An Efficient Hybrid Approach to Correcting Errors in Short Reads

Zhiheng Zhao, Jianping Yin, Yong Li, Wei Xiong, and Yubin Zhan

School of Computer, National University of Defense Technology, 410073 Changsha, China
wader_zzh@hotmail.com

Abstract. High-throughput sequencing technologies produce a large number of short reads that may contain errors. These sequencing errors constitute one of the major problems in analyzing such data. Many algorithms and software tools have been proposed to correct errors in short reads. However, the computational complexity limits their performance. In this paper, we propose a novel and efficient hybrid approach which is based on an alignment-free method combined with multiple alignments. We construct suffix arrays on all short reads to search the correct overlapping regions. For each correct overlapping region, we form multiple alignments for the substrings following the correct overlapping region to identify and correct the erroneous bases. Our approach can correct all types of errors in short reads produced by different sequencing platforms. Experiments show that our approach provides significantly higher accuracy and is comparable or even faster than previous approaches.

Keywords: High-throughput sequencing, Error correction, Suffix array, Multiple Alignments.

1 Introduction

High-throughput sequencing technologies such as Illumina's Genome Analyzer, ABI's SOLiD, and Roche's 454, e.g. [1] open up a range of new opportunities for genome research. Unlike the Sanger method, high-throughput sequencing technologies can produce a large amount of short reads in a single run. For example, the Illumina Genome Analyzer IIx can currently generate an output of up to 640 million paired-end reads in a single run with a read length between 35 and 150. This leads to many novel applications such as genome re-sequencing, de novo genome assembly and metagenomics. However, high-throughput sequencing data is more error-prone than the Sanger sequencing method. With a significant impact on the accuracy of applications such as re-sequencing and de novo genome assembly, sequencing errors have become one of the major problems in analyzing high-throughput sequencing data.

It is a difficult task to correct errors in high-throughput sequencing data, for the computational demands for large-scale short reads limit the performance of error correction algorithms and tools. The intuitive error correction method

V. Torra et al. (Eds.): MDAI 2011, LNAI 6820, pp. 198–210, 2011.

is multiple read alignments, such as MisEd [2] for Sanger reads. If the reads could be aligned correctly, we could correct the erroneous bases which are in the minority in each column. However, multiple read alignments are extremely compute-intensive for large-scale reads and do not adapt well to short reads. Hence, some algorithms were proposed based on heuristics and alignment-free methods to determine which reads align to the same genomic position. Pevzner *et al.* [3] formulated the error correction problem as a spectral alignment problem (SAP), in which k-mers are divided into solid (correct) and insolid (erroneous) according to their multiplicity and the insolid k-mers are corrected using a minimum number of edit operations to solid k-mers until all reads only contain solid k-mers. Most of previous error correction methods are based on the SAP and use heuristics to approximate the SAP. Pevzner *et al.* [3] used a simple greedy heuristics to solve the SAP. Subsequently, Chaisson *et al.* [4] proposed a dynamic programming algorithm for the SAP and implemented a heuristic algorithm based on an approximation to dynamic programming algorithm [5] in assembly tool Euler-SR similarly with Butler *et al.* [6]. Recently, some tools which optimize the k-mers classification [7, 8] or accelerate error correction using GPU [9] are also based on the SAP.

However, because all k-mers in the SAP are independent, we cannot utilize the local context of a k-mer in the sequencing reads to identify errors. The more repetitive the genome is, the greater the chance is that a sequencing error will merely change one solid k-mer to another solid k-mer, hiding the error [8]. Shrec [10] proposes a different idea for error correction which expands the local context of erroneous bases. It first searches the common correct substrings in all reads and then identifies the erroneous bases following these substrings. Shrec is based on a generalized suffix trie data structure that holds all short reads and corrects errors with a majority voting scheme. However, it requires huge memory and therefore it is difficult to be used for large-scale read data. Shrec was extended by Salmela [11] to a mixed set of reads from various sequencing technologies, with different read lengths and error characteristics. HiTEC [12] adopts the similar idea, while using the suffix array data structure instead of suffix trie. The suffix array is more memory efficient than suffix trie and HiTEC is based on a thorough statistical analysis. This makes HiTEC more accurate and efficient. However, HiTEC can only correct substitutions in short reads with identical read lengths.

