Using Disk Based Index and Box Queries for Genome Sequencing Error Correction

Yarong Gu[†] Qiang Zhu[†] Xianying Liu[†] Youchao Dong[†] C. Titus Brown[‡] Sakti Pramanik[‡]
[†]Computer and Information Science, University of Michigan, Dearborn, MI, USA
[‡]Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

Abstract

The vast increase in DNA sequencing capacity over the last decade has quickly turned biology into a data-intensive science. Nevertheless, current sequencers such as Illumia HiSeq have high random per-base error rates, which makes sequencing error correction an indispensable requirement for many sequence analysis applications. Most existing error correction methods demand large expensive memory space, which limits their scalability for handling large datasets. In this paper, we present a new disk based method, called DiskBQcor, for sequencing error correction. DiskBQcor stores k-mers of sequencing genome data along with their associated metadata on inexpensive disk and utilizes a disk based index tree to efficiently process special box queries to obtain relevant k-mers and their occurring frequencies. It then applies a comprehensive voting mechanism and possibly an efficient binary encoding based assembly technique to verify and correct an erroneous base in a genome sequence under various conditions. Our experiments demonstrate that the proposed method is quite promising in error verification and correction for sequencing genome data on disk.

keywords: DNA sequencing, error correction, index tree, box query, algorithm.

1 Introduction

DNA sequencing has been increasingly serving studies on biological problems including biomedical diagnostics, gene expression analysis, drug resistance, and basic molecular biology. However, current sequencers have quite high random per-base error rates, ranging from 1% for Illumina HiSeq to 15% for Pacific Biosciences SMRT [9]. Dealing with sequencing errors is a significant challenge for both mapping and assembly based approaches to sequence analysis. Moreover, comparing to the conventional sequencing methods, the next-generation sequencers produce shorter read lengths, leading to more repeats [20], which makes it more difficult to decide an appropriate sequencing error correction method.

Fortunately, the low sequencing cost of the next-generation sequencers has made it possible to detect and correct the errors by providing a high redundant coverage with sequencing reads for the target genome sequence. For a sufficiently large coverage and k, almost all the sequencing errors alter the relevant k-mers to versions that do not exist in the target genome sequence. Therefore, k-mers with low counts, particularly those occurring just once or twice, usually inherit sequencing errors from the corresponding reads. Error correction and other relevant sequence analysis applications [2, 4, 7] have made counting large amounts of k-mers a paramount need for bioinformatics research.

In order to deal with the computational challenges for large k-mer datasets, a number of efficient k-mer counting methods have been proposed in the literature. Most of them use a large in-memory structure such as hash table [8], Bloom filter [22], suffix tree [7], and sorted bin set [10] to store k-mers. These in-memory structures provide efficient random access to k-mers in memory. However, they usually have a high demand on computing resources. For example, Jellyfish [8] is a popular hash based counting method run on a computer with 32 cores and 256GB RAM. Such high-end computing equipment is not common for most biology laboratories today. Quake in [6], which is a widely-used error correction method, utilizes Jellyfish to perform the counting job during its error correction process. Beyond a plain k-mer counting, Quake also takes into account the quality scores of base calls when distinguishing trusted and untrusted k-mers. Lighter in [19] and the spectral alignment based parallel error correction method in [18] utilize a Bloom filter for error correction, while SHREC in [16] and its variant in [15] utilize suffix trees for error correction.

One way to reduce the expensive memory requirement for an error correction method is to develop a new technique utilizing relatively cheap disk space. There are two challenges in doing so: (1) how to efficiently search target k-mers from a large k-mer dataset on disk, and (2) how to utilize search results to correct sequencing errors. To tackle these challenges, we

BICOB 2016, April 4-6 2016, Las Vegas, Nevada, USA
978-1-943436-03-3 / Copyright ISCA

propose a novel disk based method, called DiskBQcor, that stores the k-mers with a recently developed index structure, called the BoND-tree [1], on disk and adopts a vast majority voting mechanism to verify and correct the sequencing errors at suspicious positions in the sequencing reads generated by a sequencer for a target genome sequence. For a suspicious error position in a sequencing read, a set of special box queries (BQ) are performed on the BoND tree to count and vote for each possible base at that position and determine what the correct base is at the position. Various scenarios are considered in DiskBQcor. The extreme cases are handled by an efficient binary encoding based assembly technique. Experiments demonstrate that DiskBQcor is quite promising in achieving high accuracy for error correction with reasonable efficiency, in addition to having the scalability benefit warranted by a disk based approach.

