BIOINFORMATICS ORIGINAL PAPER Vol. 30 no. 10 2014, pages 1354–1362
doi:10.1093/bioinformatics/btu030

Genome analysis

Advance Access publication January 21, 2014

BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads

Yun Heo[1], Xiao-Long Wu[1], Deming Chen[1,*], Jian Ma[2,3] and Wen-Mei Hwu[1]

[1]Department of Electrical and Computer Engineering, [2]Department of Bioengineering and [3]Institute for Genomic Biology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Associate Editor: Michael Brudno

ABSTRACT

Motivation: Rapid advances in next-generation sequencing (NGS) technology have led to exponential increase in the amount of genomic information. However, NGS reads contain far more errors than data from traditional sequencing methods, and downstream genomic analysis results can be improved by correcting the errors. Unfortunately, all the previous error correction methods required a large amount of memory, making it unsuitable to process reads from large genomes with commodity computers.

Results: We present a novel algorithm that produces accurate correction results with much less memory compared with previous solutions. The algorithm, named BLoom-filter-based Error correction Solution for high-throughput Sequencing reads (BLESS), uses a single minimum-sized Bloom filter, and is also able to tolerate a higher false-positive rate, thus allowing us to correct errors with a 40× memory usage reduction on average compared with previous methods. Meanwhile, BLESS can extend reads like DNA assemblers to correct errors at the end of reads. Evaluations using real and simulated reads showed that BLESS could generate more accurate results than existing solutions. After errors were corrected using BLESS, 69% of initially unaligned reads could be aligned correctly. Additionally, *de novo* assembly results became 50% longer with 66% fewer assembly errors.

Availability and implementation: Freely available at http://sourceforge.net/p/bless-ec

Contact: dchen@illinois.edu

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

Received on October 29, 2013; revised on December 16, 2013; accepted on January 14, 2014

1 INTRODUCTION

Recent advances in next-generation sequencing (NGS) technologies have made it possible to rapidly generate high-throughput data at a much lower cost than traditional Sanger sequencing technology (Metzker, 2009). NGS technologies enable cost-efficient genomic applications, including *de novo* assembly of many non-model organisms (Haussler *et al.*, 2009), identifying functional elements in genomes (Frazer, 2012), and finding variations within a population (Beerenwinkel and Zagordi, 2011; Durbin *et al.*, 2010; Prosperi *et al.*, 2013; Schirmer *et al.*, 2012). In addition to short read length, a main challenge in analyzing NGS data is its higher error rate than traditional sequencing technology (Loman *et al.*, 2012; Wang *et al.*, 2012), and it has been demonstrated that error correction can improve the quality of genome assembly (Salzberg *et al.*, 2012) and population genomics analysis (Jiang *et al.*, 2009; Schirmer *et al.*, 2012).

Previous error correction methods can be divided into four major categories (Yang *et al.*, 2012): (i) *k*-mer spectrum-based (Chaisson *et al.*, 2009; Dohm *et al.*, 2008; Kelley *et al.*, 2010; Li *et al.*, 2010; Liu *et al.*, 2013; Medvedev *et al.*, 2011; Pevzner *et al.*, 2001; Qu *et al.*, 2009; Shah *et al.*, 2012; Wijaya *et al.*, 2009; Yang *et al.*, 2011, 2010), (ii) suffix tree-/array-based (Ilie *et al.*, 2011; Salmela, 2010; Schröder *et al.*, 2009; Zhao *et al.*, 2011a, b), (iii) multiple sequence alignment (MSA)-based (Kao *et al.*, 2011; Salmela and Schröder, 2011) and (iv) hidden Markov model (HMM)-based (Le *et al.*, 2013; Yin *et al.*, 2013). However, none of these previous methods has successfully corrected errors in NGS reads from large genomes without consuming a large amount of memory that is not accessible to most researchers (see detailed discussions in the Supplementary Methods). Previous evaluations showed that some error correction tools require >128 gigabyte (GB) of memory to correct errors in genomes with 120 Mb, and the others need tens of GB of memory (Yang *et al.*, 2012). For a human genome, previous approaches would need hundreds of GB of memory. Even if a computer with hundreds of GB of memory is available, running such memory-hungry tools degrades the efficiency of the computer. While the error correction tool runs, we cannot do any other job using the computer if most of the memory is occupied by the error correction tool. This can be a critical problem for data centers, where a large amount of data should be processed in parallel.

In several works, Bloom filters (Bloom, 1970) or counting Bloom filters (Fan *et al.*, 2000) were used to save a *k*-mer spectrum, which includes all the strings of length *k* (i.e. *k*-mers) that exist more than a certain number of times in reads (Liu *et al.*, 2011; Shi *et al.*, 2009; Shi *et al.*, 2010a, b). Although Bloom filter is a memory-efficient data structure, the memory reduction by previous Bloom filter-based methods did not reach their maximum potential because of the following four reasons: (i) The size of a Bloom filter should be proportional to the number of distinct *k*-mers in reads, and the number of distinct *k*-mers was conservatively estimated, thus could be much higher than the actual number. (ii) They could not remove the effect of false positives from Bloom filters. To make the false-positive rate of the Bloom filters small, the size of Bloom filters were made large. (iii) Because they could not distinguish error-free *k*-mers from erroneous ones before a Bloom filter was constructed, both of the *k*-mers needed to be saved in Bloom filters. (iv) Multiple

*To whom correspondence should be addressed.

Bloom filters (or counting Bloom filters) were needed to count the multiplicity of each k-mer.

Besides the large memory consumption of the existing methods, another problem encountered during the error correction process is that there exist many identical or very similar subsequences in a genome (i.e. repeats). Because of these repeats, an erroneous subsequence can sometimes be converted to multiple error-free subsequences, making it difficult to determine the right choice.

In this article, we present a new Bloom filter-based error correction algorithm, called BLESS. BLESS belongs to the k-mer spectrum-based method, but it is designed to remove the aforementioned limitations that previous k-mer spectrum-based solutions had. Our new approach has three important new features:

(1) BLESS is designed to target high memory efficiency for error correction to be run on a commodity computer. The k-mers that exist more than a certain number of times in reads are sorted out and programmed into a Bloom filter.

