

An Online Approach for DNA Sequencing Error Correction via Disk Based Index*

Yarong Gu[†] Qiang Zhu[†] Sakti Pramanik[‡]

[†]Computer and Information Science, University of Michigan, Dearborn, MI, USA

[‡]Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

Abstract

DNA sequencing has been widely used in many biological studies such as gene expression analysis and biomedical diagnostics. To solve the problem of sequencing errors produced by a sequencer, researchers perform error correction as a first step in sequence analysis. To overcome the scalability problem of existing memory based error correction methods, a disk based sequencing error correction method, called the DiskBQcor, was recently proposed. It utilizes a disk based index tree to store k -mers from sequencing reads. It then analyzes the results of special box queries run on the index tree to efficiently correct sequencing errors. As an offline approach, the DiskBQcor corrects errors after all the k -mers for a given genome dataset are inserted into the index tree. In this paper, we present an online approach, called the DiskBQcor*, for sequencing error correction to better utilize computing resources. It extends the DiskBQcor by introducing an online analysis process during which sequencing errors are identified and corrected in an online fashion. The relevant correction algorithms, statistical measures, and error identifying strategies are discussed. Our experiments demonstrate that the proposed online method is quite promising in error correction for sequencing genome data on disk.

keywords: DNA sequencing error correction, disk-based index structure, box query, online method.

1 Introduction

Recent years have witnessed an increasing importance of DNA sequencing in the study of a wide range of biological problems such as gene expression analysis, complex microbial ecosystems, and basic molecular biology. Short reads are generated by a sequencer (machine) such as Illumina HiSeq and Pacific Biosciences SMRT for a target genome sequence. With the low sequencing cost of current sequencers, it is possible to cover the target sequence with a high redundancy

(e.g., 20 reads per base in the target genome sequence). Numerous sequence analysis approaches have been developed to utilize fixed-length subsequences, called k -mers, of the sequencing reads (see Figure 1). Since the current sequencers have a fairly high random per-base error rate ranging from 1% to 15% [2], sequencing error correction becomes one of the top priorities in sequence analysis.

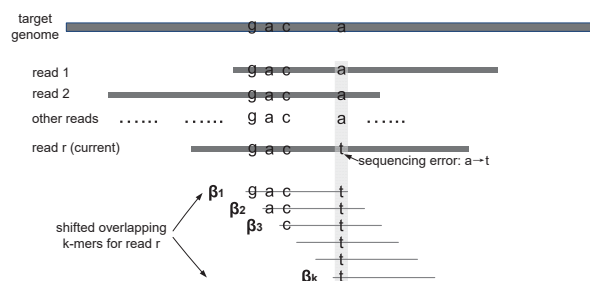


Figure 1: Sequencing reads and k -mers

The initial theory of k -spectrum [4] based error correction method works as follows: for a sufficiently large coverage and k , a sequencing error will almost certainly alter the relevant k -mer to another k -mer that does not exist in the target genome sequence. As a result, those k -mers with low counts, particularly those occurring just once or twice, usually inherit sequencing errors from the corresponding reads.

Previous studies reveal the feasibility of a k -spectrum based error correction method. Quake [5] is such a well-known error correction method. During its error correction process, Quake utilizes Jellyfish [6] to perform the counting task of a large amount of k -mers. Beyond a plain k -mer counting, Quake also takes into account the quality scores of base calls when distinguishing untrusted k -mers (i.e., having low counts) from trusted ones. Apart from this, many other studies incorporate an in-memory structure. For example, Lighter in [7] and another spectral alignment based parallel error correction method in [8] apply a Bloom filter for error correction. SHREC in [9] and a similar study in [10] utilize suffix trees for error correction. Coral [11] puts the k -mers as keys, with

*Research was supported by US NSF (under Grants #IIS-1320078 and #IIS-1319909).

their related reads as values into a hash table, and uses multiple alignments to identify errors.