In this paper, we propose an efficient hybrid approach which is based on an alignment-free method combined with multiple alignments. Our approach includes two stages. In the first stage, we construct suffix arrays on all short reads to search the correct overlapping regions as HiTEC. Each overlapping region contains a common substring shared by some reads. If the number of the reads and the length of the substring are large enough, we can consider that the common substring is from a unique genomic position and has no errors.For the reads contained in each correct overlapping region, we can consider the common substring is an anchor in the multiple aligments. Therefore, in the second stage, we

only form multiple alignments for the substrings following the common substring to identify and correct the erroneous bases.

This strategy makes our approach differentiate the alignment-based methods which require $O(m^3 + m^{2*}l_{ave}^2)$ time (m is the number of reads and l_{ave} is the average length of reads), for our approach does not form multiple alignments of all the entire reads. The worst case time complexity of our approach is $O(ml_{ave}{*}log(ml_{ave}){+}\ ml_{ave}{*}(l_{ave}\ {-}\ \gamma)^2)$ in which γ is the parameter to determine the correct overlapping region. For short reads our approach is comparable or even faster than all published approaches. Additionally, our approach is also different from Shrec and HiTEC, while they only correct errors directly following the common substring. Furthermore, the multiple alignments can be adjusted by the user-defined gap penalty and mismatch penalty. Hence, our approach can correct all types of errors in short reads produced by different sequencing platforms. Experiments show that our approach provides significantly higher accuracy than previous methods.

The rest of this paper is organized as follows. In section 2, we introduce the ideas and the methods used in our approach. We present the algorithm of our approach in section 3. In Section 4 we evaluate the performance of our approach. Finally, Section 5 concludes the paper.

2 Methods

2.1 Problem

We first give the error correction problem for our approach formally. Supposed that the reads are produced from a genome G and the length of G is N. If the sequence of G is unknown, we can use a reference genome instead of G. The genome G and the reads can be considered as strings over the alphabet {A, C, G, T, N}. Supposed the sequencing platforms produce m reads, and the length of i read is l_i, let $r{=}c_0,c_1,\ldots,c_{n-1}$ be a read of length n. $r[i,j]$ is the substring $c_i,\ c_{i+1},\ldots,\ c_j$, in which $0{\leq}i{\leq}j{\leq}n$. The reverse complement $\overline{r}$ of r, is obtained by first reversing s and then applying the transformation A $\leftrightarrow$ T; C $\leftrightarrow$ G. For example, if r = ACTG, then $\overline{r}$ = CAGT. if r is produced from the substring g of G that $g = G[j,\ k]$, we say that r maps to g. We can obtain the combined total length of the reads denoted by M, $M = \sum_{i=0}^{m} l_i$. We use *coverage* denoting the expected number of times that a position in the genome is sequenced, so *coverage* $= M/N$. All error correction methods rely on the coverage of the reads being moderately high so that every position of the genome is sequenced several times with high probability. Reads from low coverage regions cannot be corrected because there is insufficient data to infer the correct sequence. We use p denoting the per-base error rate, so for a read with the length l, the expected error bases in the read is $l{*}p$.

For a reads set $r_0,r_1,\ldots,r_{m-1}$, supposed these reads map to the substrings of G which are $g_0,g_1,\ldots,g_{m-1}$, the task of an error correction algorithm is to covert the reads $r_0,r_1,\ldots,r_{m-1}$ to $r_0^*,r_1^*,\ldots,r_{m-1}^*$ using the edit operations, making D

$= \sum_{i=0}^{m} \mid g_i - r_i^* \mid$ as small as possible. There are three types of errors: substitutions, insertions and deletions. The distribution of error types varies from one sequencing platform to another. For instance, the Roche/454 sequencing platform produces reads with insertions and deletions, due mainly to homopolymers, whereas the SOLiD and Illumina platforms are prone to substitution errors. Hence, $\mid g_i - r_i^* \mid$ can denote the Hamming distance allowing only substitutions or edit distance allowing also insertions and deletions between g_i and r_i^* according to the error characteristics of the sequencing platforms.

