BIOINFORMATICS ORIGINAL PAPER Vol. 27 no. 11 2011, pages 1455–1461
doi:10.1093/bioinformatics/btr170

Genome analysis Advance Access publication April 5, 2011

Correcting errors in short reads by multiple alignments

Leena Salmela[1,*] and Jan Schröder[2]

[1]Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, Helsinki, Finland and [2]NICTA Victorian Research Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, Melbourne, Australia

Associate Editor: Alex Bateman

ABSTRACT

Motivation: Current sequencing technologies produce a large number of erroneous reads. The sequencing errors present a major challenge in utilizing the data in *de novo* sequencing projects as assemblers have difficulties in dealing with errors.

Results: We present Coral which corrects sequencing errors by forming multiple alignments. Unlike previous tools for error correction, Coral can utilize also bases distant from the error in the correction process because the whole read is present in the alignment. Coral is easily adjustable to reads produced by different sequencing technologies like Illumina Genome Analyzer and Roche/454 Life Sciences sequencing platforms because the sequencing error model can be defined by the user. We show that our method is able to reduce the error rate of reads more than previous methods.

Availability: The source code of Coral is freely available at http://www.cs.helsinki.fi/u/lmsalmel/coral/.

Contact: leena.salmela@cs.helsinki.fi

Received and revised on March 21, 2011; accepted on March 29, 2011

1 INTRODUCTION

Next-generation sequencing technologies, such as Illumina Genome Analyzer, Applied Biosystems SOLiD and Roche/454 Life Sciences DNA sequencing platforms, produce a vast amount of reads in a single run. Vagaries of the reads produced by each sequencing machine are still being discovered.

Correction of errors in short reads is a critical task in bioinformatics. The variety of errors and biases of the current sequencing platforms must be addressed if the data is to be used to maximum effect. Error correction aims to revert these mistakes made by a sequencing platform by exploiting the redundancy of the data and judging each base in a read as correct or incorrect (and then ideally correcting it). Error correction helps to achieve high data use in *de novo* assembly. Furthermore, the computational demands of assembly algorithms are reduced significantly if reads are first corrected. Although we evaluate here the impact of error correction on *de novo* assembly, the technique could also be useful in other applications because error correction can potentially increase the mappability of reads in resequencing as well as in other applications of high-throughput sequencing.

*To whom correspondence should be addressed.

Error characteristics of the different platforms are complex, making error correction a difficult task. For instance, the Roche/454 sequencing platform produces reads with indel errors, due mainly to homopolymers, whereas the SOLiD and Illumina platforms are prone to substitution errors. Dohm *et al.* (2008), Chaisson *et al.* (2009), and Schröder *et al.* (2010) investigate quality score evolvement, error characteristics and biases of short sequencing reads.

The first error correction method that is aimed at short read datasets is built into the assembly tool Euler SR (Chaisson and Pevzner, 2007; Chaisson *et al.*, 2004). It uses the *spectral alignment* method, which first establishes a spectrum of trusted k-mers from the input data and then corrects each read so that it contains only sequences from the spectrum. Shrec by Schröder *et al.* (2009) followed, a stand-alone error correction method. Shrec is based on a parallelized suffix-trie data structure that holds a set of reads and corrects errors with a majority voting scheme. Shrec was extended by Salmela (2010) to accommodate hybrid sets of reads from various sequencing technologies, with different read lengths and error characteristics. HiTEC by Ilie *et al.* (2010) uses a suffix array of the reads to count how many times short sequences are present in the read set and use these counts to correct the reads. The approach by Yang *et al.* (2010) revisits the idea of Chaisson and Pevzner by overlaying reads with trustworthy tiles (pairs of k-mers), and correcting differences between the reads and the tiles to obtain error corrected read sets. Quake by Kelley *et al.* (2010) is another recent method that relies on spectral alignments to correct reads.

In this article, we present Coral (=CORrection with ALignments), a novel approach, which relies on multiple alignments of short reads to correct errors in the data. The idea of using multiple alignments for correcting sequencing reads is not new. For example, the preprocessing in the Arachne assembler (Jaffe *et al.*, 2003) and the MisEd error correction tool (Tammi *et al.*, 2003) use multiple alignments to correct reads from the older Sanger technology (Sanger *et al.*, 1977). Our new tool is the first to apply this approach to short read data.

