Arab Academy for Science, Technology and Maritime Transport

(AAST)

College of Computing and Information Technology

Computer Science Department

DNA Sequencing Error Correction Algorithms

A Thesis

Submitted in Partial Fulfillment to the Requirements for the

Master of Computer Science

Submitted by:

Salma Mahmoud Mohamed Gomaa

Under Supervision of

Prof.Dr.Yasser El-sonbaty
Professor of Computer Science
Arab Academy for Science and Technology and Maritime Transport

Dr.Nahla A. Belal
Assistant Professor of Computer Science
Arab Academy for Science and Technology and Maritime Transport

June - 2017

الاكاديمية العربية للعلوم والتكنولوجيا والنقل البحرى

كلية الحاسبات و تكنولوجيا المعلومات

قسم علوم الحاسب

خوارزميات لتصحيح الأخطاء الناتجة عن عملية تسلسل الحمض النووي

الرسالة

مقدمة لإستكمال المتطلبات المرجوة في

ماجستير علوم الحاسوب

مقدمة من:

سلمى محمود محمد جمعة

تحت إشراف

أ. د. ياسر علاء الدين السنباطى
أستاذ بكلية الحاسبات وتكنولوجيا المعلومات الاكاديمية العربية للعلوم والتكنولوجيا والنقل البحرى

د. نهلة أحمد بلال
أستاذ مساعد بكلية الحاسبات وتكنولوجيا المعلومات الاكاديمية العربية للعلوم والتكنولوجيا والنقل البحرى

يونيو - ٢٠١٧

Abstract. The Next-Generation sequencing technologies produce large sets of short reads within a very efficient time. But, as a side effect of this fastness, the produced short reads are subject to be infected by different types of errors on the nucleotide level; such as nucleotide erroneous substitution, nucleotide erroneous insertion and nucleotide erroneous deletion. These errors represent a great obstacle to utilize the reads in sequencing projects (such as: reads assemblers, genome aligners, etc.). Consequently, the error correction process becomes a vital and essential action that aims to reduce the error rate over the whole reads set. So, the correction of all types of the nucleotides errors becomes very challenging. H-RACER is the new error correcting tool that fix all types of nucleotide errors (substitutions, insertions, and deletions) in a mixed set of reads. It mainly depends on RACER algorithm in detecting the erroneous nucleotide and deciding its corrective value. While, the major advantage presented by H-RACER is the detection of the nucleotide corrective action and its appliance. So, H-RACER is able to correct the nucleotides substitutions errors as well as their insertions and deletions errors with the highest accuracy and the least time compared to other existing algorithms that specialize in correcting these types of nucleotides errors.

Keywords: DNA Sequencing·k-mer·Error Correction·Substitution·InDels

1 Problem Definition

Although the DNA next-sequencing generation is cheap and able to produce many reads within a relatively short time, but unfortunately it produces reads with short length and also low accuracy (which represents a vital factor in all of the DNA reads processes). In other words, it generates many short reads with many errors with different types (substitution, insertion and deletion).

2 Objective

Since the DNA reads accuracy is a very essential and vital factor for all of the process that can be applied on these reads, so the major targets of this thesis are:

1. Raising the DNA reads accuracy.
2. Correcting all of the different types of the DNA errors (substitution, insertion and deletion).
3. Accomplishing the correction process within the shortest time.

3 Introduction

The Next-Generation sequencing (NGS) high-throughput technologies [11] were originally proposed in order to help make the vast analysis of genomes less expensive and more spread, this was doable by enhancing the sequencing time, where

too many reads can be generated in a very efficient time. But unfortunately, this type of sequencing introduced two painful issues; the first issue is that the read length becomes much shorter than the conventional sequencing, while the second issue is the decrement of the accuracy, where each erroneous nucleotide can be introduced to the read sequence via one of the three erroneous actions; which are substitution, insertion and deletion. The substitution takes place when the nucleotide is replaced with another erroneous one, while the insertion is when an erroneous nucleotide is newly inserted to the read sequence, and finally the deletion results due to the deletion of a nucleotide from the sequence.
The reads accuracy is a vital factor in all processes that can be applied to the output reads. As an example, the assembly of NGS reads can not be accomplished successfully until the reads errors are corrected or eliminated. So detecting and correcting (or eliminating) the reads errors is an essential step that should precede the assembly process. This step can be accomplished either by a standalone solution or implicitly within the assembly mechanism. The frequency and quality value of the nucleotide are the two main factors used in evaluating it to be erroneous or not [13].
This newly proposed error correction methodology aims to correct all types of errors (substitutions, insertions, and deletions) taking into consideration both accuracy and time. This methodology builds its correction decisions on RACER [4] (an already existing algorithm that handles substitution errors only) with some tuning to handle the insertions and deletions errors. This paper is organized as follows; existing methodologies for error correction are demonstrated in Sect. 5, then the proposed algorithm is introduced in Sect. 7. In Sect. 8, a comparison is set between the new methodology versus existing ones through experimental data. The conclusion is shown in Sect. 9.

4 Background

4.1 Deoxyribonucleic Acid

Deoxyribonucleic Acid,or DNA, is the hereditary material in humans and almost all other organisms. Nearly every cell in an organism's body has the same DNA. Most DNA is located in the cell nucleus (where it is called nuclear DNA). The information in DNA is stored as a code made up of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all people. The order, or sequence, of these bases determines the information available for building and maintaining an organism, similar to the way in which letters of the alphabet appear in a certain order to form words and sentences. DNA bases pair up with each other, A with T and C with G, to form units called base pairs. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, sugar, and phosphate are called a nucleotide. Nucleotides are arranged in two long strands that form a spiral called a double helix. The structure of the double helix is somewhat like a ladder, with the base pairs forming the ladder's

rungs and the sugar and phosphate molecules forming the vertical sidepieces of the ladder.

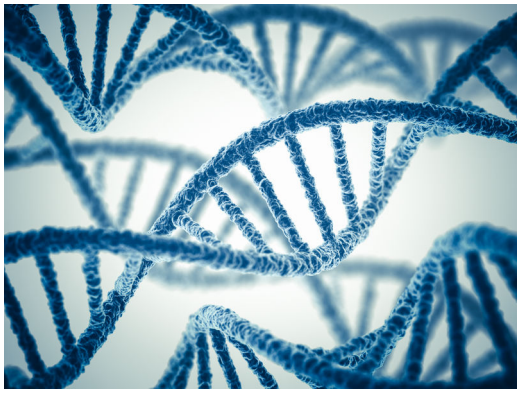

Fig. 1: This is the caption

Fig. 2: This is the caption

An important property of DNA is that it can replicate, or make copies of itself. Each strand of DNA in the double helix can serve as a pattern for duplicating the sequence of bases. This is critical when cells divide because each new cell needs to have an exact copy of the DNA present in the old cell.

4.2 DNA Sequencing

DNA sequence represents a single format onto which a broad range of biological phenomena can be projected for high-throughput data collection. Over the

past years, massively parallel DNA sequencing platforms have become widely available, reducing the cost of DNA sequencing by over two orders of magnitude, and democratizing the field by putting the sequencing capacity of a major genome center in the hands of individual investigators. These new technologies are rapidly evolving, and near-term challenges include the development of robust protocols for generating sequencing libraries, building effective new approaches to data-analysis, and often a rethinking of experimental design.

