Bioinformatics, 31(21), 2015, 3421–3428
doi: 10.1093/bioinformatics/btv415
Advance Access Publication Date: 14 July 2015
Original Paper

Genome analysis

Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data

Amin Allam*, Panos Kalnis and Victor Solovyev

Computer, Electrical and Mathematical Sciences and Engineering Division (CEMSE), King Abdullah University of Science and Technology (KAUST), Thuwal 23955-6900, Saudi Arabia

*To whom correspondence should be addressed.

Associate Editor: Gunnar Ratsch

Received on November 11, 2014; revised on May 27, 2015; accepted on July 8, 2015

Abstract

Motivation: Next-generation sequencing generates large amounts of data affected by errors in the form of substitutions, insertions or deletions of bases. Error correction based on the high-coverage information, typically improves *de novo* assembly. Most existing tools can correct substitution errors only; some support insertions and deletions, but accuracy in many cases is low.

Results: We present *Karect*, a novel error correction technique based on multiple alignment. Our approach supports substitution, insertion and deletion errors. It can handle non-uniform coverage as well as moderately covered areas of the sequenced genome. Experiments with data from Illumina, 454 FLX and Ion Torrent sequencing machines demonstrate that Karect is more accurate than previous methods, both in terms of correcting individual-bases errors (up to 10% increase in accuracy gain) and post *de novo* assembly quality (up to 10% increase in NGA50). We also introduce an improved framework for evaluating the quality of error correction.

Availability and implementation: Karect is available at: http://aminallam.github.io/karect.

Contact: amin.allam@kaust.edu.sa

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Next-generation sequencing (NGS) technologies generate at decreasing costs large amounts of data for tasks such as *de novo* genome assembly, resequencing, single-nucleotide polymorphism discovery, DNA–protein interaction discovery and identification of chromosomal rearrangements (Ilie and Molnar, 2013; Ilie *et al.*, 2011). Reads produced by NGS technologies suffer from sequencing errors in the form of substitutions, insertions and deletions of bases, which complicate further processing. Table 1 summarizes the types of errors in recent NGS technologies.

Error correction methods utilize the high data coverage to correct the erroneous bases in reads. Existing methods are classified in five categories: (i) *k*-spectrum-based methods, such as Lighter (Song *et al.*, 2014), Blue (Greenfield *et al.*, 2014), Trowel (Lim *et al.*, 2014), HECTOR (Wirawan *et al.*, 2014), BLESS (Heo *et al.*, 2014), Musket (Liu *et al.*, 2013), Reptile (Yang *et al.*, 2010), Quake (Kelley *et al.*, 2010), Hammer (Medvedev *et al.*, 2011) and the works of Chaisson *et al.* (2004), Pevzner *et al.* (2001), Qu *et al.* (2009), Wijaya *et al.* (2009), Yang *et al.* (2011) and Le *et al.* (2013) that decompose reads into the set of all *k*-mers. Error correction is based on the *k*-mer frequencies. Variants of these methods are used as preprocessing stages in several assemblers, such as SOAPdenovo (Li *et al.*, 2010), ALLPATHS-LG (Gnerre *et al.*, 2011), SGA (Simpson and Durbin, 2012) and SPAdes (Bankevich *et al.*, 2012; Nikolenko *et al.*, 2013). (ii) Suffix array/tree-based methods, such as SHREC (Schroder *et al.*, 2009), HSHREC (Salmela, 2010) and HiTEC (Ilie *et al.*, 2011), are generalizations of *k*-spectrum methods that support multiple *k* values. They build a suffix array/tree of all read suffixes and correct errors using the *k*-mer frequency weights associated with the suffix tree nodes, whereas RACER (Ilie and Molnar, 2013)

Table 1. NGS error types

Brand	Read length	Throughput	Dominant error type
Illumina	Small	Very high	Substitutions
SOLiD	Small	High	Substitutions
454 FLX	Moderate	Moderate	Insertions, deletions
Ion Torrent	Moderate	High	Insertions, deletions
Pacific Biosciences	Very long	High	Insertions, chimeric Reads

Read lengths are as follows: small ≈ 36–200 bp, moderate ≈ 200–700 bp, very long ≈ 1000–10 000 bp. Throughput is the number of reads produced over unit of time.

is based on the same concepts, without explicitly using a suffix array. Fiona (Schulz *et al.*, 2014) utilizes partial suffix trees along with statistical methods. (iii) Multiple sequence alignment (MSA)-based methods consider each read *r* as reference and perform multiple alignment of reads that share at least one *k*-mer with *r*. Coral (Salmela and Schroder, 2011) creates a consensus sequence after each alignment with *r*. DAGCon (Chin *et al.*, 2013) uses a directed acyclic graph instead of a consensus sequence. ECHO (Kao *et al.*, 2011) computes consensus bases expectation maximization algorithm that by performing pairwise alignment among reads sharing at least one *k*-mer. MuffinKmeans (Alic *et al.*, 2014) groups reads based on spectral clustering before applying MSA. (iv) Filtering methods, such as Diginorm (Brown *et al.*, 2012), exclude a substantial number of reads classified as erroneous based on *k*-mer frequencies. (v) Hybrid methods, such as LoRDEC (Salmela and Rivals, 2014), Proovread (Hackl *et al.*, 2014), LSC (Au *et al.*, 2012) and PBcR (Koren *et al.*, 2012), specifically target the correction of Pacific Biosciences reads, which are very long and chimeric, using Illumina reads.

Read error correction has two major key challenges: (i) correcting reads associated with low-covered regions of the genome; reads having high error rate and reads that can be mapped to inexact repeat regions. (ii) Handling insertion and deletion errors (Most methods consider only substitution errors. At the time of writing this article, Blue, Fiona, HECTOR, MuffinKmeans, DAGCon, Coral and HSHREC support insertion and deletion errors. The current implementation of BLESS supports substitution errors only. Hybrid methods target specifically Pacific Biosciences reads.). This is important since sequencing machines that produce long reads, which are useful for obtaining high-quality assemblies, suffer from such errors (Table 1).

We propose Karect (*K*AUST *a*ssembly *r*ead *e*rror *c*orrection *t*ool). Karect belongs to the MSA category. It considers each read *r* as reference, performs multiple alignment for a set of reads similar to *r* and stores the accumulated results in a partial order graph (POG; Lee *et al.*, 2002). Compared with existing approaches, Karect introduces novel methods to select an optimized set of reads similar to *r*; represent reads in the graph; compute weights for the graph edges and construct corrected reads.

