BIOINFORMATICS ORIGINAL PAPER

Vol. 25 no. 17 2009, pages 2157–2163
doi:10.1093/bioinformatics/btp379

Genome analysis

SHREC: a short-read error correction method

Jan Schröder[1,3,*], Heiko Schröder[2], Simon J. Puglisi[2], Ranjan Sinha[3] and Bertil Schmidt[4]

[1]Institut für Informatik, Christian-Albrecht-Universität Kiel, Herman-Rodewald-Strasse 3, 24118 Kiel, Germany, [2]School of Computer Science and Information Technology, RMIT University, Victoria 3000, [3]Department of Computer Science and Software Engineering, University of Melbourne, Victoria 3010, Australia and [4]School of Computer Engineering, Nanyang Technological University, Singapore 639798

Received on January 15, 2009; revised on May 18, 2009; accepted on June 15, 2009

Advance Access publication June 19, 2009

Associate Editor: Joaquin Dopazo

ABSTRACT

Motivation: Second-generation sequencing technologies produce a massive amount of short reads in a single experiment. However, sequencing errors can cause major problems when using this approach for *de novo* sequencing applications. Moreover, existing error correction methods have been designed and optimized for shotgun sequencing. Therefore, there is an urgent need for the design of fast and accurate computational methods and tools for error correction of large amounts of short read data.

Results: We present SHREC, a new algorithm for correcting errors in short-read data that uses a generalized suffix trie on the read data as the underlying data structure. Our results show that the method can identify erroneous reads with sensitivity and specificity of over 99% and 96% for simulated data with error rates of up to 3% as well as for real data. Furthermore, it achieves an error correction accuracy of over 80% for simulated data and over 88% for real data. These results are clearly superior to previously published approaches. SHREC is available as an efficient open-source Java implementation that allows processing of 10 million of short reads on a standard workstation.

Availability: SHREC source code in JAVA is freely available at http://www.informatik.uni-kiel.de/~jasc/Shrec/

Contact: jasc@informatik.uni-kiel.de

1 INTRODUCTION

Recently, a number of emerging (second-generation) DNA sequencing technologies has been introduced. Compared with the traditional shotgun technique (Sanger *et al.*, 1977), these technologies can generate a larger amount of read data at lower cost (Mardis 2008, Pop and Salzberg 2008, Strausberg *et al.*, 2008). Examples of already available second-generation technology include products from Illumina, 454 Life Sciences and Applied Biosystems. However, the length of produced reads is significantly shorter than for the classical Sanger technique. For example, the Illumina Genome Analyzer can generate up to 100 million reads in a single run with a read length between 35 and 75 and a per-base error rate between 1% and 2%.

*To whom correspondence should be addressed.

Established methods and tools for fragment assembly have been designed and optimized for shotgun sequencing (i.e. read lengths of around 500 bp and 6–10-fold coverage) and generally do not scale well for high-coverage short read data. Consequently, several new assemblers that are tailored towards short reads and high coverage have recently been introduced including SSAKE (Warren *et al.*, 2007), SHARCGS (Dohm *et al.*, 2007), Edena (Hernandez *et al.*, 2008), Velvet (Zerbino and Birney, 2008), ALLPATHS (Butler *et al.*, 2008) and Euler-SR (Chaisson and Pevzner, 2008; Chaisson *et al.* 2009). Although, these tools can assemble a few million of short reads in a reasonable amount of time, the quality of assembled contigs is highly sensitive to sequencing errors. For example, the per-base error rate for the Illumina 1G sequencing device has been reported to be between 1% and 2% (Dohm *et al.* 2008). One way to improve assembly results is to correct sequencing errors before assembly.

Re-sequencing is another important application area of second-generation sequencing technology. It requires the mapping of reads to a reference genome, which also can be simplified by error correction. Furthermore, reads are often trimmed towards their 3′-ends making them even shorter. Correcting errors in these trimmed parts, would lead to longer reads which in turn could improve assembly and mapping quality (Smith *et al.*, 2008).

In this article we present SHREC, a fast and scalable method to correct sequencing errors for high-throughput short-read sequencing data by building and traversing a generalized suffix trie built from the read data. By analyzing internal nodes of the trie, we are able to detect and correct errors accurately. In order to reduce memory consumption, we further present an efficient data partitioning method. This leads to an efficient implementation that can easily scale to up to 10 million reads on a standard workstation.

