An approach for overcoming homopolymer-length sequencing errors in search and alignment is presented. The proposed "homopolymer-length-filter" replaces homopolymers with single characters in both the reads and the reference. In some sequencing technologies, this filter removes many of the machine-induced sequencing errors, but still allows the filtered read to be aligned to the filtered reference. In some sequencing and alignment technologies, this filter can increase the processing speed.

Homopolymer Length Filters

Roy R. Lederman[†]
Technical Report YALEU/DCS/TR-1465
January 8, 2016

[‡] Applied Mathematics Program, Yale University, New Haven CT 06511

Approved for public release: distribution is unlimited.
Keywords: *DNA, alignment, assembly, homopolymer, filter.*

1 Introduction

Read alignment is a step in DNA sequencing procedures, in which one receives short sequences, called reads, and attempts to estimate the location in some reference DNA from which the reads are likely to have come.

There are two sources for differences between the reads and the "best matching" substrings of the reference. The first source is true differences between the sequenced DNA and the reference, we will refer to these as mutations. The second source for differences is errors in the sequencing process, we will refer to these as machine-errors.

Different sequencing machines produce different types of machine-errors. Some machines are known to produce primarily homopolymer-length errors [1, 2]. Homopolymers are consecutive repetitions of a letter in a string. For example, the string $CATAAAG$ has a homopolymer of length 3, composed to the letter A. When a homopolymer-length-error occurs, a wrong number of repetitions is reported. For example, a homopolymer-length-error in the string above may produce the string $CATAAAAG$. Our definition for homopolymer-length error does not include the case of complete deletion of homopolymers or single characters (producing a "homopolymer of length 0") or insertion of characters (turning a "homopolymer of length 0" to a character or a homopolymer). There are extensions for these cases.

Reads that have a large number of machine-errors are difficult to align to the reference even if they contain very few mutations, because the overall number of differences from the reference is large. Furthermore, machines that produce homopolymer-length errors may produce a relatively large number of machines-errors in reads that originate from homopolymer-rich regions of the DNA, making these regions particularly difficult to analyze.

Alignment algorithms use various approaches to find similarities between the reads and the reference. Some consider possible variations which may have occurred at every location and some rely on segments/seeds which are assumed to have very few errors. However, when multiple errors are present in the read, these algorithms may be slow or fail to find the correct result.

Special treatment of homopolymers has been proposed by several authors, for example: [3, 4]. Here, we propose that search algorithms may benefit from essentially ignoring the homopolymer-length rather than using it.

2 Preliminaries

2.1 RLE

Run-length-encoding (RLE) is an alternative representation for strings. RLE translates a homopolymer to a pair of a character ($\{A, C, G, T\}$) and a number ($\{1, 2, 3...\}$). For example, the string $Y = CATAAAG$ is encoded as $(C, 1), (A, 1), (T, 1), (A, 3), (G, 1)$, often written as $1C1A1T3A1G$. We assume that the shortest RLE representation is selected, so we represent AA as $2A$, rather than $1A1A$.

2.2 Flow space

Flow-space is a term used in pyrosequencing and semiconductor sequencing. For the purpose of this discussion, we define the discrete-flow-space and the continuous-flow-space as extensions of RLE.

The discrete-flow-space extends the RLE in two ways: the number in the pair is allowed to be 0, and there is a restriction on the form of the RLE. This restriction is referred to as the "flow sequence." For example, given the flow sequence $F = ACGTACGTACGT...$, the encoded string must take the form:
$(A, \alpha_1), (C, \beta_1), (G, \gamma_1), (T, \tau_1), (A, \alpha_2), (C, \beta_2), (G, \gamma_2), (T, \tau_2).....$
In this example, the string $Y = CATAAAG$ is encoded as $0A1C0G0T1A0C0G1T3A0C1G$.

The continuous-flow-space further extends the definition by allowing the number to be any real number (sometimes non-negative real). The continuous-flow-space is used to represent signal amplitudes in pyrosequencing and semiconductor sequencing.

3 Algorithm

We define the following filter for a string Y, which generates the filtered string $\hat{Y}$: replace each homopolymer in the string Y with a single copy of the letter that is repeated in the homopolymer. For example, $Y = CATAAAG$ is transformed to $\hat{Y} = CATAG$, removing consecutive repetitions of A in the homopolymer except for the first. This transformation encodes Y in RLE, and then removes the numbers from the RLE.