(2) BLESS can handle repeats in genomes better than previous k-mer spectrum-based methods, which leads to higher accuracy. This is because BLESS is able to use longer k-mers compared with previous methods. Longer k-mers resolve repeats better.

(3) BLESS can extend reads to correct errors at the end of reads as accurately as other parts of the reads. Sometimes an erroneous k-mer may be identified as an error-free one because of an irregularly large multiplicity of the k-mer. False positives from the Bloom filter can also cause the same problem. BLESS extends the reads to find multiple k-mers that cover the erroneous bases at the end of the reads to improve error correction at the end of the reads.

To identify erroneous k-mers in reads, we need to count the multiplicity of each k-mer. Counting k-mers without extensive memory is challenging (Deorowicz *et al*., 2013; Marçais and Kingsford, 2011; Melsted and Pritchard, 2011; Rizk *et al*., 2013; Roy *et al*., 2013). BLESS uses the disk-based k-mer counting algorithm like Disk Streaming of k-mers (DSK) (Rizk *et al*., 2013) and k-mer Counter (KMC) (Deorowicz *et al*., 2013). However, BLESS needs to save only half of the k-mers that DSK does in hash tables, because it does not distinguish a k-mer and its reverse complement.

To evaluate the performance of BLESS, this study used real NGS reads generated with the Illumina technology as well as simulated reads. These reads were corrected using BLESS as well as six previously published methods. Our results show that the accuracy of BLESS is the best while it only consumes 2.5% of the memory usage of all the compared methods on average. Our results further show that correcting errors using BLESS allowed us to align 69% of previously unaligned reads to the reference genome accurately. BLESS also increased NG50 of scaffolds by 50% and decreased assembly errors by 66% based on the results from Velvet (Zerbino and Birney, 2008).

2 METHODS

2.1 Overview of the BLESS algorithm

BLESS belongs to the k-mer spectrum-based error correction category (Pevzner *et al*., 2001). A k-mer is called solid if it exists more than M, the k-mers multiplicity threshold, times in the entire reads, and weak otherwise. If a k-mer extracted from a read is a weak k-mer, it can be considered as having sequencing errors.

Figure 1 depicts the high-level diagram of BLESS. To convert weak k-mers to solid k-mers, we need to save the list of the solid k-mers and to query a k-mer to the list efficiently. In Step 1, k-mers in reads are distributed into multiple files, and the multiplicity of k-mers in each file is counted. In Step 2, only solid k-mers are programmed into a Bloom filter, and errors in reads are corrected using the Bloom filter. Finally, in Step 3, BLESS restores the false corrections made by the false positives from the Bloom filter.

2.2 Step 1: Counting the multiplicity of k-mers

The first step in BLESS is to count the multiplicity of each k-mer, followed by finding the solid k-mers, and programming those solid k-mers into a Bloom filter. By counting the multiplicity of k-mers, we can sort out the solid k-mers that are needed for further analysis. We can also create a k-mer multiplicity histogram to be used to determine the multiplicity threshold M, if M is not predetermined by the user. The total number of solid k-mers is used to determine the size of the Bloom filter.

Supplementary Figure S1 in the Supplementary Document shows how to count the multiplicity of each k-mer. First, all the k-mers in the reads are distributed into N (default 100) files to reduce the required memory for this process. In BLESS, a k-mer and its reverse complement are treated as the same k-mer, which is called a canonical k-mer. If the middle base of a k-mer is A or C (k is always an odd number), the k-mer can be used as a canonical k-mer of itself. If the middle base is G or T, the reverse complement of the k-mer becomes the canonical k-mer of the original k-mer. A hash value is calculated for each canonical k-mer, and the file that the k-mer will be written into is determined by using the hash value. Next, the k-mer is written to the file. After this process, all the identical k-mers and their reverse complements are written into the same file.

The next step is to open each file that contains k-mers and count the number of k-mers using a hash table. After all the k-mers in the file are updated in the hash table, we check the multiplicity of each k-mer in the hash table. If the multiplicity of a particular k-mer is larger than M, it is a solid k-mer and is subsequently written to the solid k-mer list file F_s.

If M is not given by the user, the k-mer multiplicity histogram is generated, and M is determined using the histogram. The process of determining M using the k-mer multiplicity histogram is explained later (see Section 2.5). After completing this process for all the N files, we can create the solid k-mer list file F_s and determine the number of distinct solid k-mers N_s. The time complexity of counting the multiplicity of k-mers is $O(RL)$, where R is the number of reads and L is the read length.

2.3 Step 2: Correcting errors using a Bloom filter

To convert weak k-mers into solid k-mers, we must know the solid k-mer list. If this list was stored in file F_s, it would be impossible to rapidly check whether a k-mer is in the list or not. BLESS solves this problem by recording all the solid k-mers in a Bloom filter, which supports fast membership test while using little memory. An open source C++ Bloom filter library (http://www.partow.net/downloads/OpenBloomFilter.zip) is used in BLESS. When implemented, the size of the bit vector and the number of hash functions in the Bloom filter are determined using N_s and a target false positive probability. After constructing the Bloom filter, all the solid k-mers in F_s are programmed into the Bloom filter. The weak k-mers are then converted into solid ones using this Bloom filter.

