BIOINFORMATICS

Vol. 00 no. 00 2010
Pages 1–8

HiTEC: accurate error correction in high-throughput sequencing data

Lucian Ilie [1,*], Farideh Fazayeli [1] and Silvana Ilie [2]

[1]Department of Computer Science, University of Western Ontario, N6A 5B7, London, ON, Canada
[2]Department of Mathematics, Ryerson University, M5B 2K3, Toronto, ON, Canada

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: High-throughput sequencing technologies produce very large amounts of data and sequencing errors constitute one of the major problems in analyzing such data. Current algorithms for correcting these errors are not very accurate and do not automatically adapt to the given data.

Results: We present HiTEC, an algorithm which provides a highly accurate, robust, and fully automated method to correct reads produced by high-throughput sequencing methods. Our approach provides significantly higher accuracy than previous methods. It is time and space efficient and works very well for all read lengths, genome sizes, and coverage levels.

Availability: The source code of HiTEC is freely available at `www.csd.uwo.ca/~ilie/HiTEC/`

Contact: `ilie@csd.uwo.ca`

1 INTRODUCTION

DNA sequencing technologies have produced a revolution in biological research. Since the introduction of the Sanger method (Sanger *et al.*, 1977), hundreds of bacterial and eucaryotic genomes have been sequenced, including several human genomes. This led to a significant number of biological discoveries. High-throughput sequencing technologies, such as Illumina's Genome Analyzer, ABI's SOLiD, and Roche's 454, see, e.g., (Mardis, 2008), produce gigabytes of data in a single run, thus taking sequencing to a whole new level. They provide the ability to answer biological questions with revolutionary speed. Some of their many applications include whole-genome sequencing and resequencing, Single Nucleotide Polymorphism (SNP) discovery, identification of copy number variations, chromosomal rearrangements, etc. The impact of these technologies for everyday life, yet to be fully understood, will be far reaching.

Many algorithms and software tools have been created to deal with the large amount of data produced by these technologies. Two of the fundamental and most investigated problems are read mapping and genome assembly. The former assumes the existence of a reference genome and attempts to find the location of newly sequenced reads from a different genome of the same species (Campagna *et al.*, 2009; Eaves and Gao, 2009; Jiang and Wong, 2008; Jung Kim *et al.*, 2009; Langmead *et al.*, 2009; Li and Durbin, 2009; Li *et al.*, 2008a,b; Lin *et al.*, 2008; Malhis *et al.*, 2009; Rumble *et al.*, 2009; Schatz, 2009; Smith *et al.*, 2008). The latter attempts to reconstruct the genome that originated the reads (Butler *et al.*, 2008; Chaisson *et al.*, 2009; Chen and Skiena, 2007; Dohm *et al.*, 2007; Hernandez *et al.*, 2008; Jeck *et al.*, 2007; Simpson *et al.*, 2009; Warren *et al.*, 2007; Zerbino and Birney, 2008). In spite of the many different approaches these tools employ to solve their problems, they all share several common issues, such as very large data size, repeats in genomes, and sequencing errors. The first two cannot be changed and we shall concentrate here on sequencing errors. Attempts have been made to either correct such errors or discard the erroneous reads. Some assembly tools include a spectral alignment-based read correction preprocessing step (Chaisson *et al.*, 2009; Butler *et al.*, 2008), whereas others pre-filter the reads (Dohm *et al.*, 2007). The very recent approaches of Schroder *et al.* (2009); Shi *et al.* (2010); Salmela (2010) are exclusively dedicated to read correction.

The general idea for correcting reads is to use the high coverage of the current sequencing technologies in order to identify the erroneous bases in the reads. Each base is usually sampled many times and the correct value will prevail. The way such information is used can be spectral alignment (Shi *et al.*, 2010) or subtree weight in suffix trees (Schroder *et al.*, 2009). Whereas Shi *et al.* (2010) provides an efficient implementation of the Euler-SR read correction algorithm of Chaisson *et al.* (2009) by using CUDA-enabled graphics hardware (their program will subsequently be referred to as CUDA), the SHREC program by Schroder *et al.* (2009) uses a novel idea, by employing weighted suffix trees. The algorithm of Salmela (2010) is a generalization of SHREC to mixed sets of reads.

Error correction is quickly identified as a key problem in high-throughput sequencing data. Another software, Reptile, has been developed by Yang *et al.* (2010) simultaneously with ours. It is also based on the k-spectrum approach of Euler-SR and CUDA.

Our HiTEC — High Throughput Error Correction — algorithm uses a thorough statistical analysis of the suffix array built on the string of all reads and their reverse complements. It is intuitively explained in Section 2.2 and fully analyzed in the remaining of Section 2. We have tested in Section 3 our algorithm on many data sets, simulated or real, from Schroder *et al.* (2009), Shi *et al.* (2010), and Yang *et al.* (2010), as well as on several new ones. The accuracy of HiTEC is significantly higher than the accuracy of all the other programs. (The accuracy is the ratio between the

*To whom correspondence should be addressed number of corrected reads and the number of initially erroneous reads.) Further, our own testing reveals a significant difference between the accuracy obtained by running the SHREC program and that reported by Schroder *et al.* (2009). (We provide the values for both.) This is due to the fact that SHREC requires trying several parameter sets in order to find those providing the highest accuracy, which is likely not possible in real situations where the accuracy cannot be measured. HiTEC is not only more accurate but also more robust. Our algorithm works for a wide range of read lengths and coverage levels and it is the only one to do so with automatic adjustment, depending on the data set, based on our statistical analysis. In addition to high accuracy, the time and space complexities are very good. Our current serial implementation of HiTEC is comparable with Reptile and is about six times faster than the parallel implementation of SHREC on the four-processor machine we used for testing. The space consumption is comparable with Reptile and lower than that of SHREC for all tests. Nevertheless, we plan to improve the time and space of our algorithm by providing a parallel implementation. This and other further research directions are presented in Section 4 together with a summary of the achievements.

