Nucleotide Sequence Alignment and Compression via Shortest Unique Substring*

Boran Adaş[1], Ersin Bayraktar[1], Simone Faro[2], Ibraheem Elsayed Moustafa[3], and M. Oguzhan Külekci[3,**]

[1] Department of Computer Enginering, İstanbul Technical University, Turkey
[2] Department of Mathematics and Computer Science, University of Catania, Italy
[3] Department of Biomedical Enginering, İstanbul Medipol University, Turkey
{adas,bayrakterer}@itu.edu.tr, faro@dmi.unict.it,
{iemoustafa,okulekci}@medipol.edu.tr

Abstract. Aligning short reads produced by high throughput sequencing equipments onto a reference genome is the fundamental step of sequence analysis. Since the sequencing machinery generates massive volumes of data, it is becoming more and more vital to keep those data compressed also. In this study we present the initial results of an on-going research project, which aims to *combine* the alignment and compression of short reads with a novel preprocessing technique based on shortest unique substring identifiers. We observe that clustering the short reads according to the set of unique identifiers they include provide us an opportunity to *combine* compression and alignment. Thus, we propose an alternative path in high-throughput sequence analysis pipeline, where instead of applying an immediate whole alignment, a preprocessing that clusters the reads according to the set of shortest unique substring identifiers extracted from the reference genome is to be performed first. We also present an analysis of the short unique substrings identifiers on the human reference genome and examine how labeling each short read with those identifiers helps in alignment and compression.

1 Introduction

Mapping short reads onto the reference genome is the fundamental initial step in the analysis of high-throughput sequencing data, where a large number of *alignment* software packages have been developed in the last decade [7]. In this paper we observe that clustering the short reads according to a set of unique identifiers of the reference genome they include provide an opportunity to improve both *alignment* and *compression* of short reads. To the best of our knowledge this is the first time this approach is used for the analysis of nucleotide sequences.

The general approach to achieve alignment fast in small memory footprint has appeared to be indexing the reference genome, and then seeking the occurrences of short reads one-by-one by using that index. It is not always possible to

* This work has been supported by the Scientific & Technological Research Council of Turkey (TÜBİTAK), BİDEB–2221 Fellowship Program, and also with the TÜBİTAK-ARDEB-1005 grant number 114E293.

** Corresponding author.

F. Ortuño and I. Rojas (Eds.): IWBBIO 2015, Part II, LNCS 9044, pp. 363–374, 2015.

exactly align each read since sequencing errors as well as differences between the sequenced individual and the reference are unavoidable. Thus, while mapping the reads, error-tolerant approximate matches should be considered. However, although there has been many efficient text indexing schemes for searching exact occurrences of the patterns, matching with symbol insertions, deletions, and mismatches is still an active research area.

Most of the aligners run with some parameters *limiting* the maximum number of mismatches/insertions/deletions allowed to occur while mapping a short read, and thus, especially large insertions or deletions are not easy to detect. With the ever increasing length of the short reads due to the technological advance of sequencing platforms, these limitations tend to become more severe. Underlining this fact, more recent aligners [10,1,12] as well as the new versions of the previous alignment packages [15,14] prefer to use k–mers of the short reads to roughly detect the mapping position on the reference genome, and then deploy a Smith-Waterman [18] style dynamic programming to achieve the task. In other words, instead of searching the whole read, the occurrences of k–mers extracted from the short read are scanned on the reference genome. When enough number of k–mers jointly points to a unique location, the Smith-Waterman algorithm is applied on the detected short region.

The point that is open for improvement in that approach is the optimization of the k value. The number of candidate regions increase with the short k values, and then it becomes difficult to decide on the correct region. Similarly, when k is set to a large value, sequencing errors or mutations are more likely to effect the performance, which is contrary to the basic idea behind the approach.

The ever increasing size of the data generated with high-throughput sequencing technologies requires to develop special methods to tackle with the problems of the huge genomic data sets [2]. In their *compressive genomics* definition, Loh *et al.* [16] stated *"algorithms that compute directly on compressed genomic data allow analyses to keep pace with data generation"*.