Karect has the following advantages: (i) it supports substitution errors (called *mismatches*), insertion and deletion errors (called *indels*) and is compatible with most NGS technologies. (ii) It is fast, supports parallel execution on multi-core CPUs and can work with limited memory. (iii) It is effective against low-coverage regions, high error-rate regions and inexact repeat regions. (iv) Experiments on data from several genomes sequenced by various sequencing technologies show that Karect consistently outperforms existing techniques in terms of both individual-bases error correction (up to 10% increase in accuracy gain) and post *de novo* assembly quality (up to 10% increase in NGA50).

Algorithm: CorrectErrors

Input: Set of reads R
Output: Set of corrected reads R'

```
1 foreach $r \in R$ do
2     Create initial partial order graph $G_r$ for $r$
3     Select a set $C_r$ of candidate reads overlapping with $r$
4     foreach $c \in C_r$ do
5         Find valid semi-global alignment between $c$ and $r$
6         Normalize the resulting alignment
7         Encode the normalized alignment into $G_r$
8     Extract the corrected read $r'$ from $G_r$ and add $r'$ into result set $R'$
9 return $R'$
```

This article also introduces an improved framework for evaluating the quality of error correction methods.

2 Methods

The general framework of the introduced error correction mechanism is described in Algorithm CorrectErrors. The novel aspects of the algorithm are explained in the following subsections. Our framework uses POGs to accumulate partial alignment results. POGs are directed acyclic graphs that represent multiple alignment information of a set of sequences. POGs are used in various bioinformatics applications, including protein alignment (Lee et al., 2002) and the DAGCon module within the HGAP assembler (Chin et al., 2013).

2.1 Selecting candidate reads

Let *R* be the set of reads from the sequencing machine. We consider each $r \in R$ as reference read and select a set $C_r \subset R$ of candidate reads to align with *r* (line 3 in Algorithm CorrectErrors). Ideally, C_r should contain all reads that have significant overlap with *r*. However, this would result in very high computational cost during the alignment phase. Therefore, existing error correction techniques rely on heuristics to select a small set of candidate reads. For example, Coral (Salmela and Schroder, 2011) selects reads that share with *r* at least an exact *k*-mer. Unfortunately, this heuristic is too restrictive and may incorrectly discard many useful reads.

Karect employs an improved heuristic that becomes progressively less restrictive by allowing mismatches/indels. It generates all *k*-mers (Default *k* ranges from 27 to 42 based on number of reads.) of the reference *r* and each candidate read and splits each *k*-mer in three equal parts of length $l = k/3$. Let r_i be a *k*-mer of *r*; an example for r_i = `AAACCCTTT` is shown in Figure 1. Let C_{ri} be the set of candidate reads containing a *k*-mer that matches r_i. To construct C_{ri}, four types of matches are used: type (a): Karect first initializes

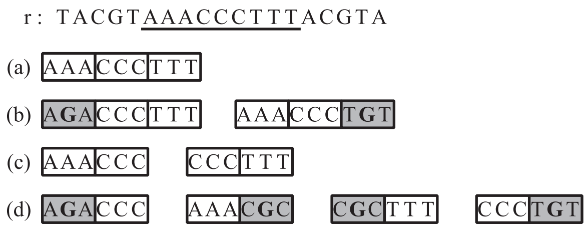

Fig. 1. Selecting candidate reads to align with *r* (i.e. the reference read being corrected). Assume $k = 9$. Each small rectangle represents a sequence of size $l = 3$. White rectangles correspond to error-free matches, whereas gray rectangles allow up to $d = 2$ errors. The figure depicts examples of *k*-mers that must be contained in candidate reads of match type (a), (b), (c) and (d). These *k*-mers match the underlined *k*-mer of the read *r*

C_{ri} to reads that share with r exactly k-mer r_i (Fig. 1a). Type (b): If less that m type (a) reads are found (m is a user-defined constraint [default $m = \max(30, \min(150, 0.6{\cdot}\text{estimated coverage}))$.]), Karect adds to C_{ri} reads that may contain up to d mismatches/indels in the l-prefix or l-suffix of k-mer r_i (Fig. 1b), where d is a user-defined parameter (default $d = 2$). To count the mismatches or indels, the Hamming or edit distance is used, respectively. Type (c): If $|C_{ri}| < m$ Karect generates two smaller k'-mers of r_i, where $k' = 2l$ and searches for exact k'-mer matches (Fig. 1c). Type (d): If $|C_{ri}| < m$, Karect searches for reads that contain up to d mismatches/indels in the l-prefix or l-suffix of the k'-mer (Fig. 1d).

To reduce the effect of bias towards specific k-mers, C_{ri} is allowed to include at most m reads sharing the same k-mer or k'-mer. C_{ri} reads are added to C_r, and the process is repeated for other k-mers of r. For more details refer to the Supplementary Document.

2.2 Alignment and normalization of candidate reads

Our goal is to correct reference read r. Karect aligns each read c in the candidate set C_r, against r (line 5 in Algorithm CORRECTERRORS). The result includes the start and end of c or r (semi-global alignment) to allow the alignment of overlaps. We use a variant of the Needleman and Wunsch (1970) algorithm; refer to the Supplementary Document for details.

To exclude candidate reads sequenced from different genome regions, an alignment is considered valid only if the overlap exceeds a threshold (Default $\tau_1 = \max(\min(0.7 \cdot \text{avgReadLen}, 35), 0.2{\cdot}\text{refReadLen})$.) τ_1 and the number of mismatches/indels within the overlap does not exceed a threshold (Default $\tau_2 = 25\%$ of the overlap.) τ_2. This rudimentary filter may still accept some reads from irrelevant genome regions. To further minimize this problem, Karect assigns a weight w_c to each read (refer to Section 2.5).

