BIOINFORMATICS ORIGINAL PAPER Vol. 26 no. 20 2010, pages 2526–2533
doi:10.1093/bioinformatics/btq468

Sequence analysis

Advance Access publication August 16, 2010

Reptile: representative tiling for short read error correction

Xiao Yang[1,2], Karin S. Dorman[2,3,4] and Srinivas Aluru[1,2,4,5,*]

[1]Department of Electrical and Computer Engineering, [2]Bioinformatics and Computational Biology Program, [3]Department of Statistics, [4]Department of Genetics, Development & Cell Biology, Iowa State University, Ames IA 50011, USA and [5]Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai 400076, India

Associate Editor: Joaquin Dopazo

ABSTRACT

Motivation: Error correction is critical to the success of next-generation sequencing applications, such as resequencing and *de novo* genome sequencing. It is especially important for high-throughput short-read sequencing, where reads are much shorter and more abundant, and errors more frequent than in traditional Sanger sequencing. Processing massive numbers of short reads with existing error correction methods is both compute and memory intensive, yet the results are far from satisfactory when applied to real datasets.

Results: We present a novel approach, termed Reptile, for error correction in short-read data from next-generation sequencing. Reptile works with the spectrum of *k*-mers from the input reads, and corrects errors by simultaneously examining: (i) Hamming distance-based correction possibilities for potentially erroneous *k*-mers; and (ii) neighboring *k*-mers from the same read for correct contextual information. By not needing to store input data, Reptile has the favorable property that it can handle data that does not fit in main memory. In addition to sequence data, Reptile can make use of available quality score information. Our experiments show that Reptile outperforms previous methods in the percentage of errors removed from the data and the accuracy in true base assignment. In addition, a significant reduction in run time and memory usage have been achieved compared with previous methods, making it more practical for short-read error correction when sampling larger genomes.

Availability: Reptile is implemented in C++ and is available through the link: http://aluru-sun.ece.iastate.edu/doku.php?id=software

Contact: aluru@iastate.edu

Received on March 16, 2010; revised on August 9, 2010; accepted on August 10, 2010

1 INTRODUCTION

High-throughput sequencing is profoundly changing the way genetics data are collected, stored and processed (Shendure and Ji, 2008). The advantages of the new technology have led to revitalization of old techniques and discovery of novel uses, with growing applications in resequencing, *de novo* genome assembly, metagenomics and beyond (Ansorge, 2009; Marguerat *et al.*, 2008). New technology inevitably comes with challenges. For many next-generation sequencers, the advantage of deeper and cheaper coverage comes at the cost of shorter reads with higher error rates compared with the Sanger sequencing they replace.

Genome assembly, the *de novo* inference of a genome without the aid of a reference genome, is challenging. Sanger reads, typically 700–1000 bp in length, are long enough for overlaps to be reliable indicators of genomic co-location, which are used in the overlap-layout-consensus approach for genome assembly. However, this approach does poorly with the much shorter reads of many next-generation sequencing platforms (e.g. 35–100 bp for Illumina Genome Analyzer II). In this context, de Bruijn graph (Idury and Waterman, 1995) and string graph (Myers, 2005) based formulations that reconstruct the genome as a path in a graph perform better due to their more global analysis and ability to naturally accommodate paired read information. As a result, they have become *de facto* models for building short-read genome assemblers, e.g. ALLPATHS (Butler *et al.*, 2008), Velvet (Zerbino and Birney, 2008), ABySS (Simpson *et al.*, 2009) and Yaga (Jackson *et al.*, 2010). Error correction has long been recognized as a critical and difficult part of these graph-based assemblers. It also has significant impact in other next-generation sequencing applications such as resequencing.

We give a brief review of several well-known error correction methods. Alignment-based error correction methods, such as MisEd (Tammi *et al.*, 2003) for Sanger reads, require refined multiple read alignments and assume unusually isolated bases to be read errors. Like the Sanger-motivated assembly algorithms, these approaches do not adapt well to short reads. Hence, Chaisson *et al.* (2004) proposed the spectral alignment problem (SAP): in a given dataset, a *k*mer is considered *solid* if its multiplicity exceeds a threshold, and *insolid* otherwise. Reads containing insolid *k*mers are corrected using a minimum number of edit operations so that they contain only solid *k*mers post-correction. Similar approaches have been adapted and used by others (Butler *et al.*, 2008). To overcome the typically long run times of SAP-based approaches, Schröder *et al.* (2009) proposed SHREC, a method based on a generalized suffix tree constructed from short-read data using both forward and reverse complementary strands. SHREC compares the multiplicity of a substring, represented by a node in the suffix tree, with its expected frequency of occurrence calculated analytically, assuming uniform sampling of the genome and uniformly distributed sequencing errors. The nodes with observed counts that deviate beyond a tolerable threshold from their expected values are considered erroneous. An erroneous node is corrected to a sibling when applicable, and all its descendants are transferred to the selected sibling. Well-engineered code is necessary to cope with the large

*To whom correspondence should be addressed.

memory requirement of the suffix tree data structure. Unlike these general purpose error correction methods, FreClu (Qu *et al.*, 2009) targets transcriptome data. The error rates for each position of a read are estimated in the same experiment via a set of control reads of a known bacterial artificial chromosome (BAC) sequence. Reads are clustered using the estimated error rates, and after error correction, FreClu could map ~5% more reads back to the reference genome.

Error correction of short-read data is particularly challenging because of the massive datasets, non-uniformly distributed read errors introduced at relatively high rates, and non-uniform coverage of the target genome. Next-generation short-read sequencers produce hundreds of millions of reads in a single run, and this trend of fast, massive data generation is continuing to accelerate. To process these data, even an efficient linear space algorithm could easily exceed available memory on a standard desktop computer. The high rate of sequencing errors also significantly increases memory usage due to the introduction of numerous erroneous *k*mers that are not present in the genome. Errors are typically identified as the unusual reads or *k*mers occurring less frequently than a threshold calculated under the assumptions of uniform coverage and error distribution. Neither assumption holds true in real data, leading to an excess of false error predictions.

