NanoBLASTer: Fast Alignment and Characterization of Oxford Nanopore Single Molecule Sequencing Reads

Mohammad Ruhul Amin[1], Steven Skiena[1], and Michael C. Schatz[2⋆]

[1]Stony Brook University, NY 11794, USA. [1]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, NY 11724, USA
1.{moamin,skiena}@cs.stonybrook.edu, 2.{schatz}@cshl.edu

Abstract. The quality of the Oxford Nanopore's long DNA sequence reads has been, to date, lower than other technologies, causing great interest to develop new algorithms that can make use of the data. So far, alignment methods including LAST, BLAST, BWA-MEM and GraphMap have been used to analyze these sequences. However, each of these tools has significant challenges to use with these data: LAST and BLAST require considerable processing time for high sensitivity, BWA-MEM has the smallest average alignment length, and GraphMap aligns many random strings with moderate accuracy. To address these challenges we developed a new read aligner called NanoBLASTer specifically designed for long nanopore reads. In experiments resequencing the well-studied *S. cerevisiae* (yeast) and *Escherichia coli* (*E. coli*) genomes, we show that our algorithm produces longer alignments with higher overall sensitivity than LAST, BLAST and BWA-MEM. We also show that the runtime of NanoBLASTer is faster than GraphMap, BLAST and BWA-MEM.

Keywords: Oxford Nanopore Technology, Single Molecule Sequencing, Sequence Alignment Algorithms

1 Introduction

Since the earliest methods for sequencing DNA were developed in the mid-1970s [1], several new technologies have been developed to reduce costs and improve throughput [2]. During this time, the worldwide capacity for sequencing DNA has nearly doubled every year, with thousands of projects underway to study the genetics of species across the tree of life [3]. One of the most recent technologies is the MinION sequencer produced by Oxford Nanopore that uses nanopore-based technology to sequence DNA. The instrument sequences DNA by monitoring the ionic current as single-stranded DNA molecules are threaded through a protein pore embedded into a highly sensitive electrically insulated membrane[4]. The minute fluctuations in current are dependent on the particular sequence of DNA

⋆ Corresponding Author

passing through the pore at any time. As such, a base-calling algorithm can use those measurements to infer the underlying sequence of the template molecule.

Using nanopore-technology, the MinION is able to sequence long fragments of DNA, including some over 100kbp [5] [6]. Furthermore, because it sequences individual molecules of DNA without amplification, the preparation for sequencing is simplified, and many of the sources of errors are eliminated compared to older "2nd generation" sequencing technologies, such as from Illumina or Ion Torrent. However, one of the main challenges of the technology is the underlying accuracy of the sequencing has been, to date, substantially lower than the older technologies, and currently has a raw read alignment identity averaging around 70% (also called "1D sequencing"). To help combat this challenge, the instrument has an enhanced mode of sequencing using the consensus of both the template and complement strands of the DNA molecule to produce "2D reads" that helps improve the alignment identity as much as 90%. However, this mode is available for only a relatively small fraction ($\sim 20\%$) of the reads.

In this paper, we introduce a novel open-source read alignment tool, called NanoBLASTer, that includes several novel enhancements to maintain high sensitivity and high performance in the presence of high error rates that would otherwise limit the performance of seeds that could be used. Using NanoBLASTer, we analyze nanopore-based resequencing data of the well-studied *S. cerevisiae* (yeast) and *Escherichia coli* (*E. coli*) genomes and find NanoBLASTer alignments are on average longer than that of LAST, BLAST and BWA-MEM. We also observe that NanoBLASTer is faster than BLAST, BWA-MEM and GraphMap. Furthermore, we find that specificity of NanoBLASTer to be many greater than GraphMap. We also report on the alignment of nanopore sequencing data for human using hg19 human genome as reference by NanoBLASTer to show that our approach can scale to larger genome size. All source code and data sets used in analysis are available at *http://ruhulsbu.github.io/NanoBLASTer/*.

Methods

Most existing DNA read alignment algorithms align a read to a reference sequence in two steps using an approach called "seed-and-extend" [11]. In the first step the aligners find high-identity short matches between the sequences called "seeds", and in the second step they extend the alignment from these seeds into lower identity regions using dynamic programming approaches. There are many variations of this approach, and tools like BLAST use fixed-length exact matches as seeds, while others like LAST use "spaced seeds" that allow for small numbers of differences between the sequences. The extension of the alignment is most commonly computed with a modified Smith-Waterman dynamic program [12], that generally require $O(n^2)$ time. Although as a performance heuristic, the reported alignments from these tools may be suboptimal unless very short seeds are applied. However, naively using short seeds can have catastrophic impact on the runtime performance, as exponentially more seeds will occur by chance as the seeds become shorter.