Consider reference read $r = \texttt{CAA}$ and candidate read $c_1 = \texttt{GAAA}$. r can be transformed to c_1 by substituting `C` with `G` at position 1 and inserting `A` at position 4. Substitutions are modeled as deletions followed by insertions. Therefore, the alignment corresponds to del(`C`,1); ins(`G`,1); ins(`A`,**4**). Now consider another candidate read $c_2 = \texttt{AAA}$. r can be transformed to c_2 by the following operations: del(`C`,1); ins(`A`,1). Observe that, inserting an `A` at position 1 generates the same string as inserting `A` at position 4. Therefore, an equivalent representation for the alignment is del(`C`,1); ins(`A`,**4**). We call this the *normalized* form of the alignment (line 6 in Algorithm CORRECTERRORS), where normalization means that operations are shifted as far as possible to the right. Normalization allows better grouping of operations of a set of candidate reads, which enables Karect to correct reference reads with high accuracy. In the previous example, after normalization it is revealed that, to correct r, we must insert an `A` at position 4, with high probability. The concept of normalization is also used in DAGCon (Chin *et al.*, 2013), but the resulting representation is suboptimal; the details are explained in the Supplementary Document. Note that normalization is not required if the sequencing technology generates only substitution errors.

2.3 Storing alignments in the POG

Each normalized alignment is stored in a POG G_r associated with the reference read r (line 7 in Algorithm CORRECTERRORS). Initially, G_r represents only r. The candidate read alignments are then added incrementally in G_r in a manner similar to DAGCon, with the difference that similar out-nodes (i.e. nodes connected by edges coming out from the same node) are merged instantly; this saves time and space. Also, in contrast to DAGCon, similar in-nodes (i.e. nodes connected by edges going to the same node) are not merged, since this is not required by our extraction algorithm; this also saves computational time. Figure 2 illustrates an example of aligning four candidate reads $c_1, \ldots, c_4$ to reference read r. The value on each edge corresponds to the number of alignments passing through that edge. We are going to modify these values in Section 2.5.

For sequencing technologies that generate only substitution errors, instead of a POG we use an array of size $|r|$ to accumulate alignment weights.

2.4 Extracting corrected read from the POG

Given POG G_r for a reference read r, the corrected read r' corresponds to a path within G_r (line 8 in Algorithm CORRECTERRORS). There are many ways to select such a path. For instance, it can be the path that maximizes the sum of edge scores, but the quality of error correction is expected to be low, because the heuristic favors longer paths. As another example, DAGCon assigns each node a score based on the weights of the out-edges and local coverage and selects the path that maximizes the sum of node scores.

We propose a novel approach. First, we normalize all edge weights such that the sum of the out-edge weights of any node is 1 (Fig. 3). The rationale is that, after normalization, edge weights will reflect the transition probability between nodes. Then, the problem is mapped to the classic problem of finding the most reliable path in a network (Petrovic and Jovanovic, 1979), which is the path that maximizes the product of edge weights. Since POGs are directed

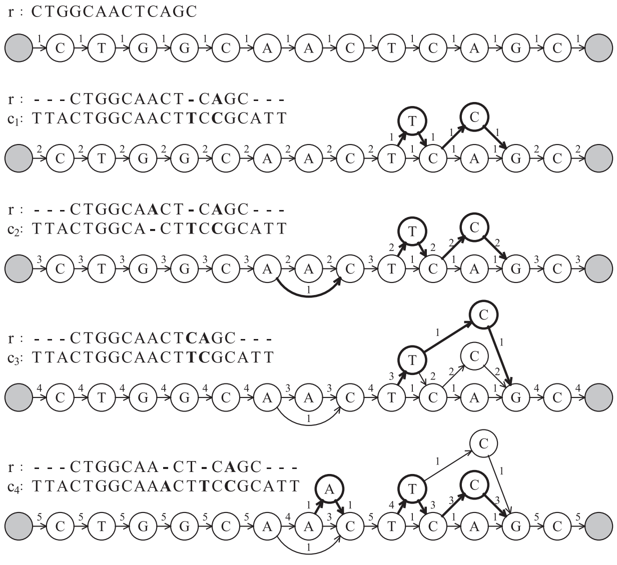
r : CTGGCAACTCAGC
r : - - - CTGGCAACT - CAGC - - -
c1: TTACTGGCAACTTCCGCATT
r : - - - CTGGCAACT - CAGC - - -
c2: TTACTGGCA - CTTCCGCATT
r : - - - CTGGCAACTCAGC - - -
c3: TTACTGGCAACTTCGCATT
r : - - - CTGGCAA - CT - CAGC - - -
c4: TTACTGGCAAACTTCCGCATT

Fig. 2. Example POG. The first row shows the initial POG for reference read *r*. In the second row, c_1 introduces an insertion and a substitution. Next, c_2 includes a deletion, an insertion and a substitution and so on. At each row, the newly introduced changes are shown in bold

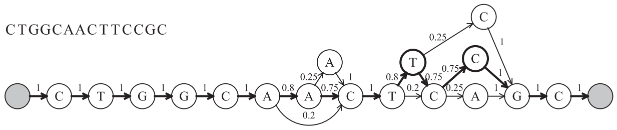

Fig. 3. Normalized POG of Figure 2. The extracted path is shown in bold

acyclic graphs, the score of the best path can be found by dynamic programming using the following recursive rule:

$$\begin{aligned} \mathrm{pW}(s) &= 1 \\ \mathrm{pW}(u) &= \max_{v \in V(G_r)}\{\mathrm{pW}(v) \cdot \overline{w}(v,u)\} \end{aligned} \tag{1}$$

The score of the best path is $\mathrm{pW}(t)$ where s and t are unique start and end nodes, $V(G_r)$ is the set of nodes in the POG, $\overline{w}(v,u)$ is the normalized edge weight between nodes $v,u \in V(G_r)$ or 0 if they are not directly connected. The corrected read r' corresponds to the nodes of the best path.

For sequencing technologies that generate only substitution errors, r' is computed as the consensus (i.e. the sequence containing the most frequent base in each position) in a gap-free multiple alignment.

2.5 Handling high-error regions

For simplicity, in Figure 2, we assumed weight $w_c = 1$ for each read c. An aligned read increases the value of all relevant edges by w_c. To accommodate for the fact that some of the aligned reads may be sequenced from a different region of the genome, for each read c we calculate w_c as follows:

$$w_c = 1 - \sqrt{\mathrm{overlapError}/(\mathrm{overlapSize} \cdot \mathrm{maxErrorRate})} \tag{2}$$

where overlapError is the edit distance between the overlapping regions of the two reads (or the Hamming distance if only substitution errors are considered), overlapSize is the size of the aligned reference read overlap and maxErrorRate is a constant (default is 0.25). Intuitively, the equation favors very similar reads that belong with high probability to the same genome region, to less similar ones that are still probable to be from the same region but highly affected by sequencing errors. This configuration enables our approach to handle high-error regions better than previous approaches. If quality values from the sequencing machine are available, we set edge weights to $w_c \cdot (1 - 10^{-0.1 q_{ci}})$, where q_{ci} is the quality value associated with the destination base at position i.