2.2 Solution

If the reads have been aligned correctly to the genome, we can identify and correct the errors by checking each column of the alignments. However, as the analysis in the first section, most of time we cannot obtain the correct alignments for the computational complexity and the accuracy of multiple alignments of large-scale reads. We suppose that the reads can map to a genome or a reference genome. Then, there are many substrings of the reads map to a identical segment of the genome for the high coverage of reads. We define these substrings as an overlapping region if they are identical. For a common substring S contained in an overlapping region, let *L(S)* denote the length of the common substring and *H(S)* denote the number of the reads across the overlapping region. Actually, a common substring is corresponding to a k-mer in the SAP and is similar to the *witness* in HiTEC and the path from the root to a node in the suffix trie in Shrec. Figure 1 shows an example of an overlapping region.

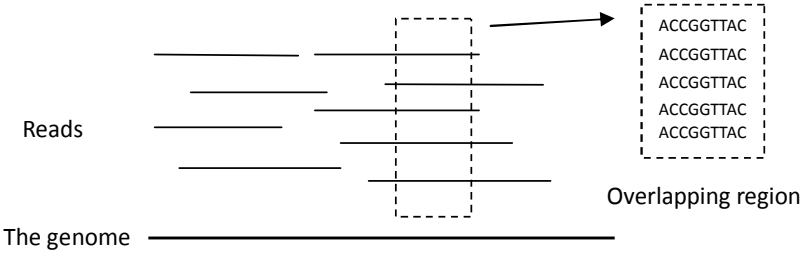

Fig. 1. An example of an overlapping region. The common substring contained in the overlapping region is S = ACCGGTTAC. *L(S)* = 9, *H(S)* = 5.

If the common substring of an overlapping region is unique , we can obtain the alignment between the reads across the overlapping region. However, the random occurrences and repeats in the genome will make a common substring identical to another one. The larger the length of the common substring is, the smaller the probability that the common substring has random occurrences or repeats. On the other hand, for a common substring S, the lager *L(S)* is, the smaller *H(S)*. If the *H(S)* is small enough, we have insufficient data to infer

whether the common substring is correct. Hence, we use two parameters γ and δ. If $L(S) = \gamma$ and $H(S) > \delta$, we define the common substring S as a correct common substring and the overlapping region as a correct overlapping region.

For the reads across a correct overlapping region, we then form multiple alignments and correct errors for the strings following the common substring. We retrieve the consensus of the multiple alignments and then check each column to correct the bases which are not identical the consensus. From the figure 2, we can see that our approach can correct mixed errors in short reads. We adjust the gap penalty and the mismatch penalty of multiple alignments so that our approach can adapt to the short reads produced from different sequencing platforms or mixed short reads.

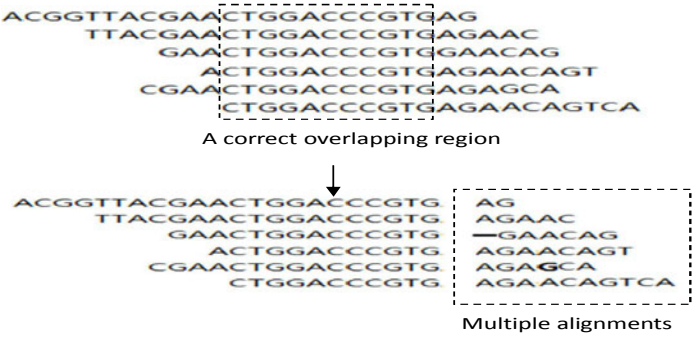

Fig. 2. Forming multiple alignments and correcting errors for the strings following the common substring in the reads across a correct overlapping region. From the multiple alignments, we can see that the third read has a deletion error and the fifth read has a substitution error.