Most of the recent error correction tools are aimed at reads from the Illumina Genome Analyzer platform and therefore they are limited to correcting substitutions which is the dominant error type in Illumina reads. Coral is easily adjustable to different error models of the various sequencing platforms. Furthermore, adjusting the error model is easily accessible to the end user who only needs to set the familiar parameters of multiple alignments, gap penalty and mismatch penalty, to appropriate values according to the error model. For example, for Illumina reads one needs to set the gap penalty to a high value effectively disallowing indels, whereas one

can set equal values to gap penalty and mismatch penalty for reads from the Roche/454 Life Sciences sequencing platform where indels are a common error type. Allowing the user to adjust the error model makes Coral also ready for data from the future single molecule sequencing technologies (Gupta, 2008) where indels may again be a common error type.

This article is organized as follows. We define the error correction problem in Section 2 and introduce the algorithms behind Coral in Section 3. In Section 4, we compare our new method with the established tools through experimental validation and then continue to investigate the impact of error correction on short read assembly. We conclude in Section 5 with discussion and suggestions for further work.

2 ERROR CORRECTION PROBLEM

Reads are random samples of nucleotide sequences from a target genome of length N. Thus, a read is a sequence of characters from the DNA alphabet {A, C, G, T, N}. There are r reads whose length varies from $m_{\min}$ to $m_{\max}$. We denote the length of read i by m_i. The combined total length of the reads is $M = \sum_{i=1}^{r} m_i$. A read may contain three types of errors: substitutions, insertions and deletions. We denote by e the maximum estimated error rate of a read. Thus, a read of length m_i may contain at most $e \cdot m_i$ errors.

Coverage is the expected number of times a position in the genome is sequenced. Therefore, coverage equals M/N. All error correction methods rely on the coverage of the reads being moderately high so that every position of the genome is sequenced several times with high probability. Reads from low coverage regions cannot be corrected because there is insufficient data to infer the correct sequence.

The task of an error correction algorithm is to detect and correct the errors in the reads. If we knew for each read its position in the target genome, we could form a multiple alignment of all the reads and correct them towards the consensus sequence of the alignment. All error correction methods use some heuristics to determine which reads align to the same genomic position and comparing this read set correct the reads towards the consensus of the set.

3 ERROR CORRECTION BY ALIGNING MULTIPLE READS

Our approach takes reads that share common k-mers and forms multiple alignments of these reads. The correction of the reads is then based on these alignments and their consensus sequences. Each step of our algorithm is discussed in detail below.

3.1 Indexing the reads

We start by indexing all k-mers that occur in the reads or their reverse complements. We create a hash table which associates a k-mer with a list of reads where it occurs. In this basic scheme, each k-mer is actually indexed twice: first in the forward direction and then in the reverse. The read lists for both are the same. To save space, we only store a k-mer and the list of reads where it occurs if the k-mer is lexicographically smaller than its reverse complement. We further note that if k is odd, a k-mer and its reverse complement cannot be equal and thus either one is lexicographically smaller. If k is even, a k-mer and its reverse complement can be equal. In this case, we do not store either of the k-mers because the orientation of the read cannot be deduced from that k-mer. We also leave out any k-mers containing indeterminate characters, i.e. N's.

To construct the k-mer index, we extract each k-mer from each read and add it to the corresponding list. This can be achieved in $O(m_i)$ time per read, where m_i is the length of the read, and so the whole construction takes $O(M)$ time.

3.2 Forming multiple alignments

The next step is to form multiple alignments. Each multiple alignment is based on one read which we call the *base read* of the alignment. We retrieve from the k-mer index all reads that share at least one k-mer with the base read. This read set together with the base read is called the *k-mer neighbourhood* of the base read.

The multiple alignment is then computed for the k-mer neighbourhood of the base read. The first read is set as the initial consensus of the alignment. The reads are then aligned against the consensus one by one using a variant of the Needleman–Wunsch algorithm (Needleman and Wunsch, 1970). We allow free gaps in the beginning and end of the alignment for both the new read and the consensus, allowing us to find prefix-suffix overlaps between the read and the consensus. Otherwise the parameters for alignment can be set by the user. After the alignment of each read, the consensus is updated according to the alignment as follows. If there is an insertion in the read, we add a new column to the multiple alignment at that position and update those reads that are already aligned to include the gap. Then we compute for each column in the multiple alignment, the most prevalent base or gap which then becomes the consensus at that position.

This procedure works reasonably well when the reads are very similar. If the reads are more divergent, the returned multiple alignment might not be very good. However, if the reads are not from the same genomic location, i.e. the reads are very dissimilar, we do not want to use the alignment for correction anyway, and thus for our purposes it is enough that the multiple alignment algorithm returns good alignments if the reads are very similar.