2.6 Handling coverage variability

Let there be two regions in the genome that are very similar (e.g. they differ by only a few bases). Assume that coverage is not uniform, which is the typical case for NGS technologies and let one of these regions be more covered than the other. For example, assume genome region AAAAAAA is covered by 50 reads, whereas region AAACAAA is covered by only 20 reads. The POGs resulting from the methodology described above, will be biased towards converting all reads from the lesser covered region, to match those of the higher covered one; obviously this is wrong. To minimize this problem, if an *original* edge weight in the POG exceeds a threshold (Default $\tau_3 = \min(100, 0.42 \cdot estCoverage)$. The estimated coverage is multiplied by 1/2 for diploid genomes to get the coverage of a single chromosome copy; refer to the Supplementary Document for details.) τ_3, Karect eliminates all other out-edges from its source node before extracting the corrected read. By original edge, we mean an edge in the initial POG of the reference read r being corrected, such as the topmost POG in Figure 2.

3 Results

We compare the error correction quality of Karect against the state-of-the-art tools for substitution errors, namely Lighter (Song *et al.*, 2014), Trowel (Lim *et al.*, 2014), BLESS (Heo *et al.*, 2014), Musket (Liu *et al.*, 2013), RACER (Ilie and Molnar, 2013), SGA (Simpson and Durbin, 2012), Quake (Kelley *et al.*, 2010), Reptile (Yang *et al.*, 2010) and Diginorm (Brown *et al.*, 2012). We also compare against the top performing tools that support insertion and deletion errors, namely Blue (Greenfield *et al.*, 2014), Fiona (Schulz *et al.*, 2014), DAGCon (Chin *et al.*, 2013), Coral (Salmela and Schroder, 2011), MuffinKmeans (Alic *et al.*, 2014) and HSHREC (Salmela, 2010). We employ a Linux machine with 2 × 6-core Intel CPUs at 2.67GHz and 192GB RAM.

3.1 Datasets

We use data from the 454, Ion Torrent and Illumina sequencing machines; Table 2 lists the details. (i) The 454 datasets consist of fragment and paired-end libraries of *Helicobacter pylori*, *Zymomonas mobilis* and *Escherichia coli*. The same datasets were also used by Finotello *et al.* (2012), but in their study, they used only the paired-end library for *Z.mobilis*, whereas we use the fragment library, too, to increase coverage. Also, we discard a portion of the reads for *H.pylori*, to obtain moderate coverage. (ii) The Ion Torrent dataset is a fragment library of *E.coli*. (iii) The Illumina datasets consist of paired-end libraries of *Staphylococcus aureus*, *Human Chromosome 14* and *Caenorhabditis elegans*. They appear in various previous studies such as GAGE (Salzberg *et al.*, 2012) and the works of Heo *et al.* (2014), Kleftogiannis *et al.* (2013), Liu *et al.* (2013) and Ilie and Molnar (2013). All datasets exhibit non-uniform coverage along the genome, which poses a major challenge for error correction; the relevant statistics appear in the Supplementary Document. We use

Table 2. Description of the datasets

Dataset	Sequencing machine	Accession number	Reference genome	Genome length	Read length	Number of reads	Number of base pairs	Coverage (×)	Aligned bases (%)
h.pylori.454[a]	454 FLX	SRR023794, SRR023796	NC_017355.1	1 588 278	233	279 235	65 048 583	40.96	95.22
z.mobilis.454[a]	454 FLX	SRR017972, SRR029606	NC_006526.2	2 056 363	190	210 814	40 087 238	19.49	96.05
e.coli.454[b]	454 Titanium	Roche	NC_000913.3	4 641 652	216	439 155	94 874 123	20.44	99.14
e.coli.ion[c]	Ion Torrent	B22-730	NC_010473.1	4 686 137	326	492 537	160 671 071	34.29	99.06
s.aureus.ill	Illumina	SRR022868 - GAGE	NC_010079.1	2 903 081	101	1 294 104	130 704 504	45.02	78.19
human.c14.ill	Illumina	GAGE	NC_000014.8	88 289 540	101	36 504 800	3 686 984 800	41.76	95.10
c.elegans.ill[d]	Illumina	SRR065390	wormbase	100 286 401	100	67 617 092	6 761 709 200	67.42	90.72

We used the GAGE reference of s.aureus.ill, which includes two plasmid sequences in addition to NC_010079.1. For the 454 and Ion Torrent datasets, we report the average read length, since reads do not have the same length.

[a] Available from NCBI SRA: http://www.ncbi.nlm.nih.gov/sra.

[b] Available from Roche: http://www.medcomp.medicina.unipd.it/Ecoli_dataset/.

[c] Available from Ion Torrent, under name B22-730: http://ioncommunity.lifetechnologies.com/welcome.

[d] c.elegans.ill reference is available at: ftp://ftp.wormbase.org/pub/wormbase/species/c_elegans/sequence/genomic/c_elegans.PRJNA13758.WS241.genomic.fa.gz.

the Roche tools to filter 454 and Ion Torrent data and employ QUAST (Gurevich *et al.*, 2013) to evaluate the resulting assemblies. Additional datasets for ultra-high coverage datasets and RNA-Seq data appear in the Supplementary Document.

3.2 Novel quality evaluation framework

Two main metrics have been used in previous work to evaluate the quality of error correction. The first one considers *individual bases* and differentiates correct base-operations from incorrect ones (Liu *et al.*, 2013; Salmela and Schroder, 2011; Yang *et al.*, 2013). The second metric is applied on the granularity of *entire reads* and differentiates correct reads from incorrect ones (Ilie and Molnar, 2013; Ilie *et al.*, 2011; Schroder *et al.*, 2009). Although each metric has its tradeoffs (Ilie and Molnar, 2013; Liu *et al.*, 2013), depending on the biological application, both are useful. For example, in the case of single-nucleotide polymorphism discovery, the base-level metric is more appropriate, whereas in the case of *de novo* assembly, the read-level metric is more suitable. For completeness, in this article, we report results for both metrics. However, the instantiations of the metrics in previous work have drawbacks. Below we propose improved versions for both metrics.