Previous work on error correction can be classified into methods designed for second-generation sequencing and those for classical shotgun sequencing. The SHARCGS assembly tool (Dohm *et al.*, 2007) avoids erroneous reads by a pre-filtering procedure: only reads that are generated at least *n* times by the sequencing machine and for which overlapping partners exist are selected. Although this method is simple and fast, it is only effective for very small error rates and very high coverage. The Euler-SR assembly program is a modification of the established Euler shotgun assembly program (Pevzner *et al.*, 2001) designed to cope with high-throughput short

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

read data. It contains a prior error correction step based on *spectral alignment* (SA) (Chaisson *et al*., 2004). The SA approach uses a set consisting of all solid l-tuples. An l-tuple is called *solid* if it occurs at least m times in the input read data set, and called *weak* otherwise. A string is called solid if all its l-tuples are solid. The SA problem for error correction takes each read r and computes a string r^* such that r^* is solid and $d(r, r^*)$ is minimal. Depending on the utilized sequencing technology $d(\cdot, \cdot)$ can either be edit distance (used for 454 sequencing data) or Hamming distance (used for Illumina sequencing data). Euler-SR contains a corresponding error correction programs to approximate spectral alignment problem (SAP). The error correction approach in ALLPATHS (Butler *et al*., 2008) is similar to the Euler-SR SAP heuristic, but uses an adaptive error threshold. In Section 5, we will show that SHREC is superior to the SAP-approach used in Euler-SR in terms of error correction accuracy.

Examples of error correction in classical shotgun sequencing include MisEd (Tammi *et al*., 2003) and the preprocessing step in Arachne (Batzoglou, 2002). Both approaches compute multiple alignments of shotgun reads and then detect errors in certain columns of these alignments. Unfortunately, the computation of such alignments is too time-consuming for the amount of reads produced by second-generation sequencers such as the Illumina Genome Analyzer. The approach presented in this article achieves its efficiency by avoiding the computation of alignments but simply traversing a space-efficient suffix trie representation of the read data.

2 METHODS

In the setting of short read fragment assembly, any error correction method relies on sufficient coverage; e.g. if the coverage at a position of an error is only two, we have no way to decide on which of the two reads contains the correct base. Obviously, the coverage needs to be significantly larger than two in order to do successful error correction. The main idea of the error correction method presented in this article is to store all reads produced by a sequencing machine plus their reverse complements in a generalized suffix trie and add leaf counts to the edges of internal nodes of the trie.

In the following, we will show that this data structure allows an effective characterization of sequencing errors. While optimal theoretical algorithms for constructing generalized suffix trees are known (Gusfield 1997), storing and manipulating them efficiently for the enormous number of reads produced by second-generation sequencers is challenging in practice and therefore requires a careful implementation. In Section 4, we provide implementation details of our data structure, which allow it to scale to tens of millions of reads on a standard workstation.

We assume a genome G of length n and a multi-set of k reads $\{R_1, \ldots, R_k\}$ each of length l. Both G and $R_i \in \sum^* = \{A, C, G, T\}^*$. Let $\Re$ denote the multi-set consisting of all reads plus their reverse complements. The coverage is defined as $c = l \times k/n$ (we assume k and n to be very large relative to l). We create a generalized suffix trie of $\Re$ with frequencies included in each internal node as follows.

Suffix trie: the *generalized suffix trie* ST($\Re$) (or just *suffix trie* for short) is a tree comprising all suffixes of all strings in $\Re$. Each edge in ST($\Re$) is uniquely labeled with a single character. To ensure the path from the root to each suffix is unique, we (logically) terminate each string in $\Re$ with a unique integer in the range $1, \ldots, 2k$. The concatenation of the edge labels on a path from the root to a leaf corresponds to a unique suffix of a string in $\Re$. Each leaf is labeled with an integer i denoting which string in $\Re$ the suffix is from (the same i which terminates the path). The *path-label* of node v, denoted path-label(v), is the string formed by concatenating the characters found on the path from the root to v. The *level* of a node v is the string length of path-label(v). The i-th *level* of ST($\Re$) is defined as the set of all nodes at level i. Level 0 is called the *top* (or *highest*) *level* of ST($\Re$) and l is called the *bottom* (or *lowest*) level.

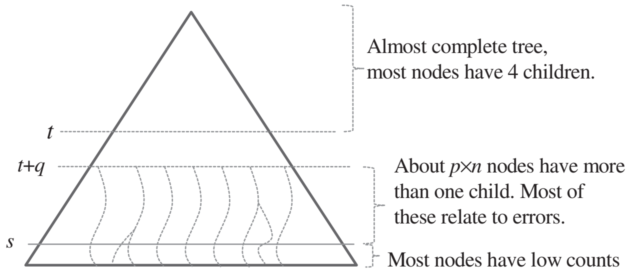
Almost complete tree,
most nodes have 4 children.
t
t+q
s
About p×n nodes have more than one child. Most of these relate to errors.
Most nodes have low counts

Fig. 1. Structure of the suffix trie ST($\Re$).