DiskBQcor focuses on error verification and correction. The detection of a suspicious error position in a sequencing read can be done by adopting a technique like the one in [21]. Specifically, through analyzing on the k-mer abundance, 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. 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 DiskBQcor to see if it is indeed an error. If so, DiskBQcor can find the correct base to replace the erroneous base at the position.

Two k-mer counting methods DSK [14] and KMC 2 [3] that utilize disk space to reduce the memory requirement were proposed in the literature. Since these two methods aim at counting all the k-mers in the given set, they do not support efficient random access to the k-mers on disk. On the other hand, the goal of our DiskBQcor is to verify and correct a suspicious error(s) at a given position(s) in a read, which implies that we need to search and count only those interested k-mers from the given dataset. As a result, efficient random disk access is required for DiskBQcor, which is realized by adopting the recent BoND-tree [1].

As mentioned earlier, Quake [6] is a widely used method to correct errors in sequencing reads with high coverage, which takes k-mer statistics as an important component to accomplish its task. However, Quake has some limitations: (1) it has to use a sufficiently large k to achieve high accuracy; (2) it cannot handle the cases in which untrusted k-mers are repeated at several places or an erroneous base occurs near the boundary of a read. Limitation (1) leads to a large size of the k-mer dataset, while limitation (2) causes Quake to trim off some true error cases from consideration. As we will see, with carefully designed box queries and other strategies, DiskBQcor is able to mitigate both limitations. To our knowledge, no similar work has been reported in the literature.

The rest of this paper is organized as follows. Section 2 discusses the details of the proposed DiskBQcor. Section 3 reports our experimental results. Section 4 concludes the paper.

2 The Method

The basic idea of DiskBQcor works as follows. The overlapping k-mers obtained from the sequencing reads for a target genome sequence along with their relevant metadata are loaded into a BoND-tree on disk. For a given suspicious error position in a sequencing read, a set of special shifted box queries are formulated and performed 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/corrected by the correct one. The relevant details are given below.

2.1 Building the BoND-tree

We can view a k-mer (e.g., "agc" with $k = 3$) as a vector in a k-dimensional Non-ordered Discrete Data Space (NDDS), where the letter (base) on each dimension of the vector is from alphabet $\{a, t, c, g\}$. In general, an 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 (domain) A_i ($1 \leq i \leq d$) consisting of a finite number of letters, where no natural ordering exists among the letters. 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 $b_1 \times b_2 \times ... \times b_d$ is called a discrete box (rectangle) in Ω_d. More concepts about an NDDS can be found in [12, 13].

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 $\{a\} \times \{g, t\} \times \{c, t\}$ on a k-mer dataset ($k = 3$) fetches those k-mers from the dataset that have letter (base) a on the first dimension, g or t on the second dimension, and c or t on the third dimension. Thus, this box query is equivalent to four exact queries to search for four individual k-mers: agc, atc, agt, and att. As we will see, box queries can be utilized to help efficiently solve the sequencing error correction problem.

The BoND-tree is a recent disk based index technique that was specially designed for supporting efficient

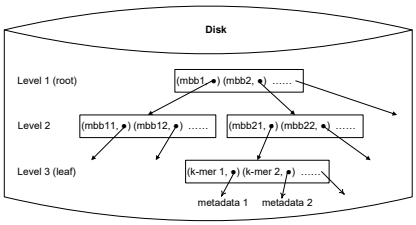

Figure 1: The BoND-tree Structure.

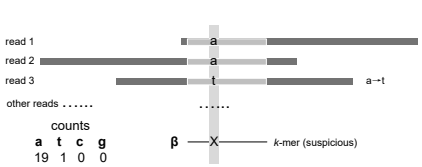

Figure 2: Voting in Case of Solo Occurrence of a Suspicious k-mer.

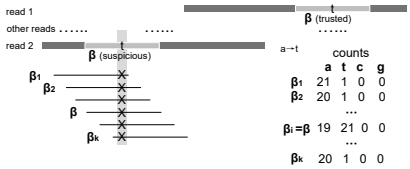

Figure 3: Voting in Case of Repetitive Occurrences of a Suspicious k-mer.

processing of box queries in an NDDS [1]. It has a balanced hierarchical indexing tree structure (see Fig. 1). Each 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 and a pointer pointing to relevant metadata. Special strategies utilizing the characteristics of an NDDS are adopted to build the tree so that box queries can be processed efficiently [1].

