Lighter: fast and memory-efficient error correction without counting

Li Song[1,2], Liliana Florea[2,1] and Ben Langmead[*1,2]

[1]Department of Computer Science, Johns Hopkins University
[2]McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine

May 25, 2014

Abstract

Lighter is a fast and memory-efficient tool for correcting sequencing errors in high-throughput sequencing datasets. Lighter avoids counting k-mers in the sequencing reads. Instead, it uses a pair of Bloom filters, one populated with a sample of the input k-mers and the other populated with k-mers likely to be correct based on a simple test. As long as the sampling fraction is adjusted in inverse proportion to the depth of sequencing, the Bloom filter size can be held constant while maintaining near-constant accuracy. Lighter is easily applied to very large sequencing datasets. It is parallelized, uses no secondary storage, and is both faster and more memory-efficient than competing approaches while achieving comparable accuracy. Lighter is free open source software available from `https://github.com/mourisl/Lighter/`.

Introduction

The cost and throughput of DNA sequencing have improved rapidly in the past several years [7], with recent advances reducing the cost of sequencing a single human genome at 30-fold coverage to around $1,000 [9]. With these advances has come an explosion of new software for analyzing large sequencing datasets. Sequencing error correction is a basic need for many of these tools. Removing errors at the outset of an analysis can improve accuracy of downstream tools such as variant callers [15]. Removing errors can also improve the speed and memory-efficiency of downstream tools, particularly for de novo assemblers based on De Bruijn graphs [23, 3].

To be useful in practice, error correction software must make economical use of time and memory even when input datasets are large (many billions of reads) and when the genome under study is also large (billions of nucleotides). Several methods have been proposed, covering a wide tradeoff space between accuracy, speed and memory- and storage-efficiency. SHREC [26] and HiTEC [12] build a suffix index of the input reads and locate errors by finding instances where a substring is followed by a character less often than expected. Coral [24] and ECHO [14] find overlaps among reads and use the resulting multiple alignments to detect and correct errors. Reptile [29] and Hammer [19] detect and correct errors by examining each k-mer's neighborhood in the dataset's k-mer Hamming graph.

The most practical and widely used error correction methods descend from the spectral alignment approach introduced in the earliest De Bruijn graph based assemblers [23, 3]. These methods count the number of times each k-mer occurs (its *multiplicity*) in the input reads, then apply a threshold such that reads with

*langmea@cs.jhu.edu

multiplicity exceeding the threshold are considered *solid*. These k-mers are unlikely to have been altered by sequencing errors. k-mers with low multiplicity (*weak* k-mers) are systematically edited into high-multiplicity k-mers using a dynamic-programming solution to the spectral alignment problem [23, 3] or, more often, a fast heuristic approximation. Quake [15], the most widely used error correction tool, uses a hash-based k-mer counter called Jellyfish [18] to determine which k-mers are correct. CUDA-EC [27] was the first to use a Bloom filter as a space-efficient alternative to hash tables for counting k-mers and for representing the set of solid k-mers. More recent tools such as Musket [17] and BLESS [10] use a combination of Bloom filters and hash tables to count k-mers or to represent the set of solid k-mers.

Lighter (LIGHTweight ERror corrector) is also in the family of spectral alignment methods, but differs from previous approaches in that it avoids counting k-mers. Rather than count k-mers, Lighter samples k-mers randomly, storing the sample in a Bloom filter. Lighter then uses a simple test applied to each position of each read to compile a set of solid k-mers, stored in a second Bloom filter. These two Bloom filters are the only sizable data structures used by Lighter.

A crucial advantage is that Lighter's parameters can be set such that the memory footprint and the accuracy of the approach are near-constant with respect to depth of sequencing. That is, no matter how deep the coverage, Lighter can allocate the same sized Bloom filters and achieve nearly the same (a) Bloom filter occupancy, (b) Bloom filter false positive rate, and (c) error correction accuracy. Lighter does this without using any disk space or other secondary memory. This is in contrast to BLESS and Quake/Jellyfish, which use secondary memory to store some or all of the k-mer counts.

Lighter's accuracy is comparable to competing tools. We show this both in simulation experiments where false positives and false negatives can be measured, and in real-world experiments where read alignment scores and assembly statistics can be measured. Lighter is also very simple and fast, faster than all other tools tried in our experiments. These advantages make Lighter quite practical compared to previous counting-based approaches, all of which require an amount of memory or secondary storage that increases with depth of coverage.