To overcome these challenges, we have developed a new alignment tool called NanoBLASTer specifically designed for aligning long nanopore reads (**Algorithm 1**). Inspired by BLAST, it also uses fixed size exact matching seeds followed by DP-based extension. However, because of the high error rate of the nanopore sequencing instruments (approximately 10% to 40% base error rate) [6] [13], the seeds that must be used are extremely short and provide relatively little specificity. Consequently, before extension, NanoBLASTer clusters neighboring seeds using a fast one-dimensional clustering technique to form longer high scoring segments we call alignment ANCHORs. The ANCHORs are then scored by length and similarity, and the top scoring ANCHORs are extended into full length alignments using a block-wise banded dynamic programming algorithm.

Algorithm 1 NanoBLASTer Overview

1: Create an inverted K-mer index of the reference genome
2: Identify and cluster neighboring K-mers seeds
3: Identify and score candidate ANCHORs
4: Extend the ANCHORs into complete alignments
5: Report high quality valid alignments in SAM format

Create Inverted *K*-mer Index of the Reference Genome

NanoBLASTer begins by constructing an inverted K-mer index to record the locations of all K-mers present in the reference genome (**Figure 1(a)**). This makes it possible in later stages of the algorithm to rapidly search for the presence of a particular read K-mer in constant or near-constant time. The reference genome will often be composed of multiple sequences, such as separate chromosomes or separate contigs that we denote Ref_i. We create two inverted K-mer indexes: one for all the Ref_i in the forward direction and another for the reverse complements of all Ref_i. Both of these indexes are stored using two lookup tables of size 4^K . For a reference genome of total length N, there are total $(N-K+1)$ K-mers. Many of these K-mers occur more than once in the reference Ref_i, so the K-mer index is keyed by the list of distinct K-mers that occur in reference Ref_i, and records the list of all the locations of that K-mer. By default, the size of K-mer used by NanoBLASTer is $11bp$. If we increase the size of K-mer then the seeds will be more specific in the genome, although the overall mapping rate may suffer given the high error rate in the nanopore reads. If we decrease the size of K-mer then NanoBLASTer will be slightly more sensitive but will dramatically increase the overall run-time since more spurious seeds will be detected.

Identify and Cluster Neighboring Seeds

Using the K-mer index, each read is scanned to compute exact matching K-mer seeds. The seeds are recorded as a **seed-tuple** consisting of the position of

K-mer in the reference and in the read sequence: ($kmer\_ref\_loc$, $kmer\_read\_loc$). Given that very short seeds must be used for the noisy reads that provide little specificity on their own, the next phase of the algorithm is to cluster the seed matches that are mutually **compatible** into K-mer neighborhoods. Two seed-tuples $seed_i$ and $seed_j$ are *compatible* if and only if the following two conditions are satisfied:

1. The read and reference positions are in ascending order:
   $kmer\_read\_loc_j \geq kmer\_read\_loc_i$ and
   $kmer\_ref\_loc_j \geq kmer\_ref\_loc_i$
2. The distance between two seed-tuples is **balanced**, meaning the ratio of the distance between the seeds in the reference and the distance in the read sequences are within 30% of each other, representing the maximum deviation