Fig. 3: This is the caption

```
>Gene1
TTTCGGCGGTGCGCTATCCGGCGGAACTTTTGCGCGTGATGGCGAGTTCCGGTCGCGGAAAGACGACCCTCGTGAATCGCCTTCGCTTT
CGATCCGCCGAGGCGATCCAAGTATCCGCATCCGGGATCGGACTCGTCAATGCGCAACCTGTGGACCGCAAGGAGATCGAGCGCAGGTC
GCGCTATGTCCACGAGGATGACCTCTTTATCGCGTCCCTAACGCGCAGGGAACACCTGATTTTCCAACGCATGGTCGGGATCGCACGAC
ATCTGACCTATCGCGACGGAGTGCGCCCGGTGGATCAGGTGATCCAGGACGTTTCCGTCACGAAATGTCACGACACGATCATCGGTGTC
GCCGCGAGGGTGAAAGGTCTGTCCGCGGGAGAAAGGAACGGTCTGCGATTCCGCTCCGAGCGTCTAACCGATCCCGCCGTTCTGATCTG
GATGACGCCACCTCCGGACTGGACTCCTTTACCCGCCACACGGTCGTCCAGGTGCTGAAGAACGTGTCCGAGAAGGCGAAGACCGTCAT
CCTGACCATTCATCACGCGTCTTCCGACGTGTTTGACGTCTTTGACAAGATCCTTCTGATGCGCGAGGCGAGGGTACGTTTCTTGGCGA
CTCCCACGGAACGCGTCGACTTCTTTTCCTA
```

Fig. 4: This is the caption

One of the core issues of Bioinformatics is dealing with a file formats. Some ad hoc simple human readable formats have over time attained the status of de facto standards. A ubiquitous example of this is the 'FASTA sequence file format',

originally invented by Bill Pearson as an input format for his FASTA suite of tools. FASTA format is a text-based format for representing either nucleotide sequences, in which nucleotides are represented using single-letter codes. FASTA format allows for sequence names and comments to precede the sequences. The first line in a FASTA file starts either with a ">" (greater-than) symbol or, less frequently, a ";" (semicolon) and was taken as a comment. Subsequent lines starting with a semicolon would be ignored by software. Since the only comment used was the first, it quickly became used to hold a summary description of the sequence, often starting with a unique library accession number, and with time it has become commonplace use to always use ">" for the first line and to not use ";" comments (which would otherwise be ignored). Following the initial line

```
;LCBO - Prolactin precursor - Bovine
; a sample sequence in FASTA format
MDSKGSSQKGSRLLLLLVVSNLLLCQGVVSTPVCPNGPGNCQVSLRDLFDRAVMVSHYIHDLSS
EMFNEFDKRYAQGKGFITMALNSCHTSSLPTPEDKEQAQQTHHEVLMSLILGLLRSWNDPLYHL
VTEVRGMKGAPDAILSRAIEIEEENKRLLEGMEMIFGQVIPGAKETEPYPVWSGLPSLQTKDED
ARYSAFYNLLHCLRRDSSKIDTYLKLLNCRIIYNNNC*

>MCHU - Calmodulin - Human, rabbit, bovine, rat, and chicken
ADQLTEEQIAEFKEAFSLFDKDGDGTITTKELGTVMRSLGQNPTEAELQDMINEVDADGNGTID
FPEFLTMMARKMKDTDSEEEIREAFRVFDKDGNGYISAAELRHVMTNLGEKLTDEEVDEMIREA
DIDGDGQVNYEEFVQMMTAK*

>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

Fig. 5: FASTA sample sequences

(used for a unique description of the sequence) is the actual sequence itself in standard one-letter code. Anything other than a valid code would be ignored (including spaces, tabulators, asterisks, etc...). Originally it was also common to end the sequence with an "*" (asterisk) character as illustrated in Fig. [TODO]. Over time, this format has evolved by consensus; however, in the absence of an explicit standard some parsers will fail to cope with very long '>' title lines or very long sequences without line wrapping. There is also no standardization for record identifiers.

In the area of DNA sequencing, the FASTQ file format has emerged as another de facto common format for data exchange between tools. It provides a simple extension to the FASTA format: the ability to store a numeric quality score associated with each nucleotide in a sequence. Early FASTQ uses PHRED quality score (which is a measure of the quality of the identification of the nucleobases generated by automated DNA sequencing where it is logarithmi-

cally linked to error probabilities as illustrated below in Fig. [TODO]). Storing PHRED scores as single characters (or bytes) gave a simple but reasonably space efficient encoding. In order that the file be human readable and easily edited, this restricted the choices to the ASCII printable characters 32–126 (decimal), and since ASCII 32 is the space character, ASCII 33–126 is used instead to encode PHRED qualities from 0 to 93 (i.e. PHRED scores with an ASCII offset of 33).

$$Q = -10 \log_{10} P$$

or

$$P = 10^{\frac{-Q}{10}}$$

Fig. 6: Phred quality scores $\boldsymbol{Q}$ are defined as a property which is logarithmically related to the base-calling error probabilities $\boldsymbol{P}$

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10,000	99.99%
50	1 in 100,000	99.999%
60	1 in 1,000,000	99.9999%

Fig. 7: Phred quality scores are logarithmically linked to error probabilities

4.3 Next-Generation Sequencing

The field of DNA sequencing technology development has a rich and diverse history. Over the past years, the incentive for developing entirely new strategies for DNA sequencing has emerged on at least four levels, undeniably reinvigorating this field. First, in the wake of the Human Genome Project, there are few remaining avenues of optimization through which significant reductions

FASTQ

```
@MISEQ-2:20:000000000-A61NM:1:1101:12299:1738 1:N:0:some_name
TGCGTCATCATCTTTGTCATCGTGTACTACGCCCTGATGGCTGGTGTGGTTTGGTTTGTGGTC
+
AAAAADAFFFFFGGGFGGFGGFHFGFHHFGAEGIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

FASTA

```
>MISEQ-2:20:000000000-A61NM:1:1101:12299:1738 1:N:0:some_name
TGCGTCATCATCTTTGTCATCGTGTACTACGCCCTGATGGCTGGTGTGGTTTGGTTTGTGGTC
```

Fig. 8: FASTQ VS FASTA

in the cost of conventional DNA sequencing can be achieved. Second, the potential utility of short-read sequencing has been tremendously strengthened by the availability of whole genome assemblies for all major model organisms, as these effectively provide a reference against which short reads can be mapped. Third, a growing variety of molecular methods have been developed, whereby a broad range of biological phenomena can be assessed by high-throughput DNA sequencing. And fourth, general progress in technology across disparate fields, including microscopy, surface chemistry, nucleotide biochemistry, polymerase engineering, computation, data storage and others, have made alternative strategies for DNA sequencing increasingly practical to realize. Next-generation DNA sequencing has the potential to dramatically accelerate biological and biomedical research, by enabling the comprehensive analysis of genomes to become inexpensive, routine and widespread, rather than requiring significant production-scale efforts. Since the next-generation sequencing aims to make the vast analysis of genomes less expensive and more spread, it works on enhancing the sequencing time, where too many reads can be generated in a very efficient time. Consequently, it negatively affects the read length and also the sequences accuracy. Actually the next-generation sequencing raises two critical issues, where the first issue is that the read length becomes much shorter than the conventional sequencing, while the second issue is the decrement of the accuracy, where each erroneous nucleotide can be introduced to the read sequence via any of the erroneous actions. There are three erroneous actions are substitution, insertion and deletion. The substitution takes place when the nucleotide is replaced with another erroneous one, while the insertion is when an erroneous nucleotide is newly inserted to the read sequence, and finally the deletion results due to the deletion of a nucleotide from the sequence.