We propose the following algorithm:

1. Filter the reference W to produce $\hat{W}$.
2. Filter all reads $\{Y^j\}_j$ to produce the filtered versions $\{\hat{Y^j}\}_j$.
3. Align $\{\hat{Y^j}\}_j$ to $\hat{W}$, possibly keeping some number of possible approximate candidates.

4. Convert the alignment coordinates back to the original coordinates and restore the original reads $\{Y^j\}_j$.

5. Refine the alignment by examining the restored reads.

The search/alignment step may use any search/alignment algorithm. In our experiments (see below) we used both a permutations-based aligner with integrated filters, and "wrapped" versions of other aligners.

In this basic form, the filter makes the search algorithm unaware of homopolymer-lengths, until the refinement is performed. Therefore, homopolymer-length-errors do not affect the algorithm. Indeed, this filter removes important information that could be used for alignment, but it often leaves enough information to allow the algorithm to generate a list of candidates. Once a list of candidates is produced, the original homopolymer-lengths can be used to refine the results.

The filter creates shorter strings, which are often processed much faster and much more efficiently than long strings. Indeed, simple truncation of the strings also produces shorter strings, but the filtered strings may have fewer errors and contain information from the entire length of the original string. Since many algorithms process shorter strings much faster, this property of the filter makes it useful even when homopolymer-length-errors are not the dominant variation.

We observe that mutations and machine-induced errors that are not homopolymer-length-errors may produce "mismatches" and "indels" in the filtered reads. However, the number of these errors is similar to the number of errors in the original reads. Reads with very large numbers of homopolymers, which may have many homopolymer-length-errors are likely to have a smaller number of variations after the filter is applied.

3.1 Generalization and extensions

In some sequencing machines, homopolymer-length errors are more likely to occur in relatively long homopolymers. It is therefore possible to truncate homopolymers at some length, rather than replace them with a single character. This is a Homopolymer-length-truncation that restricts the length of each homopolymer, as opposed to a string-truncation, which restricts the length of the entire string and cuts the end of the string. For example, if we truncate homopolymers at 2, both $Y1 = CATAAAG$ and $Y2 = CATAAAAAAG$ are transformed to $\hat{Y} = CATAAG$.

It is also possible to encode homopolymers as new letters of the alphabet. To do this, we extend the alphabet from $\{A, C, G, T\}$ to $\{A^1, A^2....C^1, C^2, ...G^1, G^2....T^1, T^2....\}$. For example: $Y = CATAAAG$ is encoded as $\hat{Y} = C^1A^1T^1A^3G^1$. In this case homopolymer errors no longer appear as indels, but rather as mismatches, which some search algorithms handle better. Homopolymer-length-truncation is possible here as well.

homopolymers can also be encoded in a discrete-flow-space or a virtual-flow-space. A flow-space representation of the string uses the machine's true flow sequence, which was used to generate the read. This representation can be extended in the ways described above. This representation is useful, when the machine misses (or adds) a character, because in the flow-space the missing character appears as a mismatch, rather than an indel. This observation about the flow-space allows us to define the virtual-flow-space, where some other flow-sequence is used. These virtual-flow-sequences may "mask" not only machine errors, but also some mutations. Several virtual-flow-sequences and non-repetitive virtual-flow-sequences may be used, possibly on overlapping segments of the reference rather than on the entire reference as a single string.

The basic idea to remove homopolymer-length information from reads does not imply that homopolymer-length information cannot be used to define some search parameters or reintroduced to support some aligner decisions. For example, a search algorithm can use information about long homopolymers to conclude that it is unlikely that the entire homopolymer was deleted. It is also possible to calculate quality measures for each character in the filtered strings (like the quality measures which are often added to regular reads). One can also store the original lengths of segments, which can help aligners distinguish between filtered reads: the strings $AATGC$ and $ATTTTTTGGGC$ are both represented by the filtered string $ATGC$, but given a filtered read $ATGC$, which had the original length 10, one can estimate that the read is related to the first string rather than the later string.

3.2 Additional applications

While we discuss this algorithm in the context of aligning/mapping reads to a reference, the algorithm can be extended to other search and string-comparison/string-alignment applications. In particular, it can be used for similarity/overlap search in libraries of reads, for error correction etc.

An additional application is assembly. This algorithm can be used to assemble filtered "skeletons" of genomes (or more generally, of an assembly graphs). These "skeletons" can then be "inflated" by reintegrating information from the original reads to estimate the correct homopolymer-lengths. Alternatively, the algorithm can be integrated into specific search parts in the regular construction of graphs.

Homopolymer-length machine-errors pose a significant problem in assembly based on pyrosequencing and semiconductor sequencing. However, in the "filtered space," some of these machines have a very low error rate. Therefore, these machines may have advantages over other sequencing machines, at least in creating the "filtered skeletons."