Method

Lighter's workflow is illustrated in Figure **??**. Lighter makes three passes over the input reads. The first pass obtains a sample of the k-mers present in the input reads, storing the sample in Bloom filter A. The second pass uses Bloom filter A to identify solid k-mers, which it stores in Bloom filter B. The third pass uses Bloom filter B and a greedy procedure to correct errors in the input reads.

Bloom filter

A Bloom filter [1] is a compact probabilistic data structure representing a set. It consists of an array of m bits, each initialized to 0. To add an item o, h independent hash functions $H_0(o), H_1(o), ..., H_{h-1}(o)$ are calculated. Each maps o to an integer in $[0, m)$ and the corresponding h array bits are set to 1. To test if item q is a member, the same hash functions are applied to q. q is a member if all corresponding bits are set to 1. A false positive occurs when the corresponding bits are set to 1 "by coincidence," that is, because of items besides q that were added previously. Assuming the hash functions map items to bit array elements with equal probability, the Bloom filter's false positive rate is approximately $(1 - e^{-h\frac{n}{m}})^h$, where n is the number of distinct items added, which we call the *cardinality*. Given n, which is usually determined by the dataset, m and h can be adjusted to achieve a desired false positive rate. Lower false positive rates can come at a cost, since greater values of m require more memory and greater values of k require more hash function

calculations. Many variations on Bloom filters have been proposed that additionally permit compression of the filter, storage of count data, representation of maps in addition to sets, etc [28]. Bloom filters and variants thereon have been applied in various bioinformatics settings, including assembly [22], compression [13], k-mer counting [21], and error correction [27].

By way of contrast, another way to represent a set is with a hash table. Hash tables do not yield false positives, but Bloom filters are far smaller. Whereas a Bloom filter is an array of bits, a hash table is an array of buckets, each large enough to store a pointer, key, or both. If chaining is used, lists associated with buckets incur additional overhead. While the Bloom filter's small size comes at the expense of false positives, these can be tolerated in many settings including in error correction.

Lighter's efficiency depends on the efficiency of the Bloom filter implementation. Specifically Lighter uses a "blocked" Bloom filter to decrease overall number of cache misses and improve efficiency. This comes at the expense of needing a slightly larger filter to achieve a comparable false positive rate to a non-blocked filter, as discussed in Supplementary Note 1.

In our method, the items to be stored in the Bloom filters are k-mers. Because we would like to treat genome strands equivalently for counting purposes, we will always *canonicalize* a k-mer before adding it to, or using it to query a Bloom filter. A canonicalized k-mer is either the k-mer itself or its reverse complement, whichever is lexicographically prior.

Sequencing model

We use a simple model to describe the sequencing process and Lighter's subsampling. The model resembles one suggested previously [20]. Let K be the total number of k-mers obtained by the sequencer. We say a k-mer is *incorrect* if its sequence has been altered by one or more sequencing errors. Otherwise it is *correct*. Let ϵ be the fraction of k-mers that are incorrect. We assume ϵ does not vary with the depth of sequencing. The sequencer obtains correct k-mers by sampling independently and uniformly from k-mers in the genome. Let the number of k-mers in the genome be G, and assume all are distinct. If κ_c is a random variable for the multiplicity of a correct k-mer in the input, κ_c is binomial with success probability $1/G$ and number of trials $(1-\epsilon)K$: $\kappa_c \sim Binom((1-\epsilon)K, 1/G)$. Since the number of trials is large and the success probability is small, the binomial is well approximated by a Poisson: $\kappa_c \sim Pois(K(1-\epsilon)/G)$

A sequenced k-mer survives subsampling with probability α. If κ_c' is a random variable for the number of times a correct k-mer appears in the subsample, $\kappa_c' \sim Binom((1-\epsilon)K, \alpha/G)$, which is approximately $Pois(\alpha K(1-\epsilon)/G)$.