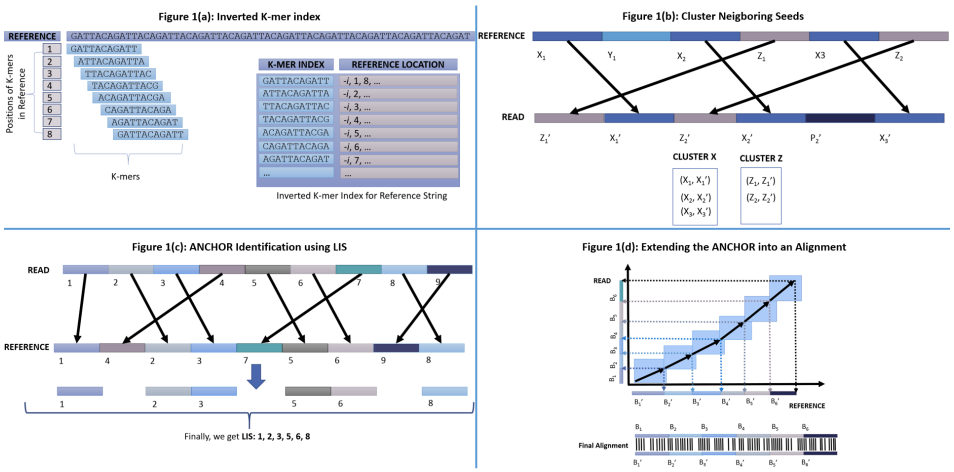

Fig. 1. NanoBLASter Algorithm: (a) We first find the locations of each K-mer in the reference Ref_i. Then we create the index on K-mer using a lookup table that stores a pointer to the list of reference locations for the respective K-mer. For a particular K-mer of reference Ref_i, we first insert $-i$ in the list of locations to indicate the beginning of that K-mer locations list, and then insert the K-mer locations in due order. (b) Clustering is done in two steps. First we find the K-mers of read in the reference and make a list of K-mer pairs ($kmer\_ref\_loc, kmer\_read\_loc$): [($X_1$, X_1'), (X_2, X_2'), (Z_1, Z_1'), (X_3, X_3'), (Z_2, Z_2')]. This list is sorted in terms of $kmer\_ref\_loc$. In the second step we find **1-D** cluster of the K-mers by neighborhood, and output cluster X and cluster Z. (c) Applying LIS to evaluate candidate ANCHORs. The LIS algorithm selects the longest subsequence of W-mers that occur in the same order in the read and reference sequences. (d) The extension starts at the position of an ANCHOR using the first B bases from both the read and reference (shown here as $B1$ and $B1'$). It then uses a banded global alignment algorithm to calculate the similarity matrix of the first block. Starting at the best similarity score for the first block, it then repeatedly attempts to extend the alignment using blocks of the next B bases.

observed in real alignments:

$$0.70 \leq \frac{(kmer\_read\_loc_{x+1} - kmer\_read\_loc_x)}{(kmer\_ref\_loc_{x+1} - kmer\_ref\_loc_x)} \leq 1.30$$

Algorithm 2 NanoBLASTER Seed Clustering

```
Initialize kmer_pair_list with seed-tuples pairs (kmer_ref_loc, kmer_read_loc)
Sort kmer_pair_list by reference coordinate
while kmer_pair_list has unprocessed items do
    Initialize a new cluster C with the first available tuple T and mark as processed
    ▷ T is also the last element of the cluster C
    for the next unprocessed entry E after T to the end of the kmer_pair_list do
        ▷ Determine max_reference_delta using read_len (length of current read),
        kmer_read_loc of tuple T and the constant SEED_TUPLE_DIST (as defined
        in configuration file of NanoBLASTer program; by default it is 10kb long)
        max_reference_delta :=
        min(1.3 * (read_len − T.kmer_read_loc), SEED_TUPLE_DIST)
        if the reference distance to next unprocessed entry is larger than
        max_reference_delta then
            ▷ Short-circuit the clustering when the distance is too far
            End for loop and advance while loop to the next available unprocessed entry
        end if
        if E has the same reference position or same read position as T then
            ▷ Filter tandem repeats to reduce the size and number of clusters
            Mark E as processed on the kmer_pair_list, but do not add to C
        else if E is compatible with the seed-tuple T then
            Add E as the new last element of C, and mark as processed in the
            kmer_pair_list.
            Update the last element T using E
        else
            Keep E in the kmer_pair_list for later consideration
        end if
    end for
    Add each cluster C to a vector for further processing
end while
```

Clusters are formed using an iterative 1-dimensional clustering of the seed-tuples sorted by reference K-mer positions, allowing for seeds to be repetitive in either the reference genome or the read itself. (**Algorithm 2, Figure 1(b)**). The algorithm makes use of the "reference" or "read" distance, meaning the distance between the location of the seeds in the reference or read sequence, respectively. At the end of this step, all the valid clusters are sorted in terms of their size in descending order.