2 METHODS

2.1 Strings and suffix arrays

This section contains the basic definitions for strings and recalls the suffix array data structure. Consider the alphabet of four bases $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. A string is any finite sequence over Σ. The set of all strings over Σ is denoted by Σ^*. The length of a string s is denoted by $|s|$ and, for $1 \le i \le |s|$, $s[i]$ is the ith letter of s. A substring of s is any consecutive sequence of letters from s, i.e., $s[i \mathinner{..} j] = s[i]s[i+1] \cdots s[j]$; in particular, for $|s| = m$, $s = s[1 \mathinner{..} m]$. For $i = 1$ we obtain a prefix and for $j = m$ a suffix of s. The reverse complement $\bar{s}$ of s, is obtained by first reversing s and then applying the transformation $\mathtt{A} \leftrightarrow \mathtt{T}, \mathtt{C} \leftrightarrow \mathtt{G}$. For example, if $s = \mathtt{CAT}$, then $\bar{s} = \mathtt{ATG}$. Clearly, $\bar{\bar{s}} = s$.

Let suf_i denote the suffix $s[i \mathinner{..} m]$ of s. Assuming a total order on the alphabet Σ, the *suffix array* of s, denoted SA, gives the increasing lexicographical order of the suffixes of s, i.e., $\mathsf{suf}_{\mathsf{SA}[1]} < \mathsf{suf}_{\mathsf{SA}[2]} < \cdots < \mathsf{suf}_{\mathsf{SA}[m]}$. For example, the suffix array of the string ACTAACACTG is shown in the second column of Fig. 1. The suffix array is often used in combination with the longest common prefix (LCP) array that gives the length of the longest common prefix between consecutive suffixes of SA, that is, $\mathsf{LCP}[i]$ is the length of the longest common prefix of $\mathsf{suf}_{\mathsf{SA}[i]}$ and $\mathsf{suf}_{\mathsf{SA}[i-1]}$; see the fourth column of Fig. 1. By definition, $\mathsf{LCP}[1] = 0$.

The suffix array data structure has been introduced by Manber and Myers (1993); the SA array can be computed in $\mathcal{O}(m)$ time and space by any of the algorithms of Kärkkäinen and Sanders (2003); Kim *et al.* (2005); Ko and Aluru (2005); the LCP array can be computed also in $\mathcal{O}(m)$ time and space by the algorithm of Kasai *et al.* (2001). However, suboptimal algorithms exist which behave much better in practice. We have used the `libdivsufsort` library of Yuta Mori[1] in our program. Also, since we need only bounded LCP values, we preferred a direct computation of the LCP, thus avoiding (Kasai *et al.*, 2001) altogether.

[1] `libdivsufsort`: A lightweight suffix sorting library, `http://code.google.com/p/libdivsufsort/`.

i	$\mathsf{SA}[i]$	$\mathsf{suf}_{\mathsf{SA}[i]}$	$\mathsf{LCP}[i]$
1	4	AACACTG	0
2	5	ACACTG	1
3	1	ACTAACACTG	2
4	7	ACTG	3
5	6	CACTG	0
6	2	CTAACACTG	1
7	8	CTG	2
8	10	G	0
9	3	TAACACTG	0
10	9	TG	1

Fig. 1. The SA and LCP arrays for the string ACTAACACTG. The longest common prefixes of consecutive suffixes are underlined; their length gives the LCP values.

2.2 Basic idea for correcting errors

The basic idea for correcting errors in reads is intuitively explained below. Consider a genome $\mathscr{G}$ of length L that will be modeled as a random string over Σ, where each letter appears with equal probability 0.25. Assume also that n reads, $r_1, r_2, \ldots, r_n$, each of length ℓ, have been produced from $\mathscr{G}$ with the per-base error p. That is, each read has been obtained by randomly choosing a substring of length ℓ of $\mathscr{G}$ and then changing each letter into any of the other three with equal probability $\frac{p}{3}$. (Reads containing any letter not in Σ are discarded.) We call the position *erroneous* if its letter is different from the corresponding one in the genome and *correct* otherwise.

We construct the string of all reads and their reverse complements

$$R = r_1\$\,\bar{r}_1\$\,r_2\$\,\bar{r}_2\$\ \cdots\ r_n\$\,\bar{r}_n\$\ ,$$

where \$ is a letter not in Σ. Denote the suffix array and longest common prefix array of R by SA and LCP, respectively.

The basic idea for correcting an erroneous position in a read is as follows. Assume that the read r_i, sampled starting on position j of the genomes, contains an error in position k and that the previous w positions, $r_i[k-w \mathinner{..} k-1]$, are correct. That is $r_i = xuay$, where $x, u, y \in \Sigma^*$, $|x| = k - w - 1$, $|u| = w$, and $a \in \Sigma$. The letter a is different from the letter $b = \mathscr{G}[j + k - 1]$ that actually appears in the genome. However, many other reads will have sampled that region correctly, that is, they contain the correct substring ub. The fact that u is followed more often by b than by a will let us suspect that a should actually be b and, if the evidence is strong enough, we shall replace a by b. The string u is witnessing that a is an error and also that it should be changed into b. Formally, a *witness* is any substring u of R, of an a priori fixed length w, such that u contains no occurrence of \$. For $a \in \Sigma$, the *support* of u for a, $\mathsf{supp}(u, a)$, is the number of occurrences of the string ua in R. See Fig. 2 for an example.

