads that were unaligned in realignment method, for different values of THRESH_SCORE value. For THRES_SCORE values above 0.6 mostly all reads went through realignment method resulting in highest number of reads with better result. Most of the reads however had the same score value. The number of reads drops linearly from THRESH_SCORE 0.3 to 0.55. The number of unaligned reads is very low, while the number of worse reads is always lower than the number of better reads.

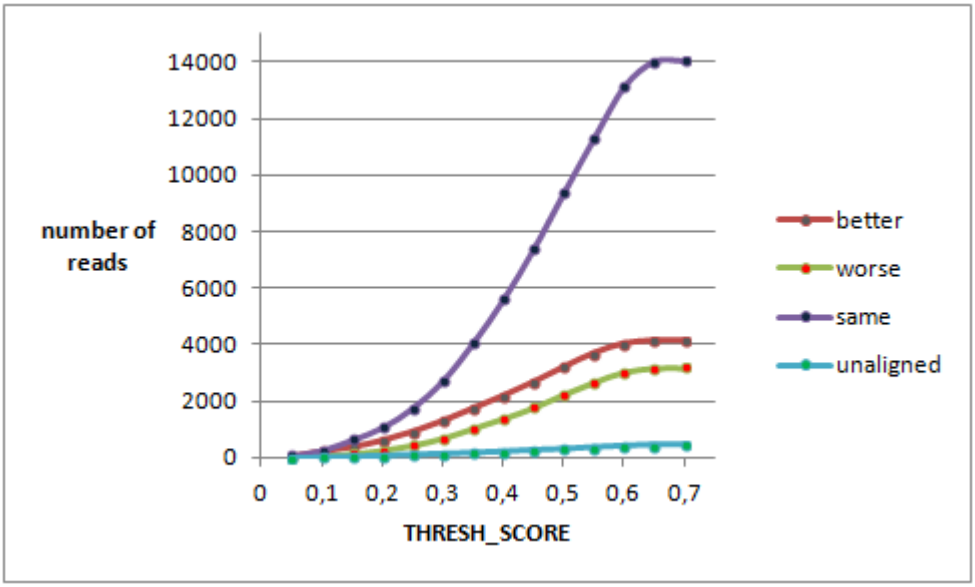

Figure 5.17: Distribution of read results for different values of THRESH_SCORE parameter

Figure 5.18 shows the ratio of start and stop metrics after the realignment method to the metrics before the realignment method. Both start accuracy and stop accuracy improve by more than 1%, while start or stop accuracy improves by 0.5%. This resulted in improvement of start and stop accuracy of 2.5% with THRESH_SCORE above 0.6. Start and stop accuracy rapidly drops from THRESH_SCORE 0.5.

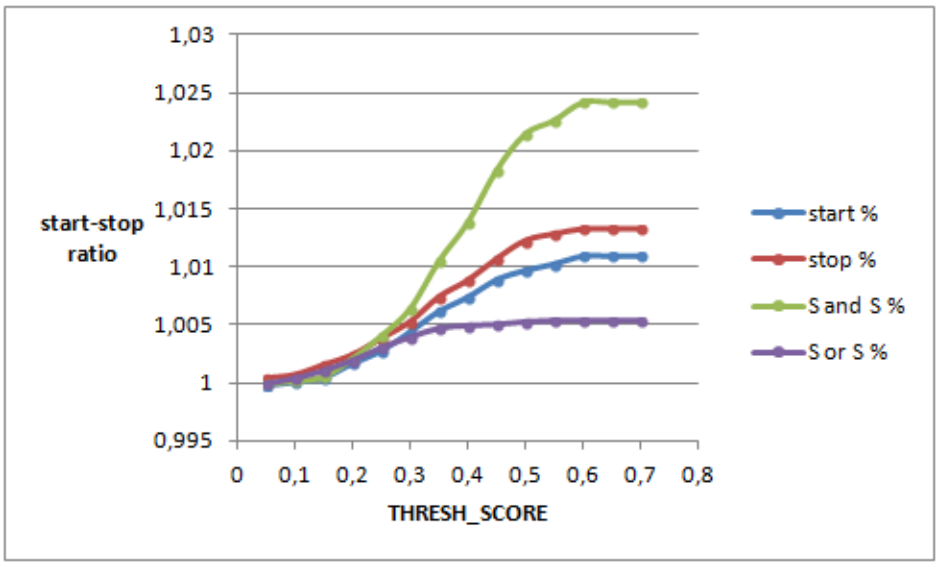

Figure 5.18: Distribution of start-stop metric improvement for different values of THRESH_SCORE parameter