Let read r be a sequence of symbols {A, C, G, T} with length L. The i-th base of read r is denoted by $r[i]$, where $0 \leq i \leq L-1$. The form $r[i, j]$ is a substring from the i-th base to the j-th base of r. The pseudo code of the correction process for a read r is shown in Supplementary Figure S2. This process is initiated from finding all the solid k-mer islands in r. A

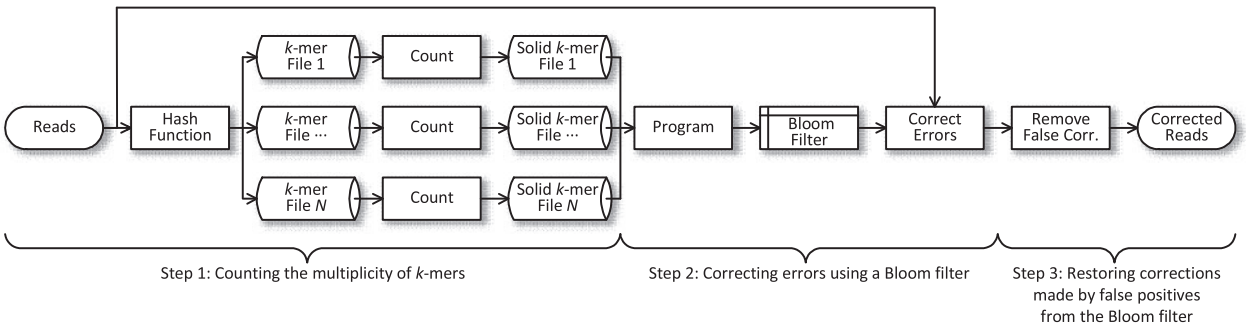

Fig. 1. The high level block diagram of BLESS. The cylinders and the rectangle with extra lines depict data written to disk and memory, respectively

solid k-mer island consists of consecutive solid k-mers, which is in neighborhoods with weak k-mers or the end of the read. To find them, all the k-mers from $r[0, k-1]$ to $r[L-k, L-1]$ are converted to their canonical forms and the canonical forms are queried to the Bloom filter. If the Bloom filter output for a k-mer is true, then the k-mer is solid. If a solid k-mer island has a solid k-mer with quality scores <10, the k-mer is removed from the solid k-mer island.

The relation between solid k-mer islands and weak k-mers is shown in Supplementary Figure S3A. Weak k-mers have errors, but the errors cannot be in the bases that overlap solid k-mers. This is because the errors that are in the overlapped bases would make the solid k-mers erroneous, while we assume that solid k-mers do not have errors. Therefore, a weak k-mer can be converted to a solid one by modifying bases that do not overlap with solid k-mers.

The weak k-mers that exist between two consecutive solid k-mer islands SI_i and SI_{i+1} can be corrected by using the rightmost k-mer of SI_i and the leftmost k-mer of SI_{i+1}. This makes all the corrected bases between SI_i and SI_{i+1} covered by k consecutive solid k-mers. If an erroneous base exists in the first or last $k-1$ bases of a read, it is not possible to get consecutive k-mers covering the erroneous base. BLESS solves this problem by extending a read on both ends.

When there is no solid k-mer island in a read, BLESS tries to change the first k-mer to a solid one by substituting low quality bases with different bases. If the first k-mer is successfully converted to solid k-mer(s), the solid k-mer(s) are traced to the right.

The detailed explanation and examples of the BLESS error correction algorithm can be found in the Supplementary Document.

2.4 Step 3: Restoring corrections caused by false positives

Although it would be rare, it is still possible that BLESS could change an erroneous base to a wrong one because of the false positives from the Bloom filter. Therefore, BLESS has a step to remove all the corrections caused by the false positives from the Bloom filter. To achieve this, BLESS first extracts candidate k-mers that contain modified bases and counts their multiplicity to check whether their multiplicity values are higher than the threshold M. After all the candidate k-mers are extracted, they are split into N files using the same hash function that was used for splitting k-mers in the original reads into N files. We have already saved the hash tables containing the multiplicity of k-mers in a hard disk; the hash tables are reloaded into memory, and the multiplicity of each candidate k-mer is checked. If the multiplicity of a candidate k-mer is smaller than M, we may conclude that the modification for the k-mer is made based on false positives, and the correction is reversed. The time complexity of this step is proportional to the number of corrected bases, and it can be expressed as $O(RL)$.

2.5 Determining parameters

Output quality of BLESS is affected by the choice of the k-mer multiplicity threshold, M. The distribution of k-mer multiplicity in the original reads is the mixture of error-free k-mers and erroneous k-mers. The multiplicity of error-free k-mers is known to follow the Poisson distribution and the multiplicity of erroneous k-mers can be fit to the gamma distribution (Kelley *et al.*, 2010; Yang, 2011). The histogram of the multiplicity of k-mers usually has the curve like the red line in Supplementary Figure S4 if k is in a reasonable range. Such a histogram can be decomposed into the histogram of error-free k-mers (blue line) and erroneous k-mers (gray line). If M is too small, many erroneous k-mers may be recognized as solid k-mers (i.e. larger false positives and smaller false negatives). On the other hand, if M is too large, many error-free k-mers become weak k-mers (i.e. larger false negatives and smaller false positives).

We define the optimal value of M, $M_{optimal}$, as the M value that minimizes the sum of false positives and false negatives. In BLESS, the histogram like the red line in Supplementary Figure S4 can be easily generated because BLESS already calculated the multiplicity of each k-mer. In the histogram, the sum of false positives and false negatives becomes the minimum when M is the valley point of the U-shape curve with the following two assumptions: (i) as M increases from the value point, the corresponding value of the gray line becomes smaller and the corresponding value of the blue line becomes larger and (ii) as M decreases from the valley point, the corresponding value of the gray line becomes larger and the corresponding value of the blue line becomes smaller. This is a reasonable assumption if error-free k-mers and erroneous k-mers can be fit into the Poisson and gamma distribution, respectively, and two distributions are away from each other.

If we assume that the current M value is the valley point and M moves to the right, the sum of false positives and false negatives increases even though the number of false positive decreases. Similarly, if M moves to the left, the sum of false positives and false negatives also increases even though the number of false negatives decreases. Therefore, the sum of false positives and false negatives becomes its minimum when M is the valley point of the histogram of the multiplicity of k-mers.

Choosing the appropriate k is also needed to get more accurate results from BLESS. If k is too long, the average multiplicity of solid k-mers becomes smaller. On the other hand, if k is too short, there may be too many unnecessary paths in the error correction process (see the Supplementary Document). This will increase not only the probability that wrong corrections are made but also BLESS's runtime. Unfortunately, BLESS currently cannot automatically determine the optimal k value. However, our empirical analysis shows that the k value that satisfies the following two conditions usually generates the results close to the best one: (i) $N_s/4^k \leq 0.0001$ where N_s represents the number of unique solid k-mers (BLESS reports N_s) and (ii) number of corrected bases becomes the maximum at the chosen k value.