Our algorithm consists of computing the support values and using them, based on thorough statistical analysis, to correct erroneous bases. Note that our support values are similar with the weights of the suffix tree vertices in SHREC. The similarity with SHREC stops here as we use more efficient data structures for computation and completely different procedure for correcting.

```
r_i1    CGTCTCCTCCAAGCCCTGTTGTCTCATACC
r_i2         TCCTCCAAGCCCTGTTGTCTCTTACCAGGA
r_i3     GTCTCCTCCAAGCCCTGTTGTCTCTTACCC
r_i4             TCCAAGCCCTGTTGTCTCTTACCCGCATGT
r_i5            CTCCAAGCCCTGTTGTCTCTTACCCGGATG
r_i6                CAAGCCCTGTTGTCTCTTACCCGGATGTTC

𝒢   ...  CGTCTCCTCCAAGCCCTGTTGTCTCTTACCCGGATGTTC  ...
```

Fig. 2. An example of an error covered by six reads; the genome region where the reads came from is shown at the bottom. The letter (inside the frame) following the witness $u = \texttt{CTGTTGTCTC}$ (underlined) should be T and not A. The support values are $\mathsf{supp}(u, \texttt{T}) = 5$ and $\mathsf{supp}(u, \texttt{A}) = 1$. If we omit the grey part, then the remaining suffixes are lexicographically sorted, as in SA.

2.3 Statistical analysis

We now formalize the idea in the previous section. For a given witness u, we define the *cluster* of u as the set of all positions in R where an occurrence of u followed by a letter different from \$ starts. The size of this cluster is

$$\mathsf{clust}(u) = \sum_{a \in \Sigma} \mathsf{supp}(u, a) \, .$$

All these positions are consecutive in SA and so are all occurrences of u supporting the same letter. This makes it easy to compute the support values and cluster size, given the suffix array. The clusters, corresponding to witnesses of a given length w, are easily found using the LCP values: a cluster consists of all consecutive positions with LCP values w or higher so that the $(w + 1)$st letter is not \$. In Fig. 2, the occurrences of a witness are shown in the order in which they appear in the suffix array.

Assume for now that any witness u of length w does not appear elsewhere in the genome since additional occurrences would make the identification of the errors more difficult. Due to repeats in the genome, this may not be possible but what we can do is reduce the probability of random occurrences in our Bernoulli model. We have two competing goals here. A long u will be less likely to appear again in $\mathscr{G}$ but will be covered by fewer reads, thus reducing its useful support. We shall return to this issue.

We first estimate the support given by a witness u to a letter a following it. We need to distinguish between the case when the witness contains errors and when it does not. As a witness may appear with errors in some reads and without errors in others, precise definitions are needed. A witness is *correct* if it occurs as a substring of $\mathscr{G}$ and *erroneous* otherwise.

Consider the case of a correct witness $u = \mathscr{G}[i \mathinner{.\,.} i + w - 1]$ supporting a correct letter $a = \mathscr{G}[i + w]$. A read covering both has to start within the interval $[i - \ell + w + 1 \mathinner{.\,.} i]$ in $\mathscr{G}$. The probability that a given read starts in this interval and contains no errors inside ua is $q_c = \frac{\ell - w}{L}(1 - p)^{w+1}$. If R_c is the number of such reads, then

$$\mathrm{Prob}(R_c = k) = \binom{n}{k} q_c^k (1 - q_c)^{n-k}$$

and thus the expected number of pairs (u, a), both u and a correct, given that $\mathsf{supp}(u, a) = k$, is

$$W_c(k) = \left(\begin{smallmatrix} n \\ k \end{smallmatrix}\right) q_c^k (1 - q_c)^{n-k} L \, .$$

Assume next that w is correct but a is an error. We now need those reads covering ua that have no error inside u but the original letter in a's position, say b, has been replaced by a. Again, there are $\ell - w$ positions where they can start but the probability that a given read starting in that interval has the errors exactly as specified is $q_e = \frac{\ell - w}{L} \frac{p}{3} (1 - p)^w$. The number R_e of such reads has the probability distribution

$$\mathrm{Prob}(R_e = k) = \binom{n}{k} q_e^k (1 - q_e)^{n-k} \, .$$

The expected number of such (u, a) pairs, u correct, a erroneous, given that $\mathsf{supp}(u, a) = k$, is

$$W_e(k) = \left(\begin{smallmatrix} n \\ k \end{smallmatrix}\right) q_e^k (1 - q_e)^{n-k} L \, .$$

In the case when u is erroneous and a is correct, we may assume that u contains only one error, since otherwise the support is much lower. Then, we are interested in all the reads covering ua and containing the same error as u. Therefore, the reasoning is very similar with the one for the case when u is correct and a is erroneous. In the remaining case, when both u and a are erroneous, the support is much lower.

The above analysis helps us compute a threshold, T, that will distinguish the support by a correct witness for a correct letter from the support when either the witness or the position, or both, are erroneous. Often, there is an interval of values k where both $W_e(k)$ and $W_c(k)$ are very small and any T in this interval is good. Also, this interval grows when the error rate decreases and hence a good value of T remains good when some of the errors have been corrected. An example is shown in Fig. 3 where the values of $W_e(k)$ and $W_c(k)$ are plotted for error 0.01 in the left plot; the right one shows the region where both $W_e(k)$ and $W_c(k)$ are very small.

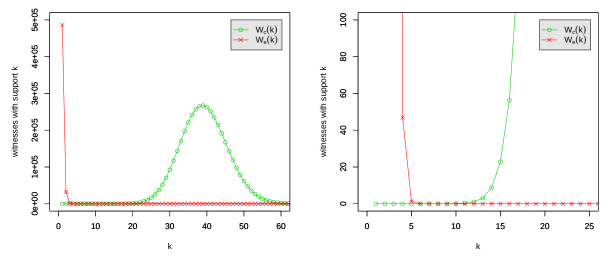

Fig. 3. The values of $W_c(k)$ and $W_e(k)$ for $L = n = 4.2$ mil., $\ell = 70$, $w = 21$, $p = 0.01$ are shown in the left plot whereas the right one shows the region of the left one where the values of both $W_c(k)$ and $W_e(k)$ are very low. The value of the threshold T in this example equals 9.