3 Algorithm

Our approach has two stages: searching the correct overlapping regions, forming multiple alignments and correcting errors. We will present each stage detailedly as well as the full algorithm and the computational complexity.

3.1 Searching the Correct Overlapping Regions

We use the suffix array data structure to search the correct overlapping regions. We first give the definition of the suffix array. For a alphabet Σ, $\mid \Sigma \mid$ is the size of alphabet. A string is any finite sequence over Σ. Let $T{=}c_0,c_1,\ldots,c_{n-1}$ be a string of length n. $T[i,j]$ is the substring c_i, $c_{i+1},\ldots,$ c_j, in which $0{\leq}i{\leq}j{\leq}n$. We add a special character "$\$$" at the tail of the string T and denote the suffix of T at the position of i by $S_i{=}T[i,n]{=}c_i$ $,c_{i+1},\ldots,c_{n-1},\$$. $\$$ is smaller than any character in Σ. The suffix array of T on the alphabet Σ, which we denote as SA, is an array of length n containing the positions of string T such that $S_{SA[i]}$ gives

the increasing lexicographical order, i.e., $S_{SA[0]}<S_{SA[1]}< \ldots <S_{SA[n-1]}$. Besides suffix array, the longest common prefix (LCP) array is often used to compute the length of the longest common prefix between suffixes. LCP[i] is the length of the longest common prefix of $S_{SA[i]}$ and $S_{SA[i-1]}$. The suffix array data structure has been proposed by Manber and Myers [13], and the suffix array construction algorithm can be seen from [14] for a survey.Figure 3 shows the suffix array and LCP array of string ACCTATACCGTA.

i	$SA[i]$	$S_{SA}[i]$	$LCP[i]$
0	12	$	0
1	11	A$	0
2	6	ACCGTA$	1
3	0	ACCTATACCGTA$	3
4	4	ATACCGTA$	1
5	7	CCGTA$	0
6	1	CCTATACCGTA$	2
7	8	CGTA$	1
8	2	CTATACCGTA$	1
9	9	GTA$	0
10	10	TA$	0
11	5	TACCGTA$	2
12	3	TATACCGTA$	2

Fig. 3. The suffix array and LCP array of string ACCTATACCGTA

We also use the libdivsufsort library of Yuta Mori [15] which is a fast and lightweight suffix array construction algorithm to compute the full suffix array of R as HiTEC. $R = r_1\$\overline{r_1}\$r_2\$\overline{r_2}\$\ldots\$r_m\$\ \overline{r_m}\$$. The algorithm requires O($nlogn$) time in which n is the length of R. $n = 2^*(M + m)$. If $n< 2^{32}$, it requires $5n$+O(1) bytes space for suffix array.

With the suffix array of R, we can get all the correct overlapping regions. We first computing the LCP array of R. Because each read in R has been separated by $, we use an variation of LCP array called LCP_$ array instead of LCP array, for we only need to compute the longest common prefix before any $ of $S_{SA[i]}$ and $S_{SA[i-1]}$ for LCP_$[$i$]. The LCP_$ array also be computed by the algorithm of Kasai *et al.* [16] in O(n) time and space. Then, we scan the LCP_$ array. If LCP_$[$i$ + j] $\geq \gamma$ for j = 1,2,...,k and $k \geq \delta$, $0 \leq i < n$, the γ-long prefix of $S_{SA[i]}$,$S_{SA[i+1]}$,...,$S_{SA[i+k]}$ is a correct overlapping region.

Constructing the suffix array needs O($nlogn$) in time. Computing the LCP_$[$i$] array and searching the correct overlapping regions both need O(n) in time. So this stage requires O($nlogn$) in time and $6n$+O(1) bytes space.