When the overlapping k-mers obtained from the given sequencing reads are loaded into a BoND-tree in DiskBQcor, the canonical representation of each k-mer, which is either the k-mer itself or its reversed complement, is calculated and inserted into the tree. The metadata associated with each k-mer in the corresponding leaf node in DiskBQcor is a list of ids for the reads that contain this k-mer or its reserved complement (indicated by a flag), which is saved in one or more linked disk blocks. Note that a conventional memory-based method can only afford saving the k-mers and their counts in their in-memory structures due to their restricted scalability for handling large datasets. Hence, more detailed information such as the read ids are not stored with the k-mers. Since DiskBQcor uses disk space, all necessary metadata can be saved with the k-mers, which reduces a large amount of unnecessary dynamic computing overhead.

2.2 Vast Majority Voting

Given a sequencing read (e.g., $r = acctgga[t]tcgtag$ .....) and a suspicious error position in the read (e.g., $[t]$ in r), the possibly erroneous base (e.g., t) at the position can be verified and corrected (if an error) by a voting approach described as follows.

We can choose a suspicious k-mer (e.g., $gga[t]tc$ with $k = 6$ in the above example) that covers the suspicious error position, replace the possibly erroneous base at the position (e.g., t) by each of 4 possible bases (i.e., a, t, c, g) to form four k-mers (e.g., $ggaatc$, $ggagtc$, $ggattc$, and $ggactc$), and use these k-mers as exact queries to find the number of occurrences for each of them at the suspicious position. Since there usually is a high coverage (e.g., about 20 times) with sequencing reads at the suspicious error position, most of the reads typically contain the correct base at the position. Using a vast majority voting rule, we can discover if the possibly erroneous base (e.g., t) is indeed an error and, if so, what the correct base is.

One optimization that can be further done here is to group the 4 exact queries into one box query by using set $X = \{a, t, c, g\}$ at the suspicious position of the suspicious k-mer to indicate all the possible bases at that position. In other words, the box used for the box query in the above example is: $\{g\} \times \{g\} \times \{a\} \times X \times \{t\} \times \{c\}$ (simply denoted by $ggaXtc$ in the remaining discussion). The votes/counts can be found by analyzing the box query result. Since only one (box) query instead four (exact) queries is executed on the BoND-tree, the efficiency is usually further improved due to the reduced latency and shared processing. Once an erroneous base has been verified, the error correction is just a matter of replacing.

Fig. 2 shows an example for error verification and correction. Assume that the average coverage is 20, which implies that each position in the target genome sequence is covered by about 20 sequencing reads on average. In the figure, reads 1, 2, 3, and seventeen other sequencing reads are produced by a sequencer to cover a certain range of the target genome sequence. Assume that the sequencer has produced an erroneous base (altering a to t) at the indicated position for read 3, making any k-mer that contains the position suspicious. After performing a box query whose box is obtained by replacing the erroneous base by set $X = \{a, t, c, g\}$ in a chosen suspicious k-mer β on the BoND-tree, the counts for a and t can be found to be 19 and 1, respectively[1]. Clearly, the correct base at the suspicious position is a.

In general, the correct base is the one with the maximum count: $\max_{y \in X} \{count(y)\}$ provided that this maximum count is significantly larger than the count of any other base at the position. If the possibly erroneous base at the position has the maximum count, it implies that this base is not an error. Otherwise, this base is determined to be an error.

However, the above simple voting approach cannot handle the situation when a suspicious k-mer β that we

[1]The count for base x (with respect to suspicious k-mer β) is the count for modified β with x being placed at the suspicious position.

use to convert into a box query for a suspicious error position happens to also appear as a trusted (without error) k-mer in another place of the target genome sequence. Fig. 3 shows such a scenario, where β appears as a suspicious k-mer in read 2 and also appears as a trusted k-mer in read 1. Assume that the average coverage is still about 20 (times) in this example. In such a case, the counts for the erroneous base t and the correct base a are about the same (e.g., $21 = 1 + 20$ vs. $19 = 19 + 0$) since the counts for t in β from both places are combined/added. Hence, it is difficult to determine which base is correct at the suspicious error position.

To solve this problem, we consider all the (shifted) suspicious k-mers $\beta_1, \beta_2, ..., \beta_k$ that cover the suspicious error position as shown in Fig. 3. For each β_i ($1 \leq i \leq k$), we form a box query by replacing the possibly erroneous base (e.g., t) at the suspicious error position by set $X = \{a, t, c, g\}$.

