Correction of sequencing errors in a mixed set of reads

Leena Salmela
Department of Computer Science, PO Box 68 (Gustaf Hällströmin katu 2b), FI-00014 University of Helsinki, Finland

ABSTRACT

Motivation: High-throughput sequencing technologies produce large sets of short reads that may contain errors. These sequencing errors make de novo assembly challenging. Error correction aims to reduce the error rate prior assembly. Many de novo sequencing projects use reads from several sequencing technologies to get the benefits of all used technologies and to alleviate their shortcomings. However, combining such a mixed set of reads is problematic as many tools are specific to one sequencing platform. The SOLiD sequencing platform is especially problematic in this regard because of the two base color coding of the reads. Therefore, new tools for working with mixed read sets are needed.

Results: We present an error correction tool for correcting substitutions, insertions and deletions in a mixed set of reads produced by various sequencing platforms. We first develop a method for correcting reads from any sequencing technology producing base space reads such as the SOLEXA/Illumina and Roche/454 Life Sciences sequencing platforms. We then further refine the algorithm to correct the color space reads from the Applied Biosystems SOLID sequencing platform together with normal base space reads. Our new tool is based on the SHREC program that is aimed at correcting SOLEXA/Illumina reads. Our experiments show that we can detect errors with 99% sensitivity and >98% specificity if the combined sequencing coverage of the sets is at least 12. We also show that the error rate of the reads is greatly reduced.

Availability: The JAVA source code is freely available at http://www.cs.helsinki.fi/u/lnsalmel/hybrid-shrec/

Contact: leena.salmela@cs.helsinki.fi

Received on March 1, 2010; revised on April 1, 2010; accepted on April 5, 2010

1 INTRODUCTION

The high-throughput sequencing machines such as SOLEXA/Illumina, Applied Biosystems SOLID and Roche/454 Life Sciences produce millions of short reads in a single run. The reads may contain errors, which continues to present a challenge to de novo assemblers. The error rate of the reads can be reduced with trimming and by correcting the reads. The different sequencing platforms have their own benefits and shortcomings. For example, the distribution of error types varies from one platform to another (Shendure and Ji, 2008). The dominant error type in SOLEXA/Illumina and SOLID reads is substitution, reads produced by the 454/Roche platform tend to have many insertions and deletions because of the technology’s inability to accurately translate the length of homopolymer runs correctly and the dominant error type in Helicos reads is deletion.

While most sequencing platforms produce base space reads, i.e. the reads are sequences of bases A, C, G and T, the SOLID platform produces reads in color space (Applied Biosystems Incorporated, 2008a). The SOLID sequencer interrogates bases in overlapping pairs so that each base is sequenced twice. The pairs are coded with four colors as shown in Figure 1.

Due to different characteristics of the sequencing platforms, it is an attractive idea to combine reads produced by several platforms. This kind of mixed read sets could improve the results of both error correction and de novo assembly. There are not many tools that can take full benefit from a mixed set of reads especially if part of the reads are SOLID color space reads. We present this kind of tool for error correction in this article.

Many tools for error correction of reads from second generation sequencers use the spectral alignment method first introduced in the EULER-SR assembler (Chaisson et al., 2004; Pevzner et al., 2001). This method first computes the spectrum of the reads which consists of all l-tuples that are frequent enough in the read set. Then a string r* is computed for each read r such that all l-tuples in r* are in the spectrum and the distance between r and r* is minimized. The distance measure can be, e.g. the Hamming distance allowing only substitutions or edit distance allowing both insertions and deletions. A similar approach is taken in several error correction tools such as SOLID Accuracy Enhancement Tool (SAET, http://solidsoftwaretools.com/gf/project/saet/) and as preprocessing in many assemblers such as ALLPATHS (Butler et al., 2008) and SOAPdenovo (Li et al., 2010).

In the alignment approach, multiple alignments are computed for the reads and then errors are detected and corrected based on the columns of these alignments. This approach has been used with reads from the classical Sanger sequencing (Sanger et al., 1977). Examples of such tools are MisEd (Tammi et al., 2003) and the preprocessing step in Arachne (Batzoglou et al., 2002). The problem of this approach is that it is not feasible to compute the multiple alignments for millions of reads produced by the newer sequencing technologies. Recently, the error correction tool SHREC (Schröder et al., 2009) extended this approach to SOLEXA/Illumina reads by...
We use the following model of DNA sequencing. We have \( k \) reads randomly sampled from a genome of length \( n \). The length of the reads can vary. The reads may have errors with some rather small probability. The errors may be substitutions, insertions or deletions. We further assume that the errors are distributed randomly and that the coverage is sufficient so that each position of the genome is present in several reads.