In that sense, compressing fastq files has been one of the most active research topics during the last few years [6], and many solutions have been proposed to represent those files as small as possible in size [5,8,4,11,3]. However, as stated in *compressive genomics* definition, the real challenge in fastq compression is more than the efficient archival of data, where we need support for operations to be achieved directly on compressed data such as efficient random access to any short-read as well as retrieving/extracting the reads mapped to a specific region of interest on the genome. The recent survey by Giancarlo *et al.* [9] lists the capabilities of the compressors in the genomic area in that sense.

The Idea and Our Contribution

A *unique substring* of the reference genome is a substring which is repeated only once in the whole sequence. In this paper we start by the observation that if a unique substring of the reference genome appears in a short read, then this short read can be mapped directly to the unique location of that substring identifier on the reference genome. This approach allows us to avoid to investigate any

other k–mers since the detected substring is unique on the reference, and thus, points to its location unambiguously.

Moreover we observe that clustering the short reads according to a set of unique identifiers they include provide us an opportunity to *combine* compression and alignment. Thus, conforming to *compressive genomics* approach, we present an alternative path in high-throughput sequence analysis pipeline, where instead of applying an immediate whole alignment, a preprocessing that clusters the reads according to the set of shortest unique substrings identifiers extracted from the reference genome is performed first. At the end of this preprocessing operation each read is assigned to a substring identifier. That binding represents a rough alignment as we know the position of the unique substring on the reference, and therefore, the rough position of the read. Once each read is associated with its unique substring identifier, the user may use this information both for the alignment and compression, and even combining these two operations.

For the alignment, assume the user has a specific region of the interest on the genome, and wants to see the reads sequenced from this section. One simply selects the shortest unique substring identifiers of that region from the previously prepared dictionary, and retrieves the reads labelled with these substring identifiers. The labels of the reads tell the rough position of the read, and a Smith-Waterman type alignment may be called for full alignment information.

For the compression task, the user may create the buckets which represents regions on the genome. These buckets store the short reads which include the unique substrings identifiers of the selected region, and can be compressed efficiently due to their high redundancy originating from the fact that they all repeat the information from the same region.

A combined approach would be first to create the buckets and keep them compressed, and then, answer the alignment queries by extracting and generating the *full* alignment information of the reads from the related buckets.

Organization of the Paper

The paper is organized as follows. In Section 2 we briefly describe the process we used for identifying the set of the shortest unique substrings from the human genome and we analyze and describe in Section 3 the set extracted substrings. In Section 4 we describe our dictionary matching algorithm for mapping the set of short reads in their positions in the human genome. Finally we present our results in Section 5 and draw our conclusions in Section 6.

2 Shortest Unique Substring Identifiers of the Genome

Shortest Unique Substring (SUS) finding [17] has received significant attention very recently, and efficient methods have been developed to solve the problem [19,13]. Each position on a text has a corresponding SUS for sure, where there might be more than one SUS for some positions. Interested readers may refer to the regarding publications for the proofs and more detailed discussions. Formally we have the following definition.

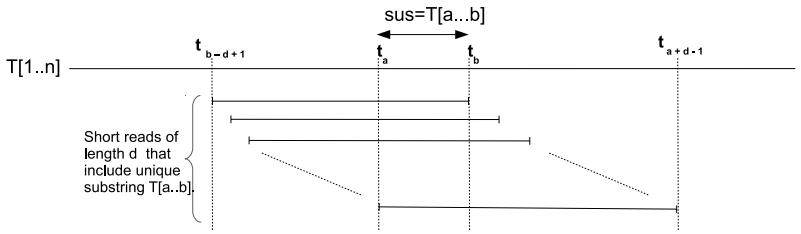

Fig. 1. Illustration of the short reads matching with the SUS identifier $T[a \dots b]$ assuming a constant read length d