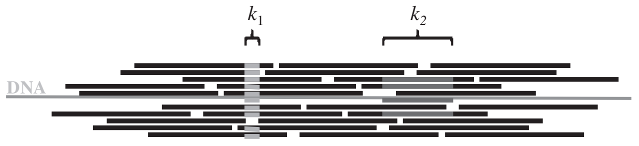

Fig. 2. An example of a piece of genomic DNA with a coverage of about 10, highlighting all sub-strings covering a sub-string of the DNA of lengths k_1 and k_2, respectively.

Weighted suffix trie: we *weigh* an edge of ST($\Re$) by the number of leaves in the sub-tree below this edge. The edge (v, w) is called the *associated edge* of node w (note each node has only one associated edge). The weight of a node is the weight of its associated edge. Observe that the weight of node w is precisely how often the string path-label(w) appears as a sub-string in the set of reads, $\Re$.

The weighted suffix trie of the read data has the following general characteristics.

(1) In the top t levels, with $t = \min\{\log_4(n), \log_4(k)\}$, the tree is almost complete, i.e. almost every node has four children.

(2) For any given string of length $t+r$ its expected number of appearances in a random string of length n is $n/(t+r)^4$. Thus, if all reads are correct, the expected number of nodes at level $t+r$ that have more than one child is only $1/r^4$-th of all nodes at this level.

(3) With a sequencing error rate of p and sufficient coverage, we expect at each level of the tree that at least $p \times n$ of all nodes have more than one child [the share of nodes with more than one child is significantly higher at higher levels of the tree; see (1)].

(4) The closer we get to the root of the tree, the less certain we can be whether a node leads us to an erroneous read. We determine the parameter q such that $1/4^q < p$ is satisfied. Thus, most nodes below level $t+q$ point us at a read with an error.

(5) Nodes below level s can have a too small weight to distinguish reliably between correct reads and erroneous reads.

Figure 1 illustrates the structure of ST($\Re$).

We now introduce the main criterion that allows us to decide whether a branch of the trie stems from a correct or from an erroneous read.

Any sub-string of length k has the expected occurrence of $c \times (l-k+1)/l$ in the set of reads. The example in Figure 2 illustrates this for $c = 10$. The sub-string of length k_2 appears three times in the set of reads, while the sequence of length k_1 appears nine times. If one of the reads has an error, a substring containing this error will only appear more than once if there is another read with the same error.

An error at position k of a read with $k > t+q$, (assuming that there is no error in the positions 1 to $k-1$) is likely to create a node at level $k-1$ in the

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

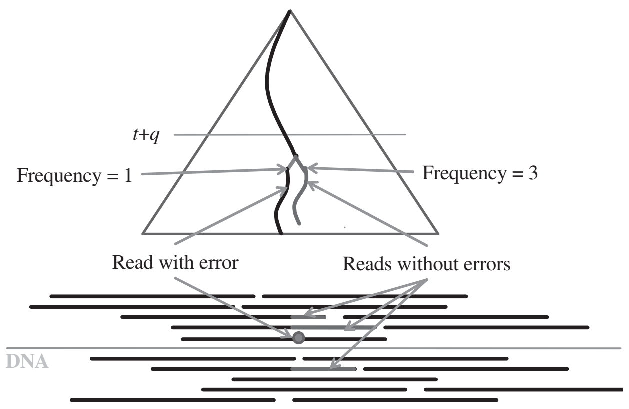

Fig. 3. The typical error scenario.

suffix trie with two children. Figure 3 illustrates this scenario for one child corresponding to the erroneous read and one child corresponding to correct reads covering the same area. In general, correct reads and reads with errors have the following characteristics visible in the suffix tree.

- Correct read: due to the coverage of the input data, any substring of length $k < l$ of a read has an expected occurrence of $(l-k+1) \times c/l$ (see Fig. 2 for an illustration).
- Erroneous read: a substring of length $k < l$ that contains an error has an expected occurrence of only $1+(l-k+1) \times p \times c/(3 \times l)$.

In the above formulas we have not taken account of the possibility that the same sub-string might appear several times at different positions of the genomic DNA being sequenced. This is always possible but highly unlikely in random strings as long as we investigate only nodes below level $t+q$. However, long repeats within genomic DNA happen. This is not entering the formulas as we know neither their frequency nor their length.

The above criterion is used as the main criterion in our error correction algorithm presented in the next section. We can apply this criterion to complete reads as well as to prefixes, suffixes and any other sub-strings of reads; i.e. we can use it to evaluate whether a sequence of weights attached to a suffix of a read is likely to have been produced by a correct read or by a faulty read.