In the following, we define true positive (TP) an instance that was wrong initially but became correct after the application of the error correction algorithm, denoted as: $\mathrm{TP} \quad := \text{wrong} \rightarrow \text{correct}$. In a similar way, we define true negative $\mathrm{TN} := \text{correct} \rightarrow \text{correct}$; false positive $\mathrm{FP} := \text{correct} \rightarrow \text{wrong}$ and false negative $\mathrm{FN} := \text{wrong} \rightarrow \text{wrong}$. We use the common definitions of $\text{Recall} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$; $\text{Precision} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$; $\text{FScore} = 2 \cdot \text{Precision} \cdot \text{Recall}/(\text{Precision} + \text{Recall})$ and $\text{Gain} = (\mathrm{TP}-\mathrm{FP})/(\mathrm{TP}+\mathrm{FN})$.

3.2.1 Individual base-operations metric

The metric lists the edit operations to transform the original and corrected read to the reference region, respectively, and counts the differences (Yang *et al.*, 2013). It works when only substitution errors are considered, but it is not well defined in the presence of insertions and deletions (Heo *et al.*, 2014), because in this case there are multiple ways to list the edit operations. We propose an alternative definition based on the actual distance, instead of the list of edit operations.

Let o and c be the original and corrected read, respectively. The distance between them is the number of bases that were wrong in o but got corrected in c (i.e. true positives TP), plus the number of bases that were correct in o but got wrongfully altered in c (i.e. false positives FP); formally $D(o,c) = \mathrm{TP} + \mathrm{FP}$. D is the Hamming distance for substitution-only errors or the edit distance, otherwise; in both cases, D can be computed unambiguously. Let P_o be the set of several genome regions where o can be mapped with minimum distance, and p_c be the member of P_o that has the smallest distance to c; formally $p_c = \operatorname{argmin}_{p \in P_o} D(p,c)$. Using the same methodology as above, we define $D(o,p_c) = \mathrm{TP} + \mathrm{FN}$ and $D(c,p_c) = \mathrm{FP} + \mathrm{FN}$. Solving the system of equations results in $\mathrm{FN} = \frac{1}{2}(D(o,p_c) - D(o,c) + D(c,p_c))$; $\mathrm{TP} = D(o,p_c) - \mathrm{FN}$ and $\mathrm{FP} = D(o,c) - \mathrm{TP}$.

3.2.2 Entire reads metric

We align the original read o with the reference genome and, similar to the previous section, we obtain the set P_o of (possible many) genome regions where o can be mapped with minimum distance D. Let c be the corrected read after applying the correction algorithm on o. o and c are considered *correct* if they match exactly any region $p \in P_o$, else they are considered *wrong*. On the basis of this definition, we use the formulas from Section 3.2 to calculate TP, FP and TN.

3.3 Correction quality

We evaluate Karect against existing tools in terms of recall, precision, Fscore and gain, using both the individual base operations and the entire reads metrics; Tables 3 and 4 summarize the results. For each dataset, the best result of each column is shown in bold. We

Table 3. Quality of error correction on datasets containing insertion and deletion errors

Dataset	Error	Base-operations (%)				Whole reads (%)				Time	Memory
	Correction	Recall	Precision	FScore	Gain	Recall	Precision	FScore	Gain	(min)	(GB)
h.pylori.454	**Karect**	95.85	**99.77**	**97.77**	**95.63**	97.33	99.94	**98.62**	**97.28**	**1.13**	1.47
	DAGCon	**96.85**	97.70	97.27	94.57	**97.45**	99.42	98.42	96.88	1.83	—
	Fiona	88.43	97.22	92.62	85.90	82.94	**99.99**	90.67	82.94	2.68	0.81
	Blue	90.93	75.41	82.44	61.28	53.63	99.81	69.77	53.52	1.77	**0.62**
	Coral	87.66	68.03	76.60	46.46	62.66	64.68	63.65	28.44	16.77	3.69
	MuffinKmeans	77.42	81.62	79.47	59.99	65.30	91.54	76.22	59.26	6.63	0.80
	HSHREC	70.34	69.67	70.00	39.72	63.23	98.63	77.06	62.35	9.57	13.03
z.mobilis.454	**Karect**	95.02	**99.61**	**97.26**	**94.64**	93.60	**99.87**	**96.63**	**93.48**	**1.17**	0.92
	DAGCon	**96.85**	91.11	93.89	87.40	**94.30**	98.61	96.41	92.97	1.50	—
	Fiona	90.66	95.78	93.15	86.67	82.77	99.84	90.51	82.64	2.28	0.97
	Blue	89.33	77.17	82.80	62.90	55.20	99.86	71.10	55.12	**1.17**	**0.32**
	Coral	88.50	74.00	80.61	57.41	66.29	72.36	69.19	40.97	3.88	3.41
	MuffinKmeans	60.33	79.32	68.53	44.60	51.38	92.01	65.94	46.91	4.92	0.97
	HSHREC	73.42	86.71	79.51	62.17	73.83	98.87	84.53	72.98	5.58	8.39
e.coli.ion	**Karect**	96.27	**99.24**	**97.73**	**95.53**	89.62	99.74	94.41	89.39	**4.23**	3.61
	DAGCon	**97.92**	95.25	96.57	93.03	**91.34**	99.06	**95.04**	**90.47**	5.98	—
	Fiona	84.80	98.45	91.12	83.47	71.88	**99.89**	83.60	71.80	4.75	1.34
	Blue	89.03	77.33	82.77	62.93	48.06	99.68	64.85	47.91	6.25	**1.15**
	Coral	87.41	86.51	86.96	73.78	63.86	89.81	74.65	56.62	51.15	5.26
	MuffinKmeans	72.15	69.74	70.92	40.84	49.28	92.57	64.32	45.33	15.13	2.19
	HSHREC	69.54	86.32	77.02	58.52	57.45	99.60	72.87	57.22	20.33	14.36

We compare against tools which support these types of errors. All programs use 12 threads. More datasets appear in the Supplementary Document.