Figure 5.19 shows the coverage amount and precision improvement with the realignment method. While coverage precision does not improve more than 0.5%, coverage amount improves for 2.5% for THRESH_SCORE above 0.55.

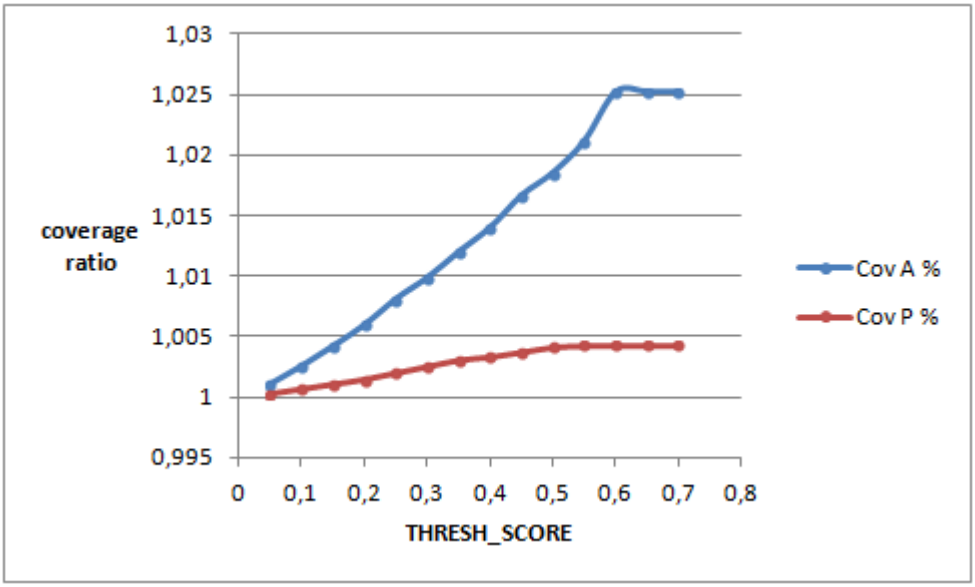

Figure 5.19: Distribution of coverage metric improvement for different values of THRESH_SCORE parameter

Figure 5.20 shows the execution time for different values of THRESH_SCORE. The execution time was measured for whole alignment process build in debug mode. This is why the execution time achieves very high values, but for the purpose of this test it is only important to analyze in what manner the different values of THRESH_SCORE affect the total execution time of the whole alignment process. The execution time is liner for THRESH_SCORE from 0.25 to 0.6. For the worst case, the execution of the realignment method causes execution time to rise by 20%.