Definition 1 (Shortest Unique Substring). *Given a text* $T[1, n]$ *of length* n*, the shortest unique substring covering the specific location* i*, for any* $1 \leq i \leq n$*, is the shortest string of length* ℓ*,* $T[a \dots a+\ell-1]$*, such that* $1 \leq a \leq i \leq a+\ell-1 \leq n$ *and* $T[a \dots a+\ell-1] \neq T[b \dots b+\ell-1]$*, for each* $1 \leq b \leq n-\ell+1$*.*

The most obvious usage of SUS detection appears in displaying the results of a string search on a target text. Assume we are searching the occurrences of a keyword that appears more than once in the given text. Thus, while displaying the results, it is helpful to display a bit of the context including the detected position of the occurrence. In such a scenario, the length of the to-be-displayed context may be tuned according to the SUS of that position, which uniquely informs about the position of appearance.

In this study, we introduce a novel preprocessing based on the SUS signatures extracted from the reference genome that would help in sensitive read mapping and compression. With that purpose we extract the SUS identifiers from the reference genome, and build a SUS dictionary, where each substring is stored with the position of its occurrence on the reference. Notice that this is an operation that needs to be done on a target reference just once.

Fig.1 illustrates how SUS identifiers can be used in the alignment process. Assume that $T[a \dots b]$ is such a SUS and d represents the short read length. The reads that include $T[a \dots b]$ are shown in the figure. If we do not let any insertions or deletion during the mapping, the leftmost appropriate read including this SUS should map to $T[b-d+1 \dots b]$, and similarly the rightmost one to $T[a \dots a+d-1]$.

The good thing is that once we caught the SUS in the read, we have the flexibility to allow larger error thresholds, since we know exactly the address of the short read matching with the SUS. Thus, to let insertion and deletions, the region might be extended a bit further to the right and left, and then, the short reads may be aligned to that extended region via a cache-oblivious dynamic programming as performed in [10,12].

Careful readers will quickly realize that in this scenario the length of the SUS identifier should be less than or equal to read length d. In addition to that, we seek an exact match between the SUS and short reads. Surely, we know that the possibility of a mismatch becomes more significant as the length of the

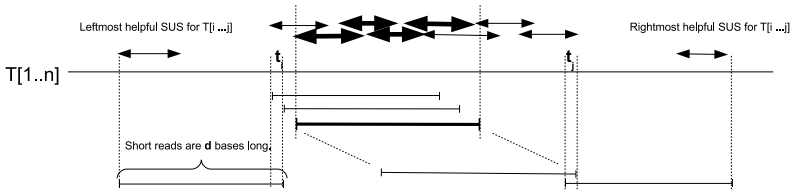

Fig. 2. A short read generally includes more than one SUS

SUS increases. Hence, long SUS identifiers are not supposed to help much, and we neglect in the SUS dictionary the ones that are longer than a predefined threshold λ during the operation. During our experiments in this study on human reference genome, we set that threshold to be $\lambda = 30$, which depends on the empirical experience that we can expect the sequencers today to be able to read that much of consecutive bases without any error.

Fortunately, a short read includes generally more than one SUS identifier as shown in Fig.2, where the sample read and the SUS candidates are marked bold. This becomes useful as we may still expect to have appropriate length SUS candidates, when we exclude the long SUS from the dictionary. Having more than one candidate helps in case of errors also, since an exact match of the short read at least with one of the SUS is enough to map it appropriately. For example in the Fig.2, the read can be located on the reference once one of the four possible SUS occur in it without an error. Below we give the formal definition of the SUS set of a region.