In this article, we present Reptile, a scalable short-read error correction method that effectively addresses the above challenges. We draw upon the *k*-spectrum approach pioneered in earlier results, but explore multiple alternative *k*mer decompositions of an erroneous read and use contextual information specified by neighboring *k*mers to infer appropriate corrections. Reptile also incorporates quality score information when available. We present algorithmic strategies to store *k*mer Hamming distance graphs and efficiently retrieve all graph neighbors of a *k*mer as candidates for correction. We compare our results with SHREC (Schröder *et al.*, 2009), a high-quality short-read error correction method, and one of the more recent in a line of continuously improving error correction protocols. In all experiments with Illumina datasets, Reptile outperforms SHREC in percentage of errors corrected and accuracy of true base assignment. Futhermore, a significant reduction in memory usage and run time makes Reptile more applicable to larger datasets. As with most current approaches including SHREC, Reptile is targeted to short reads with substitution errors, assuming insertion and deletion errors are rarely produced by short-read sequencing technology (Dohm *et al.*, 2008).

2 NOTATIONS

Let $R=\{r_1,r_2,\ldots,r_n\}$ be a collection of short reads sequenced from genome G. For simplicity (but without loss of generality), we assume each read r_i has a fixed length L. The coverage of the genome by the reads is given by $Cov=\frac{nL}{|G|}$, where $|G|$ denotes the genome length. Define the *k-spectrum* of a read r to be the set $r^k=\{r[i:i+k-1]\,|\,0\leq i<L-k+1\}$, where $r[i:j]$ denotes the substring from position i to j in r. We index the positions of a string starting from 0. The *k*-spectrum of R is given by $R^k=\bigcup_{i=1}^{n} r_i^k$.

Let α and β be two strings such that $\alpha[(|\alpha|-l):(|\alpha|-1)]=\beta[0:(l-1)]$ for some $0\leq l<\min(|\alpha|,|\beta|)$. We define the *l-concatenation* of α and β, denoted $\alpha||_l\beta$, to be the string γ of length $|\alpha|+|\beta|-l$ such that $\gamma[0:(|\alpha|-1)]=\alpha$ and $\gamma[(|\gamma|-|\beta|):(|\gamma|-1)]=\beta$.

The Hamming distance between two strings α_1 and α_2 for $|\alpha_1|=|\alpha_2|$, denoted $hd(\alpha_1,\alpha_2)$, is the number of positions at which they differ. For a *k*mer $\alpha_i\in R^k$, define its *d-neighborhood* $N_i^d=\{\alpha_j\in R^k\,|\,hd(\alpha_i,\alpha_j)\leq d\}$. Its *complete d-neighborhood* $N_{ci}^d=\{\alpha_j\,|\,hd(\alpha_i,\alpha_j)\leq d\}$ contains all *k*mers within Hamming distance d, whether or not they occur in R^k.

3 METHODS

The success of any error correction method relies on an adequate coverage of the target genome. If we know the genomic location of every read, we could layout all reads that contain a specific genomic position into a multiple alignment (Fig. 1) and correct all erroneous bases to the consensus base under the reasonable assumption that errors are infrequent and independent. For instance, base T in r_3 would be considered a sequencing error to be corrected to the consensus base A.

The main idea underlying Reptile is to create approximate multiple alignments, with the possibility of substitutions, in the absence of location information. Multiple alignments with substitutions could be created by considering all reads with pairwise Hamming distance less than some threshold, but such alignments are already hard (Manthey and Reischuk, 2005) and even in high coverage situations, the occurrence of many exactly coincident reads, e.g. r_0 and r_1 in Figure 1, are rare. We therefore resort to alignments on subreads, the substrings of a read.

Storing R, let alone all its subreads, could be memory intensive, not to mention the memory required to store information required for error correction. Inspired by the idea for bounding memory usage with de Bruijn graphs in short-read assembly, we work with *k*mer subreads of input data, where the memory of storing the *k*-spectrum R^k is bounded by $O(\min(4^k,n(L-k+1)))$. Typically, k is chosen so that the expected number of occurrences of any *k*mer in the genome should be no more than one, i.e. $4^k>|G|$. Therefore, choosing $10\leq k\leq 16$ is sufficient for microbial or human genomes, in which case the *k*-spectrum would fit within 4 GB RAM regardless of input size.

Focusing on reasonably short *k*mers has several advantages. First, we expect an adequate number of *k*mers to align to the same position along the genome even with relatively low coverage (e.g. 40×). High local coverage is needed to identify erroneous bases. For instance, in Figure 1, there exist five subreads, four copies of α_2 and one copy of α_2', aligning to the same starting position in the genome, but this number reduces to three for the longer subread $\alpha_2||_0\alpha_3$. Second, it is less compute intensive to identify N_i^d when k is small, since there are fewer ways to select d out of k positions. Last, sequencing errors in *k*mers are much less frequent compared with full-length reads, so d need not be large.

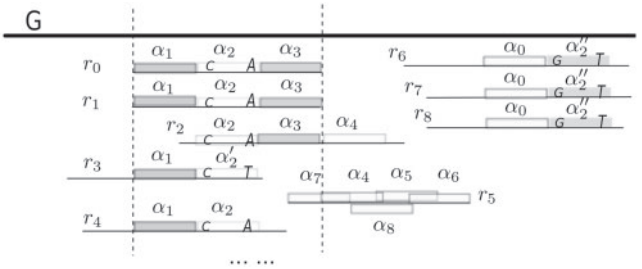

Fig. 1. G is the target genome, shown as a bold line; r_i's ($0\leq i\leq 8$) represent reads, shown as thin lines; α_j ($0\leq j\leq 8$), α_2' and α_2'' are *k*mer instances in the reads, shown as rectangles. Every read is drawn aligned to its origin of sequencing position on the target genome. The bases at two positions in the *k*mers α_2, α_2' and α_2'' are shown. All other positions in these *k*mers match across all three variants.

Nevertheless, relying solely on short $k$mers can easily lead to ambiguities when resolving erroneous bases. For instance, in Figure 1, without knowing the alignment, it is unclear if α_2' should be corrected to α_2 or α_2'', since both $hd(\alpha_2', \alpha_2) = hd(\alpha_2', \alpha_2'') = 1$ and α_2 and α_2'' have a similar higher frequency. However, the contexual information of α_2' available from read r_3 (in this case, α_1) uniquely identifies α_2 as the right correction. We seek a way to use contextual information to help resolve errors *without* increasing k and lowering local coverage.

Contextual information is provided by noting which $k$mers co-exist in observed reads. For instance, $\alpha_1 ||_0 \alpha_2$ exists in r_0, r_1 or r_4, and $\alpha_5 ||_{.5k} \alpha_6$ exists in r_5 in Figure 1. The following definitions formalize the notion of contextual information.