3.2 Forming Multiple Alignments and Correcting Errors

We form the multiple alignments for each correct overlapping region in the second stage. Supposed that there are k reads across the correct overlapping region

which are r_{i_1}, r_{i_1},...,r_{i_k}, and for the read r_{i_j} ($0 \leq j \leq k$),the common substring ends at p_j. Hence, we need to form multiple alignments for the string set $S = \{r_{i_1}[p_1,l_{i_1}],r_{i_2}[p_2,l_{i_2}], \ldots, r_{i_k}[p_k,l_{i_k}]\}$, in which l_{i_j} is the length of the read r_{i_j}.

Next we compute the consensus $r_{consensus}$ of S using the majority voting scheme. For the i-th column of S, let occurrences(x)($x \in$ {A,C,G,T}) represents the occurrences of x. If $\exists x_i$ occurrences(x_i) $> h/2$ and occurrences(x_i) ≥ 2 (h is the number of non-null elements in the i-th column of S), $r_{consensus}[i] = x_i$; otherwise, $r_{consensus}[i]$ = N. If $r_{consensus}[j] \in$ {A,C,G,T} and the later characters of $r_{consensus}[j]$ are all N, we truncate the end of $r_{consensus}$ from the j-th character and set the length of $r_{consensus}$ to j+1;

Then we align each string in S to the consensus using the Needleman-Wunsch algorithm[17]. We allow free gaps both for the string and the consensus if the users choose gapped alignment for substitution errors and indel errors. Otherwise we disallow free gaps only for the substitution errors. For the gapped alignment, we do not count the gap penalty at the end of the string and the consensus for they need to be aligned from the left. If the score of the alignment exceeds the threshold, we correct the string according to the consensus. Otherwise, we consider the string is the substring of a read which is a random occurrence or a repeat of the genome and skip the string.

It seems that we only correct errors in the "right" region of each correct overlapping region. Actually, because we search the correct overlapping regions from the suffix array which is constructed from all the reads and their reverse complement. For each correct overlapping region Γ, we will find the corresponding correct overlapping region $\overline{\Gamma}$ of its reverse complement and correct errors in the "right" region of $\overline{\Gamma}$. The "right" region of $\overline{\Gamma}$ is corresponding to the "left" region of Γ, hence our approach forms multiple read alignments in fact.

There may exist inconsistency when a read traverses across two or more correct overlapping regions. To address this problem, we use $left_i$ and $right_i$ to keep the correct segment of read r_i. We set $left_i = 0$ and $right_i = 0$ initially. In the second stage, when we find r_i across correct overlapping region for the first time, we first set $right_i$ to the right end position of the common substring of r_i. Then, if the string $r_i[right_i, l_i]$ has p-long prefix aligned to the consensus in the multiple alignments, we set $right_i = right_i + p$. Similarly, when we find the reverse complement $\overline{r_i}$ of read r_i across correct overlapping region for the first time, we first set $left_i$ to the length of the right segment beyond the common substring of $\overline{r_i}$. Then, if the string $\overline{r_i}[left_i, l_i]$ has q-long prefix aligned to the consensus in the multiple alignments, we set $left_i = left_i$ - q. We consider that the substring $r_i[left_i, right_i]$ is the corrected segment of read r_i. Therefore, when we correct read r_i in another multiple alignment, if the suspicious bases are in the substring $r_i[left_i, right_i]$, we ignore them simply. Otherwise, we correct them and update $left_i$ and $right_i$.

We give the detailed algorithm of the second satge in Algorithm 1.

Algorithm 1. align_correct()

given: a correct overlapping region Γ, reads r_1,r_1,...,r_m, the length of r_i is l_i for $1 \leq i \leq m$, *left*[m] and *right*[m]
output: the corrected bases, *left*[m] and *right*[m]