As each position of the genome is sampled several times, it is possible to detect an error in a read by aligning it against other reads. If the errors are distributed randomly, most of the reads that overlap the erroneous read do not have the same error. The alignment is thus very good except for the error position in the erroneous read, and so we can use the alignment to detect and correct the error. We utilize a suffix trie to compute these alignments efficiently.

Note that if the reads are from a diploid organism, they may contain single nucleotide polymorphisms (SNPs) in addition to sequencing errors. SNPs look similar to sequencing errors in the multiple alignment as they single nucleotide polymorphisms (SNPs) in addition to sequencing errors.

We will extend the SHREC algorithm (Schröder et al., 2009), and so next we will give an overview of the data structures and methods used by it. For now we will assume that the reads are in base space.

Let \( R \) be the set of reads and their reverse complements. The generalized suffix trie \( ST(R) \) is a tree that contains all the suffixes of the strings in \( R \). We concatenate a unique symbol \( 1\ldots2k \) to the end of each string in \( R \) so that each suffix is unique. The edges of the tree are labeled with a character from the alphabet \{A,C,G,T\}. Each node may have only one child labeled with the same character. We call the concatenation of edge labels from the root to a node the path-label of that node. For each suffix of a string in \( R \), there is a leaf such that the path-label of the leaf is the suffix. The weight of a node is the number of leaves in the subtrie rooted at that node. Note that this is exactly the number of suffixes in the subtrie. The level of a node is the length of the path from the root to the node.

In the top levels of the trie, almost all nodes have four children as almost all strings of length equal to the level of the children can be found in the genome. Further down in the trie almost all nodes have only one child. This happens at some level \( r \) such that \( 4^{r} \) is greater than the length of the genome, i.e. most strings of length \( r+1 \) appear at most once in the genome. If a node at this level has more than one child, it is likely that the branching is caused by a sequencing error especially if the weight of one of the children is very small indicating that the subtrie rooted at that child has only a few suffixes in it. Still deeper in the trie the weights of the nodes become too small to distinguish between erroneous and correct children.

The SHREC algorithm traverses the generalized suffix trie and identifies the erroneous children at the intermediate levels of the trie. It then attempts to correct the read suffixes that go through the erroneous child by a substitution. Insertions and deletions can be detected in the same way as substitutions and we will explain our algorithm to correct this kind of errors in the next section.

The above-mentioned methods can be easily adapted to correct a set of color space reads. The reads are now strings from the alphabet \{0,1,2,3\} and the reverse complements of these reads can be formed just by reversing the string. The correction algorithm otherwise works exactly in the same way.

Correcting a combination of base space and color space reads is more intricate. If we transform a color space read to base space, a single error can change all the bases starting from the error, and thus it is not feasible to correct color space reads in base space. However, if we transform a base space read into color space, the errors remain local. As each base is used only in determining two colors in the color space representation of the read, a single error in base space can only affect two colors in color space. Thus, base space and color space reads can be corrected together in color space. If a read was originally a color space read, we will allow substituting, inserting and deleting a color when correcting the read, and if the read was originally a base space read, we will allow such color space transformations that correspond to substituting, inserting or deleting a base in the corresponding base space representation. Note that since SNPs change two colors in a color space read, SNPs are not corrected in color space reads.

The SOLID color code has the following algebraic interpretation (Applied Biosystems Incorporated, 2008b). A color represents a pair of bases and it can thus be seen as a function that transforms the first base of the pair to the second one. These color functions form an algebraic group where the operator is function composition. An algebraic group is closed so the composition of two colors is also a color. Figure 2 gives the compositions of all color pairs. If we think of the translation of a base space read to a color space read, then the composite function of two colors corresponds to a function that maps a base in position \( i \) to the base in position \( i+2 \). The changes reflected in the color space by an error in base space can now be characterized with the composite colors. A substitution causes two colors to change but the composite color remains the same, an insertion changes a color \( e \) to two colors \( c_{1} \) and \( c_{2} \) such that \( e \) is the composite color of \( c_{1} \) and \( c_{2} \), and a deletion changes two colors to their composite color. Figure 3 shows some examples.