DEFINITION 1. *$t = \alpha_1 ||_l \alpha_2$ $(0 \le l < k)$ is a tile of read r if t is a substring of r, and $|\alpha_1| = |\alpha_2| = k$.*

DEFINITION 2. *$t' = \alpha' ||_l \beta'$ is a d-mutant tile of $t = \alpha ||_l \beta$ if $hd(\alpha, \alpha') \le d$ and $hd(\beta, \beta') \le d$.*

DEFINITION 3. *$T_r = (t_1, t_2, \ldots, t_m)$ is a tiling of read r if $r = t_1 ||_{l_1} t_2 \ldots ||_{l_{m-1}} t_m$ such that (1) t_i $(1 \le i \le m)$ is a tile of r, and (2) $l_i \ge 1$ $(1 \le i < m)$.*

Note that if t_i is specified as $r[j:j']$, then the overlaps between consecutive tiles can be inferred; i.e. l_i's can be derived from t_i's. Multiple tilings exist for any read. For example, both $((\alpha_7 ||_{.5k} \alpha_4), (\alpha_5 ||_{.5k} \alpha_6))$ and $((\alpha_7 ||_0 \alpha_8), (\alpha_8 ||_0 \alpha_6))$ are tilings of r_5. We also extend the concept of d-mutant tiles to tilings. For instance, we can think of $((\alpha_1 ||_0 \alpha_2'), (\alpha_2' ||_0 \alpha_3))$ as a one-mutant tiling of $T_{r_0} = ((\alpha_1 ||_0 \alpha_2), (\alpha_2 ||_0 \alpha_3))$. Similarly, $((\alpha_1 ||_0 \alpha_2'')(\alpha_2'' ||_0 \alpha_3))$ is a two-mutant tiling of T_{r_0}. Formally:

DEFINITION 4. *A d-mutant tiling of $T_r = (t_1, t_2, \ldots t_m)$ of read r is a sequence of tiles $(t_1', t_2', \ldots t_m')$ such that (1) $|t_i| = |t_i'|$ and $|t_i||_{l_i} t_{i+1}| = |t_i'||_{l_i} t_{i+1}'|$, where l_i is implicitly determined by r for $1 \le i < m$, and (2) t_i' is a d-mutant tile of t_i for $1 \le i \le m$.*

If read r contains errors and T_r is a tiling of r, then we expect to find a tiling T_s of the true read s as one of the d-mutant tilings of T_r, where constituent tiles of T_s have higher coverage than those of T_r. However, in some cases, T_s will not be found among the d-mutant tilings.

3.1 The algorithm

For ease of presentation, we assume R can fit in main memory (this requirement will be relaxed later). The algorithm consists of two major phases. We first provide a brief overview, and subsequently describe the algorithm in more detail and analyze its time and space complexity.

(1) Information extraction.
   (a) Derive the k-spectrum R^k of R.
   (b) Derive Hamming graph $G_H = (V_H, E_H)$, where $v_i \in V_H$ represents $\alpha_i \in R^k$ and $\exists\, e_{ij} = (v_i, v_j) \in E_H \iff hd(v_i, v_j) \le d$, for a given threshold d.
   (c) Compute tile occurrences.

(2) Individual read error correction.
   (a) Place an initial tile t at the beginning of the read.
   (b) Identify d-mutant tiles of t.
   (c) Correct errors in t as applicable.
   (d) Adjust tile t placement and go to step 2b, until tile placement choices are exhausted.

Given a read $r \in R$, any of its constituent $k$mers α is a vertex v in the Hamming graph. The d-neighborhood of α is accessible via the edges incident to v. Hence, if α contains at most d substitution sequencing errors, the $k$mer it should be corrected to exists in its d-neighborhood. By building local, approximate alignments of tilings constructed from d-neighborhoods, our strategy identifies a tiling of the true read as a high frequency tiling.

3.1.1 Phase 1: information extraction Constructing R^k involves one linear scan of each read in R. This takes $O(nL)$ time. We maintain R^k in sorted order using $O(|R^k| \log |R^k|)$ time. The space requirement for R^k is given by $|R^k| = O(\min(4^k, n(L-k+1)))$. Any non-*ACGT* characters (due to difficulty in base calling) are initially converted to *A*, which will be validated or corrected later by the algorithm.

During error correction, it is important to have fast access to the d-neighborhood of any $k$mer, ideally in constant time per neighboring $k$mer. One could do so by storing the entire Hamming graph G_H, but it would require large amount of memory. If we assume G as a random string, and errors accumulate independently with probability p_e, then the probability that a node is linked to another is $p_k = \sum_{e=1}^{d} \binom{k}{e} [0.25^{k-e} 0.75^e + (1-p_e)^{k-e} p_e^e]$, including the chance that another random $k$mer in the genome is within d Hamming distance of the current $k$mer and the chance that the current $k$mer contains up to d errors. Thus, the expected memory usage is $O\left(\binom{|R^k|}{2} p_k\right)$.

Alternatively, we could recover all edges associated with a given $k$mer α_i by checking whether each $k$mer in its complete neighborhood, $\alpha_j \in N_{ci}^d$, exists in R^k. If $\alpha_j \in R^k$, then there is an edge between v_i and v_j in the Hamming graph. This takes $O(\log |R^k|)$ time using binary search. There are $\binom{k}{d} 4^d$ possible candidate mutant $k$mers for each $\alpha_i \in R^k$, so it takes $O\left(\binom{k}{d} 4^d |R^k| \log |R^k|\right)$ time to identify all edges.

To reduce the average time for inferring N_i^d of α_i, we replicate R^k in memory and sort each replicate on a different subset of positions in the $k$mer string, using the following strategy:

(a) Store indices 0 to $k-1$ in a vector A.
(b) Divide A evenly into c $(d < c \le k)$ chunks, each of size $\lfloor k/c \rfloor$ or $\lceil k/c \rceil$.
(c) For every choice of $\binom{c}{d}$ chunks, sort R^k by masking the indices from selected chunks and store the sorted results separately.

To identify N_i^d of α_i, we query α_i against each sorted k-spectrum R_j^k $(0 \le j < \binom{c}{d})$ by binary search considering only indices used for sorting R_j^k. All $k$mers that belong to the d-neighborhood of α_i are adjacent to α_i in at least one R_j^k. If sequencing errors accumulate independently, then the expected number of elements of N_i^d found in every R_j^k is $h = |R^k| / 4^{k - d\lceil k/c \rceil}$. Hence, we need approximately $\binom{c}{d} h |R^k| \log |R^k|$ expected time to recover all edges of the Hamming graph, i.e. $O(|R^k| \log |R^k|)$ time assuming both $\binom{c}{d}$ and h are constants. Typically, $|R^k| << 4^k$, therefore, choosing a larger c value will use more memory, but less expected run time. As an example, in a real *Escherichia coli* dataset with 160× coverage, storing 13 copies of R^k required only ~560 MB memory, but the average number of hits per 13mer in each 13-spectrum was less than one. Therefore, identifying each element of N^1 for a 13mer took constant time on average.