obtain the reads r_{i_1}, r_{i_2},...,r_{i_k} across Γ
compute the end position p_j of the common substring of r_{i_j}, for $1 \leq j \leq k$
obtain the string sets $S = \{r_{i_1}[p_1,l_{i_1}],r_{i_2}[p_2,l_{i_2}], \ldots, r_{i_k}[p_k,l_{i_k}]\}$
for each column i in S **do**
    $r_{consensus}[i]$ = N
    compute occurrences(A),occurrences(C),occurrences(G),occurrences(T)respectively
    h ← the number of non-null characters in the column
    if $\exists x$(occurrences(x) > $h/2$ and occurrences(x)≥ 2) ($x \in$ {A,C,G,T}) **then**
        $r_{consensus}[i] = x$
    end if
end for
truncate $r_{consensus}$
for each string r_i in S **do**
    score = align(r_i,$r_{consensus}$)
    if score < ε(min(l_i, $l_{consensus}$))(ε is an user-defined threshold) **then**
        continue
    else
        Get all the suspicious bases of r_i
        for each suspicious base p **do**
            if p is included in r_i[$left_i$, $right_i$] **then**
                continue
            else
                correct(p)
                update $left_i$ and $right_i$
            end if
        end for
    end if
end for

3.3 Choosing Parameters

Now we present how to choose the parameters γ and δ. The larger γ and δ, the smaller the probability that the correct overlapping region has random occurrences or repeats. However, we cannot set γ and δ too large because it will miss many errors. We use the same method as Quake [8] to determine γ. For a γ-long string, the expected occurrences in the genome G which length is N are $2N/4^{\gamma}$,hence, we use the following formula to determine γ.

$$\frac{2N}{4^{\gamma}} \approx 0.01$$

$$\gamma \approx \log_4 200N \tag{1}$$

If there are no errors, the expected occurrences of a γ-long string in the reads is *coverage*. For the p per-base error rate, the expected occurrences of a γ-long string without error is

$$occurrences(\gamma) = coverage - coverage(1 - (1 - p)^{\gamma})$$

To increase the discriminative power for identifying the location of errors in the reads with low coverage, we set δ as equation 2.

$$\delta = \frac{occurrences(\gamma)}{2} = \frac{coverage - coverage(1 - (1 - p)^{\gamma})}{2} \quad (2)$$

For the reads produced from an approximately 5 Mbp genome such as *E. coli*, if the coverage is 70, the per-base error rate is 1%, we will set γ to 15 and δ to 31.

3.4 The Full Algorithm and Complexity

Now we can give the full algorithm and the computational complexity of our approach. The full algorithm is presented in Algorithm 2.

Algorithm 2. The full algorithm of our approach

given: reads r_1,r_1,...,rm, the length of r_i is l_i for $1 \leq i \leq m$, N, p and *coverage*
output: the corrected reads

```
1: compute γ and δ
2: initialize the arrays left[m] and right[m]
3: constructed R and compute SA and LCP_$ array
4: scan LCP_$ array to search the correct overlapping regions
5: for each correct overlapping regions Γ do
6:    align_correct()
7: end for
```

Let l_{ave} be the average length of the reads. Constructing the suffix array needs O($MlogM$) time and computing LCP_$ array needs O(M) time. There are M/δ correct overlapping regions. For each correct overlapping regions, computing the consensus needs O(δ*(l_{ave} - γ)) and using the Needleman-Wunsch algorithm to align each string to the consensus needs O(δ*(l_{ave} - γ)^2), so the time complexity of align_correct function is O(δ*(l_{ave} - γ)^2). Hence, the worst case time complexity of our approach is O($MlogM$ + M*(l_{ave} - γ)^2) which is also O(ml_{ave}*$log(ml_{ave})$+ ml_{ave}*(l_{ave} - γ)^2). Our approach is more efficient than the alignment-based methods for large-scale short reads and is comparable or even faster than the alignment-free methods. Our approach requires only $6M$+O(1) bytes space which makes it memory-efficient.

4 Evaluation

In this section, we evaluate the performance of our approach. We use the simulation data sets which are created from several bacterial genomes as previous programs. These genomes can be downloaded from GenBank under the accession

numbers. We create two read data sets: S1 and S2. S1 is a read data set for evaluating our approach compared to alternative approaches (SHREC [10], HITEC [12], and Quake [8]) with only substitution errors, for SHREC and HITEC can only correct substitutions. S2 is a read data set for evaluating our approach compared to Ext-SHREC [11] and Quake with all types of errors (Ext-SHREC does the same work as SHREC when the reads have the same length and only have substitution errors). The datasets used for our performance evaluation are summarized in Table 1.