### 3 ALGORITHM

First, we will review the SHREC algorithm (Schröder et al., 2009) for correcting substitutions in a set of reads of equal length. After that we will introduce our improvements to this algorithm.

#### 3.1 The SHREC algorithm

The SHREC algorithm (Schröder et al., 2009) starts by building a generalized suffix trie of the reads and then it attempts to correct the errors at the intermediate level of the trie.

![Fig. 2. The composite colors for each color pair.](image)

![Fig. 3. The color changes caused by errors in base space. When the nucleotide C is substituted with T, two colors change. In this example, 30 is changed to 12. We note that the composite color remains the same. 3. When the nucleotide C is deleted, the two colors 30 are replaced by their composite color 3. We further note that insertion is the reverse of a deletion.](image)
We will now modify the statistical model of the SHREC algorithm to account for reads of varying length. Let us assume that the read set contains reads of \( r \) different lengths and let us denote these lengths by \( l_1, l_2, \ldots, l_r \). Let the number of reads of length \( l_i \) be \( k_i \). Let \( m \) be sufficiently large so that each sequence of length \( m \) appears only once in the genome. The weight \( W_m \) of a node at level \( m \) in the generalized suffix trie is the number of substrings whose path in the trie passes through that node. We will use \( W_{m,i} \) to denote the contribution of reads of length \( l_i \) to the weight of a node at level \( m \). Thus, the weight of a node at level \( m \) is

\[
W_m = \sum_{i=1}^{r} W_{m,i}.
\]

Now if \( l_i < m \), \( W_{m,i} = 0 \). Otherwise each read is a Bernoulli trial for getting the path-label of the node. There are \( n \) substrings of length \( m \) in the genome and a read of length \( l_i \) samples \( a_i = l_i - m + 1 \) of them. Thus, the probability of success for the Bernoulli trial is \( a_i/n \). The total number of trials is \( k_i \) and so \( W_{m,i} \) is distributed according to the binomial distribution \( \text{Bin}(k_i, a_i/n) \). Thus, the expected value of \( W_{m,i} \) is

\[
E(W_{m,i}) = \frac{k_i a_i}{n} \text{ if } l_i \geq m, \quad 0 \text{ otherwise},
\]

and the variance of \( W_{m,i} \) is

\[
\sigma^2(W_{m,i}) = \frac{k_i (a_i/n - a_i^2/n^2)}{n} \text{ if } l_i \geq m, \quad 0 \text{ otherwise}.
\]

By the linearity of expectation we get

\[
E(W_m) = E\left(\sum_{i=1}^{r} W_{m,i}\right) = \sum_{i=1}^{r} E(W_{m,i}).
\]

If we assume that \( W_{m,i} \) for different \( i \) are independent variables then

\[
\sigma^2(W_m) = \sum_{i=1}^{r} \sigma^2(W_{m,i}) = \sum_{i=1}^{r} \sigma^2(W_{m,i}).
\]

The above assumption of the independence of \( W_{m,i} \) may be problematic if the length of a read depends on its genomic location. As in the original SHREC algorithm, we try to correct nodes whose weight is below \( E(W_m) - \alpha \cdot \sigma(W_m) \), where \( \alpha \) is the strictness parameter of the method.

### 3.3 Correcting insertions and deletions

Just like substitutions, insertions and deletions in the reads cause extra branching in the generalized suffix trie of the reads. Figure 5a shows how the insertion of a \( T \) in a read has created a low weight node in the generalized suffix trie. The insertion can be corrected by deleting the \( T \) in which case the substrates rooted at the children of the low weight node will be merged. Therefore, we compare these substrates to figure out if a deletion is a feasible way to correct a low weight node. If we find sibling substrates that contain the substrates rooted at the children of the low weight node, we have confirmed that a deletion can correct the erroneous node. We then find the reads whose suffixes are in the substrates rooted at the erroneous node and make the deletion.