3 RESULTS

To assess the performance of BLESS, we corrected errors in five different read sets from various genomes using BLESS and six other error correction methods. All the evaluations were done on a server with two Intel Xeon X5650 2.67 GHz processors, 24 GB of memory and Scientific Linux. The version and parameters of all tools used in the experiments can be found in the Supplementary Document.

3.1 Datasets used in the evaluation

We used three datasets generated by the Illumina sequencing technology and two simulated read sets. The characteristics of each read set are summarized in Table 1. The first read set, labeled D1, is the fragment library of *Staphylococcus aureus* used in the Genome Assembly Gold-standard Evaluations (GAGE) competition (Salzberg *et al.*, 2012). The second genome (D2) is high coverage (160×) low error rate (0.5%) *Escherichia coli* reads. The third read set is the fragment library of human chromosome 14 (D3) reads that were also used in the GAGE competition. To check the scalability of BLESS, we also used simulated reads generated from GRCh37 human chromosome 1 (D4). The reads were generated using simLibrary and simNGS (http://www.ebi.ac.uk/goldman-srv/simNGS), after all Ns in the reference sequence were removed. The head of each read indicates the index of the reference sequence where the read is from. Using the information, we also generated an error-free version of D4 ($D4_{Error\text{-}Free}$ hereafter). The last dataset D5 was generated to evaluate the improvement of *de novo* assembly results after error correction. Four read sets with 10–40× of read coverage and their error-free versions were generated from the first 10 Mb of the reference sequence for D4 using simNGS. Because D5 can be treated as a subset of D4, we only report *de novo* assembly results for D5 here, and all the other evaluation results for D5 are in the Supplementary Document.

To provide a controlled assessment of the accuracy of corrections made by BLESS, errors in the input read sets are identified using the error correction evaluation toolkit (ECET) (Yang *et al.*, 2012). ECET first aligns reads to the reference sequence using BWA (Li and Durbin, 2009) and identifies a set of differences between the reads and the reference. ECET evaluates corrected reads by counting how many differences in the set are removed. In our evaluations, insertions and deletions were not included in the set because insertions and deletions can be corrected by substitutions and ECET regards these substitutions as wrong modifications. For example, if a genome sequence contains ACGT and a read from the genome has one insertion between C and G (i.e. ACAG), the insertion error can be corrected by substituting the third (fourth) base A (G) with G (T). ECET counts the third and fourth bases as wrong modifications.

3.2 Error correction accuracy

We compared BLESS with the following existing error correction tools: Quake (Kelley *et al.*, 2010), Reptile (Yang *et al.*, 2010), HiTEC (Ilie *et al.*, 2011), ECHO (Kao *et al.*, 2011) and Musket (Liu *et al.*, 2013). We chose these tools to compare mainly because they cover the three major categories of error correction methods see the review by Yang *et al.* (2012), i.e. *k*-mer spectrum-based, suffix tree-based and MSA-based methods. To the best of our knowledge, PREMIER (Yin *et al.*, 2013) is the only HMM-based error correction tool for DNA reads, and it was not included in our comparison because its source code is not available. In addition, we also considered Bloom filter-based methods that were previously published. We selected DecGPU (Liu *et al.*, 2011) to compare with BLESS because it is the only Bloom filter-based method that can run without a GPU.

The comparison results of BLESS and the other six error correction tools are summarized in Table 2. The outputs of the error correction tools were converted to target error format (TEF) files using the software in ECET to measure the accuracy of the

Table 1. Details of the NGS read sets used to evaluate BLESS

Genome	Accession number		Genome length (bp)	Read length (bp)	Number of reads	Coverage (X)	Per-base error rate (%)
	Reference	Read					
D1: *S.aureus*	NC010079 NC010063.1 NC012417.1	SRR022868	2 903 080	101	1 096 140	38.1	2.1
D2: *E.coli*	NC_000913	SRR001665	4 639 675	36	20 693 240	160.6	0.5
D3: Human Chr14	NC000014.8	N/A	88 289 540	101	36 172 396	41.4	1.4
D4: Human Chr1.	NC000001.10	N/A	225 280 621	101	89 220 048	40.0	0.6
D5 (10×): Human Chr1 10 Mbp 10×	NC000001.10	N/A	10 000 000	101	990 100	10.0	0.6
D5 (20×): Human Chr1 10 Mbp 20×	NC000001.10	N/A	10 000 000	101	1 980 198	20.0	0.6
D5 (30×): Human Chr1 10 Mbp 30×	NC000001.10	N/A	10 000 000	101	2 970 298	30.0	0.6
D5 (40×): Human Chr1 10 Mbp 40×	NC000001.10	N/A	10 000 000	101	3 960 396	40.0	0.6

Note: [Genome Length] Length of genomes without Ns, [Number of Reads] Number of reads after all paired reads that contain Ns are removed. To use the software in ECET, all the reads containing Ns were removed using the program in ECET. The software removes reads without considering the paired-end information, but this may cause problems in our evaluation process because several evaluations use the paired-end information. To keep the paired-end information intact, both reads in a pair were removed if either of them contained Ns; [Coverage] Number of Reads × Read Length/Genome Length; [Per-base Error Rate] Mismatches/[(Total Number of Reads - Unaligned Reads) × Read Length].