We use several techniques to speed up the computation of alignments. If the k-mer neighbourhood of the base read is very large, we discard the alignment even before computing it as it is likely that the set contains reads from several genomic positions. We also discard the alignment before computing it if the k-mer neighbourhood is very small so that the depth of the alignment could not be sufficiently high for error correction.

Most of the reads are in the k-mer neighbourhood of several reads and thus they are aligned and corrected several times. If a read has already been aligned and corrected, it is often error free. Thus, many reads align error free against the consensus. Therefore, we use the indexed k-mer that caused the read to be included in this alignment to locate the most probable position for alignment. Then we try to align the read without errors against the consensus at that position. If this succeeds, we can skip the full alignment computation which is time consuming.

If at least one of the k-mers that a read shares with the base read occurs only once in the read, we use that to locate the most probable position for alignment. We then speed up the computation by using a banded Needleman–Wunsch algorithm where the width of the band equals $2 \cdot e \cdot m_i + 1$ where m_i is the length of the read.

If we find that a read only aligns against the consensus with many errors indicating that it is not from the same genomic position as the previous reads, we immediately discard this multiple alignment and move on to process the next one.

If gaps are not allowed, we use a linear method to establish a gapless alignment between the read and the consensus sequence. We locate the most probable position for alignment based on the k-mers that the read shares with the base read as outlined above and then form a gapless alignment at this position.

It takes $O(m_i n)$ time in the worst case to align a read of length m_i against a consensus sequence of length n. The length of the consensus sequence should not exceed twice the maximum length of a read. In the worst case, the k-mer neighbourhood of each read contains L reads where L is the limit on the size of the k-mer neighbourhoods. We use each read as a base read, and thus computing the multiple alignments takes $O(M m_{\max} L)$ in the worst

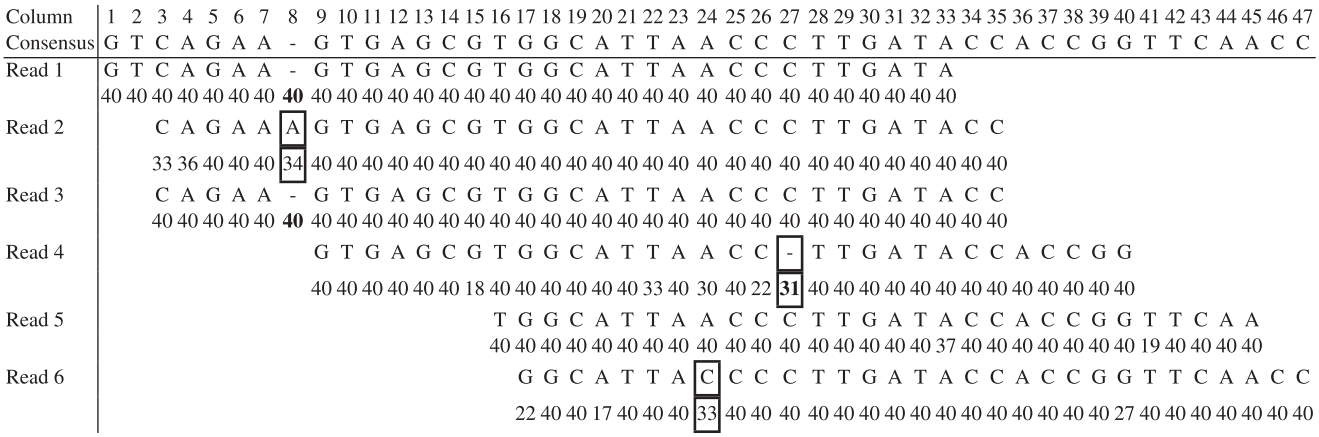

Fig. 1. An example multiple alignment of six reads with quality scores and three sequencing errors shown in boxes. The quality of the multiple alignment is 0.968. (i) In column 8, there is an insertion in read 2. The quality scores of reads 1 and 3 shown in bold face are computed as averages of the quality scores of the bases flanking the gap. The support of the consensus at that column is 0.67 and the weighted support is 0.70. (ii) In column 24, there is a mismatch in read 6. Here the support of the consensus is 0.83 and the weighted support is 0.86. If the support and the weighted support meet the corresponding thresholds, the quality score of the substituted base will be 38. (iii) In column 27, there is a deletion in read 4. The quality score of read 4 shown in bold face is calculated as an average of the bases flanking the gap. The support of the consensus is 0.83 and the weighted support is 0.87. If the support and the weighted support meet the corresponding thresholds, the quality score of the inserted base will be 40.

case. However, in practise the runtime is better as the k-mer neighbourhoods are not that large on average and for many reads we can quickly compute error-free alignments because the read aligns perfectly against the consensus.