4.4 NGS Sequences Errors Correction

The reads accuracy is a vital factor in all reads processes that can be applied to the output reads. As an example, the assembly of next-generation sequencing reads can not be accomplished successfully until the reads errors are corrected

T	C	T	C	G
T	C	A	C	G

Fig. 9: **Substitution Error** - **T** has been erroneously substituted with A

T	C	T		C	G
T	C	G	T	C	G

Fig. 10: **Insertion Error** - G has been erroneously inserted

or eliminated. So detecting and correcting (or eliminating) the reads errors is an essential step that should precede the assembly process. This step can be accomplished either by a standalone solution or implicitly within the assembly mechanism. The nucleotide frequency and its quality value are two main factors that are used in evaluating the nucleotide to erroneous or not [13].

4.5 Correction Concepts and Definitions

There are common concepts and definitions used by most of the corrective algorithms. Most of the corrective algorithms generates all the possible sub-sequences (of length k) from a read, which is called *k-mer* fig. [TODO]; while the term k-mer frequency represents the number of repetition of a k-mer (a specific sub-sequence of length k) in all the reads. Some of the corrective algorithms set a threshold for the k-mer frequency while classifying the k-mer into strong and weak ones. The strong k-mers are those k-mers that are repeated all over the reads x times, where x is greater than the preset threshold; while the weak k-mers are those ones that are repeated y times all over the reads, where y is less than the k-mer frequency threshold. The filtration step that works on classifying the k-mers into strong and weak ones is called "Spectrum Alignment". The spectrum alignment depends on the k-mers frequencies and/or the nucleotides quality values. The number of reads that include a given nucleotide in the sequence is called

T	C	**T**	C	G
T	C	-	C	G
T	C	C		G

Fig. 11: **Deletion Error** - T has been erroneously deleted

"Coverage" and it can be calculated as:

$$C = L.N/G$$

where L is the average read length, N is the number of reads and G is the genome length.
Some of the corrective algorithms uses the spectrum alignment in their correction decisions so that the correction takes place by obtaining the nucleotides substitutions that leads to reduce the weak k-mers count by converting them into strong k-mers. In some other algorithms, the correction takes place using the tree breadth-first search by traversing multi out-going edges nodes, and removing the fewer reads paths, then re-aligns them to the existing path.
There are some corrective algorithms that depends on the reads alignments, where the correction takes place by aligning reads with a common k-mer, then it fixes the misaligned nucleotides based on their occurrences and quality values. The suffix array also is used in some of the corrective algorithms, where it is built using a string of reads, and the correction takes place with the letter that appears most at each position. Also the suffix trie is used in some other algorithms, where the edges are labelled with DNA letters, while the correction is based on the number of leaves in the sub-trie rooted at the node.
The k-mer hashing table is also used in some corrective algorithms, it is used in storing the total times each nucleotide appears before and after a k-mer, where the error is corrected via the counts.
Also the k-mer discontinuities is used in some algorithms where the frequencies of adjacent k-mers are calculated and the correction is based on the removal or minimizing the discontinuity. All of the concepts, stated above, represent the major concepts and paradigms used in the algorithms specialized in correcting the DNA sequencing errors.
The accuracy evaluation of the corrective algorithms depends on calculating some factors as, the true positive rate, the false positive rate, the false negative rate, and the true negative rate. The true positive rate (TP) is the count of the properly corrected nucleotides, while the false positive rate (FP) is the count of the non-erroneous nucleotides that have been corrected improperly. The false negative rate (FN) is the count of the erroneous nucleotides that haven't been detected as erroneous by the algorithm, while the true negative rate (TN) is the count of the non-erroneous nucleotides that have been properly detected as non-erroneous ones. These four factors are the basics factors that are used in calculating the specificity, sensitivity and the accuracy of the algorithm. The specificity rate represents the ability of the algorithm to properly corrects the erroneous nucleotides, and it can be calculated as:

$$Sensitivity = TP/(TP + FN)$$

While the sensitivity rate represents its ability to detect the erroneous nucleotides, and it can be calculated as:

$$Specificity = TN/(TN + FP)$$

And finally the accuracy represents the all over error rate, and it can be calculated as:

$$Accuracy = (TP + TN)/(TP + FP + FN + TN)$$

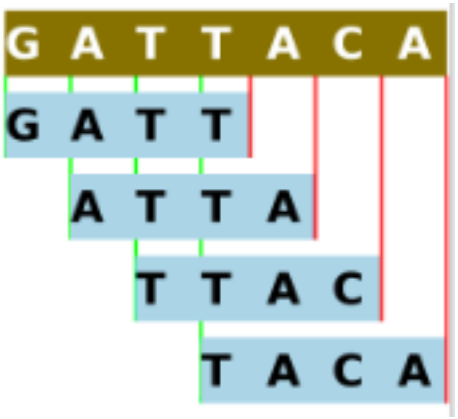

Fig. 12: **K-mer** - All the possible sub-sequences (of length k) from a read

5 Related Work

The error correction methodologies can be either an implicit process within the assembly methodology or a standalone solution that reproduces reads after correction. The assemblers that have embedded error correction are Euler [2], Velvet [15], AllPaths [1] and SOAP [6]. While the standalone methodologies are Coral [10], Quake [5], Reptile [14], HSHREC [9], HiTEC [3], RACER [4], Pollux [7], Parallel Error Correction with CUDA [12], and Error Corrector (EC) [8].

5.1 Substitution Only Corrective Algorithms

They are algorithms that are able to correct only the substitution DNA reads errors, these algorithms are: Euler [2], Velvet [15], AllPaths [1], SOAP [6], Quake [5], Reptile [14], HiTEC [3], RACER [4], Parallel Error Correction with CUDA [12], and Error Corrector (EC) [8]. TODO- Fig12

Euler Euler [2] assembly method runs a filtration step called spectrum alignment that aims to classify the k-mers into two categories according to their frequencies all over the reads. Strong k-mers are the ones with high frequencies, while weak k-mers are the ones with lower frequencies. The correction takes place by executing a greedy exploration for base call substitutions aiming to reduce

the weak k-mers count.
Since in de novo genome sequencing the sampled genome is not known, EULER uses the number of times a k-mer appears to determine if it is in Gk(the set of k-mers in a genome G), where a threshold M determines whether a k-mer is a solid or a weak one. The read that contains all solid k-mers is referred to as a solid read. Hence, the goal of EULER is to correct the reads so that they are all solid reads.
And the set of all k-mers is updated after each iteration of corrections, so that the reads are deleted if they are not able to be corrected after the last iteration. After then, the assembly of the genome begins after the error correction.
The data sets with low coverage require that the threshold M to be set with a low value so that valid k-mers are not considered weak, but this will cause correct reads to be considered incorrect. Where increasing the length of k, will help in reducing the false positive rate and increasing the difficulty of determining the correction changes.
Also in Euler, the reads with erroneous ends are trimmed, because the ends of reads tend to have more errors than the rest of the read, so it's difficult to determine the correct changes to the ends of reads.