4 Implementation and results

We implemented a preliminary prototype of the algorithm as a modified edition of a basic permutations-based mapper prototype which we introduced in [10, 11], using the pigeonhole principle to align filtered reads in the presence of indels.

We also implemented "Earplugs" to demonstrate how existing aligners can be "shielded" from "homopolymer-length-noise." "Earplugs" produces a filtered version of the reference and of the reads. The filtered reference and the filtered reads are then indexed and aligned as if they were any standard reference and reads, using standard alignment software. At the final step, "Earplugs" converts the output "SAM" files to "reinflated SAM" files, transforming back from "filtered coordinates" to the original reference coordinates.

Both the modified permutations-based aligner and "Earplugs" are early prototypes that demonstrate the algorithm. The current versions are not yet available for standard alignment pipelines. "Earplugs" will be available at http://alignment.commons.yale.edu.

We tested the permutations-based mapper and several other aligners: BWA[5], BWASW[6], BOWTIE2[7] and CUSHAW2[8], with and without "Earplugs shielding." All the software packages were used in single thread mode. CUSHAW2 was used in "have ssse3 = 0" mode, CUSHAW2 may be significantly faster when used on systems that support SSE4 extensions in addition to the SSE2 extensions that it used on our computers. Our implementation does not use processor extensions. BOWTIE2, BWASW and CUSHAW2 are more memory-efficient than this implementation of a permutations-based aligner. We used reads generated by Mason[9] from a human reference genome.

The search times are the search times reported by the software, or the overall run time (including the initial filtering) in the case of BWA. We experimented with several different parameters. In some setups we configured the aligners to report several possible alignments in order to reduce the effect of the internal criteria and filters used by each software package.

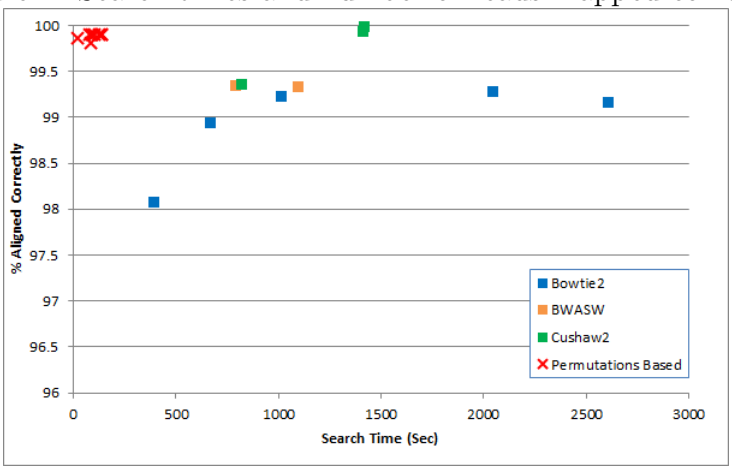
The results for BWA are not presented in the figures because BWA works better with shorter reads. However, "wrapping" BWA with "Earplugs" often allows BWA to align many more reads. For example, in an experiment with shorter reads (200bp), BWA aligned 63.6% of the reads correctly in 1096 seconds, whereas the same mode of BWA with "Earplugs" aligned 83.7% in 282 seconds. In another experiment, where we used a simpler model to simulate a high rate of homopolymer-length errors in homopolymer-rich regions of the genome, BWA aligned 2.5% of the reads in 206 seconds, whereas BWA with "Earplugs" aligned 98.6% in 45 seconds. A node with E5620 CPUs was used in these experiments.

Preliminary results indicate that the permutations-based algorithm with the integrated filter is significantly faster than the other software packages and that it produces comparable or significantly better results in many cases. "Earplugs" accelerates existing software considerably. For long reads, there is usually little reduction in accuracy, and

there is often an increase in accuracy. Since the processing time of many algorithms depends on the length of the reads, some of the speed increase can be attributed to the shorter reads.

The results obtained using a permutation-based aligner with a built-in filter indicate that integrating the filter into alignment algorithms may make these algorithms considerably faster and more accurate.