Candidate ANCHOR Identification

To find an ANCHOR, we extract a substring of length S ($S = 40bp$ by default) from both reference Ref_i and read r_j starting at the position indicated by the first seed-tuple in the cluster. By construction, the candidate ANCHOR begins with at least one exactly matching K-mer (the K-mer used to initialize the cluster). Other than this K-mer, the algorithm looks for additional shorter exact matches, denoted as W-mers ($W \geq 5$ by default), that can be used to approximate the similarity of the S-bp substrings. The algorithm uses a form of the longest-increasing-subsequence (LIS) dynamic programming algorithm to select the mutually consistent W-mers to enforce the order of W-mers in the reference is the same as their occurrences in the read and that the distance between the W-mers is small (**Algorithm 3, Figure 1(c)**). After the set of W-mers has been computed, the approximate similarity score of the candidate ANCHOR is evaluated using the following formula:

$$\frac{\text{Number of bases in } K\text{-mer and } W\text{-mers}}{S} \geq 0.60$$

Algorithm 3 Candidate Anchor Identification

1: From read r_j, find the locations of all W-mers ($W >= 5$) within the S bases of its seed K-mer ($kmer\_read\_loc$ in a seed-tuple) and make a list of W-mer locations in due order.
2: For each reference W-mer (within S bases of its respective $kmer\_ref\_loc$) in order of their occurrence, find the nearest read W-mer such that their relative positions are very close (< 5 bp apart); otherwise, for larger distance the chances of finding a random ANCHOR increase. We make a list of such read W-mer locations in due order of their occurrence in reference. Each read W-mer location is used only once.
3: Find the longest increasing subsequence (LIS) of the W-mer location list of step 2.
4: Count the total number of bases involved in the LIS from step (3) and compute the percentage of identity match for them. If it is at least 60% then the respective region is considered as an ANCHOR.
5: If we do not find an ANCHOR in rightward direction, then try in the leftward direction. This makes the algorithm more robust if there is a cluster of errors immediately before or after the K-mer seed.

Using this approach, an ANCHOR may have a number of insertions, deletions, and substitutions; yet it ensures highly similar segment pairs by matching W-mers in the same order in both read and reference. The number of ANCHORs that we discover for a read depends on the size of ANCHOR, W-mer, and percentage of identity match. We observed that reducing the substring length of the ANCHORs ($S < 40$), W-mer length ($W < 5$), or percentage of identity match ($< 60\%$) lead to many false ANCHORs which result in random read alignments. As an optimization, the algorithm skips evaluating all K-mer locations that are within B bases of a known ANCHOR as these will be automatically explored

during the ANCHOR alignment extension. Here, B is the block size for banded sequence alignment algorithm, which we discuss in the next step (default size of B is $400bp$).

Algorithm 4 Blockwise Banded Alignment

▷ Assume there is an ANCHOR A between reference sequence Ref_i and read r_j
$read\_pos \leftarrow A.read\_pos$
$ref\_pos \leftarrow A.ref\_pos$
$previous\_identity \leftarrow 55$
while not at the end of the read **or** the end of the reference **do**
    ▷ Compute an alignment block of size B at the specified locations
    $S_{ref} \leftarrow substring(Ref_i, ref\_pos, B)$
    $S_{read} \leftarrow substring(r_j, read\_pos, B)$
    Compute the similarity $matrix$ of S_{ref} and S_{read} using a banded global alignment algorithm
    Find the cell $matrix[r, c]$ with the maximum similarity score for the r bases from Ref_i and c bases from the read r_j, such that $0.75 \leq \frac{r}{c} \leq 1.25$.
    Advance ref_pos and $read\_pos$ by r and c
    Update the overall similarity of the alignment computed so far, and terminate if it drops below 55%
    To reduce the probability of random alignment, terminate extension when either $r < \frac{1}{4}B$ **or** $c < \frac{1}{4}B$
    $identity\_difference \leftarrow previous\_identity - current\_identity$
    if $identity\_difference > 20$ and $current\_identity < 0.55$ **then**
        ▷ This condition helps to generate the high quality alignments and prevent surrounding flanking regions to be aligned.
        Terminate the extension
    end if
    $previous\_identity \leftarrow current\_identity$
end while
Concatenate the alignments of the blocks together