Velvet Velvet's [15] tour bus algorithm uses breadth-first search (BFS), starting at nodes with multiple out-going edges, where candidate paths are traversed in step, moving ahead one node on all paths per iteration, until the path lengths exceed a threshold. Velvet removes the path representing fewer reads then re-aligns reads from the removed path to the remaining path.

AllPaths AllPaths [1] uses a read-correcting preprocessor related to the spectral alignment in Euler, where the reads filtration is based on quality values, which is further used in correcting some substitutional errors.

SOAP SOAP [6] filters and corrects reads using pre-set thresholds for k-mer frequencies. It removes bubbles with an algorithm like Velvet's tour bus, with higher read coverage determining the surviving path (read).

5.2 Quake

Quake [5] determines a cut-off value which separates trusted k-mers from untrusted k-mers, using the distribution of k-mers based on their quality scores. The intersection of the untrusted k-mers is used to localize the search for an error in a read. Quake tries to evaluate the conditional probability of assigning the actual nucleotides of the sequenced fragment, given the observed nucleotides.

Quake relies on k-mer coverage and quality scores to correct reads.

Quake uses a method of evaluating the quality of a k-mer based on the associated quality scores for each base.

The k-mers weighing is based on quality scores, then: - Low coverage true k-mers will have high quality scores. - High coverage k-mers with errors will

have low quality scores. 1) Counting k-mers: - Quake begins by counting all the k-mers in the data set.

- The value of k, has a significant impact on the performance of Quake.

- For an appropriate choice for k, Quake uses: k

- Quake determines a cutoff value which separates trusted k-mers from untrusted k-mers, using the distribution of k-mers based on their quality values. 2) Localizing Errors and Correcting Reads: - The reads with untrusted k-mers are considered for correction.

- An intersection of the untrusted k-mers is used to localize the search for an error in a read.

- If the errors are near the end of a read, Quake considers every base covered by the right most trusted k-mer, and left most trusted k-mers to be correct.

- Quake tries to evaluate the conditional probability of a assigning the actual nucleotides of the sequenced fragment, given the observed nucleotides. 3) Heuristics

- Ambiguous true correction: - Repeats implies to having multiple sets of valid corrections, with a small difference in the likelihood of each correction, so a true correction is ambiguous. - Quake continues past the threshold to ensure that another valid set does not exist.

-Quake stops correcting if the region is filled with low quality scores. - Reads with a region containing - For reads with regions containing

5.3 Reptile

Reptile [14] uses the spectral alignment approach used in Euler, with the quality score information if available. Trying to create approximate multiple alignments by considering all reads with pairwise hamming distance less than a pre-set threshold. Reptile uses quality score information if it is available.

Reptile is designed to correct short reads with substitution errors.

Reptile considers all reads with pairwise Hamming distance less than a set threshold.

Reptile finds the multiple alignments by only aligning k-mers in the reads.

The size of k is chosen so that the expected number of occurrences of any k-mer in the genome should be no more than one.

Reptile uses contextual information to help resolve errors without increasing k.

The main disadvantage of Reptile is that the user have to set the parameters, so it requires running scripts and analyzing the results to obtain the proper parameters.

5.4 HSHREC

HSHREC [9] is able to correct all types of errors (substitution, insertion and deletion). The methodology depends on the alignment of a read with others using a suffix trie, where the edges are labelled with DNA letters and a node

weight is the number of leaves in the sub-trie rooted at that node. On the down levels, a node with more than one child is considered to have a substitution error, while extra branching in the generalized suffix trie is caused by indels. SHREC was designed to correct substitution errors, and HSHREC is able to correct both substitution errors and indels.

HSHREC assumes that the input contains k reads randomly sampled from a genome with read length n, where n can vary in length.

Errors are detected in a read by aligning it to the other reads using a suffix trie.

The erroneous region of the read has a low weight in the trie and this is the area that will be corrected.

Single nucleotide polymorphisms (SNPs) are single nucleotide changes that differ between individuals of the same species, they can appear as errors.

Since errors at the SNP location will only be in a few reads, and SNPs will be present in several reads, it is possible to differentiate them. Each read has a unique suffix by concatenating a unique number from 1 to 2k to the end of each string in R (the set of reads, and their reverse complements).

The edges of the trie are labeled with DNA letters, and an ambiguous letter N.

The concatenation of edge labels from the root to a node is called a path-label.

For any suffix of a sting in R, there is a path-label that contains that suffix.

The weight of a node in the trie is the number of leaves in the subtrie that is rooted at that node, and is the number of suffixes in that subtrie.

The level of a node is the length of the path from the root to the node. In the top levels of the trie almost all nodes have four or five children.

In the down levels of the trie almost all nodes have only one child.

If a child at down levels has more than one child then it is likely an error.

The node with the lower weight is likely the erroneous base.

The HSHREC algorithm traverses the trie to identify errors at the intermediate levels of the trie.

The HSHREC tries to correct each error with a substitution, and the remaining errors are treated as insertions or deletions.

SHREC compares the subtrie rooted at the low weight node to the subtries rooted at the siblings of the node.

Indels cause extra branching in the generalized suffix trie.

An insertion creates a low weight node.

Deleting the node causes the children rooted at the node and their siblings to be merged.

A comparison of the subtries before and after the deletion determine if the deletion was a proper way to correct the node.

5.5 HiTEC

HiTEC [3] uses a suffix array that is built using a string of reads and their reverse complements. The correction of an erroneous nucleotide takes place with

the letter that appears most at that position. HiTEC (High Throughput Error Correction) uses a suffix array that is built using a string of the reads and their reverse complements.

Suffix array is a more time and space efficient data structure than the suffix trie.

The most accurate read error correction program to date is HiTEC.

The length of the genome and the error rate of the data set is supplied.

HiTEC has been shown to be the most accurate algorithm for correcting substitution errors, as its parameters is set using statistical analysis.

HiTEC uses an array that stores the length of the longest common prefix (LCP) between consecutive suffixes in the suffix array. The reads and their reverse complements are stored in a string: R = r1 $

HiTEC assumes that a read ri, starting at position j of the genome, contains an error in position k and that the previous w positions, ri [k - w..k - 1], are correct.

Using the majority of one base at a position, then the erroneous base is changed to the base that appears most at that position.

For a

Repeats in the genome can cause problems with a small u.

Long u will be less likely to appear in the genome & will be covered by fewer reads. Statistical analysis helps compute a threshold, T , that is used to detect errors.

To cover the case when low coverage causes very low values for T.

Reads with no w consecutive correct positions, makes it impossible to fit a correct witness at any position. And they cannot be corrected.

As the length of witness length (w) decreases: - The number of uncorrectable reads U(w) decreases. - The probability of seeing the witness increases, so the correct positions changed incorrectly. - The number of destructible reads D(w) increases. - A value for w must be found that minimizes U(w)+ D(w).

HiTEC uses a variation of witness lengths based on the optimal witness length to achieve high accuracy. If there is no ambiguity in the correct letter then the correction is made.

If there is ambiguity, then the next two letters are checked in order to decide how to correct the read.

HiTEC stops correcting when the number of bases changed during one iteration is less than 0.01% of the total number of bases, or after nine iterations or corrections.

HiTEC only requires modest coverage to make corrections.

Datasets with high coverage are split into several sets with lower coverage that are independently corrected.