Table 4. Quality of error correction on datasets containing substitution errors (Illumina)

Dataset	Error	Base-operations (%)				Whole reads (%)				Time	Memory
	correction	Recall	Precision	FScore	Gain	Recall	Precision	FScore	Gain	(min)	(GB)
s.aureus.ill	**Karect**	**99.54**	**99.93**	**99.73**	**99.46**	**98.87**	99.99	**99.42**	**98.86**	3.25	2.58
	Fiona	83.31	99.38	90.64	82.79	83.56	99.94	91.02	83.51	5.48	1.25
	BLESS	89.10	99.75	94.12	88.87	90.56	**100.00**	95.05	90.56	6.67	**0.01**
	Blue	95.61	95.70	95.66	91.32	96.08	99.91	97.96	95.99	2.87	1.95
	SGA	71.66	99.72	83.39	71.46	84.67	**100.00**	91.70	84.67	2.49	0.09
	Musket	72.04	99.86	83.70	71.94	78.46	99.99	87.93	78.45	1.67	0.23
	RACER	89.87	97.94	93.73	87.98	88.73	99.92	93.99	88.66	0.80	1.24
	Trowel	46.89	71.95	56.77	28.60	40.11	92.37	55.94	36.80	**0.13**	0.47
	Lighter	62.86	99.65	77.09	62.64	76.27	99.95	86.52	76.24	0.17	0.10
	Coral	57.59	98.19	72.60	56.52	79.37	99.25	88.20	78.77	6.32	6.05
	Quake	37.91	—	—	—	21.37	—	—	—	5.20	0.30
	Reptile	8.90	86.83	16.14	7.55	8.26	96.36	15.21	7.94	9.67	2.21
	MuffinKmeans	57.19	40.26	47.25	-27.68	52.39	94.24	67.35	49.19	56.65	2.50
human.c14.ill	**Karect**	**85.23**	**97.74**	**91.06**	**83.26**	**71.46**	99.79	**83.28**	**71.31**	47.48	81.30
	Fiona	80.49	94.95	87.13	76.21	68.62	99.66	81.28	68.39	113.80	28.91
	BLESS	70.64	95.44	81.19	67.26	65.93	99.87	79.43	65.85	193.50	**0.14**
	Blue	78.88	87.00	82.74	67.10	64.58	99.06	78.19	63.97	35.77	17.75
	SGA	70.03	96.70	81.23	67.64	63.84	**99.98**	77.92	63.82	54.82	1.33
	Musket	67.68	94.61	78.91	63.82	59.45	99.79	74.51	59.33	40.22	2.35
	RACER	75.73	57.88	65.61	20.62	64.00	99.22	77.81	63.49	18.01	10.58
	Trowel	55.74	92.50	69.56	51.22	46.19	99.86	63.17	46.13	13.60	19.48
	Lighter	60.51	86.37	71.16	50.96	51.95	99.88	68.35	51.89	**6.10**	0.48
	Coral	72.70	75.65	74.15	49.30	69.25	87.70	77.39	59.54	285.82	58.83
	Quake	42.41	—	—	—	14.31	—	—	—	35.27	2.07
	Reptile	4.04	17.74	6.58	−14.70	1.76	12.33	3.08	-10.78	414.62	9.98
c.elegans.ill	**Karect**	**88.69**	98.77	**93.46**	**87.58**	**86.23**	99.89	**92.56**	**86.14**	102.93	147.83
	Fiona	80.82	97.27	88.28	78.55	79.87	99.89	88.76	79.78	244.67	58.48
	BLESS	78.82	98.54	87.58	77.65	81.55	99.98	89.83	81.54	328.35	**0.17**
	Blue	82.26	74.11	77.97	53.53	81.28	98.29	88.98	79.87	40.57	16.58
	SGA	77.03	**99.04**	86.66	76.28	79.55	**100.00**	88.61	79.55	81.15	2.27
	Musket	59.65	95.89	73.55	57.09	70.03	99.91	82.34	69.97	42.60	2.82
	RACER	80.63	66.52	72.90	40.04	80.14	99.80	88.89	79.97	34.75	12.38
	Trowel	54.86	93.43	69.13	51.01	56.39	99.11	71.89	55.89	23.30	29.65
	Lighter	51.63	83.96	63.94	41.77	56.35	99.96	72.07	56.33	**10.95**	0.53
	Coral	70.22	74.98	72.52	46.78	75.87	83.78	79.63	61.18	632.25	85.62
	Quake	42.41	—	—	—	14.37	—	—	—	40.95	2.15

All programs use 12 threads, except BLESS and Reptile use 1 thread. Reptile failed for c.elegans.ill. MuffinKmeans failed for human.c14.ill and c.elegans.ill. We omitted some Quake results since trimming reduces correction quality, by treating each trimmed base or read as false positive or false negative.

use the default parameters for all tools. If some parameter does not have default value, we select the value suggested by the examples associated with that tool. Since DAGCon is not a stand-alone error correction tool (it is a component inside HGAP assembler), we run it by selecting candidate reads as proposed in Section 2.1, then align them using edit distance and pass alignments to DAGCon. We disable trimming in all tools, except Quake. We run Quake such that it outputs corrected and uncorrected reads. Diginorm is not included in the tables, because it filters rather than correcting reads. More details about the setup are given in the Supplementary Document. Karect consistently outperforms existing methods for most datasets (up to 10% increase in accuracy gain). Summary of these results using sum-of-ranks approach appear in the Supplementary Document. We also test Karect with ultra-high coverage datasets and RNA-Seq data. The results appear in the Supplementary Document.

3.4 De novo assembly

We evaluate the effect of error correction on *de novo* assembly using the following top performing assemblers: Celera (Miller *et al.*, 2008) and Newbler (from Roche company) for the 454 and Ion Torrent datasets; and Velvet (Zerbino and Birney, 2008), SGA (Simpson and Durbin, 2012) and Celera for the Illumina datasets. Celera requires read quality values as input. Since Fiona, DAGCon and HSHREC do not output such information, we manually set their quality values to ‘I’. Newbler is tested both with and without quality values as input; the later case is denoted by Newbler (NQ). We do not test Newbler with diginorm, since diginorm alters the read headers in a way that Newbler is unable to detect paired reads. SGA is run after filtering out its initial error correction stage. More details about the setup are included in the Supplementary Document. Assemblies are evaluated using QUAST (Gurevich *et al.*, 2013). All contigs/scaffolds are split such that each of them aligns to the reference genome; contigs/scaffolds with less than 500 bp are excluded. For both contigs and scaffolds, we report the standard metrics: NGA50, LGA50, coverage, error rate, and number of global and local misassemblies and unaligned sequences; we report also N symbols rate for scaffolds; refer to Gurevich *et al.* (2013) for details.