A disk-based sequencing error correction method, called the DiskBQCor [1], was recently introduced to overcome the drawback of aforementioned methods, i.e., requiring large expensive memory space. It stores all k -mers in an efficient index structure, called the BoND-tree [12], on disk and adopts a vast majority voting mechanism based on the results of special box queries (BQ) run on the index to verify and correct the sequencing errors at suspicious positions in the sequencing reads. Experiments demonstrated that the DiskBQCor could achieve high accuracy for error correction with reasonable efficiency, besides the scalability benefit warranted by a disk based approach.

We notice that the DiskBQCor corrects sequencing errors after all the reads are generated from the sequencer, namely, in an offline fashion. It presents a limitation on utilizing the computing resources and time. While reads are still being generated, the computing resources are idle and wasted in waiting. Meanwhile, a large k -mer set has to be loaded into the BoND-tree all in once, which leads to a long time delay. Using a large, complete BoND-tree can certainly lead to a better accuracy for error correction since the information for the entire read set is available. However, it needs to handle a large tree when performing insertions and queries, which degrades efficiency.

In fact, sequencing error correction does not have to wait until the entire set of sequencing reads is obtained. According to [1], the coverage is not a significant factor affecting the performance of the DiskBQCor. For various coverages, the accuracy remains relatively steady. For example, after changing the coverage from 20 to 10, the accuracy decreases very little (only 0.06%) in the experiment. This interesting observation has inspired us to develop a new online sequencing error correction method that utilizes and extends the DiskBQCor. The voting mechanism used by the DiskBQCor distinguishes the correct and incorrect bases from their relative counts. Given a reasonable minimum coverage at which a correct base is already distinguishable from the incorrect ones, the voting mechanism can be applied immediately. In other words, error corrections can start earlier and the accuracy gradually improves while the index tree grows. In this way, a better utilization of the computing resources is achieved since the DNA sequencing and error correction are performed in parallel.

In this paper, we present an online DNA sequencing error correction method, called the DiskBQCor*, by extending/utilizing the previous DiskBQCor. While the DiskBQCor focuses only on error verification and correction, the DiskBQCor* can detect the suspicious

error positions in a sequencing read besides performing error correction. Specifically, through analyzing on the k -mer abundance [13], the separation between the high-abundance k -mers and the low-abundance k -mers becomes clear. A suspicious error position is typically located at the boundary between these two types of k -mers. By choosing a proper cutoff value, suspicious error positions in sequencing reads can be identified. The base at each suspicious error position can be checked (verified) by invoking the DiskBQCor to see if it is indeed an error. If so, the DiskBQCor can find the correct base to replace the erroneous base at the position. Note that the DiskBQCor* is operating in an online fashion, so that we can start error detection and correction in the early stage while DNA sequencing is still in process, resulting in a better utilization of computer resources.

The remainder of the paper is organized as follows. Section 2 gives an overview of the basic concepts and notations. Section 3 presents the technical details of the DiskBQCor*. Section 4 reports our experimental results. Section 5 concludes the paper.

2 Preliminaries

To facilitate the discussions, we give an overview of relevant concepts in this section. A Non-ordered Discrete Data Space (NDDS) Ω_d is a multi-dimensional vector space, where d is the number of dimensions in Ω_d . Each dimension in Ω_d has an alphabet A_i ($1 \leq i \leq d$) that is consisted of a finite number of elements having no natural ordering among them. A k -mer (e.g., “*attgac*” with $k = 6$) can be considered as a vector in a k -dimensional NDDS, where the element on each dimension of the vector comes from alphabet $\{a, t, c, g\}$.