The disadvantages of HiTEC are: - It does not correct reads with ambiguous letters. - It can only correct data sets if the reads are all the same length - It does not run in parallel mode.

5.6 RACER

RACER [4] is able to correct data sets that have varying read lengths. Using a hash table that stores the total times each nucleotide appears before and after each k-mer, where the error is corrected via the counts. Rapid and Accurate Correction of Errors in Reads.

RACER replaces the suffix array approached used in HiTEC with a more time and space efficient hash table.

The hash table stores the k-mers in each read, and the total times each base appears before and after each k-mer.

The optimal k-mer length is calculated with a similar statistical analysis as HiTEC.

After the k-mers and the counters have been calculated, the reads are corrected based on the counts.

RACER encodes the input sequences using two bits to represent each base. The k-mer and its reverse complement must be searched, and only the one with the smaller encoding key is stored.

RACER is able to correct data sets that have varying read lengths, and runs in both serial and parallel mode.

RACER requires three parameters to be calculated: 1. A threshold for determining and correcting an error. 2. A k-mer length K that will minimize the number of false positives. 3. A k-mer length k that will maximize the number of corrections.

5.7 Pollux

Pollux [7] calculates the k-mer frequencies in the entire set of reads. Identifying the discontinuities by comparing the frequencies of adjacent k-mers within reads, assuming that individual k-mers are not erroneous. The discontinuities within reads are used to find error locations and evaluate correctness. The correction is chosen to be the one that removes or minimizes the k-mer count discontinuity. Pollux decomposes each read into k-mers and then calculates the k-mer frequencies in the entire set of reads. The nucleotide is represented with a two-bit alphabet and a hash-table strategy is used to help in using large k-mers (with length 31) to avoid common short repeats which might otherwise confound the correction procedure. Identifying the individual k-mers as not erroneous, then identifying the discontinuities by comparing the frequencies of adjacent k-mers within reads. The discontinuities within reads are used to find error locations and evaluate correctness. A read that is not erroneous is assumed to have a k-mer count profile that is reflective of a random sampling process, given local coverage. In contrast, a read that contains an error is likely to have k-mer counts that deviate unexpectedly from this random process. Evaluating multiple k-mers containing bases following the erroneous base to ensure our correction is appropriate. The correction is chosen to be the one that removes or minimizes the k-mer count discontinuity. If there exist multiple corrections which achieve this, the correction is chosen to be the one that improves the most k-mer

counts maximally beyond the current erroneous location. Pollux corrects each read independently and does not update recorded k-mer counts as a result of correction.

5.8 Parallel Error Correction with CUDA

Parallel Error Correction with CUDA [12] uses the spectrum alignment besides a voting algorithm for the single-mutation using each letter, hence errors can be fixed based on high values in the voting matrix. CUDA (Compute Unified Device Architecture) is an extension of C/C++ to write scalable multi-threaded programs for CUDA-enabled GPUs.

CUDA programs can be executed on GPUs with NVIDIA's Tesla unified computing architecture.

CUDA programs contain a sequential part, called a kernel.

Error correction consists of a set of reads R that need to independently access the spectrum Tm.l(R), so the correction of an individual read ri.

The kernel is then invoked using a thread for each ri Algorithm 1: Single-mutation voting algorithm used in the CUDA kernel.

Input: Read ri[0...L-1] and spectrum Tm.l(R). Output: Voting matrix Vi[][] of size L*4

Steps: 1. Initialize the voting matrix by zeros: Vi[j][c] := 0, j := 0...L-1, c

2. For each l-tuple in ri: if it isn't an l-tuple in Tm.l(R): For each nucleotide in the l-tuple For each letter in c If the replacement of the nucleotide by the letter makes the l-tuple Increase the voting matrix for this nucleotide & this letter by 1. - Errors can be fixed based on high values in the voting matrix.

- In case of having two errors are close to each other:

The single-mutation voting algorithm cannot correct the errors.

It is still possible to identify the read as erroneous and to trim it at certain positions or discard it.

Algorithm 2: Single-mutation fixing/trimming/ discarding procedure used in the CUDA kernel.

Input: Read ri[0...L-1] and Voting matrix Vi[][] of size (L-(l+1))*4.

Output: Fixed, trimmed, discarded or unchanged read

Steps: 1. maxV is the max value in the given voting matrix (maxj,maxc) is the indices of the maxV

2. If (maxV = 0) return ri; input read is error-free Else: a. rci string concatenation after replacement for correction Set corrected_flag = true & trimmed_flag = false b. For each l-tuple in rci: If it isn't an l-tuple in Tm.l(R): Set corrected_flag = false Else Set trimmed_flag = true c. If (corrected_flag) return rci; corrected read Else If(trimmed_flag) return rci[k1...k2]; the longest substring of rci in which all l-tuples belong to Tm.l(R) Else return

Time complexity of the kernel is dominated by the spectrum membership test in the voting algorithm.

(L-l).(1+p.3.l), where p is the number of l-tuples of the read that do not belong to the spectrum.

Hashing can perform membership tests to a hashing table in constant time. The single-mutation voting algorithm contains the data-dependent that checks if it isn't an l-tuple in Tm.l(R) for each l-tuple in the given read.

Threads, for which this statement is true, execute another 3.l membership tests.

Threads, for which this statement is false, need to wait for these threads within the same warp to finish these tests.

Thus, the number of waiting threads per warp is decreasing for higher error-rates.

So, The speedup increases for higher error rates. Our implementation outputs four datasets: 1. unchanged reads, 2. corrected reads, 3. trimmed reads, and 4. discarded reads.

The union of unchanged, corrected and trimmed reads is usually taken as an input dataset to a subsequent de-novo DNA fragment assembly algorithm. Accuracy:

Identification: identifying reads as erroneous or error-free. Correction: correcting reads, which have been identified as erroneous.

The identification of erroneous reads can be defined as a binary classification test.

Sensitivity = TP/(TP+FP) Specificity = TN/(TN+FN)

The accuracy of the actual correction/trimming operation; i.e. if correction/trimming is done at the correct read positions.

Also, the amount of corrected/trimmed reads relative to the number of discarded reads is measured.

5.9 Error Corrector

Error Corrector (EC) [8] an error correction algorithm for correcting short reads with substitution errors only. Using k-mers hashing tables to find the neighbours of each of the reads, where each read is corrected using its neighbours.

Euler	Velvet	AllPaths	SOAP	Quake	Reptile	CUDA	HiTEC	RACER	EC
2004	2008	2008	2010	2010	2010	2010	2011	2013	2015
Spectrum Alignment	Tree BFS	Spectrum Alignment	Tree BFS	Spectrum Alignment	Spectrum Alignment	Spectrum Alignment	Suffix Array	Hash Table	Hash Table
K-mer Freq.	Nuc. Freq.	K-mer Freq. with Nuc. QV	K-mer Freq.	K-mer Freq. with Nuc. QV	K-mer Freq. with Nuc. QV	K-mer Freq. with Votes	Nuc. Freq.	Nuc. Freq.	Nuc. Freq.