The above method provides an exact solution for identifying all edges in the Hamming graph. Alternatively, a simpler recursive approximation derives N^d by inferring N^1 for every element in N^{d-1}. This stategy might be more biologically meaningful (Qu *et al.*, 2009), but is only an approximation since an edge between two vertices v_i and v_j could be recovered only if there exists a path connecting them such that adjacent vertices represent $k$mers that differ by exactly one position. In this case, choosing a smaller k, using a larger dataset, or having a higher sequencing error rate all improve the chance to identify all edges.

Tiles are l-concatenations of consecutive or overlapping $k$mers found in reads. Here, we use one fixed value of l but several different values of l can be used to consider tiles with different lengths. We compute the multiplicities of tiles by a linear scan of every read to record all tiles, followed by a sort of the collected tiles and one linear scan of the sorted list. This process takes $O(|R^{2k-l}| \log |R^{2k-l}|)$ time, where $|R^{2k-l}| = O(\min(4^{2k-l}, n(L-2k+l+1)))$. Meanwhile, we record the number of occurrences of each tile, where every position has a quality score exceeding some threshold Q_c. Typically, a quality score is associated with every base of a short read. The score indicates the probability p_e that the corresponding base is sequenced incorrectly.

For instance, Illumina GenomeAnalyzer encodes the quality score as $Q = -10\log_{10}(p_e/1-p_e)$. A higher score indicates a more reliable base call.

To deal with the double strandedness of the target genome, we consider both the forward and reverse complementary strands of every read. Edge identification in the Hamming graph takes twice the time, but no additional memory is needed since R^k is already generated using both strands.

3.1.2 Phase 2: error correction We use the contextual information in read r to identify sequencing errors through the process of choosing a tiling T_r and comparing it with its d-mutant tilings. In particular, if r contains x errors, and we choose any tiling T_r, then an error-free tiling T_s belongs to the collection of d-mutant tilings of T_r if $d \geq x$. Under the standard assumption of uniform coverage, the tiles of T_s should be substantially more abundant than at least some of the tiles of T_r with errors. After T_s is identified, the true read s can be readily inferred from T_s.

In practice, x could be large, and sequencing errors tend to cluster toward the 3′ end of a read. Since we prefer d to be small to limit memory usage, run time and false error detection, it is entirely possible that T_s is not one of the d-mutant tilings of T_r. On the other hand, an alternate tiling Γ_r of r may lower the maximum number of mutations per $k$mer to below d such that Γ_s with high frequency tilings *is* one of the d-mutant tilings of Γ_r. In the case that there is no such tiling Γ_r, we examine a subset of constituent tiles in Γ_r. If a high coverage path of these selected tiles is present, the tiles are corrected. With some errors removed, a tiling may now exist that contains the true read among its d-mutant tilings.

These observations are sufficient to motivate the following procedure for identifying and correcting read errors. Place a tile t on r and attempt to correct t via comparisons with its d-mutant tiles (tile correction). If t is validated or corrected, move to the next tile in the standard tiling and repeat. If t cannot be corrected or validated, look for an alternative tiling, presumably one that avoids clusters of more than d errors that are thwarting attempts to find error-free tiles within the d neighborhoods. We first describe tile correction in Algorithm 1, then the overall procedure for read correction in Algorithm 2.

Tile correction. For each tile in R, we have recorded its multiplicity O_c in R and the number O_g of those instances where the quality score of every base exceeds Q_c. If a short read dataset comes with unreliable or missing quality score information, we set $O_g = O_c$. Otherwise, O_g is a better estimate of the number of error-free occurrences of each tile.

The tile correction procedure is given in Algorithm 1. A decision to correct tile t is based on a comparison of the high-quality occurrence counts O_g of t compared with its d-mutant tiles. As a rule of thumb, there must be compelling evidence before a correction is made. Any tile is automatically validated if its occurrence count exceeds an upper threshold C_g (lines 1–2). A low occurrence tile with no d-mutant tiles is validated only if it occurs more than a low threshold C_m times (lines 4–6). When there exist d-mutant tiles of t with much higher frequencies (at least C_r times, $C_r > 1$) than t and which differ at low-quality bases ($< Q_m$) in t, it is likely that t contains error(s) and the true tile is one of its d-mutant tiles. In such cases, a correction is possible only if there is a unique d-mutant tile with the closest Hamming distance, including a mutation at a low-quality base in t (line 14). We will replace t with this tile. Otherwise, we choose not to modify t to avoid false corrections. Similar reasoning applies to t with very low multiplicity (lines 17 and 18). Since there exist a constant number of d-mutant tiles of t, and correcting t requires comparison of every base between t and $t' \in T$, Algorithm 1 takes constant time.

Read correction. The overall procedure for correcting a read is given in Algorithm 2. In line 1, an initial tile is chosen and d_1 and d_2, two parameters specifying the maximum Hamming distance allowed in the two constituent $k$mers while identifying mutant tiles, are initialized to d. The algorithm terminates when no more plausible tiles can be identified. Within the while loop, d-mutant tiles are identified for the current tile (line 3), which is validated or corrected using Algorithm 1 (line 4). The new placement of tile t_{next} is chosen according to which of the three decisions is made by Algorithm 1 (line 4). Repeated placement of t_{next} according to decisions, [D1] through [D3] (lines 6–8), gradually forms a validated or corrected tiling of read r, although some reads may never be fully validated.

Algorithm 1 Tile t error correction.