Let b_i ($1 \leq i \leq d$) be a subset of alphabet A_i (i.e., $b_i \subseteq A_i$). The Cartesian product $b_1 \times b_2 \times \dots \times b_d$ is called a discrete box (rectangle) in Ω_d . A (discrete) box query q on a dataset S in an NDDS is defined as a query with a specified box w that returns all the vectors from S that lie within w . For example, a box query with box $\{a, g\} \times \{t\} \times \{a, c\}$ on a k -mer dataset ($k = 3$) fetches those k -mers from the dataset that have element (base) a or g on the first dimension, t on the second dimension, and a or c on the third dimension. Thus, this box query is equivalent to four exact queries to search for four individual k -mers: *ata*, *atc*, *gta*, and *gtc*. As suggested in [1], box queries can be utilized to help efficiently solve the sequencing error correction problem. More concepts about an NDDS can be found in [14, 15].

The BoND-tree is a disk based index structure specially designed to support efficient processing of box queries in an NDDS [12]. Each (leaf or non-leaf) node

consists of a set of entries and occupies one disk block. Each entry in a non-leaf node consists of a pointer pointing to a subtree and the minimum bounding box (mbb) for all the vectors stored in the subtree. Each entry in a leaf node consists of a vector/ k -mer (as a key) and a pointer pointing to relevant metadata (i.e., the list of relevant read ids in our application). Special strategies making use of the characteristics of an NDDS are adopted to build the tree so that box queries can be processed efficiently on the tree [12]. The structure of an BoND-tree is illustrated in Figure 2.

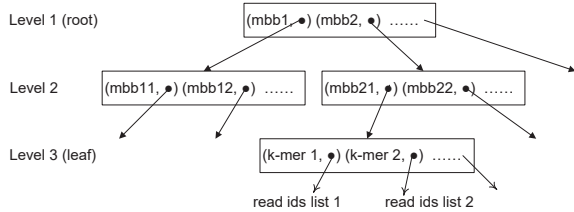


Figure 2: Structure of the BoND-tree

The DiskBQcor adopts the BoND-tree to store the k -mers and performs sequencing error verification and correction by using the results of several specially designed box queries run on the BoND-tree. When the overlapping k -mers are obtained from the given sequencing reads, they are inserted into the BoND-tree. The list of ids of the reads that contain a given k -mer is associated with the k -mer in the corresponding leaf node and saved in one or more linked disk blocks.

To verify and correct an erroneous base at a suspicious position in a sequencing read, the DiskBQcor performs a set of special box queries to count and vote for each possible base at the position and determine what the correct base is at the position. Figure 3 gives an example to illustrate the idea of a simple voting case. In the example, read 40 has an erroneous base ‘g’ at the illustrated position. A box query q is obtained from a k -mer covering the position by using a set $X = \{a, g, t, c\}$ at the suspicious position. The query will return the four k -mers having the four possible bases (i.e., a, g, t, c), respectively, at the suspicious position and their counts. Assume the read coverage is 40. The counts for k -mers with a, t, c, g at the position are 39, 0, 0, 1, respectively. Since a wins the majority of the votes, read 40 has an erroneous base (i.e., g) at the suspicious position and the correct base at the position is a . In general, a comprehensive vast majority voting mechanism using the k -mer counts is employed to handle various scenarios such as the solo occurrence of a suspicious k -mer, the repetitive occurrences of a suspicious k -mer, and the occurrence of multiple errors within the distance of k positions/bases. The extreme cases that cannot be handled by the vast majority voting mechanism are handled by an efficient binary

encoding based assembly technique. The more details of the DiskBQcor can be found in [1].

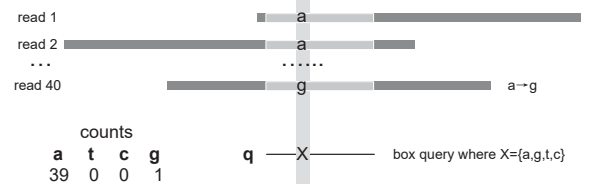


Figure 3: Error verification via voting

3 The Method

Let us now describe the details of the online sequencing error correction method DiskBQcor*.