Table 1. Datasets used for performance evaluation.The reads in S1 only contain substitution errors. The reads in S2 contain all types of errors with the same probability for each type of errors and the length of reads varies from 60 bps to 120 bps.

Dataset	ID	Reference genome(GenBank)	Genome length(MB)	Error rate(%)	coverage	length of reads (bp)
S1	A1	NC_001139	1.1	1	70	70
	A2			2		
	A3			3		
	B1	NC_007146	1.9	1		70
	B2			2		
	B3			3		
	C1	NC_003923	2.8	1		70
	C2			2		
	C3			3		
	D1	NC_000913	4.7	1		70
	D2			2		
	D3			3		
S2	E1	NC_003923	2.8	1	70	60∼120
	E2			2		
	E3			3		
	F1	NC_000913	4.7	1		60∼120
	F2			2		
	F3			3		

We also use the *accuracy* to evaluate the performance of the algorithm we measured as HITEC [12]. The accuracy is defined as the ratio between the number of corrected reads and the number of initially erroneous reads. We use err_{bef} denoting the number of erroneous reads before correction and err_{aft} denoting the number of erroneous after correction. Then,

$$accuracy = \frac{err_{bef} - err_{aft}}{err_{bef}}$$

If we denote the number of erroneous reads that are corrected, correct reads that are left unchanged, correct reads that are wrongly changed, and erroneous reads that are left unchanged by TP, TN, FP, FN (true/false positive/negative) respectively, we have $err_{bef} = TP + FN$, $err_{aft} = FP + FN$ and therefore

$$accuracy = \frac{TP - FP}{TP + FN}$$

We called our approach MyHybrid for short. The tests shown in Table 2 and Table 3 were performed on a desktop computer with Intel Xeon E5420 4-core processor at 2.50GHz, 8GB RAM, running RHEL 5 x86_64 server. All algorithms to be compared use the default parameters. We also evaluate the time and memory required by the algorithm we measured in Table 4. Note that the test runs on a 64bit Linux, the memory used by the measured programs is almost two times as many as that running on 32bit operating systems.

From the tests, we can see that the accuracy of our approach is comparable to HiTEC and Quake for the data set S1 which only contains substitution errors. With various error rates, our approach performs more steadily than the other three programs. For the data set S2 which contains mixed errors, our approach is more efficient than Ext-SHREC and Quake. This makes our approach more available for the real read data which has complex error characters. Furthermore, in addition to obtaining very high accuracy, our approach has also very good time and space complexities. Our approach outperforms HiTEC and SHREC and is approximate to Quake on computational performance.

Table 2. Accuracy comparison for the data set S1

dataset	Accuracy(%)			
ID	SHREC	Quake	HiTEC	MyHybrid
A1	95.12	97.45	98.79	97.62
A2	87.04	96.38	97.60	97.04
A3	79.75	92.67	94.39	96.56
B1	92.64	98.49	99.03	98.25
B2	83.77	94.00	98.54	98.30
B3	64.30	91.78	96.27	97.16
C1	90.42	96.41	99.24	98.48
C2	73.08	95.33	97.73	98.06
C3	58.33	91.92	94.55	96.57
D1	88.05	97.17	98.62	98.40
D2	72.64	94.43	95.95	97.03
D3	57.95	92.80	92.16	94.33

Table 3. Accuracy comparison for the data set S2

dataset	Accuracy(%)		
ID	Ext-SHREC	Quake	MyHybrid
E1	85.02	94.69	93.97
E2	74.80	89.06	92.14
E3	60.75	85.33	89.36
F1	80.64	95.64	92.02
F2	68.31	90.47	90.35
F3	55.42	87.66	89.98