Table 2. Accuracy, memory usage and runtime comparison on error correction algorithms

Data	Software	Accuracy			Memory (MB)	Wall-clock time (min)	Number of threads
		Sensitivity	Gain	Specificity			
D1	BLESS	**0.895**	**0.894**	**1.000**	**11**	6	1
	DecGPU	0.076	0.002	0.998	1556	**2**	12
	ECHO	0.710	0.707	**1.000**	6063	96	12
	HiTEC	0.859	0.838	0.999	2127	12	1
	Musket	0.709	0.703	**1.000**	362	**2**	12
	Quake	0.145	0.144	**1.000**	644	8	12
	Reptile	0.564	0.518	0.999	1232	7	1
D2	BLESS	**0.968**	**0.967**	**1.000**	**14**	23	1
	DecGPU	0.333	−0.028	0.998	2171	5	12
	HiTEC	0.920	0.880	**1.000**	14 096	83	1
	Musket	0.934	0.926	**1.000**	347	**3**	12
	Quake	0.838	0.837	**1.000**	8339	74	12
	Reptile	0.957	0.951	1.000	1008	52	1
D3	BLESS	**0.674**	**0.644**	**1.000**	**150**	180	1
	DecGPU	0.096	−0.058	0.998	2223	**28**	12
	Musket	0.575	0.537	**1.000**	3763	31	12
	Quake	0.128	0.126	**1.000**	2126	62	12
	Reptile	0.577	0.529	0.999	11 783	453	1
D4	BLESS	**0.892**	**0.870**	**1.000**	**372**	459	1
	DecGPU	0.358	−0.017	0.998	2473	82	12
	Musket	0.888	0.866	**1.000**	7815	**56**	12
	Quake	0.583	0.539	**1.000**	8863	188	12
	Reptile	0.807	0.704	0.999	19 007	1775	1

Note: Sensitivity, gain and specificity are defined as TP/(TP + FN), (TP − FP)/(TP + FN) and TN/(TN + FP), respectively. The best value for each data is shown in bold. For each tool, many different combinations of parameters were applied, and the output that showed the best gain was chosen.

corrected reads. In each dataset, we counted the following: erroneous bases successfully corrected (true positive, TP), correct or erroneous bases erroneously changed (false positives, FP), erroneous bases untouched (false negatives, FN) and the remaining bases (true negative, TN). Then, sensitivity, gain and specificity were calculated using these four values. Sensitivity, defined as TP/(TP + FN), shows how many errors in the input reads are corrected. Gain, defined as (TP − FP)/(TP + FN), represents the ratio of the reduction of errors to the total number of errors in the original reads. Gain can be negative if the number of newly generated (FP) errors is greater than the number of corrected errors. Specificity, defined as TN/(TN + FP), shows the fraction of error-free bases left unmodified. DecGPU and Quake cut bases that they cannot correct, and these trimmed bases are considered as FPs in ECET. In our evaluation, trimmed bases were excluded from FPs and thus not used to calculate sensitivity, gain and specificity because considering trimmed bases as FPs made gain of DecGPU and Quake worse than what they really were.

While some error correction tools such as HiTEC are able to independently choose appropriate parameters, the error correction quality of other tools depends on parameters that have to be set by the users. We generated the corrected read sets that provided the best gain using each error correction tool to compare the best results from the methods. To generate such read sets, the values of all the key parameters of each tool were scanned in a continuous fashion within their respective ranges until the gain of each tool reached the maximum. The parameters that were used to generate outputs can be found in the Supplemental Document. BLESS, Musket, Quake, Reptile and DecGPU were able to generate results for all the four datasets. ECHO did not complete the error correction for D2 even after 60 h of running, so we could not produce ECHO results for D2 and larger datasets (i.e. D3 and D4). HiTEC also failed to correct errors in D3 and D4 because it ran out of memory.

As shown in Table 2, BLESS consistently outperforms the other correction tools for all the input datasets. For D1–D4, the sensitivity of BLESS is higher than that of the other methods, whereas the difference between sensitivity and gain of BLESS is smaller than those of the other methods. This suggests that BLESS can correct more errors in the reads and that the results from BLESS always have fewer errors than those from other tools. The results for D5 are summarized in Supplementary Table S1 in the Supplementary Document.

The higher accuracy of BLESS comes from its ability to use longer k-mers. If k is too short, an erroneous k-mer may be recognized as solid, because it is more probable that a short erroneous k-mer exists in other parts of the genome. Even though an erroneous k-mer is recognized as a weak k-mer, it may be possible to convert it to multiple solid k-mers if k is too short. Supplementary Figure S5 shows how the number of

distinct k-mers changes and approaches N_{ideal} in the reference sequence of D2 and D4 as k increases. N_{ideal} represents the number of distinct k-mers in the reference sequence when all the k-mers in it are distinct.More repeats can be differentiated by using longer k, which is helpful in removing ambiguities in the error correction process. The number of distinct k-mers for *E.coli* becomes 96% of N_{ideal}, when k is 15. However, the same ratio for human chromosome 1 is only 50% for the same k value. When k becomes 31, this ratio for human chromosome 1 surpasses 90%. Note that a longer k value does not always guarantee better error correction results, as the average multiplicity of k-mers decreases as k increases. However, if k is too short, it would be more difficult to differentiate solid k-mers from weak ones and k should be increased until a sufficient average k-mer multiplicity is guaranteed.

In the NGS reads that were generated using the Illumina technology, errors are usually clustered at the 3′ end of the reads. Therefore, correcting errors in that region is an important feature of error correction methods, although correcting such errors is more difficult than correcting errors in the middle of the reads. BLESS can correct errors at the end of the reads as accurately as in other parts through a reads extension. To assess the number of corrected errors in each position of the reads, we calculated the number of TPs and sensitivity at each position. Figure 2A shows the number of TPs in each corrected read set for D1. In this graph, Reference refers to the number of errors in each position of the original reads, which rapidly increases at the 3′-end of the reads. Figure 2B shows the ratio of TPs to the number of errors (i.e. sensitivity) in each position of the reads in D1. We observed that BLESS maintains high sensitivity even in the regions where most of the errors are clustered, as indicated by the overall flat contour of the line shown in the figure. The figures for the other inputs can be found in Supplementary Figure S6.

3.3 Memory usage

The peak memory usage and runtime of each method is also displayed in Table 2. The average memory usage of BLESS is only 2.5% of the other methods. On average, BLESS consumes 5.6% of the memory that DecGPU does, which is another Bloom filter-based method. DecGPU programs k-mers into a counting Bloom filter, which helps the multiplicity of k-mers to be saved with small memory with a certain false-positive probability.