3.1 Basic Ideas

A DNA sequencer generates reads one by one for a target genome sequence. The generated reads are randomly distributed and covering the underlying genome sequence. The sequencer may produce sequencing erroneous bases at some positions of a read. The error rate is usually low, comparing to the correct bases in reads. Like other sequencing error correction methods, our goal is to correct a high percentage of errors in the sequencing reads.

As the sequencing reads arriving one by one, the sequencing errors are corrected in two phases. Before the minimum (average) coverage is reached, the reads are decomposed into k -mers and inserted into the BoND-tree (Insertion Phase) without corrections. After the minimum coverage is reached, error corrections start (Correction Phase) while reads continue to be decomposed into k -mers and inserted into the BoND-tree. The details of these two phases will be discussed in Sections 3.3 and 3.4.

Note that the index tree is built at the time during the DNA sequencing in an online fashion. In this way, the cost of building a large index tree is amortized over the DNA sequencing process, which mitigates the user’s pain from the long duration of building such an index tree in the conventional offline fashion. Since the DNA sequencing and the error correcting are done in parallel, the sequencing and computing resources are better utilized.

The quality of the index tree (i.e., the BoND-tree), in terms of keeping correct k -mers, is gradually improving as the tree grows larger. The index tree with all corrected k -mers can be obtained once the DNA sequencing is completed. This final tree with all the corrected k -mers and useful meta data can be used for sequence analysis applications such as sequence

alignment, terminus searching, variant detection and error verification. If we want to maintain an index tree for corrected k -mers for further incoming reads, this procedure can continue to proceed on top of the current index tree.

3.2 Measures

Coverage Statistics Given a sample DNA sequence, we assume that the sequencer outputs sequencing reads one by one. The average coverage of each base at a specific time during the sequencing process is calculated through:

$$\bar{c} = \frac{N * l}{G} \quad (1)$$

in which N is the total number of reads arrived, l is the length of one read, and G is the whole genome size. $\bar{c}$ is an important indicator of the current loading situation in the index tree.

Changing rates The count change rate $\delta(k_1, k_2)$ for two consecutive shifted k -mers k_1 and k_2 is calculated as:

$$\begin{aligned} \delta(k_1, k_2) &= \frac{\Delta c(k_1, k_2)}{\Delta x} = \frac{\text{count}(k_2) - \text{count}(k_1)}{1} \\ &= \text{count}(k_2) - \text{count}(k_1) \end{aligned} \quad (2)$$

where Δx denotes the number of steps between two k -mers, which is 1 for two consecutively shifted k -mers k_1 and k_2 . A large negative $\delta(k_1, k_2)$ usually indicates an error has occurred (i.e., k_2 hits an erroneous position (from left to right), resulting a dramatically reduced count), and may also indicate entering a low-count (coverage) area from a high-count area; while a large positive $\delta(k_1, k_2)$ may indicate k_2 entering a correct area without erroneous positions, and may also indicate entering a high-count area from a low-count area (see Figure 4). We can always assume the erroneous case since the error correction is an idempotent operation (i.e., no hurt to perform it several times).

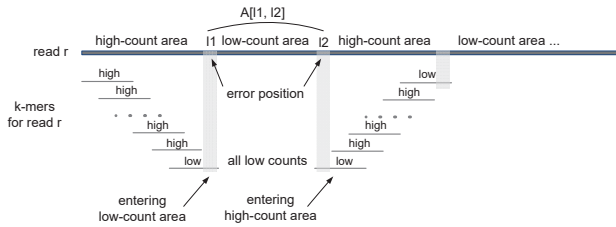


Figure 4: Change of k -mer counts for low-count and high-count areas in a read

3.3 Insertion Phase

In the Insertion Phase, our goal is to decompose the reads generated by the sequencer into shifted

overlapping k -mers, and insert the k -mers into the BoND-tree, without considering if a k -mer contains an error. This is the initial tree building phase, during which the overlapping k -mers obtained from the sequencing reads for a target genome sequence along with their relevant read ids are loaded into the tree.