If a read contains an insertion error in a homopolymer run (e.g. one of the A’s in the read GAAAATC is an insertion error), the
Correction of sequencing errors

![Diagram of base space and color space reads](attachment:diagram.png)

**Fig. 6.** The subtrees to compare when correcting a substitution, insertion or deletion of a base space read in color space.

error can be corrected by deleting any one of the bases in the run. However, only the last A creates a low weight node in the suffix trie, and thus we correct the error by deleting the last base in the run.

Figure 5b shows how the deletion of A has created an extra branch in the suffix trie. This deletion can be corrected by inserting an A in the read, and then the subtree rooted at the low weight node would be merged with the subtree rooted at the corresponding child of one of its sibling nodes. Therefore, we compare these subtrees to determine if an insertion is a feasible way to correct a low weight node. If the subtree rooted at the low weight node is included in a subtree rooted at a child node of the sibling nodes, we have identified an insertion that corrects the erroneous node. We then identify the reads whose suffixes are in the subtree rooted at the erroneous node and insert the appropriate character to these reads. If a read contains a deletion error in a homopolymer run, our algorithm detects a low weight node only after the last base in the run, and thus we correct the error by inserting the missing base to the end of the run.

3.4 Correcting indeterminate bases

It is rather straightforward to adapt the SHREC method to handle indeterminate bases. We now build the generalized suffix trie using the five-letter alphabet \{A,C,G,T,N\}. When running the correction algorithm over the suffix trie, we always regard a node whose base is N to be erroneous. We then attempt to correct the error by a substitution and a deletion. We do not attempt to correct the node by inserting a character before N as the same could be achieved by correcting the indeterminate base by substitution.

3.5 Combining base space and color space reads

The combined correction of base space and color space reads first transforms all the base space reads into color space. Then the generalized suffix trie is built on all the reads and their reverse transforms all the base space reads into color space. Then the generalized suffix trie is built on all the reads and their reverse transforms all the base space reads into color space. Then the algorithm then traverses the trie and identifies low weight nodes at the intermediate level of the trie. Color space reads whose suffixes are found in subtrees rooted at low weight nodes can then be corrected exactly as outlined above.

Base space reads whose suffixes are found in subtrees rooted at low weight nodes are corrected by using color transformations that correspond to substituting, deleting or inserting one base in the base space read. A substitution in base space changes two colors \(c_1\) and \(c_2\) into two colors \(c_3\) and \(c_4\) so that the composite color of \(c_1\) and \(c_2\) is the same as the composite color of \(c_3\) and \(c_4\). Therefore to correct a low weight node, we need to substitute both the color on the edge to the low weight node and the color on the edge between the low weight node and its child. To determine if this kind of substitution is a feasible way to correct a read, we compare the subtree rooted at the child node with such children of the siblings of the low weight node that the compositions of the colors on the edges are the same. Note that because of the rules of the color space, each sibling can have only one such child. See Figure 6a for an example.

An insertion in base space replaces a color \(c\) in color space with two colors whose composite color is \(c\). Thus, to correct a low weight node, we replace the colors on the edges adjacent to the low weight node by their composite color \(c\). To determine whether a deletion like this is a good way to correct a base space read, we compare the subtree rooted at the child of the low weight node with the subtree rooted at such a sibling of the low weight node that the color on the edge to the sibling is \(c\). See Figure 6b for an example.

A deletion in base space replaces two colors in color space with their composite color. In this case, a low weight node can be corrected by replacing the color \(c\) on the edge to the low weight node with two colors whose composite color is \(c\). Now we compare the subtree rooted at the low weight node with the subtrees rooted at such children of the siblings of the low weight node that the composition of the colors on the edges adjacent to the sibling is \(c\). Figure 6c shows an example of correcting a deletion.

After identifying how to correct a low weight node, we fetch the color space representations of the base space reads whose suffixes are in the subtree rooted at that node and correct them. If the base space reads contain no indeterminate bases, we can correct them fully in color space and after correction translate them back to base space. Indeterminate bases can be handled by consulting the original base space read to figure out the composite color when we encounter indeterminate colors in the reads translated to color space and by correcting the base space read directly whenever we correct the corresponding color space read.