After these k (shifted) box queries are performed on the BoND-tree, the following $k \times 4$ counting matrix representing all the counts with respect to the k suspicious k-mers can be obtained:

$$\begin{pmatrix} a_1 & t_1 & c_1 & g_1 \\ a_2 & t_2 & c_2 & g_2 \\ ... & & & \\ a_k & t_k & c_k & g_k \end{pmatrix} \quad (1)$$

where a_i, t_i, c_i, g_i ($1 \leq i \leq k$) are the counts for suspicious k-mer β_i when its possibly erroneous base at the suspicious error position is replaced by base a, t, c, g, respectively. Let the vote of base y be

$$v(y) = \min_{1 \leq i \leq k} \{y_i\} \quad (2)$$

where $y \in \{a, t, c, g\}$. For the example shown in Fig. 3, we have $v(a) = 19, v(t) = 1, v(c) = 0$, and $v(g) = 0$.

It is noted that $v(y)$ is usually close to the coverage number (e.g., 20) if base y is a correct one at the suspicious position since the counts for all the β_is are usually close to the coverage number when y is placed in the suspicious position[2]. Hence, $v(y)$ is relatively large in this case. On the other hand, when base y is an error, y_i is usually small if β_i does not appear as a trusted k-mer in another place, and y_i is large if β_i also appears as a trusted k-mer in another place. Hence, $v(y)$ is usually small if there exists at least one β_i that does not appear as a trusted k-mer in another place. Note that the chance for every β_i is repeated as a trusted k-mer in another place is small especially when k is reasonably large. Therefore, we can apply the following voting rule to determine a correct base:

Vast Majority Voting Rule (VMVR): The correct base at the suspicious error position is determined to be the one that has the following maximum vote:

$$MV = \max_{y \in X} \{v(y)\} \quad (3)$$

[2]For a reasonable sequencer, it should produce most reads at each position of the target genome correctly.

if MV is significantly larger than the vote for any other base at the position.

If the vote for the possibly erroneous base e at the suspicious error position is close to MV, e is determined to be not an error. Otherwise, it is an error. For Fig. 3, since $MV = v(a)$ (=19) and no other votes are close to MV, base t is determined to be an error at the suspicious error position, and base a is identified as the correct one at the position.

The condition for the vast majority voting mechanism based on Formula (3) to fail to work is that the votes for two or more bases are close to MV. We will present a strategy to handle error correction under this condition in Section 2.4.

2.3 Correcting Multiple Errors

Let us consider the situation in which a read has multiple suspicious error positions that need to be verified and corrected. If these suspicious error positions are located at least k positions/bases (distance) apart from each other in the read, they can be verified and corrected individually by the procedure presented in Section 2.2. However, when the distance between two consecutive suspicious error positions is less than k positions/bases, the above method cannot be directly applied. The reason is explained as follows.

Fig. 4 shows an example in which we have two erroneous bases g and t at suspicious error positions p1 and p2, respectively, in read 3. Assume that the distance between p1 and p2 is $k - 2$. When we form all the (shifted) suspicious k-mers β_1, β_2, ..., β_k that cover the suspicious position p2, we notice that the first two suspicious k-mers β_1 and β_2 also cover the other suspicious position p1. After the box queries that are obtained by replacing the erroneous base at p2 with set $X = \{a, t, c, g\}$ in each suspicious k-mer, the counting matrix is computed as shown in the figure. We can see that the counts for correct base a at p2 from β_1 and β_2 are 0 although most of reads have this correct base a at p2. The reason for this phenomenon is that β_1 and β_2 also contain another erroneous base g at p1, which prevent them from matching the corresponding correct reads at p2. As a result, this counting matrix cannot be used by the VMVR described in the last section since the correct base a at p2 cannot be identified in this way.

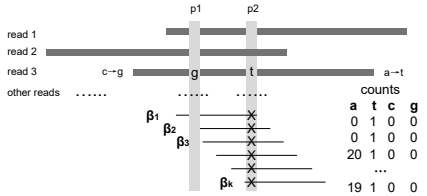

Figure 4: Multiple errors on one read.

In such a case, we adopt the following approach to

correcting multiple errors in a read. We consider each suspicious error position in a read from the right to the left. For each suspicious position, we form all those (shifted) suspicious k-mers that cover the current suspicious position but do not cover the next suspicious position (e.g., β_3, ..., β_k for position p2 in Fig. 4), convert them into box queries by using $X = \{a, t, c, g\}$ at the current suspicious position, use their results from the BoND-tree to form counting matrix (1), and apply the VMVR to determine if the possibly erroneous base at the current suspicious position is indeed an error. Once the current error is corrected in the read, we consider the next rightmost suspicious error position and correct its possibly erroneous base in a similar way. This process continues until all the possibly erroneous bases in the given read are verified and corrected. Note that, if the rightmost k-mer in the read contains more than one erroneous base, the process can start from the leftmost (or any) k-mer in the read that contains only one erroneous base and then expand the corrected region bigger and bigger. If no k-mer containing only one erroneous base can be found, we adopt the alignment method in the following subsection to correct the erroneous bases covered by the rightmost k-mer first and then apply the method in this subsection to correct the remaining erroneous bases (if any) in the read.