BLESS requires less memory than previous Bloom filter-based methods for the following reasons. First, BLESS can count the multiplicity of k-mers and find out the list of solid k-mers without constructing Bloom filters. Therefore, we eliminate the need to estimate the number of distinct k-mers. We also do not need to program weak k-mers into the Bloom filter. Second, BLESS uses a Bloom filter instead of a counting Bloom filter. Previous methods use counting Bloom filters to count the multiplicity of k-mers, and this information is then used to identify solid k-mers. In BLESS, however, we already know the list of solid k-mers. Therefore, it is not necessary to know the multiplicity of k-mers to identify solid k-mers anymore, and solid k-mers can be programmed into a Bloom filter instead of a counting Bloom filter. Finally, BLESS is able to remove false corrections that are generated by false positives from the Bloom filter. Therefore, the target false-positive probability of the Bloom filter used in BLESS does not need to be very small, which helps to reduce the size of the Bloom filter. Table 2 also compares the runtime of BLESS with the other methods. This is discussed in more detail in the Supplemental Document.

3.4 Alignment

To evaluate the impact of error correction on read mapping, we compared the number of reads that could be aligned to the reference sequence with Bowtie (Langmead *et al*., 2009) before and after error correction. In Table 3, each column denotes the percentage of exactly aligned reads out of all the reads. We used the paired-end alignment capability of Bowtie, and the reads that could not be aligned uniquely in the reference sequences were counted. The detailed Bowtie parameters used can be found in the Supplemental Document.

All error correction methods reduced the number of unaligned reads, but BLESS outperformed the others for all the four inputs. After errors were corrected using BLESS; 81% of the entire reads and 69% of the initially unaligned reads could be

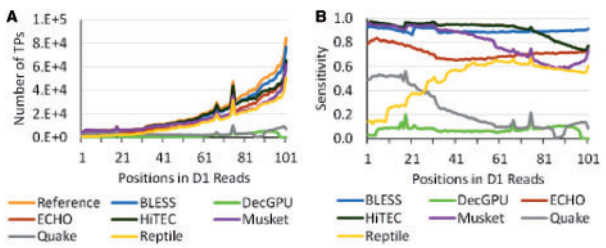

Fig. 2. The number of TPs and per-base sensitivity calculated in each position of the D1 reads. (**A**) The number of TPs calculated separately in each position of D1. 'Reference' shows the entire number of mismatch errors of the uncorrected reads. The other lines show the number of corrected errors made by each error correction tool. (**B**) The ratio of TP to Reference (i.e. number of errors in uncorrected reads) in each position of the D1 reads

Table 3. Ratio of the number of exactly aligned reads to the number of entire reads in percentage

Software	D1	D2	D3	D4
Uncorrected	19.5	73.5	42.8	36.4
Error-Free	N/A	N/A	N/A	80.3
BLESS	**75.1**	**96.5**	**74.1**	**77.2**
DecGPU	36.6	90.7	55.7	63.8
ECHO	53.6	N/A	N/A	N/A
HiTEC	70.1	95	N/A	N/A
Musket	66.9	95.3	69.6	74.3
Quake	58.1	94.3	72	65.9
Reptile	48.5	96.1	66.4	67.5

Note: Alignment was performed using the paired-end alignment of Bowtie. The best value for each column is shown in bold. [Error-Free] Result for $D4_{Error\text{-}Free}$. When we ran Bowtie, the maximum and minimum values of insert length were set, and this prevented 19.7% of Error-Free reads from being aligned to the reference sequence.

aligned to the reference on average without any mismatches. This ratio was higher than the ratio of the other methods.

D4 is a simulated read set, and we know where each read should be aligned. For each aligned read in the BLESS output, we compared the aligned position and the position where it originated. In all, 99.94% of the aligned reads were aligned to the correct positions. Even though this evaluation could not be done for D1–D3, the percentage of D1–D3 will not be very different from the D4 result because the same strict Bowtie options were used for all the datasets. The alignment results for D5 can be found in Supplementary Table S2.

3.5 *De novo* assembly

Error correction can improve not only read alignment but also *de novo* assembly results. To compare the effect of error correction methods on *de novo* assembly, scaffolds were generated using two de Bruijn graph (DBG)-based assemblers Velvet (Zerbino and Birney, 2008) and SOAPdenovo (Li *et al*., 2010) with four D5 read sets (10, 20, 30 and 40× read coverage). A string graph-based assembler SGA (Simpson and Durbin, 2012) was also used to show the effect on non-DBG-based assemblers. Scaffolds were also made using the output reads of each error correction tool, and all the scaffold sets were compared with one another.

The output quality of Velvet and SOAPdenovo is sensitive to the choice of k. Therefore, all the odd numbers between 35 and 89 were applied to Velvet and SOAPdenovo as k for each input read set. The k value that gave the longest corrected scaffold NG50 was selected. NG50 is the length of the longest scaffold, S, and that the sum of the lengths of scaffolds whose lengths are greater than or equal to S is greater than or equal to half the length of the genome length (Earl *et al*., 2011). For SGA, since the most important parameter is the minimum overlap, all the numbers from 50 to 90 were tested for each dataset to find the value that generated the longest corrected scaffold NG50. The parameters used for each dataset are described in Supplementary Tables S3–S5. Each scaffold set was evaluated using the GAGE assembly evaluation toolkit (Salzberg *et al*., 2012).

Table 4 shows the Velvet assembly results for D5 (40×). Corrected NG50 is equal to NG50 except that corrected NG50 is calculated after the scaffolds are broken at places where assembly errors occur (Salzberg *et al*., 2012). The GAGE software generates contigs by splitting scaffolds whenever a run of Ns are found. Errors in contigs include single mismatches, indels, inversions and relocations. Errors in scaffolds are the summation of indels, inversions and relocations. Genome coverage shows how many bases in the reference sequence are covered by the scaffolds. Error-Free row shows the assembly results for $D5_{\text{Error-Free}}$ (40×).

The assembly results of BLESS were better than the others in terms of assembly length and accuracy. Corrected NG50 was improved from 670 to 1004 kb after errors were corrected by BLESS. BLESS also reduced the number of errors in the contigs from 1321 to 449, and improved genome coverage from 99.5 to 99.8%. The complete *de novo* assembly results of Velvet, SOAPdenovo and SGA can be found in Supplementary Tables S3–S5.