Algorithm 3.1 Insertion Phase Procedure

Require: (1) incoming reads; (2) minimum (average) coverage C_m

Ensure: BoND-tree BT with initial k -mers being inserted

```

1: repeat receive a new read  $r$ 
2:   Decompose  $r$  into shifted overlapping  $k$ -mers
3:   for each shifted  $k$ -mer  $k$  from left to right do
4:     Insert  $k$  into BoND-tree  $BT$ 
5:   end for
6:   update  $\bar{c}$ 
7: until  $\bar{c} \geq C_m$ 

```

In Algorithm 3.1, steps 1-7 constitute a loop to continuously insert k -mers into the BoND-tree. Specifically, step 2 decomposes the arriving read into shifted overlapping k -mers. Steps 3-5 insert all the k -mers generated from the read into the tree.

3.4 Correction Phase

In the Correction Phase, our goal is to continue to decompose the generated reads into k -mers, and compare them with those already stored in the tree. Specifically, through analyzing on the k -mer abundance (counts), the separation between the high-abundance k -mers and the low-abundance k -mers will become clear. A suspicious error position is typically at the boundary between these two types of k -mers. The base at each suspicious error position can be checked (verified) by invoking the DiskBQcor to see if it is indeed an error. In this phase, for a given suspicious error position in a sequencing read, the DiskBQcor formulates a set of special shifted box queries and performs them on the BoND-tree to retrieve the relevant k -mers and their counts. With a special voting mechanism, the possibly erroneous base at the given position can be verified positively or negatively, and the correct base at the position can be identified if an error is found. In the latter case, the erroneous base at the suspicious position in the corresponding read is replaced by the correct one. We then update the relevant high-count area and the relevant low-count area on the read. The Correction Phase is described in Algorithm 3.2.

In Algorithm 3.2, steps 1-5 are similar to those in the Insertion Phase procedure, except that we keep the count of each overlapping k -mer to find a low-count area (with $\delta(k_1, k_2) \leq C_l$, where the low-count threshold value $C_l < C$ and C is the current (average) coverage satisfying $C_m \leq C \leq C_M$), and also keep the read ID sets to correct the (sequential) error(s) if such an

Algorithm 3.2 Correction Phase Procedure

Require: (1) incoming reads; (2) desired (average) coverage C_M
Ensure: BoND-tree BT with k -mers being inserted

```

1: repeat receive a new read  $r$ 
2:   Decompose  $r$  into shifted overlapping  $k$ -mers
3:   Insert these  $k$ -mers into  $BT$  and find their counts, read
   ID sets and calculate count change rates  $\delta(k_1, k_2)'s$  for each
   pair of consecutively shifted  $k$ -mers  $k_1$  and  $k_2$  in  $r$ 
4:   Use the count change rates  $\delta(k_1, k_2)'s$  to find area
   boundaries with dramatic changing  $\delta(k_1, k_2)'s$ 
5:   Let  $LL$  = the list of low-count areas in  $r$  ordered from left
   to right
6:   while  $LL$  is not empty do
7:     Let  $A[l_1, l_2]$  be an area in  $LL$ 
8:     Let  $(p, direction, pk) = \text{find\_start\_error\_position}($ 
        $A[l_1, l_2], LL, r, BT)$ 
9:      $\text{more\_error\_in\_A} = \text{true}$ 
10:    while  $\text{more\_error\_in\_A}$  do
11:       $\text{error\_correct}(A[l_1, l_2], BT, p, direction, pk)$ 
12:      Check if  $A[l_1, l_2]$  has any low-count  $k$ -mer left
13:      if  $A[l_1, l_2]$  has low-count  $k$ -mer left then
14:         $(p, direction, pk) = \text{find\_start\_error\_position}($ 
           $A[l_1, l_2], LL, r, BT)$ 
15:      else  $\text{more\_error\_in\_A} = \text{false};$  remove  $A[l_1, l_2]$ 
        from  $LL$ 
16:      end if
17:    end while
18:  end while
19: until  $\bar{c} \geq C_M$ 