To cover also the case when such a region, with very low values of both $W_e(k)$ and $W_c(k)$, does not exist (as it happens for low coverage), we increase the value of T by an experimentally computed constant of two:

$$T = \min(\{k \mid W_c(k) > W_e(k)\}) + 2\,. \qquad (1)$$

A problem is that some reads will have their errors distributed in such a way that there are no w consecutive correct positions. That makes it impossible to fit a correct witness at any position and therefore such reads cannot be corrected using the current procedure. We approximate first their number and then adjust the algorithm to correct most of those as well. Denote by $f_w(k,\ell)$ the number of possible ways to place k errors in a read of length ℓ such that any interval of length w contains at least one error. The value of $f_w(k,\ell)$ can be easily computed using this recursive formula:

$$f_w(k,\ell) = \begin{cases} \binom{\ell}{k}, & \text{if } \ell < w, \\ 0, & \text{if } k < \left\lfloor \frac{\ell}{w} \right\rfloor, \\ \sum_{i=1}^{w} f_w(k-1, \ell-i), & \text{otherwise.} \end{cases}$$

Then, the expected number of such reads is $f_w(k,\ell)p^k(1-p)^{\ell-k}n$. These values for error rates 0.01, 0.02, and 0.03 are shown for a 4MB genome in the left plot in Fig. 4. The number of reads with k errors decreases with k but $f_w(k,\ell)$ increases and so the maximum is reached somewhere around 4-5 errors.

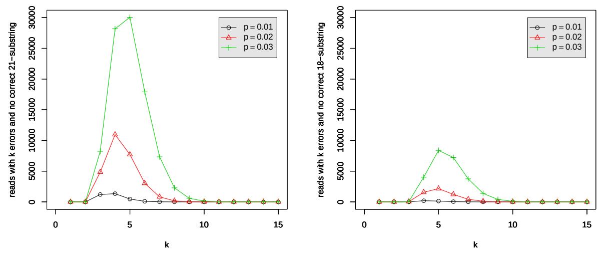

Fig. 4. The number of reads with a given number of errors and no error-free interval of length w for $L = n = 4.2$ mil. and $\ell = 70$. The left plot uses $w = 21$ and the right $w = 18$.

We are interested in the total number of reads uncorrectable with a witness of length w, that is,

$$U(w) = \sum_{k=1}^{\ell} f_w(k,\ell)p^k(1-p)^{\ell-k}n\,.$$

The percentage this number represents, in our example in Fig. 4, for a witness of length $w = 21$, out of the total number of expected erroneous reads, $E_e = (1-(1-p)^{\ell})n$, is 0.15 for $p = 0.01$, 0.87 for $p = 0.02$, and 2.56 for $p = .03$. We need to lower therefore the length of the witness in order to correct some of these reads. When the witness length is reduced to 18, the percentages drop to 0.02, 0.17, and 0.68 respectively (see the right plot in Fig. 4).

However, another problem appears. The number of uncorrectable reads drops with the decrease of the witness length but the probability of the witness occurring more than once in the genome is no longer negligible causing correct positions to be wrongly changed as follows. Assume that ua appears in $\mathscr{G}$ and that ua is sampled by a read as va, that is, a is correct but u contains errors that change it into v. The length of v is also w and the probability of v appearing in $\mathscr{G}$ is non-negligible. Assume v occurs in $\mathscr{G}$ at some position followed by $b \neq a$. Then the support given by v to b will be very large and to a very small, causing a to be changed, incorrectly, into b.

We now approximate the number of such errors. We need that u has errors, a is correct, v appears in $\mathscr{G}$ and $b \neq a$. The probability for this to happen is

$$q_w = (1-(1-p)^w)(1-p)\Big(1-\Big(1-\frac{1}{4^w}\Big)^L\Big)\frac{3}{4}\,.$$

The probability that at least one correct position in one read is changed this way is $1-(1-q_w)^{\ell-w}$ and the expected number of destructible reads, that is, correct reads the are turned erroneous this way is

$$D(w) = (1-(1-q_w)^{\ell-w})(1-p)^{\ell}n\,.$$

Thus, lowering the witness length w decreases the number $U(w)$ of uncorrectable reads but increases the number $D(w)$ of destructible reads. We therefore need the w that minimizes $U(w)+D(w)$:

$$w_m = \arg\min_w (U(w)+D(w))\,. \qquad (2)$$

It can be seen from Fig. 5 that the optimal values for $p = 0.01, 0.02, 0.03$ are $w_m = 19, 17, 16$, respectively.

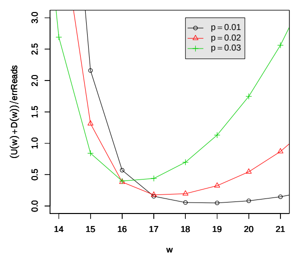

Fig. 5. The values of $U(w)+D(w)$ as percentages of the total number of erroneous reads, E_e, for $L = n = 4.2$ mil., $\ell = 70$, and $p = 0.03$.