We model incorrect k-mers similarly. The sequencer obtains incorrect k-mers by sampling independently and uniformly from k-mers "close to" a k-mer in the genome. We might define these as the set of all k-mers with low but non-zero Hamming distance from some genomic k-mer. If κ_e is a random variable for the multiplicity of an incorrect k-mer, κ_e is binomial with success probability $1/H$ and number of trials ϵK: $\kappa_e \sim Binom(\epsilon K, 1/H)$, which is approximately $Pois(K\epsilon/H)$. It is safe to assume $H \gg G$. $\kappa_e' \sim Pois(\alpha K\epsilon/H)$ is a random variable for the number of times an incorrect k-mer appears in the subsample.

Others have noted that, given a dataset with deep and uniform coverage, incorrect k-mers occur rarely while correct k-mers occur many times, proportionally to coverage [23, 3].

Stages of the method

First pass. In the first pass, Lighter examines each k-mer of each read. With probability $1-\alpha$, the k-mer is ignored. k-mers containing ambiguous nucleotides (e.g. "N") are also ignored. Otherwise, the k-mer is canonicalized and added to Bloom filter A.

Say a distinct k-mer a occurs a total of N_a times in the dataset. If none of the N_a occurrences survive subsampling, the k-mer is never added to A and A's cardinality is reduced by one. Thus, reducing α can in turn reduce A's cardinality. Because correct k-mers are more numerous, incorrect k-mers tend to be discarded from A before correct k-mers as α decreases.

The subsampling fraction α is set by the user. We suggest adjusting α in inverse proportion to depth of sequencing, for reasons discussed below. For experiments described here, we set $\alpha = 0.05$ when the average coverage is 70-fold. That is, we set α to $0.05\frac{70}{C}$ where C is average coverage.

Second pass. A read position is overlapped by up to x k-mers, $1 \leq x \leq k$, where x depends on how close the position is to either end of the read. For a position altered by sequencing error, the overlapping k-mers are all incorrect and are unlikely to appear in A. We apply a threshold such that if the number of k-mers overlapping the position and appearing in Bloom filter A is less than the threshold, we say the position is *untrusted*. Otherwise we say it is *trusted*. Each instance where the threshold is applied is called a *test case*. When one or more of the x k-mers involved in two test cases differ, we say the test cases are distinct.

Let $P^*(\alpha)$ be the probability an incorrect k-mer appears in A, taking the Bloom filter's false positive rate into account. If random variable $B_{e,x}$ represents the number of k-mers appearing in A for an untrusted position overlapped by x k-mers, $B_{e,x} \sim Binom(x, P^*(\alpha))$. We define thresholds y_x, for each x in $[1, k]$. y_x is the minimum integer such that $p(B_{e,x} \leq y_x - 1) \geq 0.995$.

Ignoring false positives for now, we model the probability of a sequenced a k-mer having been added to A as $P(\alpha) = 1 - (1 - \alpha)^{f(\alpha)}$. We define $f(\alpha) = max\{2, 0.1/\alpha\}$. That is, we assume the multiplicity of a weak k-mer is at most $f(\alpha)$, which will often be a conservative assumption, especially for small α. It is also possible to define $P(\alpha)$ in terms of random variables κ_e and κ'_e, but we avoid this here for simplicity.

A property of this threshold is that when α is small, $P(\alpha/z) = 1-(1-\alpha/z)^{0.1z/\alpha} \approx 1-(1-\alpha)^{0.1/\alpha} = P(\alpha)$, where z is a constant greater than 1 and we use the fact that $(1 - \alpha/z)^z \approx 1 - \alpha$.

For $P^*(\alpha)$, we additionally take A's false positive rate into account. If the false positive rate is β, then $P^*(\alpha) = P(\alpha) + \beta - \beta P(\alpha)$.

Once all positions in a read have been marked *trusted* or *untrusted* using the threshold, we find all instances where k trusted positions appear consecutively. The k-mer made up by those positions is added to Bloom filter B.

Third pass. In the third pass, Lighter applies a simple, greedy error correction procedure similar to that used in BLESS [10]. A read r of length $|r|$, contains $|r| - k + 1$ k-mers. k_i denotes the k-mer starting at read position i, $1 \leq i \leq |r| - k + 1$. We first identify the longest stretch of consecutive k-mers in the read that appear in Bloom filter B. Let k_b and k_e be the k-mers at the left and right extremes of the stretch. If $e < |r| - k + 1$, we examine successive k-mers to the right starting at $k_e + 1$. For a k-mer k_i that does not appear in B, we assume the nucleotide at offset $i + k - 1$ is incorrect. We consider all possible ways of substituting for the incorrect nucleotide. For each substitution, we count how many consecutive k-mers starting with k_i appear in Bloom filter B after making the substitution. We pick the substitution that creates the longest stretch of consecutive k-mers in B. If more than one candidate substitution is equally good, we call position $i + k - 1$ ambiguous and resume the rightward scan at k-mer $k_i + k$. Li: when you say we call the position ambiguous, does that mean we don't edit it? The procedure is illustrated in Figure **??**. If k_b is not the leftmost k-mer in the read, we apply a similar procedure but moving leftward.