```

area exists. Steps 6-18 correct the error(s) in each low-count area. Specifically, step 7 picks up a low-count area $A[l_1, l_2]$. Because of the initial Insertion Phase, $A[l_1, l_2]$ must be adjacent to a left high-count area (with $\delta(k_1, k_2) > C_l$) in read r or adjacent to a right high-count area in r (or both). Function $\text{find_start_error_position}()$ in step 8 finds the start suspicious error position (i.e., the first or last low-count position in $A[l_1, l_2]$ that is adjacent to a high-count area) and returns three parameters p , $direction$ and pk that will be used in function $\text{error_correct}()$, where p is the start suspicious error position found in $A[l_1, l_2]$, pk is the position index on the relevant k -mer containing p , and $direction$ is either 1 (i.e., the error verification/correction will take place from the left-most low-count k -mer in $A[l_1, l_2]$ to the right) if the current low-count area in $A[l_1, l_2]$ is adjacent to a left high-count area, or -1 (i.e., the error verification/correction will take place from the right-most low-count k -mer in $A[l_1, l_2]$ to the left) if the current low-count area in $A[l_1, l_2]$ is adjacent to a right high-count area. Steps 9 - 17 corrects all the errors in area $A[l_1, l_2]$. In particular, for each suspicious position in the area, step 11 invokes the error correction function $\text{error_correct}()$ that utilizes the DiskBQcor in [1] to verify if the base at position p is indeed an error. If so, the DiskBQcor can find the correct base to replace the erroneous base at the position. After the base is successfully corrected, we update every k -mer overlapping the position by

deleting it from the BoND-tree first, and reinserting the correct k -mer into the tree. Meanwhile, we search for its associated read ID set, and correct all the reads in the set. The DNA sequencing process continues until the desired (average) coverage C_M is reached. The online correction procedure then terminates.

4 Experiment

Experiments were conducted on a Dell PC with a 3.2 GHz Intel Core i7-4790 CPU, 12 GB RAM, 5 TB Hard Drive, and Linux 3.16.0 OS. The DiskBQcor* was implemented in the C++ programming language.

Genome data was collected from *E. coli* 536 (GenBank: NC008253, 5.5 M) with $k = 15$ and read length = 36. The threshold value C_l to separate a low-count area from a high-count area was set to be a ratio 0.2 of the current coverage C , i.e., when the current coverage is 10, if a k -mer K has a count of less than or equal to 2, it belongs to a low-count area; otherwise it is classified to a high-count area.

In the first experiment, the desired coverage C_M was set to 40. We tested the impact of the minimum coverage C_m on the correction accuracy. In other words, we ran the Insertion Phase until the minimum coverage C_m was reached, and then proceeded with the Correction Phase until the desired coverage C_M reaches 40. Table 1 shows the comparison of the observed experimental results when we set the minimum coverage C_m from 5 to 9. From the table we can see that the accuracy is decreasing while C_m is increasing, which is understandable because the initial set of reads were inserted into the index tree without error correction before the minimum coverage C_m was reached; yet mis-corrections were decreasing when C_m was increasing.

Table 1: Correction Accuracy, low count ratio=0.2, no initial repeat

Minimum Coverage (C_m)	Corrections (TP)	Mis-corrections (FP)	Errors Kept (FN)	Untrusted (trimmed)	Sensitivity	Specificity
5	864148	152008	185837	50096	0.8230	0.9993
6	845307	119307	207969	42152	0.8026	0.9995
7	824330	103459	231143	37837	0.7810	0.9995
8	802396	95665	254884	34995	0.7589	0.9996
9	779951	92190	278650	33385	0.7368	0.9996

We can correct sequencing errors for the reads in the initial set used in the Insertion Phase by feeding them (re-insertion) through the Correction Phase after C_M is reached, which will actually provide us with a more beneficial set of data in the index tree. In the second experiment, the coverage times were still set to 40. Similar to the first experiment, we changed the minimum coverage C_m , and ran the Insertion Phase until C_m is reached (let R be the initial set of the reads),

and then proceeded with the Correction Phase until the desired coverage $C_M = 40$ was reached. After that, we feeded the reads in set R again to the Correction Phase to correct their sequencing errors. Table 2 shows the comparison of the observed experimental result when we set the minimum coverage C_m from 5 to 9. From the table we can see that the sensitivity increases when C_m increases. The increase rate of sensitivity is higher at first, and becomes more steady when C_m is set to 9.