Table 4. Time and space comparison between SHREC, Quake, HiTEC and our approach for the data set S1

dataset	Time(s)				Memory(MB)			
ID	SHREC	Quake	HiTEC	MyHybrid	SHREC	Quake	HiTEC	MyHybrid
A1	1651	244.7	257.6	138.7	3002	448	1408	1462
A2	2540	265.9	386.4	142.2	3014	456	1408	1462
A3	3789	307.5	579.6	149.0	3528	440	1408	1462
B1	2517	278.6	478.9	205.7	3206	510	2476	2520
B2	3861	380.4	714.3	238.4	3890	518	2476	2520
B3	4960	421.9	1075.0	244.5	5742	526	2476	2520
C1	4033	346.1	785.5	294.8	3970	1048	3652	3738
C2	5580	381.0	1174.8	321.9	4864	1082	3652	3738
C3	7842	474.2	1767.3	345.6	6004	1124	3652	3738
D1	5947	429.3	1529.4	976.0	6190	1060	6014	6184
D2	8612	468.0	1911.7	1065.4	7100	1176	6014	6184
D3	14960	502.4	3441.1	1092.8	7452	1248	6014	6184

5 Conclusion

In this paper, we propose a novel and efficient hybrid approach for correcting errors in short reads. Our approach can correct all types of errors in short reads produced by different sequencing platforms. Experiments show that our approach provides significantly higher accuracy and is comparable or even faster than previous approaches.

References

1. Mardis, E.R.: The impact of next-generation sequencing technology on genetics. Trends Genet. 24, 133–141 (2008)
2. Tammi, M.T., Arner, E., Kindlund, E., Andersson, B.: Correcting errors in shotgun sequences. Nucleic Acids Res. 31, 4663–4672 (2003)
3. Pevzner, P.A., Tang, H., Waterman, M.S.: A new approach to fragment assembly in DNA sequencing. In: RECOMB 2001, pp. 256–267 (2001)
4. Chaisson, M.J., Pevzner, P.A., Tang, H.: Fragment assembly with short reads. Bioinformatics 20, 2067–2074 (2004)
5. Chaisson, M.J., Brinza, D., Pevzner, P.A.: De novo fragment assembly with short mate-paired reads: Does the read length matter? Genome Res. 19, 336–346 (2009)
6. Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I.A., Belmonte, M.K., Lander, E.S., Nusbaum, C., Jaffe, D.B.: ALLPATHS: De novo assembly of whole-genome shotgun microreads. Genome Res. 18, 810–820 (2008)
7. Yang, X., Dorman, K.S., Aluru, S.: Reptile: representative tiling for short read error correction. Bioinformatics 26, 2526–2533 (2010)
8. Kelley, D., Schatz, M., Salzberg, S.: Quake: quality-aware detection and correction of sequencing errors. Genome Biology 11(11), R116 (2010)
9. Shi, H., Schmidt, B., Liu, W., Muller-Wittig, W.: A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware. J. Comput.Biol. 17, 603–615 (2009)

10. Schroder, J., Schroder, H., Puglisi, S.J., Sinha, R., Schmidt, B.: SHREC: a short-read error correction method. Bioinformatics 25, 2157–2163 (2009)
11. Salmela, L.: Correction of sequencing errors in a mixed set of reads. Bioinformatics. 26(10), 1284–1290 (2010)
12. Ilie, L., Fazayeli, F., Ilie, S.: HiTEC: accurate error correction in high-throughput sequencing data. Bioinformatics 27(3), 295–302 (2011)
13. Manber, U., Myers, G.: Suffix arrays: a new method for on-line search. SIAM J. Comput. 22(5), 935–948 (1993)
14. Simon, J., Puglisi, W.F., Smyth, A.: A taxonomy of suffix array construction algorithms. ACM Comput. Surv. 39(2), 1–31 (2007)
15. Mori, Y.: Short description of improved two-stage suffix sorting algorithm, `http://homepage3.nifty.com/wpage/software/itssort.txt`
16. Kasai, T., Lee, G.H., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
17. Needleman, S.B.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)