2.4 Alignment Strategy

As the condition at the end of Section 2.2 indicates, the VMVR does not work if the votes for two or more bases are close to the maximum vote. Fig. 5 shows such a scenario, in which segment δ (with length $2k-1$) covered by all the k suspicious shifted k-mers around the suspicious error position (with erroneous base t changed from correct base a) in read 2 also appears in read 1 that has no error. In such a case, every (shifted) suspicious k-mer in read 2 is repeated (as a trusted one) in read 1. Hence, the counts for the two places are combined/added, which are close to the coverage number. As a result, we are unable to determine whether a or t is correct at the suspicious position in read 2 since they have similar votes (e.g., $v(a) = 19$ and $v(t) = 20$). Although such a case does not normally occur (especially when k is large), we need to have a way in our DiskBQcor to handle such a case. The idea of an alignment based strategy to handle such a case is described as follows.

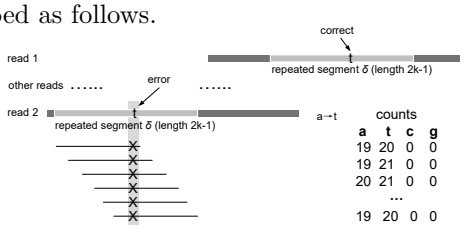

Figure 5: Reads with a long repeated segment.

For a given suspicious error position p in a read r that the VMVR cannot handle, we take a suspicious k-mer β that contains position p from read r and convert β into a box query q by replacing the possibly erroneous base[3] at position p in β by set $X = \{a, t, c, g\}$. From the result of q, we can find the set S of candidate reads that might be alignable with read r. Our goal is to identify those candidate reads in S that can indeed be aligned with r. After r is aligned with these candidate reads, the correct base at position p in r can be determined since most of them have the correct base at the position.

The problem now boils down to how to align the reference read r with a candidate read $r_1 \in S$. For this special alignment problem, we introduce an efficient method here to solve it in two stages (see Fig. 6). In the first stage, we try to match a (short) seed segment σ from r that contains the suspicious error position p with a (short) segment σ_1 from r_1. If such σ_1 can be found, the alignment moves to the second stage. Otherwise, reads r and r_1 are not alignable. In the second stage, we first place r and r_1 into their aligned positions according to the alignment of σ and σ_1. We then try to match the pairs of corresponding bases at the remaining positions from r and r_1. If the total number of mismatches between r and r_1 is within a tolerance, r and r_1 are aligned successfully. Otherwise, they are not alignable.

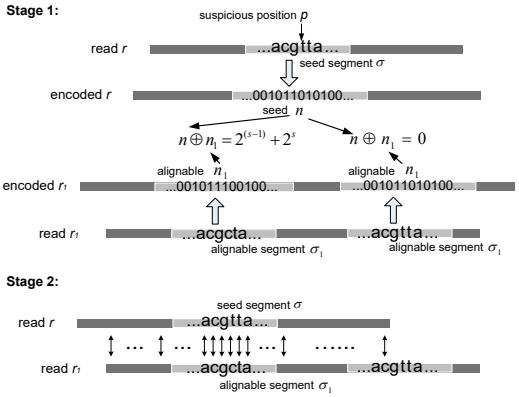

Figure 6: Two-stage Alignment Method.

To efficiently identify a segment(s) σ_1 from r_1 that is alignable with seed segment σ in r in the first stage, we use a binary encoding based technique. The key idea is described as follows. We first encode each of reads r and r_1 into a binary string/sequence by changing base "a" to "00", "t" to "01", "c" to "10", and "g" to "11" in the read. For example, a read "$aggctacgttaattga$" is converted into a binary sequence "00111110010010110101000001011100". This binary representation allows us to efficiently apply one integer (32 bits) operation to compare 16 bases together instead of using 16 separate base comparisons.