Table 2: Correction Accuracy, low count ratio=0.2, with initial repeat

Min Coverage (C_m)	Correc-tions (TP)	Mis-corrections (FP)	Errors Kept (FN)	Untrusted (trimmed)	Sensi-tivity	Speci-ficity
5	983910	160568	59908	54623	0.9426	0.9993
6	989022	129783	56900	47530	0.9456	0.9994
7	991979	115810	54984	44038	0.9475	0.9995
8	994004	109934	53541	42088	0.9489	0.9995
9	995218	108196	52511	41311	0.9499	0.9995

In order to examine the impact of the threshold value C_l on the accuracy of sequencing error correction, we conducted another experiment, in which the threshold ratio between C_l and C was set to change while keeping the minimum coverage C_m to 7. Table 3 shows the observed experimental results. Note that the ratio started at 0.15 because we wanted the Correction Phase to run as soon as the coverage reached 7, in which case the threshold value C_l was 1.

Table 3: Correction Accuracy, $C_m = 7$, with initial repeat

Ratio (Thres-hold)	Correc-tions (TP)	Mis-corrections (FP)	Errors Kept (FN)	Untrusted (trimmed)	Sensi-tivity	Speci-ficity
0.15	993609	88933	56005	37777	0.9466	0.9996
0.20	991979	115810	54984	44038	0.9475	0.9995
0.25	982929	212117	55262	68075	0.9468	0.9990

From Table 3 we can see that the ratio = 0.20 is the best for the minimum coverage C_m of 7, although the difference of the sensitivity among the three ratios was less than 0.001.

The overall accuracy is comparable to existing error correction methods. [13] has a higher sensitivity of 0.997 and a lower specificity of 0.688. Other methods all have a specificity of almost 1.0. According to BLESS [19], Quake has a sensitivity of 0.838, and BLESS has a sensitivity of 0.968. Since our proposed DiskBQcor* is an online method, it has a benefit of better utilizing the computing resources while DNA sequencing is still ongoing.

5 Conclusions

The dramatic increase in DNA sequencing capacity has quickly turned biology into a data-intensive science.

However, current sequencers suffer from the problem of having high random per-base error rates. Hence, sequencing error correction is crucial to many sequence analysis applications in bioinformatics. Most existing sequencing correction techniques cannot scale well to large datasets due to their requirements on huge expensive computer memory space. An inexpensive disk-index-tree based sequencing error correction method, called the DiskBQcor, was proposed recently [1]. However, it is an offline method, which can be applied only after the DNA sequencing is complete. Furthermore, it also assumes that suspicious erroneous bases in a read are known in advance.

In this paper, we have introduced a new online sequencing error correction method, called the DiskBQcor*. It is built on the top of the DiskBQcor, but extends the latter to allow the online sequencing error correction process and, in meantime, provides a count-change-rate mechanism to identify suspicious erroneous positions. The relevant statistical measures are also presented. Our experimental results demonstrate that the DiskBQcor* is quite promising in correcting sequencing errors in an online fashion. Since the proposed online error correction method stores the k -mers from reads in a disk-based index structure BoND-tree [12], it provides two basic benefits: the scalability warranted by the disk-based approach, and the efficiency gained from the index structure. Furthermore, memory restrictions with existing memory-based error correction methods require that only a summary of the reads (typically a combined summary of the k -mers and associated quality information) be kept in memory, limiting the opportunity to make use of positional information and sample metadata (e.g., sequencing chemistry, machine version) used to generate the data. The proposed method can maintain relevant extra metadata that not only can be used in error correction but also can benefit other sequence analysis applications such as local alignment searching, sequence assembly, and terminus searching.