Extending the ANCHOR into an Alignment

All valid ANCHORs identified are then extended leftward and rightward into complete alignments using a blockwise banded dynamic programming algorithm (**Algorithm 4, Figure 1(d)**). The extension begins from the first ANCHOR of the list, prioritized by the size of the cluster that contained it. If the alignment from one ANCHOR extends past the coordinates of another, then the later is not considered for extension. The default block size of the alignments, B is 400 for NanoBLASTer since the runtime of alignment increases quadratically with the size of B. As the alignment identity of the most noisy nanopore reads is approximately 55%, the band size is set to be at least half of the block size. This configuration helps NanoBLASTer from prematurely ending the alignment if it encounters a relatively long region of errors. On the other hand, NanoBLASTer

terminates aligning a read when it finds a sharp decline of the percentage of alignment identity from the previous block B_{i-1} to current block B_i. This condition helps NanoBLASTer to avoid aligning random or very low identity regions around a good sequence.

Reporting the Alignment

If the sequence similarity of the alignment is greater than the user specified minimum (55% by default) and covers at least the user specified length (4% of the read length by default), then it is considered a valid alignment and is reported in SAM format [14]. NanoBLASTer first processes the top L clusters (based on the size of the clusters in descending order) to search for valid ANCHORs and alignments. If it successfully finds a valid alignment in this set, it continues with the next L clusters to search for additional alignments. The search terminates when L consecutive clusters fail to generate a valid alignment.

Results

In this section, we examine the error and read characteristics of Oxford Nanopore sequencing reads by aligning and studying two MinION sequencing datasets of the *S. cerevisiae* W303 (yeast) and *E. coli* K-12 MG1655 (*E. coli*) genomes, respectively. The full details of how the *S. cerevisiae* and *E. coli* samples were sequenced are described in the manuscripts of Goodwin et al. (2015) and Quick et al. (2014) respectively [15] [6]. For the *E. coli* dataset, we only considered R7 reads for our analysis, as in GigaDB only the R7 reads had been made available in fasta format. We also include an in depth comparison to several existing alignment tools including LAST, BLAST, BWA-MEM and GraphMap. At the end of this section, we present the memory consumption of NanoBLASTer for aligning both the datasets as well as for human nanopore sequence data.

Characteristics of Oxford Nanopore Reads

In this section, we show the characteristics of nanopore reads and their alignments reported by NanoBLASTer using two datasets of yeast and *E. coli*:

1. *S. cerevisiae* W303 (yeast) dataset: There are total 410,344 reads sequenced with the MinION available in the yeast dataset providing an average coverage of 196x for the yeast reference genome (12.1Mbp). Of these, there are 78,225 "2D", 230,606 "1D template" and 101,513 "1D complement" reads with an average read length of 7.04kbp, 5.55kbp and 5.53kbp respectively.
2. *E. coli* K-12 MG1655 (*E. coli*) dataset: This dataset has total 111,128 sequences, providing 146x coverage for the *E. coli* reference genome (4.6Mbp). There are 24,221 "2D", 58,450 "1D template" and 28,457 "1D complement" reads with an average read length of 6.56kbp, 6.08kbp and 5.41kbp.

From these two datasets, we excluded 12,117 reads of yeast dataset and 922 reads of *E. coli* dataset from analysis because they were less than 100bp long. Using NanoBLASTer parameters "C10", that balance runtime and sensitivity, NanoBLASTer aligned 192,613 reads (46.94%) of yeast dataset and 68,650 reads (61.78%) of *E. coli* dataset. Details of the alignment statistics are shown in (**Table 1**). We observed that reducing the minimum acceptable similarity of 55% introduced many false alignments where the read sequence and the reverse of the read sequence (effectively a random string) would often produce alignments of this quality by chance. Interestingly, the quality does not seem to be position specific across the length of the read, and the per base error rate is on average consistent across the length of a read for insertions, deletions, or substitutions. Finally, we analyzed the coverage of the alignments across both the yeast and *E. coli* genome to evaluate any major biases. The overall distribution generally resembled the expected Poisson distribution with lambda of $\sim 100.44x$ coverage for yeast and $\sim 98.13x$ coverage for *E. coli*

Table 1. NanoBLASTer alignment statistics for yeast and *E. coli* data set.