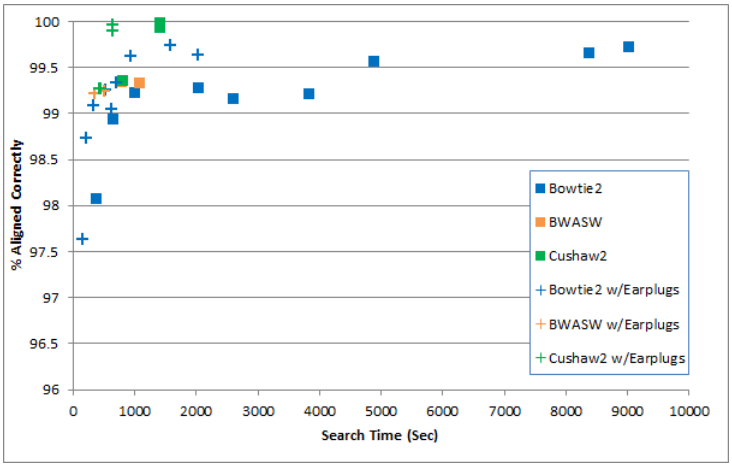
Figure 1: Search times and number of reads mapped correctly



10^5 reads, 500bp long, human genome

AMD Athlon II X2 250 CPU, 32GB RAM

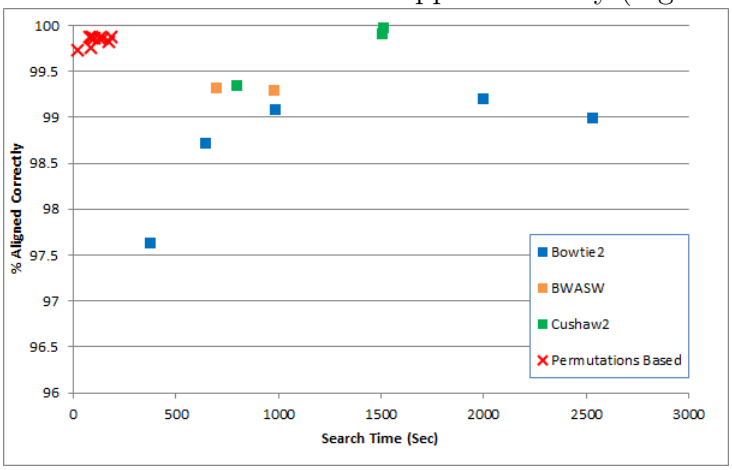
Figure 2: Search times and number of reads mapped correctly: aligners with and without "Earplugs"



10^5 reads, 500bp long, human genome

AMD Athlon II X2 250 CPU, 32GB RAM

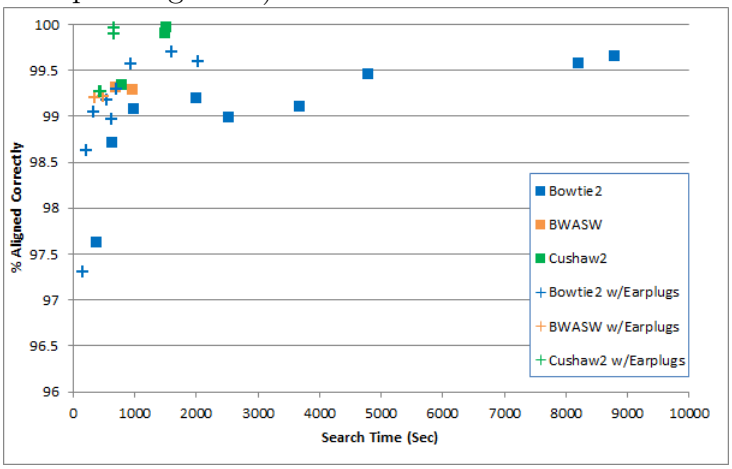
Figure 3: Search times and number of reads mapped correctly (higher sequencing noise)



10^5 reads, 500bp long, human genome, Mason parameter $k = 0.2$

AMD Athlon II X2 250 CPU, 32GB RAM

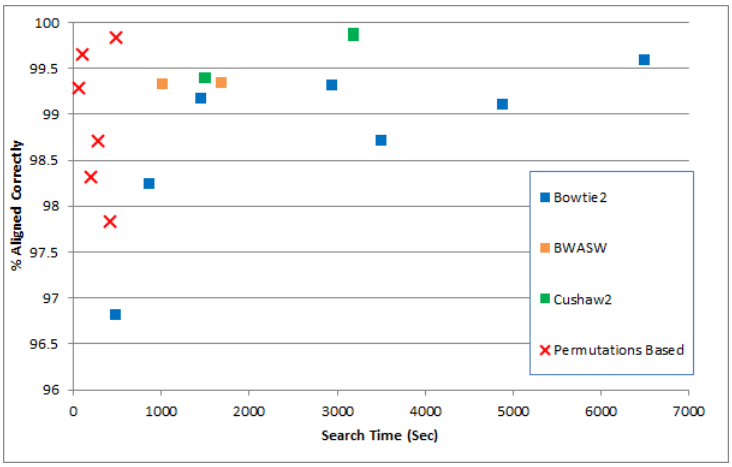
Figure 4: Search times and number of reads mapped correctly: aligners with and without "Earplugs" (higher sequencing noise)