Let n be an integer (representing seed segment σ)

[3]If β contains multiple possibly erroneous bases, each of them is replaced by set X.

from r that contains the possibly erroneous base e (in the binary form) at the suspicious position p. Assume that e is represented by the s-th and $(s+1)$-th bits (counting from the right) in n. We then check if n is alignable with each integer n_1 shifted one base at a time from the left to the right in candidate read r_1[4]. If all the corresponding bits from n and n_1 match except that the s-th and/or $(s+1)$-th bits may mismatch, we say n and n_1 are alignable. Specifically, we perform an Exclusive OR ($\oplus$) operation on n and n_1 to test if they are alignable as follows:

$$n \oplus n_1 = \begin{cases} 0 & \Rightarrow \textit{ n and } n_1 \textit{ are alignable} \\ & (\textit{they match exactly}) \\ 2^{(s-1)} & \Rightarrow \textit{ n and } n_1 \textit{ are alignable} \\ & (\textit{they mismatch only at s-th bit}) \\ 2^s & \Rightarrow \textit{ n and } n_1 \textit{ are alignable} \\ & (\textit{they mismatch only at } (s+1)\textit{-th bit}) \\ 2^{(s-1)} + 2^s & \Rightarrow \textit{ n and } n_1 \textit{ are alignable} \\ & (\textit{they mismatch only at} \\ & \quad s\textit{-th and } (s+1)\textit{-th bits}) \\ \textit{otherwise} & \Rightarrow \textit{ n and } n_1 \textit{ are not alignable.} \end{cases}$$

In this criterion for alignability, we allow only one base mismatch at the suspicious error position p. If seed segment σ also contains another erroneous base(s) that is not at position p, the chance to find an alignable segment σ_1 in candidate read r_1 is small since this would require that σ_1 also happen to have the same erroneous base(s) at the relevant position(s) other than p. Thus, the alignment would fail at the first stage in such a case. To increase the chance to use a seed segment that meets the no-other-error requirement, we adopt the following strategy (see Fig. 7). We first use an integer n that represents a seed segment σ having position p in the middle. If n is found to be alignable with one or more integers in r_1, the alignment process moves to the second stage. Otherwise, we use an integer n' that represents a seed segment σ' having position p as its leftmost position. If n' is found to be alignable with one or more integers in r_1, the alignment process moves to the second stage. Otherwise, we use an integer n'' that represents a seed segment σ'' having position p as its rightmost position. If n'' is found to be alignable with one or more integers in r_1, the alignment process moves to the second stage. Otherwise, r and r_1 are considered to be not alignable.

Note that, if r has no other erroneous base at a position with a distance $\leq 16/2 = 8$ bases from position p, seed segment σ (i.e., n) will meet the above no-other-error requirement. If r has only one erroneous base at a position (other than position p) with a distance $\leq 16/2 = 8$ bases from p, seed segment σ' (i.e., n') or σ'' (i.e., n'') will meet the above no-other-error requirement. The above seed segment choosing strategy fails only when r has two or more erroneous bases at positions (other than position p) with a distance $\leq 16/2 = 8$ bases from p and each side of p has at least one of such erroneous bases. However, the chance for such a case to occur is very small.

[4]If an integer for a base b near the left/right boundary of r_1 does not have enough bases on the left/right side of b, we may pad it with relevant bases from n on the missing left/right side.

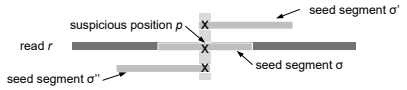

Figure 7: Strategy for Choosing Seed Segment.

In the second stage, for integer n_1 that is alignable with n (or n' or n''), we try to align reads r and r_1 according to the alignment of the segments represented by integers n and n_1. For each pair of corresponding bases from r and r_1 that are not covered by the segments represented by n and n_1, we compare them to see if they match. If the total number of mismatches is within a given tolerance, r and r_1 are aligned successfully. Otherwise, we try another n_1 (if any) that is alignable with n to see r and r_1 can be aligned using the new segment represented by n_1. If none of the integers from r_1 that are alignable with n leads to a successful alignment of r and r_1, reads r and r_1 are considered to be not alignable.

3 Experimental Results

To examine the performance of our disk based sequencing error correction method, DiskBQcor, we conducted extensive experiments. The performance was evaluated in terms of accuracy, efficiency and scalability. DiskBQcor was implemented in the C++ programming language. All the 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 performance data was measured based on the average from three executions.

The genome data used in the experiments was collected from *E. coli* 536 (GenBank: NC008253, 5.5 M). Different coverages with sequencing reads were tested in the experiments. Simulated 36 bp reads with an error rate at 0.5% were used. Different sets of overlapping k-mers for various k values were obtained from these reads for the experiments. For each error position p, all the shifted k-mers covering p are generated. For each shifted k-mer, its possibly erroneous base at p is replaced by $X = \{a, t, c, g\}$ to form the corresponding box query used for our error correction process.