If gaps are not allowed, the optimization technique described above allows us to compute the alignment of a read of length m_i against the consensus in $O(m_i)$ time. In this case, computing the multiple alignments takes $O(ML)$ in the worst case.

After computing the multiple alignment, we check if there are any reads that do not share at least k positions with the base read. This basically means that the read was so dissimilar to the base read that not even their common k-mers are aligned to the same position. If such reads are found, we discard them and recompute the multiple alignment for the other remaining reads. If there are still reads that do not share at least k positions with the base read, the whole multiple alignment is discarded.

3.3 Correction of reads

To measure the quality of the multiple alignment, we compute for each aligned read the fraction of aligned positions where the read agrees with the consensus sequence of the alignment. The positions where both the consensus and the read have a gap are ignored when computing the fraction. The *quality of the multiple alignment* is then defined as the minimum of the above values. If the quality is higher than $1-e$, the alignment will be used to correct the aligned reads. Figure 1 shows an example of a multiple alignment. Here the fractions of aligned positions where the reads agree with the consensus sequence are 1.000, 0.970, 1.000, 0.969, 1.000 and 0.968. The lowest of these is 0.968 which is also the quality of the alignment.

For each position of the consensus, we define the *support* of the consensus as the number of reads that agree with the consensus at that position divided by the total number of reads aligned at that position. For each aligned read, we then compare the read and the consensus sequence in each aligned position. If they do not agree, we correct the read to agree with the consensus if the support of the consensus is at least t_s where the *support threshold* t_s ($0.5 \leq t_s < 1.0$) is a parameter of the method. See Figure 1 for examples of computing the support of the consensus.

If quality scores are available for the base calls, we make an additional check before allowing correction. If there is a gap in the aligned read, we set the quality score for the gap to the average of the quality scores of the bases flanking the gap. We then compute the *weighted support* of a position of the consensus sequence by dividing the sum of the quality scores of the bases agreeing with the consensus with the sum of the quality scores of all bases aligned at that position. We then correct the base only if this exceeds the *quality threshold* t_Q ($0.5 \leq t_Q < 1.0$), which is another parameter of the method. The quality score of the corrected base is set to the average quality of the bases agreeing with the consensus at that position. See Figure 1 for examples of computing the weighted support of the consensus.

The combined length of all reads in a multiple alignment cannot exceed $Lm_{\max}$ and at most $eLm_{\max}$ positions are considered for correction. The quality threshold t_Q can be computed in $O(L)$ time for those positions and thus the worst case complexity is $O(Lm_{\max})$ to detect potential errors and $O(eL^2 m_{\max})$ to correct the detected errors.

3.4 Complexity

The worst case runtime is dominated by the computation of the multiple alignments and is thus $O(Mm_{\max}L)$ if gaps are allowed and $O(ML)$ if only mismatches are allowed.

The reads contain M bases in total and thus the total number of k-mers is also $O(M)$. The k-mer index consist of a hash table that hashes k-mers to their occurrence lists and the occurrence lists for each different k-mer. There cannot be more than $O(M)$ different k-mers and thus the hash table takes at most $O(M)$ space. Each position in the reads creates at most one entry in the occurrence lists and thus the space complexity of the occurrence lists is $O(M)$. Therefore, the total space complexity of the k-mer index is $O(M)$. The space complexity of computing the multiple alignments is $O(Lm_{\max} + m_{\max}^2)$. If the number of reads exceeds the maximum read length, the space of the k-mer index clearly dominates. Thus, the overall space complexity is $O(M)$.

3.5 Choosing parameters

If all reads in the k-mer neighbourhood of a read originate from close by positions in the genome, their multiple alignment will be suitable for error correction. Thus, we should choose the length of a k-mer so that with high probability each k-mer occurs only once in the genome. Therefore, k should be chosen so that $k \geq \lceil \log_4 N \rceil$ where N is the length of the genome. If k is larger than half the length of a read, then the middle base of the read is contained in every k-mer of the read. Furthermore, if the middle base is

Table 1. Datasets for evaluation of error correction performance

Dataset key	*D*1	*D*2	*D*3	*D*4
Reference organism				
Name	*Escherichia coli*		*Staphylococcus aureus*	
Reference sequence	NC_000913		NC_003923	
Genome size (Mb)	4.6		2.8	
Dataset				
Accession number (SRA)	SRR000868	SRR022918	NA	SRR022866
Read length (bp)	56 – 625	47	35	76
Number of reads	0.23M	7.1M	3.8M	8.9M
Number of bases (Mb)	59	338	138	673
Sequencing platform (library name)	Roche 454 (2121724656)	Illumina GAII (Solexa-4733)	Illumina GA I	Illumina GAII (Solexa-8293)
Mapping specifics				
No. of reads mapped to ref.	0.22M	3.4M	3.3M	4.2M
Percentage	98.8	48.4	86.9	47.7
Genome coverage	12.2×	35.0×	40.6×	113.7×

The dataset *D*2 is not archived in NCBI's Sequence Read Archive, but publicly available on http://www.genomic.ch/edena.php.

erroneous, the read does not contain any correct k-mers and most likely it will not participate in any correct multiple alignments. Therefore, k should also be smaller than half the length of a read.