1. **if** $O_g(t) \geq C_g$ **then**
2. t is valid; *return*.
3. **end if**
4. **if** t has no d-mutant tiles t' **then**
5. **if** $O_g(t) \geq C_m$ **then**
6. t is valid; *return*.
7. **end if**
8. *return* due to insufficient evidence.
9. **end if**
10. **if** $O_g(t) \geq C_m$ **then**
11. Select d-mutant tiles $T = \{t' \mid \frac{O_g(t')}{O_g(t)} \geq C_r\}$.
12. If $T = \emptyset$, t is valid; *return*.
13. For every $t' \in T$, record those positions differed from t and corresponding quality scores.
14. Correct t to t' and *return* if 1) $hd(t,t') \leq hd(t,t'')$ for all $t'' \in T$. 2) at least one of the corrected bases has quality score less than Q_m in t.
15. If t' is not unique, *return* due to insufficient evidence (ambiguities).
16. **else**
17. **if** t has only one d-mutant tile t' s.t. $O_g(t') \geq C_m$ **then**
18. Correct t to t'; *return*.
19. **else**
20. *return* due to insufficient evidence.
21. **end if**
22. **end if**

To better understand how the rules in lines 6–8 choose a tiling, we illustrate with an example in Figure 2 for $d = 1$. An initial tile t_0 is chosen as shown. Since there exists one error in each of the constituent $k$mers, t_0 can be corrected. Tile t_1 is chosen as the next tile according to [D2], but two sequencing errors within the second $k$mer of t_1 lead to an inconclusive decision. Hence, t_1 is not selected in the tiling and an alternative tile τ_1 is chosen according to [D3(a)]. The algorithm iterates and if tiles can be validated or corrected at every stage, we are able to complete a tiling moving from 5′ to 3′ along the read. Unfortunately, non-uniform coverage and the existence of more than one reasonable d-mutant tile can lead to an inconclusive decision in Algorithm 1 regardless of tiling choice. In Figure 2, the 5′ to 3′ tiling encounters an inconclusive dead-end at arrow a_3. To move past dead-end tiles, a non-overlapping tile τ_2 is chosen by [D3(b)]. A small unvalidated gap is left in the middle of the 5′ to 3′ tiling of this read. The example shows only the tiling from 5′ to 3′. The same strategy is applied in the 3′ to 5′ direction.

We briefly analyze the run time of Algorithm 2. Tiles are sorted, so tile information is accessed in $O(\log(nL))$ time. Therefore, line 3 requires $O(\log(nL))$ time. Once some tiles have been corrected, the search space for new d-mutant tiles shrinks when d_1 or d_2 is set to 0 in line 5. The maximum number of non-overlapping tiles in a tiling is the constant $L/|t|$. Hence, the time spent in correcting each read is $O(\log(nL))$, and the overall run time of Algorithm 2 is $O(nL\log(nL))$.

3.1.3 Choosing parameters Although analytical calculations like those adopted in existing methods can be used to choose parameters of Algorithm 1, we choose their values based on the input data to help avoid the unrealistic assumptions of uniformly distributed read errors and uniform genome coverage. Given short-read data R, we examine the empirical distribution of quality scores and choose threshold Q_c such that a given percentage (e.g. 15–20%) of bases have quality score value below Q_c. This value could

Algorithm 2 Read error correction.

1. Initialize: $t \leftarrow t_0$, where t_0 is a prefix of r; $d_1, d_2 \leftarrow d$.
2. **while** $t \neq \emptyset$ **do**
3. Identify d-mutant tiles of $t = \alpha_1 ||_l \alpha_2$ as the set $\{t' = \alpha'_1 ||_l \alpha'_2 \,|\, (\alpha'_1, \alpha'_2) \in N_1^{d_1} \times N_2^{d_2}\}$.
4. Correct t (Algorithm 1).
5. Based on the decisions made on t in the former step, tile t_{next} of r will be chosen according to [D1]–[D3] as follows. If there is insufficient space to place a tile toward the end of a read r, t_{next} will be chosen as the suffix of r.
6. [D1] t is valid: select t_{next} such that the suffix–prefix overlap between t and t_{next} equals α_2; $d_1 \leftarrow 0$.
7. [D2] t was corrected to $t' = \alpha'_1 ||_l \alpha'_2$ and t is replaced with t' in r: select t_{next} such that the suffix–prefix overlap between t' and t_{next} equals α'_2; $d_1 \leftarrow 0$.
8. [D3] Insufficient evidence to correct t: set t_{next} to be one of the following. Let $r[i_1 : i_2]$ be a maximal validated or corrected region of r that overlaps with the 5′ region of t, where $i_2 - i_1 + 1 \geq |t|$, and $r[i_2 + 1 : i_3]$ be a maximal uncorrected region from previous iterations due to insufficient evidence.
   a. If $r[i_2 + 1 : i_3] = \emptyset$, then $t_{\text{next}} \leftarrow r[i_2 - |t| + 2, i_2 + 1]$ and $d_1 \leftarrow 1$.
   b. If $r[i_2 + 1 : i_3] \neq \emptyset$, then $t_{\text{next}} \leftarrow r[i_3 + 1, i_3 + |t|]$.

   Similarly, if $r[i_1 : i_2]$ overlaps with 3′ end of t and $r[i_3 : i_1 - 1]$ is a maximal uncorrected region from previous iterations, then if $r[i_3 : i_1 - 1] = \emptyset$, set $t_{\text{next}} \leftarrow r[i_1 - 1 : |t| + i_1 - 2]$ and $d_2 \leftarrow 1$. In above cases, t_{next} is valid only if it was not assigned with the same value in previous iterations (unless t_{next} became the suffix of r). If such a choice does not exist, $t_{\text{next}} \leftarrow \emptyset$.
9. $t \leftarrow t_{\text{next}}$.
10. **end while**

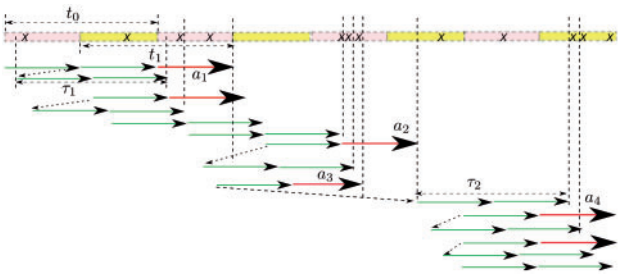

Fig. 2. An illustration of choosing a tiling of a read for $d = 1$. A read is represented on top by a concatenation of rectangles, where each rectangle denotes a $k$mer. Each tile is represented by a concatenation of two adjacent arrows, which denote its $k$mer composition. For simplicity of illustration, we choose the read length to be divisible by k and each tile is a 0-concatenation of two adjacent $k$mers. X's denote sequencing errors. Each bold arrow, a_i ($1 \leq i \leq 4$), denotes tile with insufficient evidence for correction. The placement of an alternative tile is indicated by a dotted arrow.