3 ALGORITHM

The proposed weighted suffix trie is analyzed for imbalances of edge weights of neighboring edges which points to potential errors in the associated reads. The initial step of the algorithm is constructing the trie from all reads and their reverse complements. Subsequently, the trie is traversed as follows.

(1) Perform a depth-first traversal of ST($\Re$) inspecting the nodes from level s up to level $t+q$ for potential errors

(2) Identify all nodes v with at least two children where one of the children w has a smaller than expected weight.

(3) For each identified node w find the set of reads $R(w)$ belonging to the suffixes in the sub-tree below w.

(4) For each read $R_i \in R(w)$ examine if correction to a sibling of w fits the suffix.

   (a) If so, calculate error-position in R_i and correct the nucleotide to the edge label of siblings' associated edge.

   (b) Otherwise, mark R_i as erroneous.

(5) After all nodes have been analyzed: if there are marked reads that have not been corrected during the algorithm remove them from the set of reads before assembly.

We now discuss how the steps of the algorithm are performed in more detail:

Step 1: as described in the previous section, the part of the ST($\Re$) where errors can be effectively identified is between levels s and $t+q$. Still, we traverse the tree below level s as well in order to gather leave information for Step 3.

Step 2: the expected weight of a node depends on its level and the coverage of the genome. We are using the expected value $E(m)$ and the SD $\sigma(m)$ of the weight of a node at level m. The number of positions in G that can cover a sub-string of length m using a read of length l is $a = l-m+1$. Then

$$E(m) = ka/n \text{ and } \sigma(m) = k(a/n - a^2/n^2).$$

A node at level m is identified as having a smaller than expected weight if its weight is below $E(m) - x \cdot \sigma(m)$, where x is a parameter. A low value of x (e.g. $x=1$) will result in detection of all errors but many false positives (FPs) too—even with an ideal standard distribution of the reads only ~80% of the node weights are within this range. A higher value of x reduces the chances of FPs but may result in missing errors that have a high number of occurrences. Our experiments have shown the best results in terms of correction rate versus FPs for $5 \leq x \leq 7$ for synthetic read data. However, tests with experimentally obtained read datasets have obtained the best results outside this range. Thus, our current implementation provides the SHREC algorithm with a static user-defined value for the expected visits of a node.

Step 3: Once a node is identified the algorithm retrieves the corresponding reads and positions in which the error has been detected. This *suffix-to-read* mapping requires a careful implementation if memory usage is to be kept low and we describe one such approach the next section.

Step 4a: In this case, an error has been found in the correct position in the corresponding read. This can be determined by analyzing the sub-tree beneath the faulty node and its neighbors: the erroneous sub-tree should exactly fit into one of the neighboring trees. This implies that a correction of the nucleotide represented by the erroneous node to the nucleotide represented by the matching neighbor will fix the error. The position of the error in the original read can be obtained by comparing the read-length (l), the suffix-length (l') and the position of the error in the suffix (pos) as $l - l' +$ pos.

Step 4b: in this case, an error has been found at a position different from its actual occurrence. This can happen if an error re-routes the suffix to another matching part of the tree. The erroneous suffix then matches a part of this sub-tree before it branches out. Hence, identification of nodes in this case does not lead us to the actual error position. This situation can also be detected by analyzing the sibling(s) of an identified node: the correction of the single nucleotide to a sibling's edge label will most likely not lead to a perfect match of the sub-tree. Therefore, the algorithm does not perform a correction in this case but only marks the read as erroneous.

Also note that reads with more than one error are more difficult to correct in ST($\Re$). Even if the correct position of the first error is discovered, it will not lead us to a perfectly matching path in the corrected sub-tree because of the second error. A practical solution

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

to this problem is as follows: instead of demanding a perfectly matching sub-tree after a correction, we consider it sufficient if the paths only for a certain amount of bases following the corrected base match. A more accurate way would be to analyze the hamming distance of suffixes following the corrected nucleotide but would be significantly more time consuming.

Algorithmic options: we discuss three options of the presented error correction algorithm.

(1) *Identify-only approach* (*no actual error correction*): a simple method to handle errors in the read is by ignoring erroneous reads and hope for enough error-free reads to perform subsequent *de novo* genome assembly. This approach can obviously only work if the error rate is low and the coverage is high. The algorithm would only need to identify nodes with an unexpectedly low weight, find the corresponding reads and delete them. This method is often not sufficient for common error rates and correction of the reads does not add much complexity to the algorithm so we abandon this idea.

(2) *Static approach* [*correcting errors without updating* ST($\Re$)]: once an error is found the corresponding read is identified as explained above and the error is corrected. Afterwards the read will be marked so the same error would not be corrected again.