The parameter e should be chosen to reflect the maximum expected error rate in the reads. However, setting e to a large value risks over correcting the reads because then even poor multiple alignments will be used for correction.

The support threshold t_s is related to the coverage of the read set. If we estimate that the lowest coverage in any part of the genome is c then t_s should be set to $(c-1)/c$. Again a too low value risks over correction because corrections will then be made even if the support for the consensus is weak, while a too high value leaves reads from areas with low coverage uncorrected. The quality threshold should be set similarly.

Finally, one can set the parameters governing the computations of the alignments which are match reward, mismatch penalty and gap penalty. These should be set according to the the error model of the sequencing technology used. For example, Illumina reads gap penalty should be set high to disallow gaps in the alignments.

4 RESULTS

The aims of the experiments were 2-fold. First, we measured the quality of correction, and its effect on subsequent assembly. Secondly, the computational resources required for the new method were compared to alternative approaches (SHREC, Quake and Reptile).[1]

Experimental setup: for testing we used the datasets listed in Table 1. Unlike in other publications on the topic, we consider only real sequencing data and no simulated reads. We believe that none of the available read simulators can grasp the true characteristics of next-generation sequencing technologies, and thus observations on simulated data may be invalid. All tests were conducted on an otherwise idle AMD Opteron machine with 4 2.6 GHz CPUs, 32 GB main memory and 1024K L2 Cache. The operating system was Ubuntu Linux version 8.04.4 LTS. The compiler was g++ (gcc version 4.2.4) executed with the -O3 option. Times given are the average of two runs and were recorded with the Linux/Unix time command.

To test the performance of the short read error correction tools under investigation, we first have to distinguish correct from erroneous bases in the experimental data. We accomplish this by mapping the reads to their respective references and defining mismatches and indels as errors in the sequence reads. This is common practise, as can be seen in the publications by (Chaisson and Pevzner, 2007; Schröder *et al.*, 2009; Yang *et al.*, 2010). However, this method bears risks, since genomic variants such as SNPs can be defined as errors, and ambiguously mapping reads can lead to false classifications of bases as well. This is a general problem with short read data for applications like error correction, assembly or mapping, since the actual genomic sequence is hardly ever known, as that is the target of discovery. To minimize the risk of false classification, we only consider uniquely mapping reads for our experiments. Also, this disadvantage is the same for all of the tested methods. We used SOAP by Li *et al.* (2008) for mapping the Illumina reads and SHRiMP by Rumble *et al.* (2009) to map the Roche/454 reads.

We can then assess the performance of an error correction method by identifying how many of the alleged errors in the data it can correct [*true positives* (TPs)], how many errors get introduced falsely [*false positives* (FPs)], how many errors remain undetected [*false negatives* (FNs)] and how many correct bases remain unchanged [*true negatives* (TNs)]. From these numbers, other statistics like specificity, sensitivity or gain can be inferred, which will be explained below.

Such statistics, however, are not entirely satisfactory to assess the quality of corrections of a method, because it is unclear which statistic (when optimized) will indeed yield the best performance for other applications depending on the data. Since error correction of short read data really is a preprocessing step for other applications like assembly, we assess performance on the impact on this key

[1]We also attempted comparison to a very recent method HiTEC by (Ilie *et al.*, 2010), but could not complete the experiments in time, after correspondence with the authors lead to fixing a bug after submission.