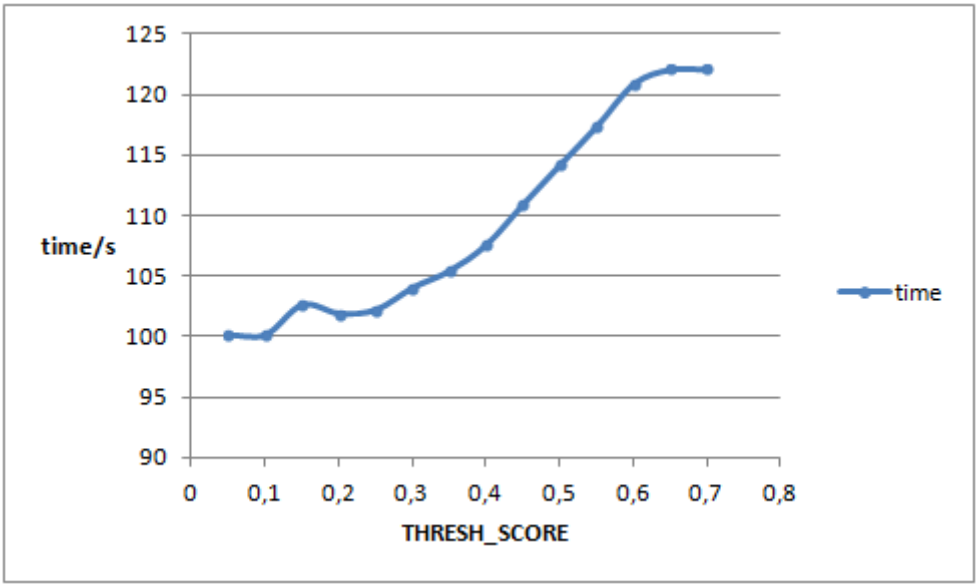

Figure 5.20: Execution time of realignment method

These results showed that some improvement can be achieved with realignment of reads, but in this case, realignment did not make a greater improvement of results. With execution time rising for up to 20% realignment of reads might not be worth using in the process. However this evaluation showed that some improvements can be achieved and that further analysis of coverage information could produce even better realignment results. Also, this evaluation showed that maybe the way of improving the whole alignment process could be achieved with local alignment and not with realignment of reads, because realignment process did not improve precision and accuracy of the results more than 2.5%. Also, this suggests that the CM itself might not be improvable with coverage information but it should rather be intrinsically improved and modified.

5.7. Best Solution and BBMap Comparison

After all research the best resulting method was chosen. That was a two phase method with read realignment. The first phase method was executed with the following parameters:

- KEYLEN: 13
- KEY_DENSITY: high
- SCORE_CUTOFF: low
- READLEN: 2000

The second phase method was executed with the following parameters:

- KEYLEN: 11
- KEY_DENSITY: low
- SCORE_CUTOFF: high
- READLEN: 500

Read realignment was executed with the same parameters as first phase and with THRES_SCORE = 0.55.

This resulting method was used to map the same dataset as used in previous tests. It achieved very high hit rate of 98,19%, leaving only 1.8% of reads unaligned. Coverage precision was 75.26%, leaving 25% of the bases from resulting alignments to be wrongly aligned. Coverage amount was 80,24%, which means that 80,24% of total number of bases were found in the genome. While coverage metrics are high, to achieve higher precision and accuracy other ideas should be explored. Start accuracy was 73,88%, while stop accuracy was 71,02%. Start or stop accuracy was 95,14% which means that almost every aligned read had at least start or stop aligned perfectly. This suggest that this method could be used to get correct starts and stops of the alignment and then use some other method to get higher precision. Start and stop precision was 49,77% which means that half of the reads had start and stop position mapped correctly. The whole alignment process had execution time of 17.54 seconds. BBMap had hit rate of only 51%. Why is this the case is left unknown since BBMap uses many complex parts of alignment process with the core method. TheBBMap alignment process had execution time of 6,34 seconds which is about three times quicker than the developed solution, but the hit rate is obviously much lower which suggest that using different improvement methods on the core BBMap method does produce much better results.

6. Conclusion

In this thesis, a new rna-seq aligner was developed from the aligning method used in BBMap tool. Basic principles of this method are explained so that its recreation can be available to everyone. It is shown that this method has very high hit rate especially for shorter reads with lower error rate, but it also shows good results for longer reads with higher error rate, produced by sequencing tools such as PacBio.

The analysis of core method showed that it can produce different results depending on used parameters. However, the results were always more precise and had more hit rate than other tools that were evaluated in [4]. Start and stop locations of alignment were precisely aligned that shows great potential in using this method as a basis for further improvements that would produce more precise alignments. Also, core method produce high hit rates for different read lengths and used parameters.

Few examples of improvements of this method have been proposed and evaluated such as coverage calculation and repetition of core method for unaligned reads or reads with low quality alignment. Coverage information showed that results can be further improved by detecting regions with high coverage and that it has potential for further research. Unaligned reads were successfully aligned using the additional information, while the realignment of low scored reads produced only slight improvements in the results.