Definition 2 (Shortest Unique Substring Set of a Region). *Assume a region of interest $T[i \dots j]$ on the reference genome $T[1 \dots n]$ is specified and the constant length of the short reads is d. The* SUS *set of the specified region is the list of the* SUS *strings from the* SUS *dictionary, whose beginning positions on the reference genome are between $i - d + 1$ and $j + d - 1$.*

With the concern of aligning all the reads corresponding to an arbitrary region of interest $T[i \dots j]$, we seek the leftmost and rightmost SUS identifiers that are helpful to construct the region. With the term *helpful*, we mean there exists a chance that a short read including this SUS may cover at least one base from the target region. This is depicted in Fig.2 as when the selected leftmost (rightmost) SUS appears leftmost (rightmost) on a short read, that short read may cover the position t_i (t_j).

3 SUS Analysis of the Human Reference Genome

In this section we analyze the SUS identifiers we extracted from human reference genome GRCh38[1]. During our analysis we concatenated all chromosomes of the

[1] Available at `http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/`.

genome into a single string and changed everything other than A, C, G, T, N to N, and replaced consecutive repeating Ns with a single N letter. Considering the DNA sequencing technology, where short reads may originate from both the forward and reverse strands of the DNA, we appended the reverse complement of this string to its end, and thus, the resulting whole human genome is of length $5875280183 \approx 5.87$ billions bases.

For each position on this string, we have detected the corresponding SUS with the method of [13]. The operation took roughly 75 minutes on a machine with 256 GB memory and Intel Core 2 Quad processor running Linux Centos 6.2.

There may be more than one SUS (with the same length) for a position. We break the tie by choosing the leftmost one in such a case. Moreover one SUS may be shared by *consecutive* positions on the target string, and thus, we counted the number of distinct SUS in the SUS database of the whole genome. We found that $1924177251 \approx 1.92$ billion of the 5.87 billion items in the SUS database are unique when both the forward and reverse strands are taken into account.

Since long SUS identifiers are not useful in our strategy, we excluded the ones that are longer than the threshold value, which we set as 30 in our study (the longest SUS detected is nearly 1.2 million bases long). In addition, some of the SUS identifiers are either right or left extensions of neighbouring shorter ones. For instance assume a SUS is $T[a \dots b]$, and while searching for the SUS covering position $b+1$, it might be the case that $T[a \dots b+1]$ may be returned as the SUS of that position by the algorithm. We also get rid of such extension patterns, and create our final SUS dictionary composed of $963836205 \approx 1$ billion SUS identifiers.

A SUS has the potential to cover a position if it is in vicinity of d bases to that position. That is because, when that SUS appears at the very beginning or end of a short read, then that short read covers all λ positions to the right or left as shown in Figure 1. Certainly, it is much better for a position to have the chance of being covered by large number of distinct SUS.

For some positions on the human genome it might not be possible to detect a SUS identifier longer than the selected threshold 30 bases. If such positions does not have a neighbouring SUS in close vicinity, then the reads originating from this area has the danger of not being caught by any of the SUS identifiers from the dictionary. To measure this problem, we define below the theoretical SUS coverage of an individual position.

Definition 3 (Theoretical sus coverage of a position). *The leftmost short read possible to cover an inspected position i is $T[i-d+1 \dots i]$, and the rightmost short read including position i is $T[i \dots i+d-1]$. Notice that these short reads may be produced from both the forward and reverse strands by the sequencing equipment. Any* SUS *identifier $T[a \dots b]$, such that $i-d+1 \leq a \leq b \leq i+d-1$, may appear in those short reads covering the position i. Therefore, we define the theoretical* SUS *coverage of position i as the total number of such* SUS *identifiers on both the forward and reverse strands.*

Figure 3 shows the theoretical SUS coverage of the human reference genome. The short reads that include the positions, which have 0 SUS coverage, have no chance of being identified by the proposed scheme. We call these reads *orphan*,

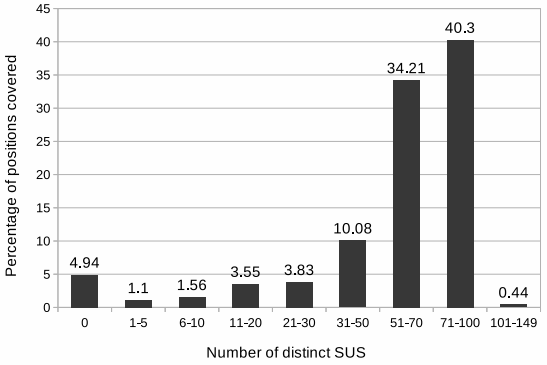

Fig. 3. The theoretical SUS coverage of the human reference genome

and observed that less than 5% of the genome remains orphan. Those orphan positions are non avoidable due to the repetitive nature of the genome, but they can be handled efficiently by the regular k–mer approaches.