Fig. 13: An illustrative summary for the substitution only corrective algorithms

5.10 Substitution, Insertion And Deletion Corrective Algorithms

They are algorithms that are able to correct all types of DNA reads errors, these algorithms are: Coral [10], HSHREC [9], and Pollux [7] TODO- Fig13

Coral Coral [10] is able to correct all types of errors (substitution, insertion and deletion). The methodology is built on scoring the alignments between short reads, where each alignment runs on a base read with all the reads that have at least one common k-mer with this base read. Then the correction takes place for the misaligned positions depending on the number of times each letter occurs in addition to quality scores of the nucleotide.
Coral uses multiple sequence alignments (MSA) between short reads to detect errors, where the calculates the gap penalty and mismatch penalty (parameters for scoring the alignments) have a significant impact on the quality of the corrections.
The main defect in Coral is that it isn't able to correct mixed data sets that contain both substitution errors with insertion and deletions ones. For substitution errors, Coral sets the gap penalty with a very high value to prevent insertions and deletions in the alignments. While for insertion and deletion errors, Coral sets the gap and mismatch penalties to be equal.
Coral can summarize his algorithm as:

1. Indexing Reads, where a hash table is used to store the k-mers and the reads that contain each k-mer. And in order to save space, Coral store the information of only the lexicographically smaller of the k-mer & its reverse. It also does not store any k-mers that contain ambiguous letters.
2. Multiple Alignments, where Coral starts each alignment with one read which is called the base read, and all the reads that share at least one k-mer with the base read are found using the k-mer hash table, and they are called neighbourhood of the base. If the neighbourhood of the base read is very large or very small, then Coral does not perform any alignments. As, if it's very large, then the reads likely came from different regions in the genome and the MSA will take a long time to compute, as the alignments will be very poor. But, if it's very small, then there is likely not enough coverage to make any significant corrections.
   Coral doesn't correct the reads that have been corrected before. So, if a read has already been corrected in a previous alignment, Coral tries to align it to the consensus without errors in the region of the k-mer. And, if a read aligns perfectly to the consensus then the rest of the read is not aligned to save time. But, if a read has many errors compared to the consensus then it stops aligning the read and moves on to the next one. And if the gaps are not allowed then a gap-less alignment is performed between a read and the consensus sequence.
3. Correcting Reads, where Coral calculates the number of misaligned positions for each read compared to the consensus sequence. And if the quality of the alignment is above a preset threshold, then it will be used to correct

the aligned reads. The support threshold for each position in the consensus sequence is calculated as: (num of times each letter occurs)/ (total num of reads aligned at that position). In case a read differs from the consensus at any position, then the letter is changed in the read provided the support is above the threshold. And if quality scores are available and there is a gap in an aligned read, then: quality score of the gap = avg (the quality scores of the bases flanking the gap).

The complexity of Coral is illustrated as follow (assuming that where M is the combined total length of the reads, L is the maximum number of reads in a neighbourhood, and r is the longest read length):

- If gaps are allowed, the worst case runtime is O(MrL).
- If only mismatches are allowed, the worst case runtime is O(ML).
- The space for the hash table that stores the k-mers is bounded by O(M).
- The space complexity for computing the MSA is $O(Lr + r^2)$.
- The overall space complexity is O(M).

HSHREC	Coral	Pollx
2010	2011	2015
Suffix Trie	Reads Alignment	K-mer Discontinuities
Nuc. Freq.	Nuc. Freq. with QV	K-mer Freq.

Fig. 14: An illustrative summary for the substitution, insertion and deletion corrective algorithms

6 The First Proposed Algorithm

Aiming to correct all types of errors

Hashing the k-mers into integers

Flexible to run more correction iterations

Correction takes place by: Grouping k-mers, where only the three nucleotides located at the first, middle and last positions of the read can vary.

Diagnosing each k-mer group, then fixing the misalignment in the three nucleotides based on their occurrences and quality values.

Correction takes place by: Re-grouping the k-mers depending on the non-corrected ones, where only the three nucleotides with the lowest quality value in the read can vary.

Diagnosing each k-mer group, then fixing the misalignment in the three nucleotides based on their occurrences and quality values.

For Lactis, the data with smallest size: It shows the best accuracy with a very small difference (0.05%), but with the worst time with a big difference; 1 hr compared to 15, 5, and 3 mins

For Pallidum, data with larger size: It runs more than 16 hrs compared to 22, 12, and 3 mins; which enforced the interruption of the running without getting the results

For e.Coli, the largest dataset: It has been running for a day (24 hrs) without completion; which enforced the interruption of the running without getting the results

This algorithm is mainly dependent on the k-mers grouping

The kmers grouping takes place by generating all of the possible cases of the corrections of every kmer, and here goes the time defect

The main major step of the proposal implies to it's weakness point, which proves that this proposal won't get a better results

7 The Proposed Algorithm

H-RACER is the newly proposed approach for correcting all types of errors (substitution, insertion, and deletion). Although RACER is the fastest DNA error correction algorithm existent nowadays with a high accuracy, but it can not correct all types of errors, it can only correct substitutions. So, H-RACER is proposed in order to correct all types of errors. H-RACER follows the same algorithm of RACER in detecting errors and deciding their corrections, the newly added part in H-RACER is the detection of the error type in order to apply the correction properly for data sets with varying error types.

H-RACER detects the error type for an erroneous nucleotide by studying its correction value (obtained by RACER) against its neighbours, then decides the corrective action (substitute, insert, delete) according to the detected error type. Once the error detection and correction stages are done, H-RACER starts applying correction using its own methodology which mainly depends on detecting the error type in order to apply the detected correction with the proper action (substitution, insertion or deletion).

H-RACER starts the error type detection by looping on every nucleotide in the whole reads set, to check if this nucleotide is an erroneous one. For every erroneous nucleotide H-RACER checks the position of this nucleotide in the read not to be the last one in the read (so that there is at least one nucleotide following it) where the follower nucleotide is erroneous too. Then H-RACER starts to examine the erroneous and corrective values for both nucleotides (the current and its follower). H-RACER checks if the correction value of the current erroneous nucleotide is equal to the erroneous value of its follower, so it will be concluded that this current erroneous nucleotide is a result of an insertion erroneous action. Hence, H-RACER applies the correction as a deletion action for this current erroneous nucleotide. But, if it is found that the erroneous value of the current

nucleotide is equal to the correction value of its erroneous follower, so it will be concluded that this current erroneous nucleotide is a result of a deletion erroneous action. Hence, H-RACER applies the correction as an insertion action at a position directly before the current erroneous nucleotide with the correction value of it (the current erroneous nucleotide). Otherwise, if there is not any criss-cross equality relation between the erroneous and correction values of the current nucleotide and its erroneous follower (i.e. the correction/erroneous value of the current erroneous nucleotide is not equal to the erroneous/correction value of its follower), then it will be concluded that this current erroneous is a result of a substitution erroneous action. Hence, H-RACER applies the corrective action as a substitution action for the erroneous value of the current nucleotide with its correction value.
On another side, if it is found that the current nucleotide is the last one in the read or its follower nucleotide is not erroneous, then H-RACER will check the position of this nucleotide in the read not to be the first in the read (so that there is at least one nucleotide that precedes it) where the precedent nucleotide is erroneous too. Then H-RACER starts to examine the erroneous and corrective values for both nucleotides (the current and its precedent). H-RACER checks if the correction value of the current erroneous nucleotide is equal to the erroneous value of its precedent, so it will be concluded that this current erroneous nucleotide is a result of an insertion erroneous action. Hence, H-RACER applies the correction as a deletion action for this current erroneous nucleotide. But, if it is found that the erroneous value of the current nucleotide is equal to the correction value of its erroneous precedent, so it will be concluded that this current erroneous nucleotide is a result of a deletion erroneous action. Hence, H-RACER applies the correction as an insertion action at a position directly after the current erroneous nucleotide with the correction value of it (the current erroneous nucleotide). Otherwise, if there is no criss-cross equality relation between the erroneous and correction values of the current nucleotide and its erroneous precedent (i.e. the correction/erroneous value of the current erroneous nucleotide is not equal to the erroneous/correction value of its precedent), then it will be concluded that this current erroneous is a result of a substitution erroneous action. Hence, H-RACER applies the corrective action as a substitution action for the erroneous value of the current nucleotide with its correction value.
Finally, if H-RACER finds that the current nucleotide is either the last or the first in the read, or neither its follower nor precedent nucleotides are erroneous, then it will be concluded that this current erroneous is a result of a substitution erroneous action. Hence, H-RACER applies the corrective action as a substitution action for the erroneous value of the current nucleotide with its correction value.
H-RACER error detection algorithm has a complexity $O(r)$, where r is the number of reads. For more illustration check the examples shown below in Fig. 15 and Fig. 16, and the pseudo-code shown in Fig. 17.