Table 4. Summary of Velvet assembly results for D5 (40×)

Software	Scaffold corrected NG50 (kbp)	Number of errors in contigs	Number of errors in scaffolds	Genome coverage (%)
Uncorrected	671.0	1,321	0	99.5
Error-free	1239.1	543	7	99.8
BLESS	**1004.1**	**447**	2	**99.8**
DecGPU	751.6	566	2	**99.8**
ECHO	665.4	827	8	**99.8**
HiTEC	805.2	813	**0**	99.7
Musket	**1004.1**	476	3	**99.8**
Quake	850.4	553	2	**99.8**
Reptile	1004.0	466	4	**99.8**

Note: The best value for each column is shown in bold. [Error-Free] Assembly results for $D5_{\text{Error-Free}}$ (40×). An inversion error means that part of a contig or scaffold comes from a different strand with respect to the true genome. A relocation means that part of a contig or scaffold is matched with a different part within a chromosome. [Number of errors in contigs] Single mismatches + Indels + Inversion + Relocation in contigs. [Number of errors in scaffolds] Inversions + Relocation + Indels in scaffolds.

3.6 Removing false positives caused by the Bloom filter

Bloom filter may return incorrect querying results that cause false positives (Bloom, 1970). In BLESS, we designed two processes to remove such false positives. The first process is done simultaneously with the error correction. It is possible that a weak k-mer can be converted to another weak k-mer that is recognized as a solid k-mer because of a false positive from the Bloom filter. To prevent this, BLESS modifies a base only when all the k k-mers that overlap the modified base are solid k-mers. As the second process, we check the multiplicity of k-mers that include modified bases. The multiplicity of the k-mers is checked using the hash tables that were built to find solid k-mers in reads. If the multiplicity of a k-mer is smaller than M, the modified base in the k-mers is restored.

The second column of Supplementary Table S6 shows the number of corrections made by BLESS. Detected false positives in the third column are the number of FPs that were made by the false positives from the Bloom filter and detected in the final checking process. The ratio of the third column to the second was at most 0.0001%, whereas the target false positive probability of the Bloom filter in BLESS was set to its default value (i.e. 0.1%) for all the input sets. This number comes from the fact that BLESS makes a correction only if the querying results of k (or 5 at the end of reads) consecutive k-mers that cover the corrected base are true. These results demonstrate that our prefiltering method successfully prevents most of FPs made by the false positives of the Bloom filter.

3.7 Choosing parameters automatically

In BLESS, M affects the output quality, and BLESS can automatically choose this value. Supplementary Table S7 shows how close the values chosen by BLESS are to the best M that makes the gain of BLESS's output the highest. The second column represents the best M; the third column is the corresponding gain when M is the value in the second column. The fourth

and fifth columns represent M chosen by BLESS and the corresponding gain. For D1 and D3, the values that BLESS chose were the same as the best M in the second column. For D2 and D4, there are small differences between M chosen by BLESS and the best M. However, the difference between the third and fifth column was 0.001 and 0, respectively. Therefore, BLESS's auto M selection capability achieves the best gain or the nearly best gain in all the four input sets.

4 DISCUSSION

Current NGS technologies produce errors in reads, which can influence the quality of downstream analysis. Many methods have been developed to correct such sequencing errors. However, most previous methods cannot correct errors in large genomes. Even if a few methods succeeded in correcting large genomes, their outputs still contain many uncorrected errors. In addition, the memory requirement for the existing tools has been still too large for most researchers, who might only have access to computers with a moderate amount of memory.

In this work, we present a novel error correction algorithm for NGS reads, called BLESS, which has two novel features: (i) BLESS consumes much less memory than previous methods. BLESS can sort out minimum k-mers needed to correct errors, and program the k-mers in a minimum-sized Bloom filter. This makes BLESS consume much less memory than any other error correction method including previous Bloom filter-based ones. (ii) BLESS also generates more accurate results for reads from genomes with many short repeats. This is mainly because BLESS is not limited by the choices of the length of the k-mer (see Software options in the Supplementary Document for details). While the maximum k is usually 20–30 in other methods, BLESS is able to remove ambiguities in the error correction process by choosing large numbers for k without increasing memory usage. Furthermore, BLESS is efficient at correcting errors close to the ends of the reads. BLESS corrects errors at the ends of the reads by extending the ends.

BLESS was compared with previous top performers using real and simulated reads. The experimental results showed that BLESS generated more accurate results than previous algorithms while consuming only 2.5% of the memory usage of the compared methods on average. Moreover, running BLESS improved the length and accuracy of *de novo* assembly results for all the three widely used assemblers, Velvet, SOAPdenovo and SGA. BLESS also made 69% of unaligned reads exactly aligned to reference sequences.

As pointed out before, BLESS can choose large values for k. The only drawback to large k values is that the average multiplicity of such k-mers drops. If the average multiplicity of k-mers is too low, we cannot precisely distinguish erroneous k-mers using their multiplicity. Nevertheless, it is important to note thatbecause the DNA sequencing cost keeps dropping and the throughput keeps increasing, we expect to solve this problem by increasing the depth of reads. The memory usage of BLESS is proportional only to the number of solid k-mers. Because the solid k-mers eventually represent the k-mers that exist in the genome sequence, the number of solid k-mers remains constant even as the number of input reads escalates. Therefore, memory consumption of BLESS will not increase even when read depth increases as shown in Supplementary Table S1.

There are two future avenues to pursue. First, although the runtime of BLESS is already competitive as shown in Table 2, supporting multiple threads will improve BLESS's runtime further. BLESS's wall-clock time decomposition for D4 is depicted in Supplementary Figure S7. Most of the time is spent counting the number of distinct solid k-mers and correcting errors. The runtime of both processes can be improved through parallelization. In the counting step, k-mers are distributed into N files, and each file is processed in succession. This step can be parallelized without degrading memory usage. The error correction process of a read is independent of other reads, and therefore the error correction part can be easily parallelized.

Second, developing a method to automatically choose k may be added. Recent work (Chikhi and Medvedev, 2014) showed that a k-mer multiplicity histogram can be made in a short time by sampling reads and an optimal k value can be found using the histograms. This sampling-based approach will also work for BLESS. It will be helpful to reduce the runtime because it can prevent users from running BLESS multiple times with different k values.