4 SUS Dictionary Matching

In this section we describe the algorithm we used to match the SUS collected in the dictionary against the set of short reads. Before entering into details we observe that an important property of the SUS dictionary is that none of the items appear as a substring of another item. This property is formally stated by the following lemma.

Lemma 1. *Let $S = \{s_1, s_2, \ldots s_m\}$ be the* SUS *dictionary, where s_i is a unique substring of the reference genome T. There exists no s_i in S, which appears as a substring in any other s_j, with $j \neq i$.*

Proof. Assume s_i appears in s_j, where $i \neq j$. We know that s_i and s_j are unique on the reference genome by definition of SUS. We have also deleted from the set the right or left extensions of SUS identifiers while creating the dictionary, and thus, s_j cannot be a right or left extension of s_i. Hence, if s_i occurs in s_j, this means s_i is not unique, which contradicts the hypothesis. □

Based on Lemma 1 we devised an algorithm for fast scanning of the short reads against the SUS dictionary. Specifically during the preprocessing phase it builds a data structure in order to index all the SUS in the dictionary. Then This index is used to speed up the searching process in the subsequent phase, where the short reads are searched, one by one, for any occurrence of the given SUS.

In our algorithm we make use of the *longest common prefix* of two sequences as define below.

Definition 4 (Longest Common Prefix). *Given two strings, x and y over the same alphabet, the longest common prefix array (*LCP*) between x and y, in symbol* $\mathrm{lcp}(x, y)$*, is the maximal length ℓ such that $x[1 \dots \ell] = y[1 \dots \ell]$, where $\ell \leq \max(|x|, |y|)$.*

For example, if $x =$ ACATAC and $y =$ ACTTAGC then $lcp(x, y) = 2$.

In the following we describe separately the preprocessing and the searching phase of our algorithm.

The Preprocessing Phase

Let S be the SUS dictionary and let R be the set of the short reads as described above. In this section we give a description of the data structure we use for matching the SUS against the short reads and briefly describe the preprocessing of the input data.

The set S of the SUS contains DNA sequences with a length between 12 and 30 bases. We observed on the human reference genome that the shortest SUS is of length 10 bases, where 10 or 11 bases long SUS identifiers are very few. Thus, we decided to consider 12 as the bottom threshold for SUS signatures and just extended the ones with 10 or 11 bases to reach length 12.

We indicate the minimum length of an SUS in S with the symbol $m = 12$. For each $s_i \in S$, let p_i be the prefix of length m of s_i, and let r_i be the suffix of s_i of length $|s_i| - m$. It is clear that $r_i = \varepsilon$ when $s_i = m$. In this context we can write $s_i = p_i \cdot r_i$ for each $s_i \in S$.

When preprocessing the set S we compute a fingerprint $f(s_i)$ for each $s_i \in S$. The fingerprint of an SUS s_i is computed by translating its prefix p_i in an integer number as $f(s_i) = \sum_{j=0}^{m-1} code(p_i[j]) \times 4^{m-1-j}$, where $code : \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\} \to \{0, 1, 2, 3\}$ is a function which maps each character in an integer number. It is trivial to observe that the prefix of a SUS in S is uniquely described by a single fingerprint value. However there are SUS which share the same prefix, although they are different. Since the fingerprint value is computed on the prefix of length $m = 12$ of each SUS we have that $0 \leq f(s_i) < 2^{24}$ (where $2^{24} = 16.777.216$), for each $s_i \in S$.