The first set of experiments were conducted to evaluate the accuracy of our method. For each experiment, we measured the accuracy, i.e., the percentage of the errors that were properly corrected, from our method. We also measured the number of the errors that were mis-corrected and the number of the errors that were left uncorrected. Table 1 shows the results when k ranges from 12 to 16 and the coverage is fixed at 15. The results demonstrate that, in general, the accuracy of our method is increasingly better as k increases,

which is consistent with what is expected since a large k would reduce the chance for a k-mer to appear in multiple places, leading to a better determination of an error. $k = 15$ is the optimal length determined by Quake [6] for such experimental data. For this k, although the accuracy 99.87% of our method is better than the overall accuracy 90.5% of Quake, it is not a fair comparison since Quake also includes a step to determine the suspicious error positions. For those error positions that Quake tries to correct, it can correct 99.8% of them, which is comparable to the observed accuracy of our method. On the other hand, our accuracy covers all the error positions including those that Quake cannot handle such as the cases having repetitive suspicious k-mers. Hence, our method is very attractive for error correction when some suspicious error positions need to be checked. Furthermore, the experimental results also demonstrate that our method can yield a quite high accuracy even when k is relatively small (e.g., $k = 12$). Utilizing this advantage could help us reduce the space requirement by storing shorter k-mers, leading to an enhanced scalability for our method.

Table 1: Correction Accuracy for Simulated 36 bp *E. coli* with Coverage=15

k	Total errors	Correc -tions	Mis-correc -tions	Errors kept	Accuracy (%)
12	812764	786076	21	26667	97.00%
13	812869	806472	21	6375	99.21%
14	813890	811684	21	2185	99.73%
15	814122	813080	11	1031	99.87%
16	813816	813141	16	660	99.92%

To examine the impact of the coverage factor on the performance of our method, we conducted experiments with various coverages. Table 2 shows the results when the coverages are 10, 15, and 20 and k is fixed at 15. From the table, we can see that the coverage is not a significant factor affecting the performance of our method, which indicates that the VMVR is quite robust. For various coverages, the accuracy of our method remains almost steady. This advantage could help us further reduce the space requirement by adopting a low coverage genome dataset as input, which provides another way to further improve the scalability for our method.

Table 2: Correction Accuracy for Simulated 36 bp *E. coli* with k=15

Cover -age	Total errors	Correc -tions	Mis-cor -rections	Errors kept	Accuracy
10	541662	540672	4	985	99.82%
15	814122	813080	11	1031	99.87%
20	1084518	1083262	17	1239	99.88%

To evaluate the efficiency of our method, we recorded the relevant measures in our experiments. Table 3 shows the index creation time, the error correction time and the percentage of alignment cases when k ranges from 12 to 16 and the coverage is fixed at 15, while Table 4 shows the same measures when the coverage ranges from 10 to 20 and k is fixed at 15. From the tables, we can see that the error correction time for our method was relatively small, which indicates that the disk based BoND-tree can help achieve reasonable efficiency for our disk based sequencing error correction method. From the tables, we can also see that more alignment cases need to be handled when k became smaller. However, the error correction time, which includes alignment time, was still small even when a significant number of alignment cases (e.g., $k = 12$) were processed, which indicates that the binary encoding based alignment technique adopted in our method is efficient. From the tables, we can also see that, although the error correction itself did not take much time, the creation of the BoND-tree took significant amount of time. Note that the index creation time is bound to the efficiency of the index building algorithm given in [1]. To improve the index building efficiency, a bulk loading technique like those in [17, 11] could be developed for the BoND-tree. Alternatively, the index tree could be built at the time during DNA sequencing in a streaming fashion, which avoids a separate index tree building process. Once the index tree is built, our method could be implemented as a service for error verification and correction to efficiently process user's requests on checking some suspicious error positions in a genome sequence. The built index tree storing k-mers and relevant metadata could also be utilized to support other sequence analysis applications such as sequence alignment, terminus searching, and variant detection.

Table 3: Correction Time for Simulated 36 bp *E. coli* with Coverage=15

k	Creation Time (minutes)	Correction Time (minutes)	Alignments (%)
12	194.2	33.3	23%
13	216.0	20.0	6%
14	254.1	20.7	2%
15	264.0	21.3	0.6%
16	284.1	25.3	0%

Table 4: Correction Time for Simulated 36 bp *E. coli* with k=15

Cover -age	Creation Time (minutes)	Correction Time (minutes)	Alignments (%)
10	167.7	14.5	0.6%
15	264.0	21.3	0.6%
20	343.0	38.4	0.6%

We also conducted an experiment using reads with a larger size, i.e., 124 bases. The results are shown in Table 5. From the table, we can see that similar results were obtained when using a larger read size, which indicates that our method scales well with the read size.