Table 2. Performance evaluation of error correction methods on different datasets

Dataset	Method	FP	TP	FN	TN	Sensitivity	Gain	Runtime (min)	Memory (GB)
*D*1	Shrec	6.7k	138k	36.9k	105M	0.789	0.751	31.6/122.8	5.2
	Coral	12.4k	169k	13.7k	111M	0.925	0.857	5.3/18.6	1.8
*D*2	Shrec	3k	1.3M	34k	160M	0.974	0.972	16.1/57.75	9.2
	Quake	6.6k	808k	526k	157M	0.606	0.601	7.75/13.2	0.4
	Reptile	4.1k	0.7M	618k	160M	0.537	0.534	10.5	1.0
	Coral	2.6k	1.3M	32k	160M	0.976	0.974	8.12/16.0	2.6
*D*3	Shrec	3.2k	867k	103k	116M	0.894	0.890	9.5/32.1	9.2
	Coral	2.9k	911k	58k	116M	0.940	0.937	4.25/13.22	1.7
*D*4	Shrec	2.7k	1.2M	22.5k	319M	0.982	0.979	40.1/132.4	9.9
	Quake	3.4k	0.9M	293k	313M	0.758	0.755	11.75/14.6	0.7
	Reptile	34.5k	0.37M	843k	319M	0.306	0.277	28.3	1.6
	Coral	4.0k	1.2M	27.1k	319M	0.978	0.974	80.0/305.1	4.0

The runtime is given as total runtime and CPU time (where applicable).

application for short read data. For this purpose, we choose Edena by (Hernandez *et al.*, 2008), a well-established short read assembler, to run on the corrected read sets.

Parameter configuration: the following parameters have been selected for the various error correction methods during the experiments:

- Shrec: standard parameters.
- Quake: '-k 15' as suggested in the manual.
- Reptile: standard parameters.
- Coral: standard parameters (with the appropriate configurations with regards to the sequencing platforms): -illumina/-454.

Note, that we did not explore the parameter space to obtain optimal results for any of the error correction methods. In reality, when working on a read set fresh off a sequencing platform, there is no immediate feedback for the user, to identify good or bad error correction, which makes parameter choices hard. For this reason, we ran the tools with a best guess kind of configuration, to make it more indicative of a real application. For Shrec, this meant adjusting the cutoff value c, for Coral standard parameters and for Reptile standard configuration as well, since it estimates all the important statistics by itself.

Statistical error correction performance: to assess the accuracy of the different methods by numbers, we identify the above four categories of error classification and then deduct the following statistics:

- *Sensitivity* = TP/(TP + FN), the sensitivity towards erroneous bases.
- *Gain* = (TP − FP)/(TP + FN), as introduced by Yang *et al.* (2010), a statistic to combine the two intuitions of removing errors without introducing additional ones.

The results for the above datasets can be found in Table 2. We could not obtain any results for Reptile and Quake on datasets *D*1 and *D*3. *D*1 is a set of pyro-sequencing reads generated by Roche/454 sequencer and Reptile and Quake are not designed to operate with them. *D*3 on the other hand does not feature quality score values, a prerequisite for the Reptile and Quake error correction tools. The results for Shrec on dataset *D*1 have been computed with the advanced version of Shrec introduced by Salmela (2010).

The results show comparable performance between Shrec and Coral. The crucial statistics like false positive counts, specificity and gain are in close vicinity to each other. Only on datasets *D*1 and *D*3, a clear difference between the tools can be seen. *D*3 in particular has a relatively high error rate, giving full read alignments a slight edge over a *k*-mer based method.

The other two methods fall noticeably short of the performance of Coral, due to significantly lower TP rates.

When comparing the obtained results with the experiments in the original papers for Shrec and Reptile (Quake does not assess the performance on publicly available data, but on simulated reads only), some discrepancies become obvious. For instance, the performance of Shrec on the very same dataset *D*3 is inferior to that in the original publication. Note, however, that we did not explore the parameter space for any of the methods as mentioned above, so the full potential can be missed by a bit. For Reptile on the other hand, we obtained *better* results on the same dataset (*D*2) that Yang *et al.* analysed in the original publication (they achieved a sensitivity of 0.527 and gain 0.38). Still, these results fall short of the performance of Coral.

Resource demands: for the comparison of resource demands of the different methods, we refer again to the results in Table 2. Coral uses significantly less memory than Shrec (between a factor 3 and 10), but more than Reptile and Quake (about a factor of 2.5 and 6 respectively). Runtime is not as easily distinguished, since Coral's worst case complexity correlates quadratically with the read length (due to the dynamic programming nature of the alignments). As a result, the runtime on the shorter reads of *D*2 for Coral is faster than or comparable to the other methods, whereas it is worst on the long reads of *D*4. Since 'long' reads (in the vicinity of about 100 bp) are the present and future of NGS platforms, this is an issue to work on. We will discuss this further in Section 5. Quake has to be named as the most resource effective methods from these experiments, delivering better results than Reptile and having the best runtime and memory usage.

Note that the experiments were conducted on a multi-core machine as stated above. Shrec, Quake and Coral can utilize this to their benefit, whereas Reptile is a single-threaded program. For this

reason, we have given CPU time as an additional statistic in Table 2. However, multi-core machines or even clusters are the reality of most labs operating in computational biology/bioinformatics, so we consider the wall-time comparison of the method as more appropriate.