(3) *Dynamic approach* [*correcting errors with updating* ST($\Re$)]: as soon as there are two errors in one read, it is impossible to find both error-positions correctly since the first error re-routes the suffixes to the wrong sub-tree (see Step 4b). The algorithm has better chances to correct these multi-error-reads if the suffix trie is updated each time an error is corrected. The update operation needs to take the identified erroneous read and all of its suffixes out of the tree and place the corrected read and all of its suffixes in the tree again. The updated trie then allows for an easier identification of a potential second error. Furthermore, this dynamic updating approach can lead to a cascading effect. Corrections affecting nodes in a way that one after the other the structure becomes clearer and more and more errors can be identified and corrected. The drawback to this method is its time consumption. Updating a read and all its suffixes is an expensive operation: the algorithm has to update l different suffixes—and each of these needs $O(l)$ tree operations. This leads to an additional work of $O(l^2)$ for each correction.

4 IMPLEMENTATION

The implementation of the presented algorithm needs to address the following three issues.

(1) *Suffix-to-read mapping*: for each identified node v, all reads with the sub-string path-label(v) need to be retrieved. This is a frequent operation and therefore needs to be implemented efficiently.

(2) *Memory reduction*: a large number of reads and high error rates can result in a suffix trie with a prohibitively large memory footprint. Therefore, a mechanism is required that keeps memory consumption within a feasible range.

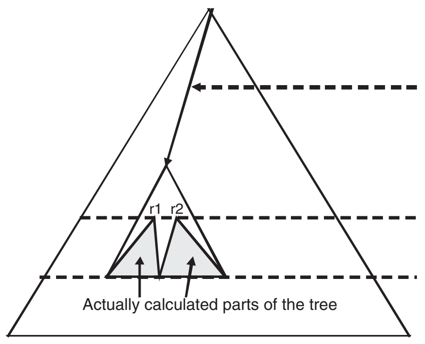

Fig. 4. Reducing the memory consumption of the suffix trie by (i) building and analyzing only the sub-trie starting with *Prefix*; (ii) limiting the levels of this sub-trie to the interval (*Startlevel*, *Endlevel*).

(3) *Trie Updating*: updating of the suffix trie data structure needs to be supported for the dynamic error correction approach.

Suffix-to-read mapping: each read in $\Re$ is assigned a unique read number id, with id $\in \{1,\dots,2k\}$ (in the following, we refer to reads and read numbers interchangeably). Our implementation uses the permutation $\Re'$ of $\Re$ to implement the suffix-to-read mapping. $\Re'$ sorts the reads of $\Re$ in lexicographical order using the *reverse* of all reads as keys (the *reverse* of a read $r=r_1 \dots r_k$ is defined as $r' = r_k \dots r_1$). We use the following trick to avoid explicit representation of leaves in the trie using $\Re'$. Access to leaf lists can then be provided with a constant amount of overhead per node by storing two indices into $\Re'$. Consider an internal node v. The leaf labels of the leaves that are direct children of v correspond precisely to the reads that end in path-label(v), i.e. the reads that have path-label(v) as a suffix. These reads appear in a contiguous portion of $\Re'$ and so rather than store these labels explicitly as children of v we need only store two indices cl, cr into $\Re'$.

Memory reduction: the total amount of memory required to store ST($\Re$) is determined by the number of nodes times the per-node memory consumption. The total amount of nodes depends on the actual read dataset but can be asymptotically estimated as follows. For error-free read data, the number of nodes is $O(l \times n)$. Each erroneous read creates additional $O(l^2)$ nodes. With the assumption that the number of erroneous reads is $O(n)$, this results in a total amount of $O(l^2 \times n)$ nodes. Obviously, the amount of memory needed for this implementation grows beyond the capacities of a regular desktop PC even for moderately sized genomes and very short read-lengths. Our implementation uses the following techniques to deal with the memory consumption problem.

First, the suffix trie is cut into smaller sub-tries, which are built and analyzed independently. Each sub-trie corresponds to a particular prefix. This prefix indicates the sub-trie in which the process will work during the algorithm. Second, two parameters are used to limit the levels of each sub-trie. This allows restricting construction and analysis of the sub-trie only to the promising parts and keeping the less significant parts from consuming memory. Figure 4 illustrates the two described memory reduction techniques.