be adjusted to consider the error rates of the particular next-generation sequencing equipment in use. Given Q_c and counts of high-quality tile ocurrences, we choose C_g so that only a small percentage (e.g. 1–3%) of tiles have high-quality multiplicity greater than C_g. C_m is chosen so that a larger percentage (e.g. 4–6%) of tiles occur more than C_m times in R. As C_m value decreases, more errors are corrected at the cost of an increased risk of false error correction. The specific values chosen depend on the histogram of tile occurrences. By default, we set $C_r = 2$ such that a low frequency tile could only be corrected to a tile with at least twice the frequency. Increasing C_r improves the confidence in error correction. Finally, we choose $k = \lceil \log_4 |G| \rceil$ when an estimate of the length of the genome is available, otherwise, a number between 10 and 16 should work. Tile size is $\sim 2k$, so $k$mer overlap is 0 to a few bases. The maximum Hamming distance d is set to one by default. But when k is chosen to be relatively large (e.g. 14 to 16), increasing d allows us to identify more sequencing errors but incurs a longer run time and increases the risk of false error prediction.

3.1.4 Overall complexity Combining the analysis for each step, the overall run time is $O(nL\log(nL))$ and the space usage is $O(|R^k| + |R^{|t|}|)$.

When the collection of input short reads R does not fit in main memory, we propose a divide and merge strategy where R is partitioned into chunks small enough to occupy just a portion of main memory. For each chunk, we stream through each read and record the k-spectrum and tile information, merging it with the data from previous chunks. Reads need not be stored in memory after they have been processed. A similar strategy is applied for error correction: R is reloaded into memory in chunks, tilings and d-mutant tilings are inferred for each read, and errors are corrected as warranted.

4 RESULTS

We evaluated Reptile on several Illumina/Solexa datasets and compared the results with SHREC (Schröder *et al.*, 2009) version 2.0, a recent high-quality short-read error correction method that is itself shown to give superior results over prior k-spectrum approaches. We omitted evaluation on simulated data because simulations with random errors or synthetic genomes do not accurately reflect actual short-read sequencing errors (Dohm *et al.*, 2008), and could even be misleading. Our test datasets are Illumina-generated short reads of well-characterized, Sanger assembled bacterial genomes. Knowledge of the genomes is needed for determining the accuracy of the error correction methods. The six experimental datasets, downloaded from the sequence read archive at NCBI, are listed in Table 1. Datasets $D1$ (Accession Number: SRX000429), $D2$ (SRR001665_1), $D5$ (SRR022918_1) and $D6$ (SRR034509_1) are Illumina reads from the *E.coli str. K-12 substr* (NC_000913) genome (~4.64 Mbp); datasets $D3$ (SRR006332) and $D4$ are Illumina reads from the *Acinetobacter sp. ADP1* (NC_005966) genome (~3.6 Mb). The first four datasets are generated by Solexa 1G Genome Analyzer, where each read has the same length 36 bp. The latter two datasets are generated using the more recent Illumina Genome Analyzer II, with read lengths of 47 bp in $D5$ and 101 bp in $D6$. $D1$ has high coverage and low error rate. $D2$ has typical coverage and low error rate. $D3$ has high coverage and high error rate. $D4$ is derived from $D3$ by randomly selecting short reads amounting to 40× coverage. This is done for evaluating performance on a low coverage, high error rate dataset. Both $D5$ and $D6$ have higher error rates. In addition, >13.9% of the reads in $D6$ contain ambiguous nucleotides, denoted by character N. Since SHREC cannot process non-*ACGT* characters, we eliminated all reads with ambiguous bases, even though Reptile has no such limitation. The number of discarded reads is indicated in column 5, Table 1.

Similar to Schröder *et al.* (2009), we evaluated error correction results with the aid of RMAP (v2.05) (Smith *et al.*, 2008), which maps short reads to a known genome by minimizing mismatches. We allowed up to five mismatches per read in the first four datasets

Table 1. Experimental datasets

Data	Genome	Read length (bp)	Number of reads (M)	Discarded reads	Cov.	Error rate (%)
*D*1	*E.coli*	36	20.8	107.7 K	160×	0.6
*D*2	*E.coli*	36	10.4	48.3 K	80×	0.6
*D*3	*A. sp.*	36	17.7	456 K	173×	1.5
*D*4	*A. sp.*	36	4.0	0	40×	1.5
*D*5	*E.coli*	47	7.0	32.7 K	71×	3.3
*D*6	*E.coli*	101	8.9	1.44 M	193×	2.2

Error rate is estimated by mapping the reads to the corresponding genome using RMAP, and finding mismatches based on uniquely mapped reads.

Table 2. Results of mapping each dataset to the corresponding genome using RMAP

Data	Allowed mismatches	Number of reads[a]	Uniquely mapped reads (%)	Ambiguously mapped reads (%)
*D*1	5	20708709	96.5	2.5
*D*2	5	10359952	96.7	2.5
*D*3	5	17675271	79.9	1.5
*D*4	5	4000000	84.1	1.6
*D*5	10	7049153	62.5	1.5
*D*6	10	8874761	63.5	1.2
	15		68.8	1.4

[a]Number of reads containing no ambiguous bases.

and allowed up to 10 mismatches (default value of RMAP) in *D*5 and fifteen mismatches in *D*6 since the reads are longer in the latter two datasets. Reads that could not be mapped to the genome, or that map to multiple locations, are discarded. The mismatches between uniquely mapped reads and the genome are considered read errors. Quality of the datasets varied as shown in Table 2, with the percentage of reads that are uniquely mapped ranging from 62.5 to 96.7%. The large percentage of unmappable reads, the higher error rates as well as the large percentage of reads with ambiguous bases indicate that *D*5 and *D*6 have lower quality than *D*1 to *D*4.

Since the goal of error correction is to identify and correct each erroneous nucleotide, we assess the quality of error correction at the base level. A true positive (TP) is any erroneous base that is changed to the true base, a false positive (FP) is any true base changed wrongly, a true negative (TN) is any true base left unchanged, and a false negative (FN) is any erroneous base left unchanged. Then Sensitivity = TP/(TP + FN) and Specificity = TN/(TN + FP). Note that these definitions are different from those used by Schröder *et al.* (2009), which target read-level error detection (whether a read is flagged as containing an error or not). This is a less stringent measure because any read containing errors was classified as TP provided at least one of its errors was detected and irrespective of whether they were accurately corrected or not.