While the coverage information is one type of method improvement, another improvement could be to improve the core method itself. This thesis provides basic information about functionality and behavior of the core method for different reads and parameters and could be used for further development of the core method. Cutting long reads to short reads but using the information which short reads come from the same long read could be the first step in building an improved core method.

Finally, very high hit rate and great accuracy in locating the start and/or stop of the read in the genome are the main strength of the core method and its improved variants. However, despite the fact that core method is build to detect splices and

gaps, they still remain the problem and are often the cause of imprecision in the resulting alignments because long reads have very high error rate. This problem could be tackled by using short local alignments on the small area where the core method aligned the read in the genome. This is a very promising improvement especially because the core method in most cases aligns the read at the location in the genome where it really was sequenced, even with the long reads with error rate of 18%. Local aligners would be the final touch that would give very precise results of the alignment, however they are much more time consuming than any aligner built in this thesis. An example of such improvement was build to make a representing string of an alignment, but its analysis was left for further research.

There is still much room for improvement in the field of RNA sequencing, but this thesis shows that the method which is used by BBMap is currently the best starting point in making further progress in development of not just RNA-seq tools, especially long read RNA-seq tools, but also in development in DNA sequencing tools. No splices and gaps could make the core method itself sufficient to align long DNA reads with high error, especially knowing that the tool does not need any changes to work with DNA data as well. But this is also a topic for further research.

7. Summary

RNA sequencing is a technology that uses next generation sequencing tools for transriptome analysis. Bioinformatics tools used are used to analyze RNA sequences to reveal a snapshot of RNA presence and quantity from a genome at a given time which is important for interpreting the functional elements of the genome and understanding development of diseases. This thesis presents a tool built from the BBMap aligning method that is used in mapping long reads with high error rate to the reference genome. Improvements of the developed method are also presented and evaluated. Results are analyzes and compared to the BBMap tool. This paper explains the aligning method used by BBMap and shows that it can be recreated in such manner that it produces highly precise alignment results.

8. Bibliography

[1] Y. Chu and D. R. Corey. RNA Sequencing: Platform Selection, Experimental Design, and Data Interpretation. *Nucleic Acid Ther*. 22(4): 271-274, 2012.

[2] C. A. Maher, C. Kumar-Sinha, X. Cao, S. Kalyana-Sundaram, B. Han, X. Jing, L.Sam, T. Barrette, N. Palanisamy, A. M. Chinnaiyan. Transcriptome Sequencing to Detect Gene Fusion in Cancer. *Nature*. 458(7234): 97-101, 2009.

[3] Qian F, Chung L, Zheng W, Bruno V, Alexander RP, Wang Z, Wang X, Kurscheid S, Zhao H, Fikrig E, Gerstein M, Snyder M, Montgomery RR. Identification of Genes Critical for Resistance to Infection by West Nile Virus Using RNA-Seq Analysis. *Viruses*. 5(7):1664-1681, 2013.

[4] I. Jerković. RNA-seq mapper, Faculty of Electrical Engineering and Computing, University of Zagreb, 2014.

[5] E. Tseng and J. G. Underwood. Full-Length cDNA sequencing on the PacBio® RS, *Pacific Biosciences, Research and Development, Menlo Park, CA,* 2013.

[6] K. F. Aua, V. Sebastianob, P. T. Afsharc, J. D. Durruthyb, L. Leed, B. A. Williamsf, H. van Bakelg, E. E. Schadtg, R. A. Reijo-Perab, J. G. Underwoodd, and W. H. Wonga. Characterization of the human ESC transcriptome by hybrid sequencing. PNAS, 2013.

[7] B. Bushnell: BBMap short read aligner, URL http://sourceforge.net/projects/bbmap/

[8] D. Kim et al: Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions, 2013

[9] G. R. Grant, M. H. Farkas, A. Pizarro, N. Lahens, J. Schug, B. Brunk, C. J. Stoeckert Jr, J. B. Hogenesch and E. A. Pierce. Comparative Analysis of RNA-Seq Alignment Algorithms and the RNA-Seq Unified Mapper (RUM). *Bioinformatics*. 2011.

[10] J. T. Witten, J. Ule: Understanding splicing regulation through RNA splicing maps, 2011