Please note that uneven distribution of the nucleotides in real genomes (favoring the letters A and T) leads to uneven numbers of nodes following each prefix. The differences can be easily up to a factor of 1000 (when, for example, comparing a prefix with high number of A, T with the one only consisting of C's). This problem

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

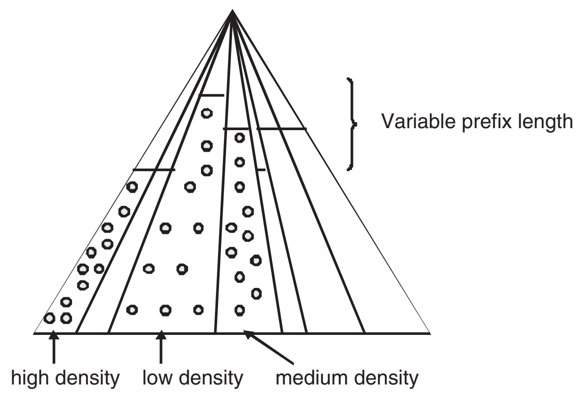

Fig. 5. Balancing the amount of memory used per sub-trie by using a variable prefix length.

can be dealt with by choosing the prefix lengths of each sub-trie depending on the quantity of C's and G's; i.e. choosing a shorter prefix length for those with a high CG content (Fig. 5).

Trie updating: the data structure containing the reads is progressively altered according to the error corrections made. As a result our partitioned implementation has a dynamic character: subsequently built sub-tries benefit from the corrections made earlier, since a correction in one sub-trie affects suffixes in several other sub-tries. Furthermore, we run two consecutive iterations of the error correction algorithm in order to support the correction of reads with multiple errors.

5 PERFORMANCE EVALUATION

We have evaluated the performance of the presented SHREC algorithm using several simulated Solexa/Illumina-style datasets as well as a real Solexa/Illumina dataset. The simulated datasets have been produced by generating random reads with a given error-rate from a reference genome sequence. In order to test scalability, we have selected yeast chromosomes [*Saccharomyces cerevisiae chromosome 5* (*S.cer5*) and *chromosome 7* (*S.cer7*)] and bacterial genomes [*Haemophilus influenzae* (*H.inf*), *Escherichia coli* (*E.col*)] as reference sequences of various lengths (ranging from 0.58 Mbp to 4.71 Mbp). Three datasets have been created for each reference genome sequence using per-base error-rates of 1%, 2% and 3%, a coverage of $C=70$, and a constant read length of $L=70$. To test the robustness of SHREC, we have further created three datasets with a lower coverage of $C=35$. Thus, the size of simulated input datasets varies from 0.3 m to 9.4 m reads. The real dataset consists of 3.43 m unambiguous reads (i.e. they do not contain any non-determined nucleotide) of length 35 each, which was downloaded from http://www.genomic.ch/edena.php and has a relatively low coverage of 43. It has been obtained experimentally by Hernandez *et al.* (2008) using the Illumina Genome Analyzer for sequencing the *Staphylococcus Aureus* strain MW2 (*H.Aci*). We have estimated the error rate of this dataset as 1% by aligning each read to the reference genome using RMAP (Smith *et al.*, 2008). The 16 datasets used for our performance evaluation are summarized in Table 1.

We have measured the runtime and error correction/identification performance of these datasets on an Intel Xeon multi-core machine with 2.6 GHz. The current version of SHREC is implemented in Java. In order to take advantage of multiple cores, we are using multithreading; i.e. each thread constructs and analyzes an independent prefix. Table 2 shows the runtime for 13 selected datasets using JVM 1.5. It can be seen that the runtime of SHREC scales linearly with the genome length and with the error-rates. This becomes clear if we look at the construction of the trie, which dominates the runtime: a genome of length n provides roughly $2n$ different possible reads or suffixes. This limits the size of the trie at each (higher) level. An error in a read however introduces new nodes to the trie: regardless of the error position in the read, l new suffixes of the read will be changed and possibly create new nodes. The complexity for the trie size follows $O(n \times l \times e)$ for each level k of the trie where $4^k > n$. The correction step requires scanning each node on one specific level and comparing sub-tries in case of a suspected error. In theory, the comparison is a depth-first search trie operation. The characteristics of the error correction trie make it run in linear runtime with the height of the compared sub-tries (four in our case). So, the overall complexity follows to $O(n \times l \times e)$ as well.

Table 1. Datasets used for performance evaluation

ID	Reference genome (GenBank)	Genome length (M)	Error rate (%)	Coverage	Number of reads (M)
A1l	*S.cer5* (NC_001137)	0.6	1	35	0.3
A2l			2		
A3l			3		
A1h			1	70	0.6
A2h			2		
A3h			3		
B1	*S.cer7* (NC_001139)	1.1	1		1.1
B2			2		
B3			3		
C1	*H.inf* (NC_007146)	1.9	1		1.9
C2			2		
C3			3		
D1	*E.col* (NC_000913)	4.7	1		4.7
D2			2		
D3			3		
E	*S.aureus* (NC_003923)	2.8	1	43	3.4

Table 2. Runtime (in seconds) for selected datasets

A1h	A2h	A3h	B1	B2	B3	C1	C2	C3	D1	D2	D3	E
185	220	270	520	620	710	865	1010	1205	2350	2725	2750	739

We have analyzed the accuracy of SHREC in terms of

(1) *identification*: identifying reads as erroneous or error free, and
(2) *correction*: actual correction of erroneous reads.