During the preprocessing phase we construct an index table B of 2^{24} locations which is used to index all the sequences of length $m = 12$ over an alphabet of 4 elements. Then, for each s_i in S, we define a bucket, $b(s_i)$, containing useful information about the SUS and insert it in B according to its fingerprint. Thus each element $B[k]$ of the table is the set of buckets of all the SUS which share the same fingerprint k. More formally we have $B[k] = \{b(s_i) \ : \ s_i \in S \text{ and } f(s_i) = k\}$, for $0 \leq k < 2^{24}$. The set $B[k]$ is represented by a linked list where the buckets are lexicographically ordered according to the corresponding SUS. In this context we indicate with $prev(s_i)$ the SUS which precedes s_i in its linked list.

The bucket of each s_i in S is a triple $b(s_i) = \{i, lcp_i, r_i\}$, where

- i is the index of the SUS in the dictionary S. Such information is used to locate the SUS and its position in the reference genome.

– lcp_i is the longest common prefix between s_i and $prev(s_i)$.
– r_i is the suffix of s_i of length $|s_i| - m$.

The Searching Phase

During the searching phase we select each short read from the set R, one by one, and search it for the occurrence of any SUS in the dictionary S.

Let t be a short read in R and let n be the length of t. During the searching of t we open a substring w of length m over t, initially aligned with the left end of t so that $w = t[1 \dots m]$. We call such a substring the *window* of t. Then the window is slided to the right character by character until it reaches the right end of t.

For each alignment of the window w at position i of t (so that $w = t[i \dots i + m - 1]$), we check if any SUS in S has an occurrence beginning at position i of t. If no SUS occurs in t at position i the next iteration is started with a new alignment of the window at position $i + 1$.

For each iteration, say at position i, the algorithm computes the fingerprint k of the window $w = t[i \dots i + m - 1]$. Then it easy to observe that only the SUS in the set $B[k]$ can occur at position i of t, since they share the same prefix as the window. Thus the algorithm checks the element of the set $B[k]$, one by one, until an occurrence is found or all possible candidates have been checked. The elements of the set $B[k]$ are checked by following a lexicographical order of the correspondent SUS.

Let $s_{i_1}, s_{i_2}, \dots, s_{i_n}$ be the n SUS in the set $B[k]$, in lexicographical order. Since we already know that the first m characters of s_{i_1} are equal to $t[i \dots i + m - 1]$, the algorithm scans the characters of the read t starting from position $i + m$ and comparing them with the corresponding characters in s_{i_1}, until the whole SUS is scanned or a mismatch is encountered. In the first case an occurrence is reported and the algorithm stops searching the read t. In the second case the algorithms discards s_{i_1} and continue comparing t with the next SUS s_{i_2}.

Suppose that the algorithm scanned j characters of s_{i_1}, starting from position $i + m$, before finding a mismatch. Thus we have $s_{i_1}[m + j - 1] = t[i + m + j - 1]$ and $s_{i_1}[m + j] \neq t[i + m + j]$.

We now recall that the value lcp_{i_2} is the maximal length of the shared prefix between s_{i_1} and s_{i_2}. Thus if $lcp_{i_2} < m + j$ we know that s_{i_2} cannot occur at position i of t. Moreover, for the same reason, none of the other SUS in the set $\{s_{i_2}, s_{i_3}, \dots, s_{i_n}\}$ can occur at position i of t. Thus in this case the scanning is stopped and a new iteration is started with a new window.

In the other case, if $lcp_{i_2} \geq m + j$ the algorithm continues comparing t and s_{i_2} starting at position $i + m + j$ of t until the whole SUS is scanned or a mismatch is encountered.

When a new iteration on the new window $w' = t[i + 1 \dots i + m]$ is started the algorithm can remember the length of the prefix which has been scanned in the previous iteration. Suppose j is the length of such a prefix, so that $s_{i_1}[m + j - 1] = t[i + m + j - 1]$ and $s_{i_1}[m + j] \neq t[i + m + j]$ and suppose $lcp_{i_2} < m + j$ so that a new iteration is started. Let k' be the new fingerprint value of the window w'.