1. Inserted Nucleotide - Deletion Correction
   (1) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (2) Correction: AC$\mathbf{T}\cdots$
   (3) Tracing:
      (1) $\underline{\text{G}}$ is an erroneous nucleotide
      (2) $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      (3) $\underline{\text{G}}$'s correction value is **T**
   (4) Conclusion: $\underline{\text{G}}$ is an erroneously inserted nucleotide
2. Deleted Nucleotide - Insertion Correction
   (1) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (2) Correction: AC$\mathbf{AG}\cdots$
   (3) Tracing:
      (1) $\underline{\text{G}}$ is an erroneous nucleotide
      (2) $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      (3) $\underline{\text{G}}$'s correction value is **A**
      (4) $\underline{\text{T}}$'s correction value is **G**
   (4) Conclusion: **A** is an erroneously deleted nucleotide
3. Substituted Nucleotide - Substitution Correction
   (1) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (2) Correction: AC$\mathbf{AC}\cdots$
   (3) Tracing:
      (1) G is an erroneous nucleotide
      (2) $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      (3) $\underline{\text{G}}$'s correction value is **A**
      (4) $\underline{\text{T}}$'s correction value is **C**
   (4) Conclusion: **A** is an erroneously substituted nucleotide with $\underline{\text{G}}$

Note: The erroneous nucleotides are the underlined ones, while the nucleotides corrections are the ones in bold.

Fig. 15: H-RACER Error Type Detection Examples

Assume a read represented as: $r = s_1 s_2 \cdots s_n$
if s_i is erroneous
Then,

1. if s_{i+1} has error Then,
   (1) Delete s_i, if the correction of s_i equals to s_{i+1}
   (2) Insert the correction of s_i before s_i, if s_i equals to the correction of s_{i+1}
   (3) Substitute s_i with its correction, otherwise
2. if s_{i-1} has error Then,
   (1) Delete s_i, if the correction of s_i equals to s_{i-1}
   (2) Insert the correction of s_i after s_i, if s_i equals to the correction of s_{i-1}
   (3) Substitute s_i with its correction, otherwise

Fig. 16: H-RACER Error Type Detection Abstraction

```
for every read in reads do
    for every nuc in read do
        if has_error(nuc)
            if not_last(nuc) AND has_error(next(nuc))
                if correction(nuc) equals to next(nuc)
                    Delete nuc from read
                else if nuc equals to correction(next(nuc))
                    Insert correction(nuc) before nuc in read
                else
                    substitute nuc with correction(nuc) in read
                end if
            else if not_first(nuc) AND has_error(previous(nuc))
                if correction(nuc) equals to previous(nuc)
                    Delete nuc from read
                else if nuc equals to correction(previous(nuc))
                    Insert correction(nuc) after nuc in read
                else
                    substitute nuc with correction(nuc) in read
                end if
            else
                substitute nuc with correction(nuc) in read
            end if
        end if
    end for
end for
```

Fig. 17: H-RACER Pseudo-Code - O(r)

8 Evaluation

8.1 Datasets and Platform

The testing was performed on a wide variety of real data sets, shown below in table 1, with different read length, genome size and coverage. It was preferred to use real data sets and to avoid any simulated ones as they do not offer a good indication of real life performance. All data sets were brought from the National Center for Biotechnology Information (NCBI).
All algorithms were executed on the same amazon elastic cloud (AWS EC2) instance with 32 vCPU and 244GiB RAM, with Linux (Ubuntu) operating system.

Table 1: Data sets used in evaluation

Name	Accession Number	Genome	Genome Length	Read Length	Number of Reads	Coverage
Lactococcus Lactis	SRR088759	NC_013656.1	2,598,144	36	4,370,050	60.55
Treponema Pallidum	SRR361468	CP002376.1	1,139,417	35	7,133,663	219.13
E.coli 75a	SRR396536	NC_000913.2	4,639,675	75	3,454,048	55.83
E.coli 75b	SRR396532	NC_000913.2	4,639,675	75	4,341,061	70.17

8.2 Results

The comparisons, shown below in tables 2, 3, 4, and 5, were established between H-RACER and algorithms specialized in correcting all types of errors (substitutions, insertions, and deletions). While the obtained measurements were brought via the verification code implemented by RACER, that has the advantage of avoiding the interference of mapping or assembling programs.

As shown below in the comparisons tables, H-RACER has the best results in accuracy and time. Actually, H-RACER aims to increase the genome reads accuracy, consequently, it aims to eliminate the errors existing in the genome reads. So, H-RACER avoided introducing errors to the reads by lowering the false positive rate and consequently increasing the specificity rate. On the other side, the false negative rate was negatively affected and the sensitivity rate was decreased as well. But, this approach did not negatively affect the accuracy, on contradictory, it resulted in getting the best accuracy.
In other words, the high accuracy of H-RACER mainly resulted from both, the remarkable lowering of false positive rate and the high raising of true negative rate. On the other side, both, the true positive and false negative rates were negatively affected. Hence, H-RACER does not have neither the highest true positive nor the lowest false positive rates. And this is what H-RACER follows in RACER's footsteps, where it is preferred to lower the algorithm sensitivity represented in raising the false negative rate and lowering the true positive rate

rather than raising the false positive rate and lowering the true negative rate. And this makes sense, as enhancing the reads overall accuracy is the main vital target. So, corrective algorithms should not introduce errors (represented in false positive rate), but it should target a higher gain to get higher accuracy, and so does RACER followed by H-RACER.

The comparisons with the other algorithms, shown below in tables 2, 3, 4, and 5, prove that although the other algorithms have higher sensitivity than H-RACER, but all of them do not explicitly beat H-RACER accuracy, except for **?**Treponema Pallidum**?**, where HSHREC beats H-RACER's accuracy by 0.07% and this is due to the high coverage rate of **?**Treponema Pallidum**?** that will be explained below.

For the short genome **?**Lactococcus Lactis**?**, illustrated in table 2, H-RACER shows the best results in specificity, gain, accuracy and time compared to CORAL, Pollux and HSHREC, although the sensitivity of H-RACER is not the best. But H-RACER gains the highest accuracy by lowering the false positive rate (as explained above).