Table 5: Experimental Results for Simulated 124 bp *E. coli* with $k = 15$ and Coverage=15

Total errors	Correc -tions	Mis-correc -tions	Errors kept	Accuracy (%)
820526	820266	5	255	99.97%
Creation Time (minutes)		Correction Time (minutes)		Alignments (%)
361		29.6		0.26%

To examine if our method would have similar performance on other genomic data, we conducted an

experiment using C. elegans chromosome I data (Genbank: NC003279.8, 15M) with read size = 124, k=16 and coverage=10. We observed that, for 1,491,895 sequencing errors in the reads, our method achieved an accuracy of 99.70%, which demonstrated a similar behavior. The error correction time and the index creation time were 207 and 937 minutes, respectively.

4 Conclusions

Current DNA sequencers suffer from the problem of having high random per-base error rates, which makes sequencing error correction a crucial task for many sequence analysis applications in bioinformatics. Existing sequencing correction techniques cannot scale well to large datasets due to the requirement on huge expensive computer memory space.

To overcome the drawback, we present a novel disk based sequencing error correction method, DiskBQcor, in this paper. The BoND-tree, which is a recently-developed index structure on disk [1], is utilized in our method to perform sequencing error verification and correction. Since input k-mers and their relevant medata are stored on disk with the BoND-tree that supports efficient processing of box queries, our method provides an efficient way for error verification and correction on disk in addition to the benefit of scalability warranted by inexpensive disk space.

Our future research includes incorporating base quality scores into our error correction method, integrating an effective error detection technique with our method, developing bulk loading and streaming building techniques for the BoND-tree, and applying our BoND-tree storing k-mers and their relevant metadata to other sequence analysis applications.

Acknowledgment: Research was supported by the US National Science Foundation (NSF) (under Grants #IIS-1320078 and #IIS-1319909). The preliminary results of this work were presented as a poster at ACM BCB'2015 [5].

References

[1] C. Chen, A. Watve, S. Pramanik, Q. Zhu. The BoND-tree: an efficient indexing method for box queries in nonordered discrete data spaces. *IEEE Trans. on Knowl. and Data Eng.*, 25(11):2629–2643, 2013.

[2] P. E. Compeau, P. A. Pevzner, *et al.* How to apply de bruijn graphs to genome assembly. *Nature Biotechnol*, 29(11):987–991, 2011.

[3] S. Deorowicz, M. Kokot, *et al.* KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.

[4] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5):1792–1797, 2004.

[5] Y. Gu, X. Liu, Q. Zhu, Y. Dong, C. T. Brown, and S. Pramanik. A new method for DNA sequencing error verification and correction via an on-disk index tree. *Proc. of ACM BCB'15*, pp. 503–504, 2015.

[6] D. R. Kelley, M. C. Schatz, *et al.* Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.

[7] S. Kurtz, A. Narechania, *et al.* A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, 9(1):517, 2008.

[8] Marçais, Guillaume, and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

[9] M. Metzker. Sequencing technologies - the next generation. *Nat Rev Genet*, 11:31–46, 2010.

[10] J. R. Miller, A. L. Delcher, *et al.* Aggressive assembly of pyrosequencing reads withmates. *Bioinformatics*, 24(24):2818–2824, 2008.

[11] G. Qian, H.-J. Seok, Q. Zhu, and S. Pramanik. Space-partitioning-based bulk-loading for the NSP-tree in non-ordered discrete data spaces. *Proc. of DEXA'08*, pp. 404–418, 2008.

[12] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach. *ACM Trans. on Database Syst.*, 31(2):439–484, 2006.

[13] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces. *ACM Trans. on Info. Syst.*, 23(1):79–110, 2006.

[14] G. Rizk, D. Lavenier, and R. Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.

[15] L. Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, 2010.

[16] J. Schroder, H. Schroder, *et al.* SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

[17] H.-J. Seok, G. Qian, Q. Zhu, A. Oswald, and S. Pramanik. Bulk-loading the ND-tree in non-ordered discrete data spaces. *Proc. of DASFAA'08*, pp. 156–171, 2008.

[18] H. Shi, B. Schmidt, *et al.* A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware. *J. Comput. Biol.*, 17(4):603–15, 2010.

[19] L. Song, L. Florea, and B. Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biology*, 15:509, 2014.

[20] Treangen, J. Todd, and S. L. Salzberg. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics*, 13(1):36–46, 2012.

[21] Q. Zhang, S. Awad, and C. T. Brown. Crossing the streams: a framework for streaming. *PeerJ PrePrints*, 2015.

[22] Q. Zhang, J. Pell, *et al.* These are not the k-mers you are looking for: Efficient online k-mer counting using a probabilistic data structure. *PLoS ONE*, 9(7):e101271, 2014.