Impact on assembly: *de novo* DNA sequence assembly is one of the key tasks applied to short reads. Assembly can benefit in two ways from error correction. First, error-free reads can be assembled more easily since they contain more error-free k-mers (or error-free overlaps, depending on the data structures used in the assembly tool). Secondly, less errors in a dataset mean less unique k-mers (or overlaps), which can reduce the memory consumption of an assembly tool significantly.

As discussed earlier, it is not entirely clear which statistics should be maximized in order to obtain good error correction. Maximizing gain, for instance, might be a sensible strategy; however, in our experience a critical factor to successful assembly is a low false positive rate while keeping the recall as high as possible. The reason for this is the fragile nature of low coverage areas in the sequencing data towards error correction. An overly ambitious tool can easily identify these areas as erroneous and 'correct' them towards something similar and more prevalent in the data, making them unavailable for the assembly tool. This tendency of over correction is to be kept in mind.

To assess the benefits of error correction on assembly, we chose the Edena assembler by Hernandez *et al.* (2008) for our experiments. De bruijn graph-based assemblers like Velvet (Zerbino and Birney, 2008) or SOAPdenovo (Li *et al.*, 2010) are more popular, but cannot show the impact of error correction as clearly, because they incorporate their own correction techniques when processing the graph. Edena is relying on error-free overlaps in the initial read set and is thus more suited for our means of evaluation. When running Edena, we varied the minimum overlap length parameter. Otherwise standard parameters were used.

Note, that Edena requires reads to be of the same length. Quake trims reads, if it considers them erroneous, but cannot correct them. Consequently, we had to discard trimmed reads from the dataset. This left 3.9M reads, or 93%. This will slightly affect the assembly results as well as the resource demands of Edena in the following.

To measure assembly performance, we mainly focus on the N50 statistic, which gives a good indication on how successful the assembly was. Figure 2 shows the N50 statistic of Edena's results when assembling datasets, which have been manipulated by different error correction methods (on the example of dataset $D4$). The overlap parameter shown on the x-axis is used in Edena for the graph construction: only overlaps of reads longer than this minimum are considered to create edges.

The graphs show the enhanced performance of Edena when working with error corrected read sets: the data points for the Coral-corrected reads exceed all the other assemblies for the investigated overlap values, which is evidence for increased robustness of the assembly. The N50 value is about 22% higher than if Edena runs on uncorrected data. Shrec and Quake also achieve a positive effect on the assembly. The reads corrected by Reptile cannot be combined into as long contigs anymore and the performance drops by about 5%.

Furthermore, we are interested in the improvement of resource demands when performing assembly on error corrected reads, since these (especially memory constraints) can be limiting factors in the assembly process. Errors in reads do not only make assembly a harder problem, but they also create nodes and edges in the respective graph structures, which would not have occurred in error-free sequencing material. These additional nodes and edges are reflected in higher processing time and memory usage of the assembler. Removing errors can improve these statistics, as can be seen in Table 3.

Again, the read set, which has been generated with Coral, achieves the best results when it comes to resource savings. This was expected given the high recall rate observed earlier.

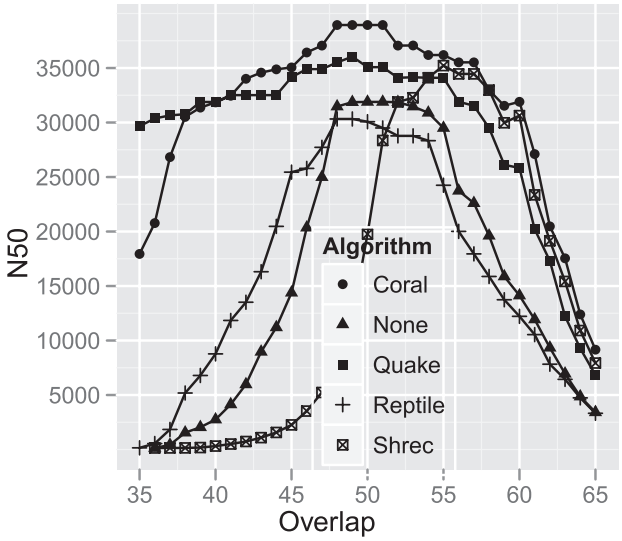

Fig. 2. Assembly performance of the Edena assembler on dataset $D4$ and error corrected versions of it.