For **?**Treponema Pallidum**?**, the short genome with a very high coverage (the average number of reads representing a given nucleotide in the reconstructed sequence) as illustrated in tables 1 and 3, H-RACER shows a lowering in the true positive rate with a raising in the false negative one rather than the expected rates. This is due to the very high coverage of **?**Treponema Pallidum**?** that increases the ambiguity for H-RACER in detecting the proper correction for some nucleotides, leading to a lowering in the true positive rate and consequently in the accuracy. But, by comparing such an accuracy with others, as illustrated in table 3, it is obvious that H-RACER shows the best results in specificity, gain, accuracy and time compared to CORAL and Pollux. While HSHREC is the only algorithm that beats H-RACER's gain and accuracy (for **?**Treponema Pallidum**?** only) with a very little difference rates (0.29% and 0.07% respectively). But, H-RACER accomplished such a correction in the best time compared to all others including HSHREC's (with a very remarkable ratio) and the best specificity as well.

For long genomes **?**E.coli 75a**?** and **?**E.coli 75b**?**, illustrated in tables 4 and 5, H-RACER obviously shows the best results in accuracy with a very perfect time, while CORAL and Pollux show lower accuracy with too much longer time, but HSHREC's running throws exception for such genomes, as SHREC (and consequently HSHREC) requires a very large space [4], [9], so it is unable to run successfully for the larger genomes on the specified machine.

Finally, the remarkable great difference in time between H-RACER and the rest of the algorithms is due to the bitwise orientation in implementation (inherited

from RACER), and also H-RACER keeps RACER's complexity, consequently, H-RACER gains the advantage of having the best time.

Table 2: Evaluation comparison table for Lactococcus Lactis

	Coral	Pollux	HSHSREC	H-RACER
True Positive	15,396,336	25,325,532	25,537,644	21,237,660
False Positive	2,039,148	7,720,920	6,053,580	19,656
False Negative	11,413,764	1,484,568	1,272,456	5,572,440
True Negative	128,472,552	122,790,780	124,458,120	130,492,044
Sensitivity	57.43%	94.46%	95.25%	79.22%
Specificity	98.44%	94.08%	95.36%	99.98%
Gain	49.82%	65.66%	72.67%	79.14%
Accuracy	91.45%	94.15%	95.34%	96.45%
Time in Minutes	5	3	15	1

Table 3: Evaluation comparison table for Treponema Pallidum

	Coral	Pollux	HSHSREC	H-RACER
True Positive	25,553,185	63,845,425	64,381,905	56,277,270
False Positive	3,462,165	8,832,320	8,133,895	223,405
False Negative	41,547,065	3,254,825	2,718,345	10,822,980
True Negative	179,115,790	173,745,635	174,444,060	182,354,550
Sensitivity	38.08%	95.15%	95.95%	83.87%
Specificity	98.10%	95.16%	95.55%	99.88%
Gain	32.92%	81.99%	83.83%	83.54%
Accuracy	81.97%	95.16%	95.65%	95.58%
Time in Minutes	12	3	22	2

Table 4: Evaluation comparison table for E.coli 75a

	Coral	Pollux	HSHSREC	H-RACER
True Positive	26,434,125	79,984,425	N/A	76,325,475
False Positive	5,549,925	31,675,650	N/A	33,000
False Negative	73,707,075	20,164,125	N/A	23,823,075
True Negative	153,362,475	127,229,400	N/A	158,872,050
Sensitivity	26.40%	79.87%	N/A	76.21%
Specificity	96.51%	80.07%	N/A	99.98%
Gain	20.85%	48.24%	N/A	76.18%
Accuracy	69.40%	79.99%	N/A	90.79%
Time in Minutes	9	16	N/A	1

Table 5: Evaluation comparison table for E.coli 75b

	Coral	Pollux	HSHSREC	H-RACER
True Positive	13,312,725	99,375,600	N/A	81,059,700
False Positive	3,681,450	37,779,750	N/A	35,925
False Negative	108,494,025	22,439,925	N/A	40,755,825
True Negative	200,091,375	165,984,300	N/A	203,728,125
Sensitivity	10.93%	81.58%	N/A	66.54%
Specificity	98.19%	81.46%	N/A	99.98%
Gain	7.91%	50.56%	N/A	66.51%
Accuracy	65.55%	81.50%	N/A	87.47%
Time in Minutes	13	21	N/A	2

9 Conclusion

H-RACER comes up with the advantage of correcting different error types which were missed in RACER. H-RACER followed in the footsteps of RACER's implementation in order to acquire the major advantages of RACER in both aspects performance and time, then added its elegant algorithm in detecting the errors types and properly applying their corrections. Consequently, H-RACER shows great results in both performance and time compared to existing algorithms specialized in correcting all types of errors for large and small genome lengths. And by comparing H-RACER with existing algorithms specialized in correcting all types of errors, it is proved that H-RACER is the fastest with the highest accuracy algorithm.
Finally, H-RACER algorithm has been implemented in C/C++ as an open source program.
Available at: drive.google.com/open?id=0B7Otgzz7lZlldkE2ZFVwcU5qN1U

References

1. Butler, J., MacCallum, I., Kleber, M., et al.: Allpaths: De novo assembly of whole-genome shotgun microreads. Genome Res. 18, 810–820 (2008)
2. Chaisson, M., Pevzner, P., Tang, M.: Fragment assembly with short reads. Bioinformatics. 20, 2067–2074 (2004)
3. Ilie, L., Fazayeli, F., Ilie, S.: Hitec: accurate error correction in high-throughput sequencing data. Bioinformatics 27, 295–302 (2011)
4. Ilie, L., Molnar, M.: Racer: Rapid and accurate correction of errors in reads. Bioinformatics. 19, 2490–2493 (2013)
5. Kelley, D., Schatz, M., Salzberg, S.: Quake: quality-aware detection and correction of sequencing errors. Genome Biol. 11, R116 (2010)
6. Li, R., Zhu, H., Ruan, J., et al.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Res. 20, 265–272 (2010)
7. Marinier, E., Brown, D., McConkey, B.: Pollux: platform independent error correction of single and mixed genomes. BMC Bioinformatics 15, 435 (2015)

8. Saha, S., Rajasekaran, S.: Ec: an efficient error correction algorithm for short reads. BMC Bioinformatics 16(Suppl 17):S2 (2015)
9. Salmela, L.: Correction of sequencing errors in a mixed set of reads. Bioinformatics 26, 1284–1290 (2010)
10. Salmela, L., Schröder, J.: Correcting errors in short reads by multiple alignments. Bioinformatics. 27, 1455–1461 (2011)
11. Shendure, J., Ji, H.: Next-generation dna sequencing. Nat Biotechnol. 26, 1135–1145 (2008)
12. Shi, H., Schmidt, B., Liu, W., et al.: A parallel algorithm for error correction in high throughput short-read data on cuda-enabled graphics hardware. J. Comput. Biol 17, 603–615 (2010)
13. Yang, X., Chockalingam, S., Aluru, S.: A survey of error-correction methods for next-generation sequencing. Brief. Bioinform. 14, 56–66 (2013)
14. Yang, X., Dorman, K., Aluru, S.: Reptile: representative tiling for short read error correction. Bioinformatics 26, 2526–2533 (2010)
15. Zerbino, D., Birney, E.: Velvet: Algorithms for de novo short read assembly using de bruijn graphs. Genome Res. 18, 821–829 (2008)