Table 5. Assembly of the *E.coli* 454 dataset (insertion and deletion errors)

Error correction	Assembler	Contigs							Scaffolds							
		NGA50	LGA50	GM	LM	UA	MM	Cov	NGA50	LGA50	GM	LM	UA	MM	Cov	*N*
—	Celera	472 473	4	3	11	0	12	99.93	1 750 103	2	4	21	0	11	99.94	46
	Celera (sff)	639 160	**3**	4	13	1	13	99.86	2 346 056	**1**	4	18	1	13	99.86	56
	Newbler	112 331	15	0	**5**	0	**2**	98.14	1 295 223	2	4	77	0	**2**	97.85	1982
Karect	Celera	**858 863**	**3**	2	13	0	10	**99.97**	2 565 113	**1**	3	17	0	10	**99.98**	90
	Newbler	112 524	14	0	10	0	9	98.20	1 109 133	2	6	80	0	9	97.80	2143
	Newbler (NQ)	110 321	15	0	10	0	9	98.21	**3 362 497**	**1**	4	82	0	9	97.84	2004
DAGCon	Celera	235 860	6	6	22	1	53	99.80	1 130 974	2	10	51	0	53	99.82	84
	Newbler (NQ)	73 648	20	1	15	1	22	97.36	618 779	3	7	115	0	22	96.70	3368
Fiona	Celera	682 339	**3**	2	10	0	11	99.95	2 569 644	**1**	**2**	17	0	11	99.96	15
	Newbler (NQ)	117 500	13	0	7	1	9	98.12	962 800	2	5	79	0	9	97.85	1983
Blue	Celera	780 219	**3**	2	9	0	10	**99.97**	2 345 312	**1**	4	**15**	0	10	**99.98**	**5**
	Newbler	111 928	14	0	7	1	8	98.20	1 492 028	2	7	77	0	8	97.88	1897
	Newbler (NQ)	112 390	14	0	9	1	8	98.21	923 165	2	8	76	0	8	97.87	2047
Coral	Celera	473 444	5	4	12	0	171	99.90	1 809 222	2	7	22	0	166	99.95	66
	Newbler	125 630	13	0	10	1	173	98.10	845 431	3	8	80	0	173	97.76	2128
	Newbler (NQ)	112 408	13	0	8	0	172	98.09	923 986	2	6	81	0	172	97.79	2186
MuffinKmeans	Celera	562 359	4	3	19	0	18	99.83	2 881 627	**1**	4	30	0	18	99.83	115
	Newbler	103 722	15	0	11	1	13	98.07	942 705	2	5	85	0	12	97.69	2414
	Newbler (NQ)	97 794	16	0	11	1	14	98.08	941 847	2	5	87	0	14	97.72	2429
HSHREC	Celera	115 342	13	2	16	0	17	99.64	1 137 457	2	9	59	0	18	99.72	224
	Newbler (NQ)	78 203	20	1	7	0	11	97.95	823 237	3	7	109	0	11	97.66	2733
Diginorm	Celera	3354	387	3	11	0	20	84.20	73 273	20	246	872	0	26	84.70	13 889

Data are filtered using the Roche tools, except for Celera (sff). GM, global misassemblies; LM, local misassemblies; UA, unaligned contigs/scaffolds; MM, rate of mismatches/indels per 100 kb; *N*, rate of *N* symbols per 100 kb; Cov, coverage (%). Newbler (NQ) means data are passed to Newbler without quality values.

Table 6. Assembly of the *S.aureus* Illumina dataset (only substitution errors)

Error correction	Assembler	Contigs							Scaffolds							
		NGA50	LGA50	GM	LM	UA	MM	Cov	NGA50	LGA50	GM	LM	UA	MM	Cov	*N*
—	Velvet	3024	295	10	0	2	9	93.88	3941	234	10	16	16	18	93.88	566
	SGA	1491	565	5	0	0	**1**	85.86	1530	555	6	0	0	2	86.01	0
	Celera	6644	135	6	4	1	20	91.92	8603	104	9	35	1	22	91.95	67
Karect	Velvet	**27 586**	**34**	7	3	0	3	97.95	**37 926**	**23**	7	6	0	4	97.96	73
	SGA	24 633	39	3	1	0	2	**98.08**	25 321	38	4	1	0	2	**98.14**	0
	Celera	24 177	38	6	5	0	13	97.18	33 840	29	8	14	0	15	97.20	32
Fiona	Velvet	20 239	47	5	3	0	8	97.79	26 282	35	7	6	0	12	97.81	116
	SGA	15 721	58	**2**	2	0	4	97.69	15 861	57	**3**	2	0	4	97.71	0
	Celera	16 003	56	8	3	0	21	96.95	23 109	42	9	15	0	24	96.97	50
BLESS	Velvet	6823	128	10	0	1	6	97.24	8853	99	12	9	5	12	97.26	302
	SGA	4676	190	**2**	0	0	**1**	95.91	4842	182	4	0	0	**1**	95.98	0
	Celera	5397	169	9	2	1	18	91.11	8393	105	11	18	0	26	91.30	166
Blue	Velvet	26 459	35	9	3	0	3	97.88	32 271	28	9	7	0	4	97.88	74
	SGA	19 863	45	4	1	0	2	97.77	21 851	43	5	1	0	2	97.87	1
	Celera	24 936	37	8	5	0	18	97.24	33 321	27	12	13	0	19	97.24	32
SGA	Velvet	8610	104	5	1	0	2	97.42	9576	92	6	3	1	4	97.51	121
	SGA	6767	131	3	1	0	**1**	96.89	6857	128	4	1	0	**1**	96.94	0
	Celera	8408	98	7	4	0	22	94.89	12 393	72	7	27	0	25	94.92	77
Musket	Velvet	13 862	68	6	2	0	5	97.63	17 295	51	7	14	0	8	97.64	161
	SGA	9020	98	4	0	0	**1**	97.49	9314	97	5	0	0	2	97.52	0
	Celera	12 366	73	13	3	0	10	96.25	18 811	49	11	17	0	15	96.30	90
RACER	Velvet	7779	113	11	4	2	15	97.39	10 690	80	23	13	0	26	97.53	284
	SGA	5754	157	6	0	3	3	96.43	5923	151	8	1	3	4	96.56	1
	Celera	6839	129	23	4	3	26	93.30	11 918	77	31	21	2	32	93.43	147
Lighter	Velvet	6399	138	9	1	1	7	97.08	8521	107	11	13	2	13	97.19	334
	SGA	4386	203	**2**	0	0	3	95.74	4450	199	**3**	0	0	4	95.79	0
	Celera	8293	106	9	3	0	14	93.65	12 022	76	12	17	0	18	93.74	91
Coral	Velvet	17 801	49	10	7	2	19	97.50	22 173	41	10	11	3	15	97.49	82
	SGA	14 286	64	4	2	1	10	97.37	14 702	62	6	4	1	10	97.37	3
	Celera	15 767	55	10	6	2	28	96.00	20 301	44	13	18	2	29	96.02	34

GM, global misassemblies; LM, local misassemblies; UA, unaligned contigs/scaffolds; MM, rate of mismatches/indels per 100 kb; *N*, rate of *N* symbols per 100 kb; Cov, coverage (%). More methods appear in the Supplementary Document.