By Lemma 1 we know that any SUS in $B[k']$, with a length less than $j-1$, can occur at position $i+1$. Thus the algorithm can discard from $B[k']$ all the SUS with a length less than $j-1$.

This process stops when an occurrence of any SUS in S is found in t or when the starting position i of the window reaches the value $|t|-m$.

Observe that the computation of the fingerprint of a given window $w' = t[i+1 \ldots i+m]$ can be computed in constant time from the fingerprint of the previous window $w = t[i \ldots i+m-1]$ by the following relation

$$f(w') = (f(w) - code(t[i]) \times 4^{m-1}) + code(t[i+m])$$

Thus the computation of all windows along a short read of length d can be done in $\mathcal{O}(d)$ time. However each iteration of the searching process requires $\mathcal{O}(\lambda - m)$ time in the worst case. Thus the worst case time complexity for searching a short read of length d for any occurrence of the SUS in S is $\mathcal{O}((\lambda - m)d)$.

Despite its quadratic worst case time complexity it turns out from our experimental evaluation that the average number of text characters inspection during the search is linear.

5 Results

We have implemented the SUS pattern matching algorithm and applied on the short reads of the whole human genome $NA18507$ which was sequenced with Illumina HiSeq2500. The machine we have conducted this matching had $32GB$ of memory, LinuxMint 17 operating system. We only used a single CPU of the available four. It took $\approx$ 10 minutes to pass over the 4 million pair-end short reads to detect SUS identifiers, and the software used 16GB memory[2].

Table 1 summarizes what percent of the short reads could be identified with how many SUS signatures. 3.74% of the short reads include 1 to 5 distinct SUS signatures, and $\approx$ 50 % have at least 30 and at most 50 distinct SUS identifiers. Remember that maximum SUS length was set to 30 bases, and the read lengths in this experiment was 101 bases per short read. When one of the two pairs

Table 1. Percentages of the short reads including SUS identifiers on the first 4 million of the pair-end sequences of the NA18507

	% of short-reads identified by X SUS signatures					
unidentified	1–5	6–10	11–20	21–30	31–50	71–100
3.19	3.74	3.42	11.16	28.23	49.73	0.53

[2] It is noteworthy that although there is a lot to do for space usage reduction and execution time enhancement, we decided to apply those changes in final release of the software and did not pay much attention at this point to implement them in this proof–of–concept study.

in a pair-end tuple is identified with an SUS, we assume we can successfully align both pairs since we know that they are in a certain distance. Considering this fact, we have observed that, of the 4 million pair-end reads it is possible to identify ≈ 96 percent directly. This means those reads uniquely map to the area pointed by the SUS identifier they include. The remaining short reads in which no SUS could be located, it is necessary to run the regular k–mer approach to decide where they can map to. These are mostly the reads originating from highly repetitive areas of the genome or highly erroneous readings.

6 Conclusions and Future Works

We have introduced clustering of the short reads according to the SUS signatures extracted from the target species' reference genome. This clustering is supposed to help in two directions so as to improve the compression and alignment. Reordering the reads in the fastq file so that the ones having neighboring SUS signatures are kept close would keep the related items in the same bucket, and hence, better compression might be available. For the alignment, once an SUS is detected inside a short read, its position can be uniquely identified on the reference genome, and thus, more sensitive alignment might be possible with running the SW algorithm with a gretaer insertion–deletion flexibility. The proposed pipeline is shown in Figure 6. Within this study we have build the SUS dictionary for the human reference genome and developed an efficient SUS matching algorithm.