The identification of erroneous reads can be defined as a binary classification test. The corresponding definitions of true positive (TP), FP, true negative (TN) and false negative (FN) are given in Table 3. Sensitivity and specificity measures are defined as:

$$\text{Sensitivity} = \text{TP}/(\text{TP}+\text{FN}); \text{Specificity} = \text{TN}/(\text{TN}+\text{FP}).$$

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

Table 3. Definitions of TP, FP, FN and TN for the read identification classification test

	Read condition	
Algorithm outcome	Erroneous	Error free
Fixed or discarded	TP	FP
Unchanged	FN	TN

Table 4. Performance of the read identification classification test measured in terms of sensitivity and specificity for each dataset (the number of reads in the four sets TP, FP, TN and FN as well as the percentage of discarded reads in TP are also shown)

	TP (discarded) (%)	FP	FN	TN	Sensitivity	Specificity
A1l	140 215 (0.44)	320	3575	141 836	0.975	0.998
A2l	208 485 (0.64)	413	7425	70 429	0.966	0.994
A3l	241 740 (0.74)	298	10 038	35 007	0.960	0.992
A1h	281 472 (0.32)	0	6238	283 181	0.978	1
A2h	418 351 (0.38)	1	12 801	140 452	0.970	1
A3h	488 156 (0.82)	0	15 783	70 074	0.969	1
B1	543 864 (0.43)	0	12 558	545 892	0.977	1
B2	806 972 (0.36)	0	24 719	271 246	0.970	1
B3	942 457 (0.77)	0	30 143	134 697	0.969	1
C1	936 523 (0.46)	2	24 112	943 698	0.975	1
C2	1 397 114 (0.74)	4	44 774	468 391	0.969	1
C3	1 625 092 (0.71)	0	53 554	232 858	0.968	1
D1	2 311 124 (0.31)	1	61 545	2 334 385	0.974	1
D2	3 443 262 (0.40)	0	111 421	1 159 156	0.969	1
D3	4 007 602 (0.36)	0	130 905	575 856	0.968	1
E	828 374 (0.90)	5865	28 012	2 537 749	0.967	0.998

Table 4 shows the specificity and sensitivity measures for 13 tested datasets. It can be seen that that SHREC identifies erroneous reads with very high sensitivity and specificity. We have further analyzed the reads that have been classified as erroneous. These reads are further sub-divided into

- *corrected reads*: reads identified as erroneous and corrected by SHREC, and
- *discarded reads*: reads identified as erroneous but not corrected by SHREC.

The amount of *corrected reads* relative to the total number of reads and relative to the total number of erroneous reads is shown Table 5. Furthermore, we have compared the correction performance of SHREC with the SAP-based error correction approach implemented in Euler-SR. The correction performance of Euler-SR on the same datasets is also shown in Table 5.

Table 5 shows better robustness of SHREC compared with the SAP-approach in Euler-SR with respect to correction accuracy. While the accuracy of SHREC remains at least 80%, for 3% error-rate datasets and low coverage, Euler-SR drops significantly. This can be explained by the larger number of erroneous reads with more than one error for higher error-rates. These reads are difficult to correct by the SAP approximation algorithm used by Euler-SR. Note that Table 5 shows only completely corrected reads. SHREC also

Table 5. Percentage of corrected reads relative to the total number of reads (*corrected*) and relative to the total number of erroneous reads (*accuracy*) by SHREC and EULER-SR (using default parameters for Illumina data)

	SHREC		Euler-SR	
	Corrected (%)	Accuracy (%)	Corrected (%)	Accuracy (%)
A1l	49.2	95.2	41.3	83.2
A2l	73.2	89.3	52.9	71.1
A3l	84.8	81.1	49.8	57.4
A1h	49.4	95.7	39.9	80.2
A2h	73.4	90.5	50.6	68.0
A3h	85.6	84.0	16.0	16.9
B1	49.4	95.3	40.8	80.3
B2	73.4	90.0	51.7	68.0
B3	85.7	83.3	13.1	13.7
C1	49.3	94.1	39.4	80.0
C2	73.5	88.2	49.9	67.7
C3	85.5	81.0	47.0	53.5
D1	49.2	93.5	40.2	80.0
D2	73.3	87.4	50.9	67.7
D3	85.3	80.0	47.7	54.4
E	27.0	88.3	9.4	33.4

delivers partially corrected reads which can still be of use during the assembly process. Taking partially corrected reads into account as well, SHREC delivers accuracy well beyond 97% even for high-error rates. A further observation is that the accuracy for the real dataset (i.e. dataset E) is lower than for the simulated datasets with a similar error-rate. There are two reasons for this. First, the lower coverage and second, that errors in real reads are not as evenly distributed as in our simulated reads (Dohm *et al*., 2008). It can be seen that the suffix-trie-based approach used in SHREC can deal well with this situation, while the accuracy of the SAP-based approach in Euler-SR breaks further down.