4 IMPLEMENTATION

We have extended the implementation of SHREC (Schröder et al., 2009) with the ideas presented in the previous section. We provide two versions of the program. One can correct sets with only base space reads as the transformation to color space is an unnecessary
5 RESULTS AND DISCUSSION

We tested our implementation both on simulated reads and real reads from the Escherichia coli genome. The simulated reads are generated from the K-12 sub-strain MG1655 (NC_000913). The length of the simulated base space reads varies uniformly at random from 75 to 125 bp and the length of the simulated color space reads varies from 40 to 50 bp. The base space reads were generated with coverages of 6, 12, and 24, and error rates 1.5% and 3.0%. The color space reads were generated with coverages of 15 and 30, and error rates 1.4% and 2.9%. In the base space reads, each type of error is equally probable but in the color space reads only 5% of the errors are insertions or deletions as substitution is the dominant error type in reads produced by the SOLiD sequencing platform.

We downloaded real reads from the E.coli strain UTI89 produced by the Roche/454 sequencing platform from NCBI short read archive (accession number SRA000156) and real SOLiD reads from the E.coli strain DH10B from Applied Biosystems (http://solidsoftwaretools.com/gf/project/ecoli2x50). The SOLiD reads were filtered to exclude low-quality reads. The coverage of these sets is typical for a project involving only one kind of reads. We used a subset of these reads so that the coverage would be closer to a project that uses a hybrid approach to genome assembly. The coverage of our subset of the 454 reads is 6 and the coverage of the SOLiD reads varies from 30 to 50. Unfortunately, we could not find base space and color space reads from the same strain of E.coli but nevertheless our experiments on these read sets show that there are benefits to be gained from a mixed set of real reads. Table 1 shows a summary of the read sets. To test the hybrid correction of base space and color space reads, we used a combination of the base space and color space read sets.

All the experiments were run on an Intel Xeon computer with eight cores operating at 3.66 GHz and 32 GB of memory. The algorithms are implemented in Java and compiled and run with JVM 1.6. We tried the algorithm with several values of the strictness parameter $\alpha$ producing different thresholds at the analyzed level of the trie and show the results for the best observed value. For the simulated read sets, the best value was determined by minimizing the number of false positives (FPs) and false negatives (FNs) when detecting erroneous reads, and for the real reads, we chose the value that allowed aligning the corrected reads best to the reference genome. Table 2 shows the runtime and memory usage of the algorithm, the best strictness value and the corresponding threshold for detecting low weight nodes at the analyzed level of the trie for the various read sets. The memory usage is measured as the peak resident set size of the program. The implementation of the algorithm uses threads. Each thread builds and analyzes a subtree of the suffix trie where all suffixes start with a prefix specific to that thread. The memory usage of the algorithm can be decreased by reducing the number of threads or increasing the length of the prefix. However, the runtime will then increase. The strictness value must be chosen so that the threshold for detecting low weight nodes at analyzed level is greater than one. The best strictness value varied from 4 to 7 except for the lowest coverage read sets for which we were forced to choose a smaller value to keep the threshold at analyzed level greater than one.

To evaluate the performance of the algorithm, we measured its ability to detect erroneous reads and the error rate of the reads before and after correction. The detection of erroneous reads is a binary classification test. Table 3 shows the definitions for true positive (TP), true negative (TN), false negative (FN) and false positive (FP). The sensitivity is defined as TP/(TP+FN) and specificity as TN/(TN+FP).