Choosing $w = w_m$ provides, theoretically, the highest accuracy for the current iteration and a greedy strategy becomes apparent. However, it turns in out that a combination of values that are close to the optimal w_m and the smallest w that causes essentially no correct reads to be changed, that is,

$$w_M = \min(\{w \mid D(w) < 0.0001E_e\})\,, \qquad (3)$$

works best in practice. While witnesses of length w_M effectively correct all but the uncorrectable $U(w_M)$ reads, those of length w_m will create large enough stretches of consecutive correct positions inside an additional $U(w_M) - U(w_m)$ reads so that they become correctable by witnesses of length w_M. Also, w_M satisfies the conditions under which we computed the parameter T and hence it will be also used for this purpose. The sequence of witness lengths used in the HiTEC algorithm, denoted $w_{\text{seq}} = w_{\text{seq}}[1\,..\,9]$, is:

$$w_m{+}1, w_M{+}1, w_M{+}1, w_m, w_M, w_M, w_m{-}1, w_M{-}1, w_M{-}1\ . \quad (4)$$

Finally, some reads will contain several errors which often require several iterations of our correcting procedure. Instead of estimating at the beginning the number of iterations needed, it is more reliable to stop the process when the number of corrected positions in a single iterations drops below a certain threshold. Such an approach, with an experimentally computed value for the threshold will be used in our algorithm.

2.4 The algorithm

The pseudo code of the HiTEC algorithm that results from the above reasoning in shown in Fig. 6. Note that only L and p are required as input parameters. An approximate value for L is probably known before the experiment. If not, then L can be estimated by a biological experiment or an expectation-maximization procedure (such as in (Myers, 2005)). An approximate value of p should be known from the machine that does the sequencing.

```
HiTEC $(r_1, \ldots, r_n)$
given: $n$ reads $r_1, \ldots, r_n$ (of length $\ell$ each); $L$ and $p$
output: $n$ corrected reads
 1. compute $w_M$ and $w_m$            // using (3) and (2), resp.
 2. compute $T$                        // using (1) with $w = w_M$
 3. $i \leftarrow 1$                   // iteration number
 4. repeat
 5.    $c \leftarrow 0$                // bases changed this iteration
 6.    $w \leftarrow w_{\text{seq}}[i]$   // from (4)
 7.    construct $R$ and compute SA and LCP
 8.    compute the clusters in SA for all witnesses of length $w$
 9.    for each witness $u$ with clust$(u) \geq T + 1$ do
10.       Corr $\leftarrow \{a \mid \text{supp}(u, a) \geq T\}$
11.       Err $\leftarrow \{a \mid \text{supp}(u, a) \leq T - 1\}$
12.       for each $a \in$ Err do
13.          if $(|\,\text{Corr}\,| = 1)$ then
14.             correct $a$ to $b \in$ Corr    // change both $r$ and $\bar{r}$
15.             $c \leftarrow c + 1$
16.          if $(|\,\text{Corr}\,| \geq 2)$ then
17.             for each $b \in$ Corr do
18.                if ($ua$, $ub$ followed by same two letters) then
19.                   correct $a$ to $b$     // change both $r$ and $\bar{r}$
20.                   $c \leftarrow c + 1$
21.    $i \leftarrow i + 1$
22. until $\big((\frac{c}{\ell n} < 0.0001)$ or $(i > 9)\big)$
23. return all $r_j$'s from $R$
```

Fig. 6. The HiTEC algorithm.

As previously mentioned, in Step 7, we used for the construction of the SA array the `libdivsufsort` library, the state-of-the-art algorithm for suffix array construction, which is significantly faster and more space efficient than the theoretically optimal algorithms. For the LCP array, we only need to check in Step 8 whether the LCP values are smaller or larger than w. We eliminate the need to store the LCP array by employing a direct computation of these values. Cache effects ensured that the time remains essentially the same.

For each witness u with a large enough cluster, the sets of correct and erroneous letters supported by u are built in Steps 10-11. If there is no ambiguity, then we correct in Step 14. If there is ambiguity, then we check also the next two letters (Step 18) in order to decide how to correct. In either case, the position of a in the string R corresponds to a position inside a read r (which can be some r_j or $\bar{r}_j$) and we correct both r and its reverse complement $\bar{r}$.

The procedure stops when the number of bases changed during one iteration drops below 0.01% of the total number of bases (Step 22). In any case, no more than 9 iterations are performed. This last condition is necessary since the ratio between the number of bases changed in one iteration and the total number of bases becomes less reliable as an indicator of the actual number of corrected reads when the number of iterations increases.

One practical improvement not mentioned in the algorithm is that, due to the excellent performance of HiTEC for modest coverage, data sets of high coverage can be split into several sets of lower coverage which are independently corrected, thus saving space.

3 RESULTS

3.1 Accuracy

The *accuracy* is defined as the ratio between the number of corrected reads and the number of initially erroneous reads. The erroneous/correct status of each read (before and after correcting) has been determined by a straightforward search (using suffix arrays) of all reads in the genome sequence. To be precise, if the number of erroneous reads before and after correction is err_{bef} and err_{aft}, respectively, then accuracy is the ratio $\frac{\text{err}_{\text{bef}} - \text{err}_{\text{aft}}}{\text{err}_{\text{bef}}}$. If we denote by TP, TN, FP, FN (true/false positive/negative) the number of erroneous reads that are corrected, correct reads that are left unchanged, correct reads that are wrongly changed, and erroneous reads that are left unchanged, respectively, then we have $\text{err}_{\text{bef}} = TP + FN$, $\text{err}_{\text{aft}} = FP + FN$ and therefore

$$\text{accuracy} = \frac{\text{err}_{\text{bef}} - \text{err}_{\text{aft}}}{\text{err}_{\text{bef}}} = \frac{TP - FP}{TP + FN}\ .$$

We have compared the accuracy of our algorithm on quite a number of data sets[2], including those of Schroder *et al.* (2009), Shi *et al.* (2010), and Yang *et al.* (2010). All programs have been run with default parameters. Several bacterial genomes, see Table 1, were downloaded from GenBank under the accession numbers given. Later on we shall refer to these genomes by the IDs given in parentheses in the first column. To be precise, we constructed a number of data sets by uniformly sampling reads with given length, coverage, and per-base error rate from the above genomes. The data sets can be considered identical with those of Schroder *et al.* (2009) and Shi *et al.* (2010) because they have been generated from the same genomes with the same parameters. Experiments show that the differences between the accuracy of a program on different data sets generated with the same parameters as above are insignificant.