10^5 reads, 500bp long, human genome, Mason parameter $k = 0.2$

AMD Athlon II X2 250 CPU, 32GB RAM

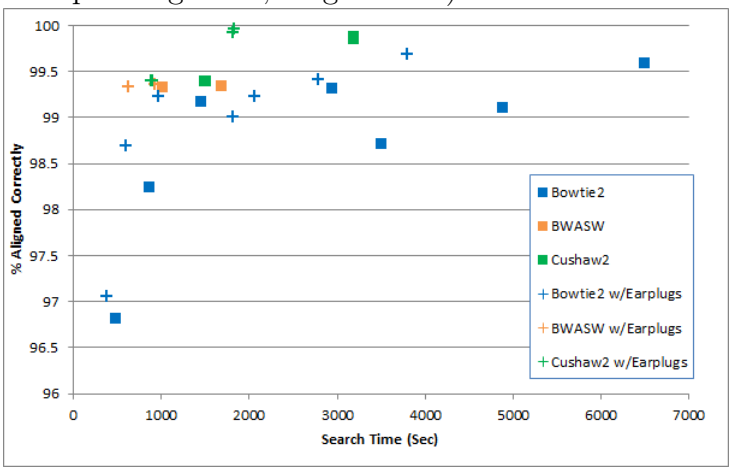
Figure 5: Search times and number of reads mapped correctly (higher sequencing noise, longer reads)



10^5 reads, 750bp long, human genome, Mason parameter $k = 0.3$

E5620 CPUs, 48GB RAM

Figure 6: Search times and number of reads mapped correctly: aligners with and without "Earplugs" (higher sequencing noise, longer reads)



10^5 reads, 750bp long, human genome, Mason parameter $k = 0.3$

E5620 CPUs, 48GB RAM

5 Conclusions

An approach for the alignment of reads in the presence of homopolymer-length-errors has been presented. The proposed approach is based on removing "noise" from reads by "ignoring" parts of the information in the reads.

It has been demonstrated that the remaining information is often sufficient for alignment, and in particular for alignment of relatively long reads. Subsequent refinements of the alignment results may use the original information. The method can be used in related applications, such as assembly.

References

[1] Gilles, A., Meglcz, E., Pech, N., Ferreira, S., Malausa, T., & Martin, J.F. *Accuracy and quality assessment of 454 GS-FLX Titanium pyrosequencing.* BMC Genomics 2011.

[2] Loman, N.J., Misra, R.V., Dallman, T.J., Constantinidou, C., Gharbia, S.E., Wain J., & Pallen M.J., *Performance comparison of benchtop high-throughput sequencing platforms.* Nature Biotechnology 30, 434439. (2012)

[3] Vacic V, Jin H, Zhu JK, Lonardi S. *A probabilistic method for small RNA flowgram matching.* Pac Symp Biocomput. 2008:75-86. (2008)

[4] Lysholm, F., Andersson, B. & Persson, B. *FAAST: Flow-space Assisted Alignment Search Tool.* BMC Bioinformatics 12: 293 (2011)

[5] Li, H. & Durbin, R. *Fast and accurate short read alignment with Burrows-Wheeler transform.* Bioinformatics 25, 1754 -1760 (2009).

[6] Li H. & Durbin R. *Fast and accurate long-read alignment with Burrows-Wheeler transform.* Bioinformatics. (2010) [PMID: 20080505]

[7] Langmead, B. & Salzberg, S. L. *Fast gapped-read alignment with Bowtie 2.* Nat Meth 9, 357-359 (2012).

[8] Liu, Y & Schmidt, B: *"Long read alignment based on maximal exact match seeds".* Bioinformatics, 2012, 28(18): i318-324

[9] Holtgrewe, M. *Mason a read simulator for second generation sequencing data.* Technical Report TR-B-10-06, Institut fr Mathematik und Informatik, Freie Universitt Berlin. (2010)

[10] Lederman, Roy R. *A random-permutations-based approach to fast read alignment.* BMC bioinformatics 14, no. Suppl 5 (2013): S8.

[11] Lederman, Roy R. *A nearest neighbors algorithm for strings.* Technical Report 1453, Yale CS (2012)

Revised January 8, 2016 (no change in content).