<table>
<thead>
<tr>
<th>Table 1. Summary of read sets used in the experiments, last two read sets are real reads and the rest of the sets are simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
</tr>
<tr>
<td>B6x1.5</td>
</tr>
<tr>
<td>B6x3.0</td>
</tr>
<tr>
<td>B12x1.5</td>
</tr>
<tr>
<td>B12x3.0</td>
</tr>
<tr>
<td>B24x1.5</td>
</tr>
<tr>
<td>B24x3.0</td>
</tr>
<tr>
<td>C15x1.5</td>
</tr>
<tr>
<td>C15x3.0</td>
</tr>
<tr>
<td>C30x1.5</td>
</tr>
<tr>
<td>C30x3.0</td>
</tr>
<tr>
<td>RB0x</td>
</tr>
<tr>
<td>RC30x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. Runtime and memory usage of correction, the best value for the strictness parameter $\alpha$ and the corresponding threshold for detecting low weight nodes at the analyzed level of the trie for various read sets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDs</strong></td>
</tr>
<tr>
<td>B6x1.5</td>
</tr>
<tr>
<td>B6x3.0</td>
</tr>
<tr>
<td>B12x1.5</td>
</tr>
<tr>
<td>B12x3.0</td>
</tr>
<tr>
<td>B24x1.5</td>
</tr>
<tr>
<td>B24x3.0</td>
</tr>
<tr>
<td>C15x1.5</td>
</tr>
<tr>
<td>C15x3.0</td>
</tr>
<tr>
<td>C30x1.5</td>
</tr>
<tr>
<td>C30x3.0</td>
</tr>
<tr>
<td>B6x1.5 + C15x1.5</td>
</tr>
<tr>
<td>B6x3.0 + C15x3.0</td>
</tr>
<tr>
<td>B12x1.5 + C30x1.5</td>
</tr>
<tr>
<td>B12x3.0 + C30x3.0</td>
</tr>
<tr>
<td>RB0x</td>
</tr>
<tr>
<td>RC30x</td>
</tr>
<tr>
<td>RB6x + RC30x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Classifications of identified and erroneous reads</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error type</strong></td>
</tr>
<tr>
<td>Identified as erroneous</td>
</tr>
</tbody>
</table>

Complicating this case, the other version corrects read sets that may include both color space and base space reads but can naturally be limited to only color space reads.
Table 4. The classification test for simulated read sets

<table>
<thead>
<tr>
<th>ID</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>TN</th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6x1.5</td>
<td>213075</td>
<td>3755</td>
<td>2682</td>
<td>60272</td>
<td>0.988</td>
<td>0.941</td>
</tr>
<tr>
<td>B6x3.0</td>
<td>262944</td>
<td>1584</td>
<td>1748</td>
<td>60386</td>
<td>0.993</td>
<td>0.985</td>
</tr>
<tr>
<td>B12x1.5</td>
<td>425389</td>
<td>861</td>
<td>2333</td>
<td>126868</td>
<td>0.995</td>
<td>0.993</td>
</tr>
<tr>
<td>B12x3.0</td>
<td>527718</td>
<td>483</td>
<td>1426</td>
<td>30010</td>
<td>0.997</td>
<td>0.984</td>
</tr>
<tr>
<td>B24x1.5</td>
<td>859386</td>
<td>59</td>
<td>3823</td>
<td>255939</td>
<td>0.996</td>
<td>1.000</td>
</tr>
<tr>
<td>B24x3.0</td>
<td>1056625</td>
<td>132</td>
<td>4113</td>
<td>60386</td>
<td>0.998</td>
<td>0.998</td>
</tr>
<tr>
<td>C15x1.5</td>
<td>1141544</td>
<td>6012</td>
<td>4360</td>
<td>806307</td>
<td>0.995</td>
<td>0.993</td>
</tr>
<tr>
<td>C15x3.0</td>
<td>1497762</td>
<td>8251</td>
<td>162186</td>
<td>833120</td>
<td>0.995</td>
<td>1.000</td>
</tr>
<tr>
<td>C30x1.5</td>
<td>2285417</td>
<td>347</td>
<td>8826</td>
<td>833120</td>
<td>0.996</td>
<td>1.000</td>
</tr>
<tr>
<td>C30x3.0</td>
<td>2285513</td>
<td>47</td>
<td>8730</td>
<td>833120</td>
<td>0.996</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table 5. The error rates (%) of the simulated read sets before and after correction