In order to demonstrate the practical gain of SHREC, we have analyzed how the performance of *de novo* assembly tools can benefit from our error correction method. We have conducted experiments with the Edena assembler (Hernandez *et al*., 2008) using the datasets A1h, A2h, A3h with the original reads and the set of reads output by SHREC as inputs. The obtained results (using the minimum overlap of 40, which gave the best assembly results) are shown in Table 6. It can be seen that assembly greatly improves by using error-corrected reads (particularly for higher error rates).

6 CONCLUSION AND FUTURE WORK

Emerging HTSR sequencing technologies present a major bioinformatics challenge. In particular, bioinformatics tools that can process massive amounts of short reads are required. In this article, we have addressed this challenge by presenting the design and implementation of SHREC, a new short read error correction method. Using a suffix trie based approach; SHREC is able to achieve high-identification sensitivity and specificity as well as high-correction accuracy for both simulated and real datasets. SHREC also outperforms the previously used SAP-based error correction approach and can deal well with low coverage and uneven error distribution.

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016

Table 6. Assembly quality produced by Edena using the original read data sets A1h, A2h and A3h compared with the corresponding error-corrected read datasets by SHREC

Dataset	Number of contigs (≥100 bp)	N50	Mean contig length (bp)	Maximum contig length (bp)
A1h	413	10 459	1437	27 932
A1h corrected	241	46 438	2421	77 021
A2h	413	5989	1409	17 949
A2h corrected	265	28 424	2207	73 079
A3h	941	496	366	1921
A3h corrected	277	29 542	2111	66 009

Furthermore, we have used a partitioning technique to allow SHREC to process 10 million reads on a standard desktop computer. Extending the current SHREC implementation to be able to process >100 million reads and to handle insertions and deletions is part of our future work. We further plan to adapt SHREC to correct read data from other sequencing technologies (such as the 2-base encoding system used by the SOLiD sequencing platform). The suffix trie representation of read data could also be used to do subsequent *de novo* short read assembly. The assembly algorithm could use the notion of extensibility based on suffix links in the suffix trie. It would be interesting to compare this method with the commonly used graph-based assembly approaches.

Funding: This work was supported by the Australian Research Council, and by the NICTA Victorian Research Laboratory. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Center of Excellence program.

Conflict of Interest: none declared.

REFERENCES

Batzoglou,S. *et al*. (2002) ARACHNE: a whole-genome shotgun assembler. *Genome Res.*, **12**, 177–189.

Butler,J. *et al*. (2008) ALLPATHS: *de novo* assembly of whole-genome shotgun microreads. *Genome Res.*, **18**, 810–820.

Chaisson,M.J. *et al*. (2009) *De novo* fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res.*, **19**, 336–346.

Chaisson,M.J. and Pevzner,P.A. (2008) A short read fragment assembly of bacterial genomes. *Genome Res.*, **18**, 324–330.

Chaisson,M.J. *et al*. (2004) Fragment assembly with short reads. *Bioinformatics*, **20**, 2067–2074.

Dohm,J.C. *et al*. (2007) SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic assembly. *Genome Res.*, **17**, 1697–1706.

Dohm,J.C. *et al*. (2008) Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acid Res.*, **36**, e105.

Gusfield,D. (1997) *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, USA.

Hernandez,D. *et al*. (2008) De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Res.,* **18**, 802–809.

Mardis,E.R. (2008) The impact of next generation sequencing on genetics. *Trend Genet.*, **24**, 133–141.

Pevzner,P.A. *et al*. (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci.*, **98**, 9748–9753.

Pop,M. and Salzberg,S.L. (2008) Bioinformatics challenges of new sequencing technology. *Trend Genet.*, **24**, 142–149.

Sanger,F. *et al*. (1977) DNA sequencing with chain-terminating inhibitors. *Proc. Natl Acad. Sci.*, **74**, 5463–5467.

Smith,A.D. *et al*. (2008) Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, **9**, 128.

Strausberg,R.L. *et al*. (2008) Emerging DNA sequencing technologies for human genomic medicine. *Drug Discov. Today*, **13**, 569–577.

Tammi,M.T. *et al*. (2003) Correcting errors for shotgun sequencing. *Nucleic Acid Res.*, **31**, 4663–4672.

Warren,R.L. *et al*. (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**, 4, 500–501.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on April 6, 2016