Dataset	2D reads aligned with identity (%)	1D template reads aligned with identity (%)	1D complement reads aligned with identity (%)
Yeast	53,756 reads with 71.70% identity	91,775 reads with 63.05% identity	47,082 reads with 61.63% identity
E. coli	20,249 reads with 70.64% identity	31,198 reads with 63.89% identity	17,203 reads with 59.73% identity

Comparison Between NanoBLASTer and Other Alignment Tools

To evaluate the performance of NanoBLASTer, we also aligned the yeast and *E. coli* reads using the LAST, BLAST, BWA-MEM and GraphMap algorithms on an 8 core Intel(R) Xeon(R) CPU X3450 @2.67GHz server with 24 GB of RAM. For each of these tools, we selected the published configurations used for aligning nanopore reads. We also used only one thread for all the tools to show fair runtime comparisons, since not all are multithreaded but is an embarrassingly parallel problem. textbfTable 2 shows that with these configurations, LAST has the fastest runtime, while NanoBLASTer has the second best runtime.

Alignment Length To differentiate between the performance of NanoBLASTer and all other tools, we further analyzed the alignment lengths reported by these tools in **Figure 2** for both datasets. We focus on the single longest alignment for each read, as we expect just one true alignment per read in this resequencing experiment. According to the experimental results, for both datasets, around one third of the LAST or BLAST alignments are completely "contained" by NanoBLASTer alignments, meaning the NanoBLASTer alignment is at essentially the same position as the LAST or BLAST alignment

Table 2. Here we show the runtime and number of alignments reported by each alignment tool for the yeast and *E.coli* dataset. We also report the runtime parameters used for each alignment tool.

Tool	Dataset	Runtime (min)	Number of alignments in SAM file	Number of reads aligned (% of total dataset)	Runtime parameters and Version
LAST	Yeast	333	1,500,341	195,103 (47.55%)	-q1 -a1 -b1; version:
	E. coli	71	151,461	73,380 (66.03%)	lastal 590
NanoBLASTer	Yeast	429	530,202	192,613 (46.94%)	-C10 ("fast" mode);
	E. coli	91	126,299	68,650 (61.78%)	version: 0.16
GraphMap	Yeast	617	361,775	359,542 (87.62%)	default mode;
	E. coli	133	103,534	103,534 (93.17%)	version: 0.21
BWA-MEM	Yeast	882	298,638	198,918 (48.48%)	-x ont2d; version:
	E. coli	179	93,113	70,903 (63.80%)	bwa-0.7.12-r1044
BLAST	Yeast	1,218	2,727,778	191,127 (46.58%)	-reward 5 -penalty -4
	E. coli	96	120,793	68,039 (61.23%)	-gapopen 8 -gapextend 6 -evalue 1e-10; version: blast-2.2.31+v

although the NanoBLASTer alignment is longer. In contrast, LAST and BLAST contains only around 17% and 10% of NanoBLASTer alignments respectively. Also, more than one third of the LAST or BLAST alignments "overlap" with the NanoBLASTer alignments, meaning that the reference coordinates of the reported alignments overlap. Similarly, nearly one fourth of the BWA-MEM alignments are completely contained by NanoBLASTer, and nearly half of the BWA-MEM and NanoBLASTer alignments overlap in terms of their reference coordinates. Whereas, GraphMap contains more than one third of NanoBLASTer alignments. We represent this analysis in a scatter plot comparing the alignment lengths of NanoBLASTer to that of other tools. In **Figure 2**, the dominance is most pronounced in cases where the NanoBLASTer alignments contain the alignments of other tool, but is also present for overlapping or "outside" alignments (meaning the reference coordinates of the alignments do not intersect).

Alignment Identity We show the boxplot of alignment identity for each of the tools in **Figure 13**. From this figure, we see that for both datasets, GraphMap has the lowest average and the greatest standard deviation in identity. We believe this is due to a number of false alignments reported by the algorithm (see below). NanoBLASTer shows moderately lower average alignment identity than that of LAST, BLAST and BWA-MEM. BLAST and LAST shows similar distributions in alignment identity. BWA-MEM shows highest average alignment identity, though it has many outliers.