Table 3. Resource demands of the Edena assembler when processing read set $D4$ or error corrected versions of it

Method	Stage	Runtime (%)	Memory, GB(%)
None	1	32.2 min	1.46
	2	33.6 s	0.73
Quake	1	28.4 min (−2.5)	1.46 (−0)
	2	43.6 s (+30)	0.63 (−14)
Reptile	1	31.7 min (−1.6)	1.36 (−7.1)
	2	28.7 s (−14.5)	0.65 (−11.2)
Shrec	1	28.9 min (−10.2)	1.31 (−10.5)
	2	25.8 s (−23.3)	0.58 (−20.5)
Coral	1	28.1 min (−12.7)	1.24 (−15.1)
	2	24.0 s (−28.6)	0.53 (−27.4)

The two stages in column 2 refer to the overlap graph creation step (1) and the processing of the graph (2) within Edena. The values in parentheses in the runtime and memory columns show the improvement over the run without any error correction. The runtime and memory usage have been averaged over two runs of the software.

5 CONCLUSION AND FUTURE WORK

In this article, we have introduced a novel error correction method. It relies on multiple alignments of reads as well as their quality scores to distinguish correct from erroneous bases.

Whereas most previous tools for error correction concentrate on one sequencing technology, Coral is applicable to a wide range of sequencing datasets. Our experiments covered data from different Illumina technologies and Roche 454 but Coral can be easily applied to datasets from other sequencing technologies including third-generation sequencing platforms by adjusting the parameters of alignments on the command line.

Coral's error correction performance is superior to that of the existing methods in all quality regards. This benefits the performance of short read data applications, as we have demonstrated by the example of *de novo* assembly. The very low false positive rate compared to Shrec prevents the issue of over correcting low coverage regions, while the high recall (compared with Reptile) allows for a steep increase of error-free k-mers or overlaps in the sequencing data, which leads to an increase in contig lengths and reduction in runtime and memory requirement of the Edena assembler.

Coral is able to outperform the competing methods because of the beneficial nature of multiple alignments. These give a more coherent picture of sequencing errors in the reads since whole reads are inspected for the decision-making process. In Shrec and Reptile, only k-mers (or tiles respectively) are investigated instead.

Future work: we are investigating the possibility to use approximate k-mers instead of exact ones for indexing the reads. This would be especially beneficial for very short reads because we could lift the requirement for k to be smaller than half the length of a read. However, as the read lengths are increasing, this is not a pressing concern. We are also planning to investigate the possibility to update the k-mer index when reads are corrected which would allow for better alignments towards the end of the correcting run. We also plan to develop methods based on sequence similarity to partition large k-neighbourhoods into smaller subgroups that would each originate from different part of the genome. The subgroups could then be corrected separately.

Furthermore, Coral is at the moment not compatible with the colour space reads from the SOLiD sequencing platform. In future, we plan to develop a version that addresses this issue.

Finally, the resource demands of Coral need to be improved due to the ever-growing throughput of modern sequencing technologies. Here, Coral has demonstrated good performance on bacterial genomes. To make Coral applicable to larger genomes, we plan to compress the k-mer index to save space and to implement guided choices of alignments and re-use of alignment information to speed up the correction process.

ACKNOWLEDGEMENTS

We thank Simon J. Puglisi (RMIT, Australia) for the initial contributions and fruitful discussions about the topic of alignment-based error correction.

Funding: This work was supported by the Australian Research Council, and by the NICTA Victorian Research Laboratory, and by Academy of Finland [grant number 118653 (ALGODAN)]. 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.

Conflicts of Interest: none declared.

REFERENCES

Chaisson,M.J. and Pevzner,P.A. (2007) Short read fragment assembly of bacterial genomes. *Genome Res.*, **18**, 324–330.

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

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.

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

Gupta,P.K. (2008) Single-molecule DNA sequencing technologies for future genomics research. *Trends Biotechnol.*, **26**, 602–611.

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.

Ilie,L. *et al.* (2010) HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, **27**, 295–302.

Jaffe,D.B. *et al.* (2003) Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Res.*, **13**, 91–96.

Kelley,D.R. *et al.* (2010) Quake: quality-aware detection and correction of sequencing errors. *Genome Biol.*, **11**, R116.

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

Li,R. *et al.* (2010) De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res.*, **20**, 265–272.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

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.

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

Schröder,J. *et al.* (2010) Reference-free validation of short read data. *PLoS One*, **5**, e12681.

Tammi,M. *et al.* (2003) Correcting errors in shotgun sequences. *Nucleic Acids Res.*, **31**, 4663–4672.

Yang,X. *et al.* (2010) Reptile: representative tiling for short read error correction. *Bioinformatics*, **26**, 2526–2533.

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