We propose two additional measures for assessing the quality of error correction:

- *Erroneous base assignment* (EBA): let n_e denote the number of erroneous bases that are correctly identified but changed to a wrong base. Then, EBA $= n_e/(\text{TP}+n_e)$ reflects how well we are able to correct an erroneous base to the true base after a sequencing error has been identified. A lower value of EBA indicates a more accurate base assignment.
- *Gain*: (TP − FP)/(TP + FN). This measures the percentage of errors effectively removed from the dataset, which is equivalent to the number of errors before correction minus the number of errors after correction divided by the number of errors before correction. Clearly, Gain should approach one for the best methods, but may be negative for methods that actually introduce more errors than they correct.

We regard these measures as important because they penalize failing to detect an erroneous base, correctly detecting an erroneous base but wrongly correcting it, and characterizing a correct base to be an erroneous base. In particular, we strongly advocate the Gain measure as it captures data quality post-error correction compared with the quality prior to the correction.

The results of running Reptile and SHREC on the six datasets are summarized in Table 3. Due to the larger memory usage of the SHREC program, we were not able to obtain results for *D*3, *D*5 and *D*6. In all other cases, Reptile had higher Gain and lower EBA than SHREC. With other parameters fixed in Reptile, we varied maximum d value used for inferring Hamming graph in *D*1 and *D*2. As expected, the run time significantly increased as d increased, since the size of d-neighborhood for each $k$mer increased. Also, we see an increase in both TP and FP and four to five times higher EBA, indicating that when we increase the search space, we run the risk of false error detection and correction but increase the chance to identifying more errors.

An inherent difficulty in using any method is the challenge of choosing optimal parameters. The results reported in Table 3 are obtained when using the parameter choices suggested in Section 3. To show that even better performance is possible, we applied a series of parameter choices to dataset *D*3 (Fig. 3). Gain improved from 63% with the default parameters to as high as 72%. We chose to report on Reptile using the default parameters for all cases in Table 3 because it is unfair to choose optimal parameters for each individual case based on our knowledge of the genome, which would generally not be known. Similarly, we used the default parameter settings for SHREC. Using a different combination of parameters may vary the results of both SHREC and Reptile. In this article, we have presented a method to select parameters for Reptile based on known quantities such as $k$mer frequency and quality score histograms. A similar guidance is needed for the SHREC program and is beyond the scope of the article. Note that we do not take into account improved results that can only be obtained by the knowledge of the genome (Fig. 3). One can observe that our method of parameter estimation based on statistics from the dataset is performing better than analytical calculations based on the assumptions of uniform error distribution and uniform coverage of genome by reads.

In addition, we compared the run time and memory usage of SHREC and Reptile. SHREC is a multi-threaded program while the current release of Reptile can only use a single core. Hence, we report run times in total CPU hours. Both methods were run on a SUN Fire X2200 workstation with dual quad-core 2.3 GHz AMD Barcelona three processors with 8 GB RAM and 4 GB swap memory, running Debian GNU/Linux x86_64. Results in Table 3 show Reptile is 3–10 times faster and uses 8–11 times less memory than SHREC. As expected, memory usage of Reptile is associated with the length of the genome and the number of errors in the data

Table 3. Results of Reptile and SHREC on Illumina sequenced short reads

Data (Cov)	Method (d)	TP	FN	FP	TN	EBA (%)	Sensitivity (%)	Specificity (%)	Gain (%)	CPU Hours	Memory (GB)
$D1$	SHREC	2819754	1183861	667435	740 842 474	1.794	70.4	99.9	53.8	-	> 8
(160x)	Reptile (1)	3164394	839221	133558	741 376 351	0.007	79	99.9	75.7	0.79	1.1
	Reptile (2)	3457717	545898	245417	741 264 492	0.028	86.4	99.9	80.2	2.49	1.1
$D2$	SHREC	1303505	422337	251228	370 981 202	1.549	75.5	99.9	61.0	3.6	7.1
(79.5x)	Reptile (1)	1169256	556586	44959	371 187 471	0.009	67.8	99.9	65.2	0.35	0.84
	Reptile (2)	1315277	410565	91205	371 141 225	0.042	76.2	99.9	70.9	1.23	0.84
$D3$	SHREC	–	–	–	–	–	–	–	–	–	–
(172.5x)	Reptile (1)	7138883	2361813	1138666	610 144 638	0.013	75.1	99.8	63.2	1.66	2.2
$D4$	SHREC	1473252	530736	613921	141 382 091	1.306	73.5	99.6	42.9	2.78	7.6
(40x)	Reptile (1)	1422949	581039	222218	141 773 794	0.091	71	99.8	59.9	0.26	0.66
$D5$	SHREC	–	–	–	–	–	–	–	–	–	–
(71x)	Reptile (1)	3551764	3189748	985674	323 583 005	0.017	52.7	99.7	38.1	0.94	1.9
$D6$	SHREC	–	–	–	–	–	–	–	–	–	–
(193x)	Reptile (1)	17158925	2947342	1298891	874 945 703	0.01	85.3	99.9	78.9	2.76	4.6

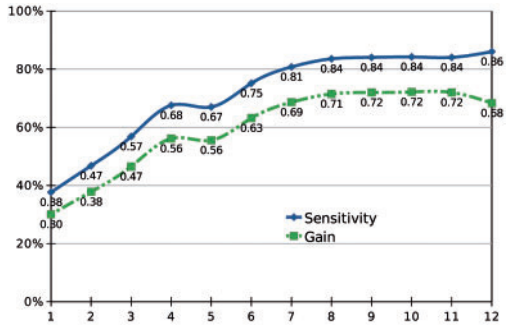

Fig. 3. Gain and sensitivity versus different parameter choices for $D3$. The first 11 sample points use parameters $k=11, d=1, |t|=22$, and (C_m, Q_c) values (14, 60), (12, 60), (10, 60), (10, 55), (8, 60), (8, 55), (8, 50), (8, 45), (7, 45), (6, 45), (5, 45), respectively. The last sample point uses parameters $(k=12, d=2, |t|=24, C_m=8, Q_c=45)$.

(compare $D1$ and $D2$), while the memory usage of SHREC increased with the number of reads, genome length and sequencing errors. In addition, although $D1$ contains many more reads (20.8M) than $D3$ (17.7M), the higher error rate significantly increased memory usage in both methods.

To enable fair comparison with SHREC, the above experiments were carried out by excluding all reads containing ambiguous bases. However, Reptile does have functionality to deal with ambiguous bases, which is useful in the following cases: (i) if a read contains few ambiguous bases, the surrounding high-quality regions provide sufficient information to infer correct bases by referencing the k-spectrum; (ii) in some datasets, neglecting reads containing ambiguous bases leads to excessive loss of data, which further distorts uniformity of the sampling. For instance, as much as 13.9% reads in dataset $D6$ contain N's.