Random String Alignment As the GraphMap produced alignments for around 90% of the reads for both the yeast and *E. coli* datasets, we further analyzed its alignments. The most frequent alignment identity was observed around 50%

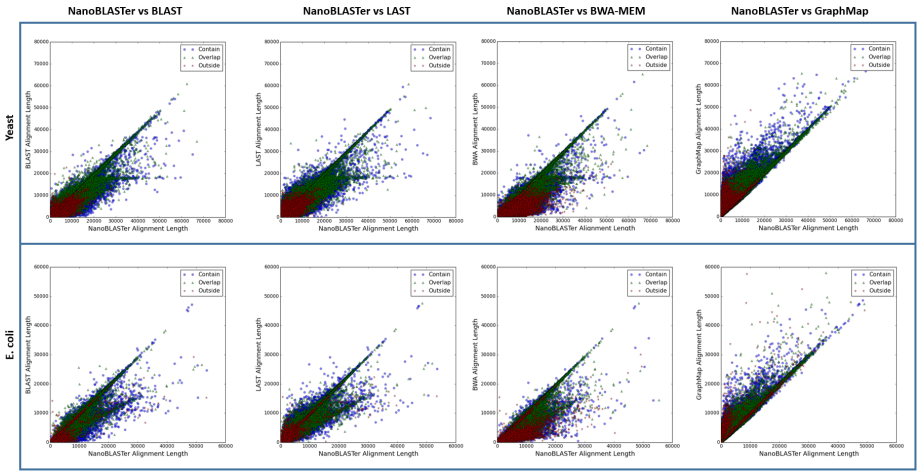

Fig. 2. Scatter plot comparing the alignment lengths of NanoBLASTer to that of BLAST, LAST, BWA-MEM and GraphMap (using longest single alignment for each read). Hree the alignments are represented in three different sets: "contained", "overlapped" and "outside" in terms of comparing the reference mapping location and alignment length reported by the aligners. For most of the reads, NanoBLASTer produces longer alignments than BLAST, LAST and BWA-MEM, but shorter alignments than GraphMap for both yeast and *E. coli*.

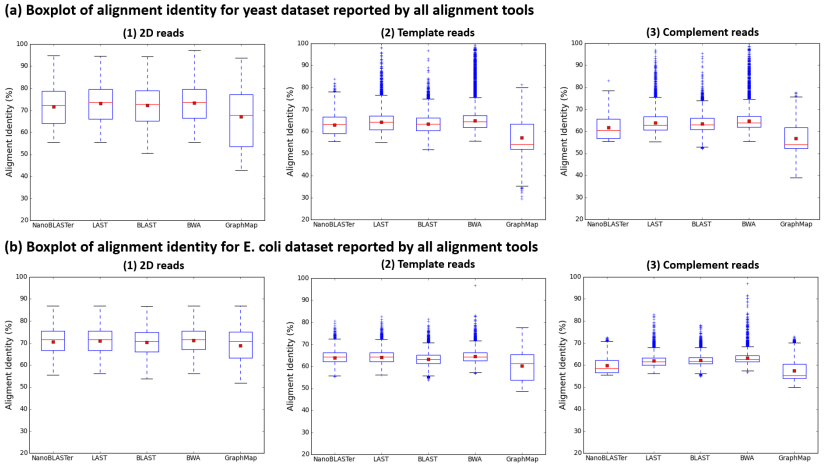

Fig. 3. Boxplot of alignment identity for each of the tools (using longest single alignment for each read). In this plot average alignment identity is indicated by the red square dot and median of the alignment identities is indicated by small red line.

for yeast dataset and around 53% for *E. coli* dataset. As the chances of false alignments increases for the alignment identity less than 55%, we also tried to

align the reverse of the yeast reads (not reverse complement, but effectively a random string) to its genome with this aligner. We found that 80% of the reads were aligned with an average 52.25% identity for 2D, 51.91% identity for template and 51.88% identity for complement reverse reads. Similarly, more than 70% of the *E. coli* reverse reads were aligned with an average 53.49% identity for 2D, 53.37% identity for template and 53.39% identity for complement reads. Whereas, NanoBLASTer aligned only 1.72% of all the yeast reverse reads, and 0.093% of all the *E. coli* reverse reads. We conclude that GraphMap produces many false alignments for the random reads, and find the alignments of reads that were only aligned by GraphMap to be of questionable reliability.