<table>
<thead>
<tr>
<th>ID</th>
<th>Original reads</th>
<th>Corrected</th>
<th>EULER-SR</th>
<th>SAET</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6x1.5</td>
<td>1.489</td>
<td>0.299</td>
<td>0.595</td>
<td>–</td>
</tr>
<tr>
<td>B6x3.0</td>
<td>2.960</td>
<td>0.882</td>
<td>1.984</td>
<td>–</td>
</tr>
<tr>
<td>B12x1.5</td>
<td>1.491</td>
<td>0.218</td>
<td>0.307</td>
<td>–</td>
</tr>
<tr>
<td>B12x3.0</td>
<td>2.960</td>
<td>0.661</td>
<td>1.094</td>
<td>–</td>
</tr>
<tr>
<td>B24x1.5</td>
<td>1.491</td>
<td>0.215</td>
<td>0.350</td>
<td>–</td>
</tr>
<tr>
<td>C15x1.5</td>
<td>1.433</td>
<td>0.196</td>
<td>–</td>
<td>0.220</td>
</tr>
<tr>
<td>C15x3.0</td>
<td>2.864</td>
<td>0.707</td>
<td>0.663</td>
<td>–</td>
</tr>
<tr>
<td>C30x1.5</td>
<td>1.435</td>
<td>0.193</td>
<td>–</td>
<td>0.188</td>
</tr>
<tr>
<td>C30x3.0</td>
<td>2.870</td>
<td>0.691</td>
<td>–</td>
<td>0.527</td>
</tr>
</tbody>
</table>

as TN/(TN+FP). Table 4 shows the results of the classification test for the simulated read sets. For all datasets the sensitivity is >98% meaning that only a few of the erroneous reads remain undetected. The specificity of the method is also >98% except for the low-coverage base space read sets B6x1.5 and B6x3.0.

To measure the accuracy of correction, we computed the error rate of the read sets before and after correction by comparing them to the original reads before errors were introduced. We compared the accuracy of our correction method with EULER-SR (Chaisson et al., 2004) in case of base space reads and to SAET by Applied Biosystems in case of color space reads. Table 5 shows that the error rate is significantly reduced by the correction. Performing hybrid correction with a mixed set of reads instead of correcting each set separately further increases the correctness of the reads for low-coverage read sets. Our method reduced the error rate much more than EULER-SR but SAET achieves in most cases better error rates than our method for color space reads.

For the real read sets, the accuracy of correction was measured by aligning the reads to the reference genome. We used BLAST (Altschul et al., 1990) to align the base space reads and SOAP2 (Li et al., 2009) to align the color space reads. Figures 7 and 8 show the proportion of reads that can be aligned to the reference genome as a function of allowed error rate. For example, 86% of the original color space reads can be aligned to the reference genome with at most one mismatch, whereas 91% of the corrected color space reads can be aligned with the same mismatch tolerance. Our new method performs better than EULER-SR but SAET achieves slightly better correction in color space than our method. The good performance of SAET might be due to its ability to correct more than one error in the reads, which is not possible for our method.
Table 6. The impact of read correction on de novo assembly

<table>
<thead>
<tr>
<th>ID</th>
<th>Number of contigs (≥100 bp)</th>
<th>N50</th>
<th>Mean contig length (bp)</th>
<th>Maximum contig length (bp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB6x</td>
<td>2587</td>
<td>2933</td>
<td>1728</td>
<td>13 555</td>
</tr>
<tr>
<td>RB6x corrected</td>
<td>1731</td>
<td>4318</td>
<td>2568</td>
<td>18 692</td>
</tr>
<tr>
<td>RC30x</td>
<td>10 868</td>
<td>438</td>
<td>348</td>
<td>2308</td>
</tr>
<tr>
<td>RC30x corrected</td>
<td>7027</td>
<td>590</td>
<td>455</td>
<td>3081</td>
</tr>
</tbody>
</table>

an l-tuple, while the bases surrounding an error must align exactly with other reads for our method to correct the error.

To demonstrate the impact of error correction on de novo assembly we ran the Velvet (Zerbino and Birney, 2008) assembler on the real read sets before and after correction. Table 6 shows that assembly is greatly improved by the error correction.

6 CONCLUSION

We have presented a tool for the hybrid correction of a mixed set of reads produced by several sequencing platforms including the SOLiD sequencing technology that produces color space reads. We showed that our method can detect errors with high sensitivity and specificity and also the error rate of the reads is reduced. We also showed that low-coverage read sets clearly benefit from hybrid correction with other read sets.

ACKNOWLEDGEMENTS

We wish to thank Andreas Bremges for implementing part of the algorithms and Simon J. Puglisi for helpful comments regarding a draft of this manuscript. We also wish to thank Rainer Lehtonen, Virpi Ahola, Ilkka Hanski, Panu Somervuo, Laru Pulin, Petri Auvinen, Esko Ukkonen, Veli Mäkinen and Niko Valimäki for insightful discussions about error correction.

**REFERENCES**