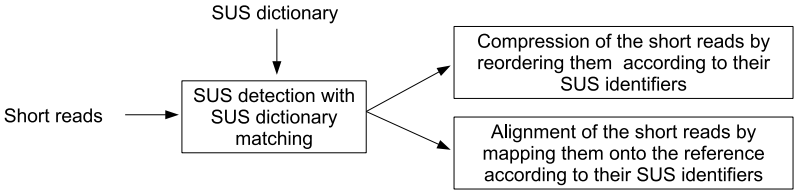

Fig. 4. Proposed sequence analysis pipeline

Next steps of the project will be building the actual alignment and compression blocks and benchmarking each against the current state–of–the–art solutions. Surely, decreasing the computational resource requirement at each step will be an important point, while it is not very much considered at this early proof-of–concept study.

References

1. Alkan, C., Kidd, J.M., Marques-Bonet, T., Aksay, G., Antonacci, F., Hormozdiari, F., Kitzman, J.O., Baker, C., Malig, M., Mutlu, O., et al.: Personalized copy number and segmental duplication maps using next-generation sequencing. Nature Genetics 41(10), 1061–1067 (2009)

2. Berger, B., Peng, J., Singh, M.: Computational solutions for omics data. Nature Reviews Genetics 14(5), 333–346 (2013)
3. Bonfield, J.K., Mahoney, M.V.: Compression of fastq and sam format sequencing data. PloS One 8(3), e59190 (2013)
4. Cox, A.J., Bauer, M.J., Jakobi, T., Rosone, G.: Large-scale compression of genomic sequence databases with the burrows–wheeler transform. Bioinformatics 28(11), 1415–1419 (2012)
5. Deorowicz, S., Grabowski, S.: Compression of dna sequence reads in fastq format. Bioinformatics 27(6), 860–862 (2011)
6. Deorowicz, S., Grabowski, S.: Data compression for sequencing data. Algorithms for Molecular Biology 8(1), 25 (2013)
7. Fonseca, N.A., Rung, J., Brazma, A., Marioni, J.C.: Tools for mapping high-throughput sequencing data. Bioinformatics 28(24), 3169–3177 (2012)
8. Hsi-Yang, F.M., Leinonen, R., Cochrane, G., Birney, E.: Efficient storage of high throughput dna sequencing data using reference-based compression. Genome Research 21(5), 734–740 (2011)
9. Giancarlo, R., Rombo, S.E., Utro, F.: Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies. Briefings in Bioinformatics, bbt088 (2013)
10. Hach, F., Hormozdiari, F., Alkan, C., Hormozdiari, F., Birol, I., Eichler, E.E., Sahinalp, S.C.: mrsfast: A cache-oblivious algorithm for short-read mapping. Nature Methods 7(8), 576–577 (2010)
11. Hach, F., Numanagić, I., Alkan, C., Sahinalp, S.C.: Scalce: Boosting sequence compression algorithms using locally consistent encoding. Bioinformatics 28(23), 3051–3057 (2012)
12. Hach, F., Sarrafi, I., Hormozdiari, F., Alkan, C., Eichler, E.E., Sahinalp, S.C.: mrsfast-ultra: a compact, snp-aware mapper for high performance sequencing applications. Nucleic Acids Research, gku370 (2014)
13. İleri, A.M., Külekci, M.O., Xu, B.: Shortest unique substring query revisited. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 172–181. Springer, Heidelberg (2014)
14. Langmead, B., Salzberg, S.L.: Fast gapped-read alignment with bowtie 2. Nature Methods 9(4), 357–359 (2012)
15. Li, H., Durbin, R.: Fast and accurate long-read alignment with burrows–wheeler transform. Bioinformatics 26(5), 589–595 (2010)
16. Loh, P.-R., Baym, M., Berger, B.: Compressive genomics. Nature Biotechnology 30(7), 627–630 (2012)
17. Pei, J., Wu, W.C.-H., Yeh, M.-Y.: On shortest unique substring queries. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 937–948. IEEE (2013)
18. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
19. Tsuruta, K., Inenaga, S., Bannai, H., Takeda, M.: Shortest Unique Substrings Queries in Optimal Time. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 503–513. Springer, Heidelberg (2014)