Our future research includes incorporating base quality scores into our error correction methods, digging into the potentials to improve the accuracy when localizing error regions, exploring our methods in a distributed environment including Hadoop/MapReduce, adopting faster approaches like bulk loading in tree building phase, and applying our BoND-tree storing all the corrected k -mers and their relevant metadata to other sequence analysis applications.

References

- [1] Y. Gu, Q. Zhu, X. Liu, Y. Dong, C. T. Brown, and S. Pramanik, "Using disk based index and box queries for genome sequencing error correction," *Proc. of The 8th International Conference on Bioinformatics and Computational Biology*, pp. 69–76, 2016.
- [2] M. L. Metzker, "Sequencing technologies-the next generation," *Nature Reviews Genetics*, vol. 11, no. 1, pp. 31–46, 2010.
- [3] T. J. Treangen and S. L. Salzberg, "Repetitive dna and next-generation sequencing: computational challenges and solutions," *Nature Reviews Genetics*, vol. 13, no. 1, pp. 36–46, 2012.
- [4] X. Yang, S. P. Chockalingam, and S. Aluru, "A survey of error-correction methods for next-generation sequencing," *Briefings in Bioinformatics*, vol. 14, no. 1, pp. 56–66, 2013.
- [5] D. R. Kelley, M. C. Schatz, S. L. Salzberg *et al.*, "Quake: quality-aware detection and correction of sequencing errors," *Genome Biology*, vol. 11, no. 11, p. R116, 2010.
- [6] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011.
- [7] L. Song, L. Florea, and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome Biology*, vol. 15, p. 509, 2014.
- [8] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig, "A parallel algorithm for error correction in high-throughput short-read data on cuda-enabled graphics hardware," *Journal of Computational Biology*, vol. 17, no. 4, pp. 603–615, 2010.
- [9] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, "Shrec: a short-read error correction method," *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, 2009.
- [10] L. Salmela, "Correction of sequencing errors in a mixed set of reads," *Bioinformatics*, vol. 26, no. 10, pp. 1284–1290, 2010.
- [11] L. Salmela and J. Schröder, "Correcting errors in short reads by multiple alignments," *Bioinformatics*, vol. 27, no. 11, pp. 1455–1461, 2011.
- [12] C. Chen, A. Watve, S. Pramanik, and Q. Zhu, "The BoND-Tree: An efficient indexing method for box queries in nonordered discrete data spaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2629–2643, 2013.
- [13] Q. Zhang, S. Awad, and C. T. Brown, "Crossing the streams: a framework for streaming," *PeerJ PrePrints*, 2015.
- [14] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, "Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 2, pp. 439–484, 2006.
- [15] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, "A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces," *ACM Transactions on Information Systems (TOIS)*, vol. 24, no. 1, pp. 79–110, 2006.
- [16] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu, "A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers," *BMC genomics*, vol. 13, no. 1, p. 341, 2012.
- [17] W.-C. Kao, A. H. Chan, and Y. S. Song, "Echo: a reference-free short-read error correction algorithm," *Genome research*, vol. 21, no. 7, pp. 1181–1192, 2011.
- [18] R. C. Edgar and H. Flyvbjerg, "Error filtering, pair assembly and error correction for next-generation sequencing reads," *Bioinformatics*, p. btv401, 2015.
- [19] Y. Heo, X.-L. Wu, D. Chen, J. Ma, and W.-M. Hwu, "Bless: bloom filter-based error correction solution for high-throughput sequencing reads," *Bioinformatics*, p. btu030, 2014.
- [20] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer, "Substantial biases in ultra-short read data sets from high-throughput dna sequencing," *Nucleic acids research*, vol. 36, no. 16, pp. e105–e105, 2008.