The data sets used by Schroder *et al.* (2009) are shown in Table 2, those of Shi *et al.* (2010) in Table 3, and Table 4 contains a mixture

[2] Most of these tests were performed on the SHARCNET high performance computers: `www.sharcnet.ca`.

Table 1. The genomes used for comparison.

Reference genome (ID)	Accession no.	Len.(bp)
Saccharomyces Cerevisiae, Chr. 5 (S.cer5)	NC_001137	576,869
Saccharomyces Cerevisiae, Chr. 7 (S.cer7)	NC_001139	1,090,946
Haemophilus Influenzae (H.inf)	NC_007146	1,914,490
Escherichia coli str.K-12 substr.MG1655 (E.coli)	NC_000913	4,639,675
Escherichia coli str.K-12 substr.DH10B (E.coli2)	NC_010473	4,686,137
Staphylococcus aureus (S.aureus)	NC_003923	2,820,462
Helicobacter acinonychis (H.acinonychis)	NC_008229	1,553,927

of read lengths and coverage levels taken from the longest genome considered.

Table 5 contains several real sets of Illumina reads. The first real data set was used also by Schroder *et al.* (2009); it is available from `www.genomic.ch/edena.php` and it was previously used by Hernandez *et al.* (2008). Both the first and the second real data sets were used by Shi *et al.* (2010). The second one is available from `sharcgs.molgen.mpg.de/download.shtml` and it was used initially by Dohm *et al.* (2007). The third one is new and is available from `clcbio.com/index.php?id=1290`, the CLCbio web site, as an example of NGS data. The reads in each of the first three data set were selected using RMAP (Smith *et al.*, 2008) as previously done, according to Schmidt (2010):

```
./rmap -c chromosome_dir -w 36 -m 3 -v -o
          output_file reads_file
```

The per-base error rate was computed by counting the total number of mismatches from the output file of RMAP. The fourth real data set has been suggested by one of the reviewers as a most recent example of Illumina reads. It has accession number ERA000206. Due to its very large size, the SHREC program could not produce any results. The performance of HiTEC is very high in spite of the fact that no selection of the reads using RMAP has been done such as for the previous three real data sets.

Table 6 contains the relevant data sets from Yang *et al.* (2010). A common feature of the sets of reads in the Sequence Read Archive (SRA, `http://www.ncbi.nlm.nih.gov/sra`) is that the sequence of the genome from which the reads were sequenced is not known and therefore, considering another genome instead can produce misleading results. Out of the data sets of Yang *et al.* (2010), the percentage of the reads that can be mapped on the indicated reference genome is around 97% for the first two data sets and around 60-70% for the other ones. This is clear indication that the genomes for the last four data sets are not the actual genomes that produced the reads. Therefore, we could meaningfully use only the first two.

In Tables 2-5 the "SHREC" column gives the accuracy values we obtained by running the SHREC program whereas the values in the "SHRECpaper" column are taken from (Schroder *et al.*, 2009). The values in the "CUDA" column are taken from (Shi *et al.*, 2010), that is, we did not test the program. (The values are missing for tests not performed by Shi *et al.* (2010).)

Several comments are in order. First, the accuracy of HiTEC is significantly higher than that of all the other programs for all experiments. The difference is significantly higher for the real data sets. While the accuracy of HiTEC is similar to the case of simulated data and is not affected by the error rate or coverage, CUDA's accuracy is much lower compared to the simulated data and SHREC's accuracy decreases dramatically with the increase of the error rate. Reptile's accuracy is comparable to that of SHREC when quality scores are unavailable (Table 2) and still significantly below the accuracy of HiTEC even when using quality scores (Table 6). HiTEC performs particularly better for lower coverage.

Second, there is a significant difference, which increases with genome length and error rate, between the accuracy of SHREC as resulting from our testing of the software and that provided by Schroder *et al.* (2009). This is due to the fact that Schroder *et al.* (2009) try a number of parameters and the best accuracy obtained was reported. As already mentioned, in our testing we have not adjusted the parameters of any programs. Note also that the accuracy values from (Shi *et al.*, 2010) are higher than the values obtained by the tests of Euler-SR reported by Schroder *et al.* (2009). Anyway, the accuracy of HiTEC is significantly higher than all the other accuracy values.

Third, the SHREC program was run with the same number of iterations as ours, as resulted from the stopping criterion in Step 22. In most cases, the accuracy of HiTEC after one or two iterations is already higher than that of SHREC after nine iterations.

Similarly with SHREC, the parameters of Reptile are fixed. That means they do not adapt to the data. As a result, Reptile could not correct any errors of the fourth data set from Table 5.

One last remark, our measure, accuracy, is different from the gain of Yang *et al.* (2010). For the data sets considered in Table 6, the gain for HiTEC is 83.33 and 82.22, respectively, whereas Reptile produces 82.81 and 72.53. So, gain indicates again that HiTEC has better performance.

Table 2. Accuracy comparison for the data sets of (Schroder *et al.*, 2009). The read length is 70bp and coverage is 70 for all data sets.