Tables 5 and 6 summarize the results for the *de novo* assembly of the *E.coli* 454 Titanium dataset (insertion and deletion errors) and the *S.aureus* Illumina dataset (only substitution errors), respectively. The best result of each column is shown in bold. Results for the other datasets are included in the Supplementary Document. The results demonstrate that, compared with existing error correcting methods, Karect improves significantly the assembly quality (up to 10% increase in NGA50). Summary of these results using sum-of-ranks approach appear in the Supplementary Document.

4 Conclusion

We presented Karect, a novel error correction technique for NGS data. Karect is based on multiple alignment, supports substitution, insertion and deletion errors and handles effectively non-uniform coverage as well as moderately covered areas. Extensive experimental evaluation demonstrates that Karect achieves superior error correction compared to existing state-of-the-art methods. Karect also enables substantially improved assemblies, when used as preprocessing step for modern assemblers.

Currently, we do not support Pacific Biosciences data, because of chimeric reads; we are working on this issue.

Funding

AA and PK are supported by the KAUST Base Research Fund of PK. VS is supported by the KAUST Base Research Fund. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Conflict of Interest: none declared.

References

Alic,A. *et al*. (2014) Robust error correction for de novo assembly via spectral partitioning and sequence alignment. In: Rojas,I. and Guzman,F.O. (eds), *Proceedings of the International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO)*, Copicentro Granada S.L., Granada, Spain, pp. 1040–1048.

Au,K.F. *et al*. (2012) Improving PacBio long read accuracy by short read alignment. *PLoS One*, **7**, e46679.

Bankevich,A. *et al*. (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol*., **19**, 455–477.

Brown,C.T. *et al*. (2012) A reference-free algorithm for computational normalization of shotgun sequencing data. *arXiv*: 1203.4802 [q-bio.GN].

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

Chin,C.S. *et al*. (2013) Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat. Methods*, **10**, 563–569.

Finotello,F. *et al*. (2012) Comparative analysis of algorithms for whole-genome assembly of pyrosequencing data. *Brief. Bioinformatics*, **13**, 269–280.

Gnerre,S. *et al*. (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl Acad. Sci. USA*, **108**, 1513–1518.

Greenfield,P. *et al*. (2014) Blue: correcting sequencing errors using consensus and context. *Bioinformatics*, **30**, 2723–2732.

Gurevich,A. *et al*. (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Hackl,T. *et al*. (2014) proovread: large-scale high-accuracy PacBio correction through iterative short read consensus. *Bioinformatics*, **30**, 3004–3011.

Heo,Y. *et al*. (2014) BLESS: bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, **30**, 1354–1362.

Ilie,L. and Molnar,M. (2013) RACER: rapid and accurate correction of errors in reads. *Bioinformatics*, **29**, 2490–2493.

Ilie,L. *et al*. (2011) HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, **27**, 295–302.

Kao,W.C. *et al*. (2011) ECHO: a reference-free short-read error correction algorithm. *Genome Res*., **21**, 1181–1192.

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

Kleftogiannis,D. *et al*. (2013) Comparing memory-efficient genome assemblers on stand-alone and cloud infrastructures. *PLoS One*, **8**, e75505.

Koren,S. *et al*. (2012) Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol*., **30**, 693–700.

Le,H.S. *et al*. (2013) Probabilistic error correction for RNA sequencing. *Nucleic Acids Res*., **41**, e109.

Lee,C. *et al*. (2002) Multiple sequence alignment using partial order graphs. *Bioinformatics*, **18**, 452–464.

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

Lim,E.C. *et al*. (2014) Trowel: a fast and accurate error correction module for Illumina sequencing reads. *Bioinformatics*, **30**, 3264–3265.

Liu,Y. *et al*. (2013) Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, **29**, 308–315.

Medvedev,P. *et al*. (2011) Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, **27**, i137–i141.

Miller,J.R. *et al*. (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.

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.

Nikolenko,S.I. *et al*. (2013) BayesHammer: Bayesian clustering for error correction in single-cell sequencing. *BMC Genomics*, **14**(Suppl 1), S7.

Petrovic,R. and Jovanovic,S. (1979) Two algorithms for determining the most reliable path of a network. *IEEE Trans. Reliab*., **R-28**, 115–119.

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

Qu,W. *et al*. (2009) Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Res*., **19**, 1309–1315.

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

Salmela,L. and Rivals,E. (2014) LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, **30**, 3506–3514.

Salmela,L. and Schroder,J. (2011) Correcting errors in short reads by multiple alignments. *Bioinformatics*, **27**, 1455–1461.

Salzberg,S.L. *et al*. (2012) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res*., **22**, 557–567.

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

Schulz,M.H. *et al*. (2014) Fiona: a parallel and automatic strategy for read error correction. *Bioinformatics*, **30**, i356–i363.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*., **22**, 549–556.

Song,L. *et al*. (2014) Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biol*., **15**, 509.

Wijaya,E. *et al*. (2009) Recount: expectation maximization based error correction tool for next generation sequencing data. *Genome Inform*., **23**, 189–201.

Wirawan,A. *et al*. (2014) HECTOR: a parallel multistage homopolymer spectrum based error corrector for 454 sequencing data. *BMC Bioinformatics*, **15**, 131.

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

Yang,X. *et al*. (2011) Repeat-aware modeling and correction of short read errors. *BMC Bioinformatics*, **12**(Suppl 1), S52.

Yang,X. *et al*. (2013) A survey of error-correction methods for next-generation sequencing. *Brief. Bioinformatics*, **14**, 56–66.

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