Memory Usage of NanoBLASTer NanoBLASTer uses two lookup tables for all the chromosomes in a dataset. For indexing a K-mer, NanoBLASTer first converts the K-mer to a unique number and then stores all the K-mer positions in a vector for it. The lookup table stores a pointer to the vector for the respective K-mer. We use "Integer" to keep the K-mer locations in the vector. We call this data structure "double index". Using this data structure, NanoBLASTer consumes 1.02 GB RAM for *E. coli*, 1.60 GB RAM for yeast. We also report that NanoBLASTer can process larger genomes by aligning human nanopore sequence data (collected from Experiment: ERX779349 of ENA, submitted by Wellcome Trust Sanger Institute) to hg19 reference genome. All the 26,791 reads were extracted from the given fastq files and accumulated in a single fasta file. NanoBLASTer successfully aligned total 13,582 reads with an average alignment length of 5.97kbp and average identity of 64.79% at the expense of 53.98 GB RAM. NanoBLASTer can also be configured to only use one lookup table (memory consumption will be reduced by half) at the expense of few minutes more of runtime.

Discussion and Future Work

We find several advantages of NanoBLASTer over the other tools: NanoBLASTer provided longer alignments than that of LAST, BLAST and BWA-MEM and it performed well for lower quality reads. By some metrics, GraphMap could be an effective choice, but it also frequently aligns random reads with similar accuracy metrics. Moreover, NanoBLASTer showed very good coverage for both the yeast ($\sim 100.44x$) and *E. coli* ($\sim 98.13x$) dataset across the respective genomes. Considering all these results, NanoBLASTer and nanopore reads in general prove itself to be useful for any number of applications, including de novo genome assembly, structural variation analysis, or many others. Currently, we keep the inverted K-mer index using lookup table in the memory for which NanoBLASTer is less memory efficient. In future release, we will optimize the memory use of NanoBLASTer using an FM-index or other compact representations. Also, we will continue our work on the part of discovering ANCHORs more efficiently. This will include investigating locality sensitive hashing techniques that have proven useful for aligning high error rate sequences from Pacific Biosciences [17].

References

1. Sanger, F., Nicklen, S., Coulson, A.R.: Dna sequencing with chain-terminating inhibitors. Proceedings of the National Academy of Sciences 74(12), 5463-5467 (1977)
2. Mardis, E.R.: A decade/'s perspective on dna sequencing technology. Nature 470(7333), 198-203 (2011)
3. Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., Robinson, G.E.: Big data: Astronomical or genomical? PLoS Biol 13(7), 1002195 (2015). doi:10.1371/journal.pbio.1002195
4. Venkatesan, B.M., Bashir, R.: Nanopore sensors for nucleic acid analysis. Nature nanotechnology 6(10), 615-624 (2011)
5. Laver, T., Harrison, J., O'Neill, P., Moore, K., Farbos, A., Paszkiewicz, K., Studholme, D.: Assessing the performance of the oxford nanopore technologies minion. Biomolecular Detection and Quantification 3, 1-8 (2015)
6. Quick, J., Quinlan, A.R., Loman, N.J.: A reference bacterial genome dataset generated on the minionTM portable single-molecule nanopore sequencer. GigaScience 3(1), 22 (2014)
7. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of molecular biology 215(3), 403-410 (1990)
8. Kielbasa, S.M., Wan, R., Sato, K., Horton, P., Frith, M.C.: Adaptive seeds tame genomic sequence comparison. Genome research 21(3), 487-493 (2011)
9. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. arXiv preprint arXiv:1303.3997 (2013)
10. Sovic, I., Sikic, M., Wilm, A., Fenlon, S.N., Chen, S., Nagarajan, N.: Fast and sensitive mapping of error-prone nanopore sequencing reads with graphmap. bioRxiv, 020719 (2015)
11. Gusfield, D.: Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press, New York, NY (1997)
12. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of molecular biology 147(1), 195-197 (1981)
13. Jain, M., Fiddes, I.T., Miga, K.H., Olsen, H.E., Paten, B., Akeson, M.: Improved data analysis for the minion nanopore sequencer. Nature methods (2015)
14. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., et al.: The sequence alignment/map format and samtools. Bioinformatics 25(16), 2078-2079 (2009)
15. Goodwin, S., Gurtowski, J., Ethe-Sayers, S., Deshpande, P., Schatz, M., McCombie, W.R.: Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. Genome Research (2015). doi:10.1101/gr.191395.115
16. Ross, M.G., Russ, C., Costello, M., Hollinger, A., Lennon, N.J., Hegarty, R., Nusbaum, C., Jaffe, D.B.: Characterizing and measuring bias in sequence data. Genome Biol 14(5), 51 (2013)
17. Berlin, K., Koren, S., Chin, C.-S., Drake, J.P., Landolin, J.M., Phillippy, A.M.: Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. Nature biotechnology (2015)