ACKNOWLEDGEMENTS

The authors thank the National Center for Supercomputing Applications at the University of Illinois for providing computational resources.

Funding: Samsung Academic Education Program (Y.H.) and In3 Award from the University of Illinois (D.C., W.M.H., and J.M.) (in part).

Conflict of Interest: none declared.

REFERENCES

Beerenwinkel,N. and Zagordi,O. (2011) Ultra-deep sequencing for the analysis of viral populations. *Curr. Opin. Virol.*, **1**, 413–418.

Bloom,B. (1970) Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, **13**, 422–426.

Chaisson,M. *et al.* (2009) *De novo* fragment assembly with short mate-paired reads: does the read length matter? *Genome Res.*, **19**, 336–346.

Chikhi,R. and Medvedev,P. (2014) Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, **30**, 31–37.

Deorowicz,S. *et al.* (2013) Disk-based k-mer counting on a PC. *BMC Bioinformatics*, **14**, 160.

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

Durbin,R. *et al.* (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

Earl,D. *et al.* (2011) Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Res.*, **21**, 2224–2241.

Fan,L. *et al.* (2000) Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, **8**, 281–293.

Frazer,K. (2012) Decoding the human genome. *Genome Res.*, **22**, 1599–1601.

Haussler,D. *et al.* (2009) Genome 10K: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *J. Hered.*, **100**, 659–674.

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

Jiang,R. *et al.* (2009) Population genetic inference from resequencing data. *Genetics*, **181**, 187–197.

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

Kelley,D. *et al.* (2010) Quake: quality-aware detection and correction of sequencing errors. *Genome Biol.*, **11**, R116.
Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25–R10.
Le,H.-S. *et al.* (2013) Probabilistic error correction for RNA sequencing. *Nucleic Acids Res.*, **41**, e109.
Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.
Li,R. *et al.* (2010) *De novo* assembly of human genomes with massively parallel short read sequencing. *Genome Res.*, **20**, 265–272.
Liu,Y. *et al.* (2011) DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI. *BMC Bioinformatics*, **12**, 85.
Liu,Y. *et al.* (2013) Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, **29**, 308–315.
Loman,N. *et al.* (2012) Performance comparison of benchtop high-throughput sequencing platforms. *Nat. Biotechnol.*, **30**, 434–439.
Marçais,G. and Kingsford,C. (2011) A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, **27**, 764–770.
Medvedev,P. *et al.* (2011) Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, **27**, i137–i141.
Melsted,P. and Pritchard,J. (2011) Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics*, **12**, 333.
Metzker,M. (2009) Sequencing technologies—the next generation. *Nat. Rev. Genet.*, **11**, 31–46.
Pevzner,P. *et al.* (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci. USA*, **98**, 9748–9753.
Prosperi,M. *et al.* (2013) Empirical validation of viral quasispecies assembly algorithms: state-of-the-art and challenges. *Sci. Rep.*, **3**, http://www.nature.com/srep/2013/131003/srep02837/full/srep02837.html. (8 February 2014, date last accessed).
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.
Rizk,G. *et al.* (2013) DSK: k-mer counting with very low memory usage. *Bioinformatics*, **29**, 652–653.
Roy,R. *et al.* (2013) Turtle: identifying frequent k-mers with cache-efficient algorithms, [arXiv:1305.1861v1, 2013].
Salmela,L. (2010) Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, **26**, 1284–1290.
Salmela,L. and Schröder,J. (2011) Correcting errors in short reads by multiple alignments. *Bioinformatics*, **27**, 1455–1461.
Salzberg,S. *et al.* (2012) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.
Schirmer,M. *et al.* (2012) Benchmarking of viral haplotype reconstruction programmes: an overview of the capacities and limitations of currently available programmes. *Brief. Bioinform* [Epub ahead of print, Dec 19, 2013].
Schröder,J. *et al.* (2009) SHREC: a short-read error correction method. *Bioinformatics*, **25**, 2157–2163.
Shah,A.R. *et al.* (2012) A parallel algorithm for spectrum-based short read error correction. In: *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. pp. 60–70.
Shi,H. *et al.* (2009) Accelerating error correction in high-throughput short-read DNA sequencing data with CUDA. In: *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, pp. 1–8.
Shi,H. *et al.* (2010a) A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware. *J. Comput. Biol.*, **17**, 603–615.
Shi,H. *et al.* (2010b) Quality-score guided error correction for short-read sequencing data using CUDA. *Procedia Comput. Sci.*, **1**, 1129–1138.
Simpson,J. and Durbin,R. (2012) Efficient *de novo* assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549–556.
Wang,X. *et al.* (2012) Estimation of sequencing error rates in short reads. *BMC Bioinformatics*, **13**, 185.
Wijaya,E. *et al.* (2009) Recount: expectation maximization based error correction tool for next generation sequencing data. *Genome Inform.*, **23**, 189–201.
Yang,X. (2011) Error correction and clustering algorithms for next generation sequencing. In: *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. pp. 2101–2104.
Yang,X. *et al.* (2011) Repeat-aware modeling and correction of short read errors. *BMC Bioinformatics*, **12**, 1–10.
Yang,X. *et al.* (2012) A survey of error-correction methods for next-generation sequencing. *Brief. Bioinform* [Epub ahead of print].
Yang,X. *et al.* (2010) Reptile: representative tiling for short read error correction. *Bioinformatics*, **26**, 2526–2533.
Yin,X. *et al.* (2013) PREMIER - PRobabilistic Error-correction using Markov Inference in Errored Reads. *arXiv*, **2013**, 1302.0212.
Zerbino,D. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.
Zhao,Z. *et al.* (2011a) An efficient hybrid approach to correcting errors in short reads. In: Torra,V. *et al.* (ed.) *Modeling Decision for Artificial Intelligence*. Springer, Berlin Heidelberg, pp. 198–210.
Zhao,Z. *et al.* (2011b) PSAEC: An Improved Algorithm for Short Read Error Correction Using Partial Suffix Arrays. In: *Proceedings of the 5th Joint International Frontiers in Algorithmics, and 7th International Conference on Algorithmic Aspects in Information and Management*. Springer-Verlag, Jinhua, China, pp. 220–232.