Reptile attempts to correct ambiguous bases in regions where their density is low. If a read contains too many ambiguous bases, it is low quality and untrustworthy. Some reads may have ambiguous bases clustered in some region, e.g. the $3'$ region, while other parts may still be of good quality. It is more meaningful to try correcting ambiguous bases in the latter parts alone, since a cluster of ambiguous nucleotides in a read makes it unlikely to pinpoint other reads that have the same genomic co-location. Formally, Reptile attempts to correct an ambiguous base b of read r, if in any substring $r[i:i+w-1]$ that contains b, there are no more than d ambiguous bases. The ratio of d to w constrains the maximum density of ambiguous bases allowed in attempting error correction. By default, w is set to k (to equal $k$mer length), while d is set to the maximum Hamming distance allowed (Section 3). To implement the above idea, all ambiguous bases satisfying the density constraint are changed to one of the bases from the set {A, C, G, T} initially (default 'A'), and will be validated or corrected later by the algorithm.

Table 4. Quality of ambiguous base correction using Reptile

Data	N	Accuracy (%)	Sensitivity (%)	Specificity (%)	Gain (%)	EBA (%)
$D2$	A	99.98	66.4	100	63.7	0.01
	C	100	66.4	100	63.8	0.01
	G	100	66.4	100	63.7	0.01
	T	100	66.3	100	63.7	0.01
$D6$	A	99.99	85.1	99.8	78.5	0.01
	C	99.99	85.2	99.8	78.4	0.009
	G	100	85.3	99.8	78.5	0.01
	T	99.99	85.3	99.8	78.5	0.009

To test the accuracy of this procedure as well as study the effect of the choice of the default base, we conducted Reptile runs on the full datasets of $D2$ (36 bp) and $D6$ (101 bp) by setting the ambiguous bases to the chosen default. The results are presented in Table 4. The default base used is shown under Column 'N'. *Accuracy* is defined to be the percentage of ambiguous bases that have been successfully corrected (again, only reads uniquely mapped by RMAP are considered as truth is unknown otherwise). The last four

columns are as defined in Table 3. As can be observed from Table 4, (i) the accuracy of ambiguous base correction is high and consistent with the overall EBA rate, (ii) changing the default base slightly influenced the results due to the resulting differences in k-spectrum composition and (iii) the sensitivity and Gain values are slightly lower than reported in Table 3, mainly because the ambiguous bases that were left uncorrected by Reptile could sometimes be uniquely mapped to the reference genome using RMAP, hence increasing the FN value.

5 DISCUSSION

The proposed error correction algorithm is conservative because it avoids changing bases unless there is a compelling under-representation of a tile compared with its d-mutant tiles. Actual errors in read r cannot be corrected if r occurs in a very low coverage region of the genome or there exist multiple candidate d-mutant tiles, probably because of genome repetition. On the other hand, a tile may be miscorrected if it contains a minor variant of a highly repetitive element in the genome or it traverses a low coverage region that is similar to other regions with normal coverage. Our method is not unique in being challenged by non-uniform coverage on repetitive genomes. Error correction for highly repetitive genomes is essential for successfully assembling larger eukaryotic genomes but none of the existing methods successfully addresses this problem, including Reptile.

Short-read mapping provides a reasonable method to evaluate error correction methods in well-assembled, low repetition genomes. Nevertheless, it is not possible to unambiguously determine all errors. There are natural polymorphisms among bacterial lines, and some presumed polymorphisms may be unrecognized assembly errors. Furthermore, the mapping software chooses among alternative mappings by invoking parsimony, but there is some chance that the true number of errors is less than the minimum. Lastly, mapping software cannot map reads that contain more than a constant number of substitutions, typically just two, with full sensitivity, although we considered 5 here and tested as many as 15 with similar results. Despite these limitations, we believe that most errors are correctly identified, and this approach can provide a fair comparison of error correction methods.

We and others (Smith *et al.*, 2008) have found that sequence quality scores provide valuable information. Our use of quality scores probably helped us account for the error patterns in next-generation sequencing data (Dohm *et al.*, 2008) without explicitly modeling them. However, it has been observed (Dohm *et al.*, 2008) that high quality scores may be too optimistic and low quality scores too pessimistic in estimating sequencing errors in Solexa data. Since quality scores may not be precise measures of misread probabilities, the current version of Reptile uses quality score information in a very simple manner, but can be modified to make more sophisticated use of quality scores if warranted. Finally, although quality scores are needed to run Reptile, it can be run effectively without scores by setting all quality scores and the threshold Q_c to the same value.

There remain several additional challenges in next-generation sequencing error correction. One challenge is to distinguish errors from polymorphisms, for example, single nucleotide polymorphisms (SNPs). Reptile could accommodate SNP prediction with modification in the tile correction stage (Algorithm 1), where ambiguities may indicate polymorphisms. Another challenge is the growing read length of upcoming high-throughput sequencers. Currently, we define tiles as concatenations of two $k$mers. it might prove useful to extend the tile definition to more than two $k$mers in order to address error correction in much longer reads.

Funding: This research is funded in part by the Iowa State University Plant Sciences Institute Innovative Research Grants program.

Conflict of Interest: none declared.

REFERENCES

Ansorge,W.J. (2009) Next-generation DNA sequencing techniques. *Nat. Biotechnol.*, **25**, 195–203.

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

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

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

Idury,R.M. and Waterman,M.S. (1995) A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, **2**, 291–306.

Jackson,B.G. *et al*. (2010) Parallel de novo assembly of large genomes from high-throughput short reads. In *24th IEEE International Parallel & Distributed Processing Symposium*, Atlanta, USA, pp. 1–10.

Manthey,B. and Reischuk,R. (2005). The intractability of computing the Hamming distance. *Theor. Comput. Sci.*, **337**, 331–346.

Marguerat,S. *et al*. (2008) Next-generation sequencing: applications beyond genomes. *Biochem. Soc. Trans.*, **36**, 1091–1096.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinformatics*, **21** (Suppl. 2), ii79–ii85.

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.

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

Shendure,J. and Ji,H. (2008) Next-generation DNA sequencing. *Nat. Biotechnol.*, **26**, 1135–1145.

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

Smith,A. *et al*. (2008) Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, **9**, 128–135.

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

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