H-RACER: Hybrid RACER to Correct Substitution, Insertion, and Deletion Errors

SALMA GOMAA[1,2], NAHLA A. BELAL[1,3,*] and YASSER EL-SONBATY[1,4]

[1] College of Computing and Information Technology, AASTMT, Alexandria, Egypt
[2] salma.gomaa89@gmail.com
[3] nahlabelal@aast.edu
[4] yasser@aast.edu

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 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

* Corresponding Author

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. 2, then the proposed algorithm is introduced in Sect. 3. In Sect. 4, a comparison is set between the new methodology versus existing ones through experimental data. The conclusion is shown in Sect. 5.

2 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].

2.1 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.

2.2 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.

2.3 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.

2.4 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).

2.5 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.

2.6 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.

2.7 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.

2.8 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.

2.9 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.

2.10 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.

2.11 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.

2.12 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.

2.13 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.

3 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. 1 and Fig. 2, and the pseudo-code shown in Fig. 3.

1. Inserted Nucleotide - Deletion Correction
   (a) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (b) Correction: AC$\mathbf{T}\cdots$
   (c) Tracing:
      i. $\underline{\text{G}}$ is an erroneous nucleotide
      ii. $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      iii. $\underline{\text{G}}$'s correction value is **T**
   (d) Conclusion: $\underline{\text{G}}$ is an erroneously inserted nucleotide
2. Deleted Nucleotide - Insertion Correction
   (a) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (b) Correction: AC$\mathbf{AG}\cdots$
   (c) Tracing:
      i. $\underline{\text{G}}$ is an erroneous nucleotide
      ii. $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      iii. $\underline{\text{G}}$'s correction value is **A**
      iv. $\underline{\text{T}}$'s correction value is **G**
   (d) Conclusion: **A** is an erroneously deleted nucleotide
3. Substituted Nucleotide - Substitution Correction
   (a) Read Sequence: AC$\underline{\text{GT}}\cdots$
   (b) Correction: AC$\mathbf{AC}\cdots$
   (c) Tracing:
      i. G is an erroneous nucleotide
      ii. $\underline{\text{G}}$ is followed by an erroneous nucleotide $\underline{\text{T}}$
      iii. $\underline{\text{G}}$'s correction value is **A**
      iv. $\underline{\text{T}}$'s correction value is **C**
   (d) 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. 1. 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,
   (a) Delete s_i, if the correction of s_i equals to s_{i+1}
   (b) Insert the correction of s_i before s_i, if s_i equals to the correction of s_{i+1}
   (c) Substitute s_i with its correction, otherwise
2. if s_{i-1} has error Then,
   (a) Delete s_i, if the correction of s_i equals to s_{i-1}
   (b) Insert the correction of s_i after s_i, if s_i equals to the correction of s_{i-1}
   (c) Substitute s_i with its correction, otherwise

Fig. 2. 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. 3. H-RACER Pseudo-Code - O(r)

4 Evaluation

4.1 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

4.2 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

5 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., Schrder, 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)