Data set		Accuracy				
Genome	err.(%)	SHREC	SHRECpaper	CUDA	Reptile	HiTEC
S.cer5	1	95.85	95.70		92.89	99.79
S.cer5	2	88.93	90.50		83.22	99.55
S.cer5	3	78.15	84.00		71.71	99.40
S.cer7	1	94.83	95.30		92.93	99.74
S.cer7	2	85.60	90.00		83.38	99.58
S.cer7	3	71.61	83.30		71.90	99.39
H.inf	1	91.21	94.10	87.50	93.00	99.73
H.inf	2	76.35	88.20	76.60	83.57	99.50
H.inf	3	55.84	81.00	63.60	72.08	99.02
E.coli	1	89.37	93.50		92.98	99.75
E.coli	2	71.38	87.40		83.45	99.42
E.coli	3	47.80	80.00		71.97	99.22

3.2 Time and space

In addition to obtaining very high accuracy, our algorithm has also very good time and space complexities, due to the use of good data structures. The tests shown in Table 7 were performed for the data sets in Table 2 on a Sun Fire V440 Server, with four UltraSPARC IIIi processors at 1593MHz, 4GB RAM each, running SunOS 5.10. Our serial implementation of HiTEC is about six times faster than the multithreaded SHREC. The space required by our algorithm is comparable to that of Reptile and both are lower than

Table 3. Accuracy comparison for the data sets of (Shi *et al.*, 2010). The read length is 35bp and coverage is 70 for all data sets.

Data set		Accuracy		
Genome	err.(%)	SHREC	CUDA	HiTEC
S.cer5	1	96.09	83.50	96.27
S.cer5	2	93.43	77.20	96.90
S.cer5	3	89.46	69.90	93.95
S.cer7	1	95.31	83.60	95.76
S.cer7	2	92.27	77.20	95.86
S.cer7	3	88.13	69.90	93.48
H.inf	1	93.34	83.50	96.39
H.inf	2	89.45	77.20	94.80
H.inf	3	83.93	69.90	89.83
E.coli	1	91.50	83.60	94.41
E.coli	2	87.06	77.20	94.37
E.coli	3	80.76	69.90	91.13

Table 4. Accuracy comparison between SHREC and HiTEC for a variety of read lengths, coverage levels, and error rates sampled from the E.coli genome.

Data set				Accuracy	
Genome	read len.	covrg.	err.(%)	SHREC	HiTEC
E.coli	70	35	1	93.44	99.75
E.coli	70	35	2	87.87	99.46
E.coli	70	35	3	80.84	99.25
E.coli	50	50	1	93.44	99.25
E.coli	50	50	2	88.85	98.75
E.coli	50	50	3	83.65	97.88
E.coli	50	35	1	93.31	99.27
E.coli	50	35	2	89.20	99.06
E.coli	50	35	3	83.85	97.91
E.coli	35	50	1	91.60	94.37
E.coli	35	50	2	87.40	94.37
E.coli	35	50	3	82.32	91.15

Table 5. Accuracy comparison for several real sets of Illumina reads.

Data set				Accuracy			
Genome	read len.	covrg.	err.(%)	SHREC	SHRECpaper	CUDA	HiTEC
S.aureus	35	42.5	1.00	74.75	88.30	48.30	93.38
H.acinonychis	36	188.8	1.60	34.83		47.40	91.26
E.coli2	35	17.8	0.38	80.65			90.40
E.coli	100	574	0.50	—			87.49

Table 6. Accuracy comparison between Reptile and HiTEC on two sets of reads from the E.coli genome that were used by Yang *et al.* (2010).

Data set				Accuracy	
Accession number	read len.	covrg.	err.(%)	Reptile	HiTEC
SRX00429	36	160	0.44	84.32	86.17
SRR001665_1	36	80	0.38	75.28	85.78

SHREC's. Reptile is slightly faster however, the running time of HiTEC includes many iterations. In fact HiTEC achieves higher accuracy sooner.

Table 7. Time and space comparison between SHREC and HiTEC. The read length is 70bp and coverage is 70 for all data sets.

Data set		Time (s)			Space (MB)		
Genome	err.(%)	SHREC	Reptile	HiTEC	SHREC	Reptile	HiTEC
S.cer5	1	3126	151	543	1340	727	399
S.cer5	2	4261	222	665	1350	531	399
S.cer5	3	7597	284	1193	1293	666	399
S.cer7	1	6511	373	1069	1502	1184	754
S.cer7	2	10649	618	1581	1512	1314	754
S.cer7	3	15823	792	2399	1745	1148	754
H.inf	1	9595	783	1971	1675	1434	1324
H.inf	2	15826	1340	2866	2090	1630	1324
H.inf	3	23319	1810	4253	3072	1771	1324
E.coli	1	23530	2741	5107	3194	1865	3210
E.coli	2	32073	5365	6290	3628	2266	3210
E.coli	3	57185	8812	11193	3437	2711	3210

3.3 Very large genomes

We have also performed measurements to predict the ability of our algorithm to correct errors in the case of very large genomes. We plotted in Fig. 7 the percentage of $U(w) + D(w)$ (uncorrectable plus destructible reads) for genome size 1GB. We used two common read lengths from Illumina: 75 and 100.

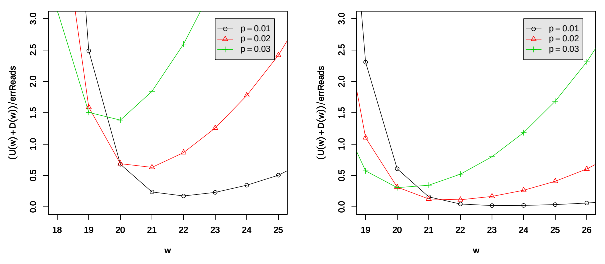

Fig. 7. The values of $U(w) + D(w)$ as percentages they represent out of the total number of erroneous reads, E_e, for $L = n = 1$ bil. The left plot uses read length $\ell = 75$ and the right one $\ell = 100$.

In practice, the error rate increases with read length but so does our algorithm's performance, only faster. While for reads of size 35 the ratio of those that can be corrected decreases below 50% for very large genomes, the situation is much better already for read size 50. In a 1GB genome, for an error rate of 0.01 and a read length 50, our algorithm can correct up to 97.72% of the erroneous reads but when we increase the read length to 100, we can correct up to 97.70% even for a very high error rate of 0.03.

4 CONCLUSION AND FUTURE RESEARCH

The main goal of this paper is to provide an algorithm that is highly efficient at correcting the errors of high throughput sequencing technologies. According to the extensive testing we have performed, our algorithm exceeds significantly the accuracy of previous algorithms. Also, a thorough statistical analysis makes our program the only one that is able to automatically adjust to the input data. The testing has been done on Illumina reads but the approach is suitable for any type of reads for which the errors consist mainly of substitutions.

The ability of our program to correct increases with the length of the reads and, according to (Zhou *et al.*, 2010), the read length is going to grow with the 3rd generation of sequencing technologies, such as single molecule sequencing or nanopore sequencing. We therefore hope that our program will be very competitive even with the rapid change of sequencing technologies.

We plan to work on a parallel implementation which will be able to handle the even more massive outputs to come. The new implementation will be capable of dealing with all types of reads as well as with mixed sets of reads. Quality scores as well as additional knowledge of the bias of the sequencing devices concerning the actual distribution of the positions of the reads in the genome could help us improve further the accuracy of our algorithm.

Acknowledgements We would like to thank the anonymous reviewers for suggesting the fourth data set in Table 5 and for pointing out to us the Reptile program.

Silvana Ilie and Lucian Ilie were each supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

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

Campagna,D., *et al.* (2009), PASS: a program to align short sequences. *Bioinformatics* **25** 967 – 968.

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.

Chen,J., and Skiena,S., (2007), Assembly for double-ended short-read sequencing technologies, in: E. Mardis, S. Kim, and H. Tang, eds., *Advances in Genome Sequencing Technology and Algorithms*, Artech House Publishers, New York, 123 – 141.

Chen,Y., *et al.* (2009), PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds, *Bioinformatics* **25** 2514 – 2521.

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

L.H. Eaves and Y. Gao (2009), MOM: maximum oligonucleotide mapping, *Bioinformatics* **25** 969 – 970.

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.

Jeck,W.R., *et al.* (2007), Extending assembly of short DNA sequences to handle error, *Bioinformatics* **23** 2942 – 2944.

Jiang, H., and Wong,W.H, (2008), SeqMap: mapping massive amount of oligonucleotides to the genome, *Bioinformatics* **24** 2395 – 2396.

Jung Kim,Y., *et al.* (2009), ProbeMatch: a tool for aligning oligonucleotide sequences, *Bioinformatics* **25** 1424 – 1425.

Kärkkäinen,J., and Sanders,P., (2003), Simple linear work suffix array construction, in *Proc. of ICALP'03*, Lecture Notes in Comput. Sci. **2719**, Springer-Verlag, Berlin, Heidelberg, 943 – 955.

Kasai,T., *et al.*, (2001), Linear-time longest-common-prefix computation in suffix arrays and its applications, *Proc. of CPM'01*, Lecture Notes in Comput. Sci. **2089**, Springer-Verlag, Berlin, 181 – 192.

Kim,D.K., *et al.*, (2005), Constructing suffix arrays in linear time. *J. Discrete Algorithms* **3**(2-4) 126 – 142.

Ko,P., and Aluru,S., (2005), Space efficient linear time construction of suffix arrays, *J. Discrete Algorithms* **3**(2-4) 143 – 156.

Langmead,B., *et al.* (2009), Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, *Genome Biol.* **10** R25.

Li,H., *et al.* (2008a), Mapping short DNA sequencing reads and calling variants using mapping quality scores, *Genome Res.* **18** 1851 – 1858.

Li,R., *et al.* (2008b), SOAP: short oligonucleotide alignment program, *Bioinformatics* **24** 713 – 714.

Li,H., and Durbin,R. (2009), Fast and accurate short read alignment with Burrows-Wheeler transform, *Bioinformatics* **25** 1754 – 1760.

Lin,H., *et al.* (2008), ZOOM! Zillions of oligos mapped, *Bioinformatics* **24** 2431 – 2437.

Malhis,N., *et al.* (2009), Slider-maximum use of probability information for alignment of short sequence reads and SNP detection, *Bioinformatics* **25** 6 – 13.

Manber,U., and Myers,G., (1993), Suffix arrays: a new method for on-line search, *SIAM J. Comput.* **22**(5) 935 – 948.

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

Myers,G., (2005), Building fragment assembly string graphs, *Bioinformatics* **21** ii79 – ii85.

Rumble,S.M., *et al.* (2009), SHRiMP: Accurate Mapping of Short Color-space Reads, *PLoS Comput Biol.* **5** e1000386.

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

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

Schatz,M., (2009), Cloudburst: highly sensitive read mapping with mapreduce, *Bioinformatics* **25** 1363 – 1369.

Schmidt,B., (2010), Personal communication.

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

Shi,H., *et al.*, (2010), A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware, *J. Comput. Biol.* **17** 603 – 615.

Simpson,J.T., *et al.*, (2009), ABySS: A parallel assembler for short read sequence data, *Genome Research* **19** 1117 – 1123.

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

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

Yang, X., Dorman, K.S., and Aluru, S., (2010), Reptile: representative tiling for short read error correction, *Bioinformatics* **26** 2526 – 2533.

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

Zhou,X., *et al.* (2010), The next-generation sequencing technology: A technology review and future perspective, *Science China* **53** 44 – 57.