When errors are located near to end of a read, the stretches of consecutive k-mers used to prioritize substitutions become very short. For instance, if the error is at the very last position of the read, we must

choose a substation on the basis of just one k-mer: the rightmost k-mer. This very often results in a tie, and no correction. Lighter attempts to avoid ties by considering k-mers extending beyond the end of the read, as discussed in Supplementary Note 2.

Scaling with depth of sequencing

Lighter's accuracy can be made near-constant as the depth of sequencing K increases and its memory footprint is held constant. This is accomplished by holding αK constant, i.e., by adjusting α in inverse proportion to K. This is illustrated in Tables 2 and 6. We also argue this more formally in Supplementary Note 3.

Quality score

A low base quality value at a certain position can force Lighter to treat that position as untrusted even if the overlapping k-mers indicate it is trusted. First, Lighter scans the first 1 million reads in the input, recording the quality value at the last position in each read. Lighter then chooses the 5th-percentile quality value; that is, the value such that 5% of the values are less than or equal to it say t_1. Use the same idea, we get another 5th-percentile quality, say t_2 value for the first 1 million reads' first base. When Lighter decides whether a position is trusted or not, if its quality score is less or equal to $min\{t_1, t_2 - 1\}$, then call it untrusted regardless of how many of the overlapping k-mers appear in Bloom filter A.

Parallelization

As shown in Figure **??**, Lighter works in three passes: (1) populating Bloom filter A with a k-mer subsample, (2) applying the per-position test and populating Bloom filter B with likely-correct k-mers, and (3) error correction. For pass 1, because α is usually small, most time is spent scanning the input reads. Consequently, we found little benefit to parallelizing pass 1. Pass 2 is parallelized by using concurrent threads handle subsets of input reads. Because Bloom filter A is only being queried (not added to), we need not synchronize accesses to A. Accesses to B are synchronized so that additions of k-mers to B by different threads do not interfere. Since it is typical for the same correct k-mer to be added repeatedly to B, we can save synchronization effort by first checking whether the k-mer is already present and adding it (synchronously) only if necessary. Pass 3 is parallelized by using concurrent threads to handle subsets of the reads; since Bloom filter B is only being queried, we need not synchronize accesses.

Evaluation

Simulated data set

Accuracy on simulated data. We compared Lighter's performance with Quake v0.3[15], Musket v1.1[17] and BLESS v0p12 [10]. We generated collection of reads simulated from the reference genome for the K12 strain of *E. coli* (NC_000913.2) using Mason v0.1.2 [11]. We let k-mer size $k = 17$ for all programs unless otherwise noted.

We simulated six distinct datasets with 101bp single-end reads, varying average coverage (35x, 75x 140x) and average error rate (1% and 3%). For a given error rate e we specify Mason parameters `-qmb` $e/2$ `-qme` $3e$, so that the average error rate is e but errors are more common toward the 3' end, as in real datasets.

We then ran all three tools on all six datasets, with results presented in Table 1. In these comparisons, a true positive (TP) is an instance where an error is successfully corrected, i.e. with the correct base substituted. A false positive (FP) is an instance where a spurious substitution is made at an error-free position. A false

Coverage		35×		70×		140×	
Error rate		1%	3%	1%	3%	1%	3%
α for lighter		0.1	0.1	0.05	0.05	0.025	0.025
Recall	quake	89.59	48.77	89.64	48.82	89.59	48.78
	musket	92.61	92.04	92.60	92.05	92.60	92.03
	bless	98.68	97.29	98.69	97.48	98.65	97.47
	lighter	**99.42**	**98.03**	**99.36**	**98.93**	**99.39**	**98.99**
Precision	quake	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
	musket	99.78	99.63	99.78	99.63	99.78	99.63
	bless	98.90	98.59	98.88	98.62	98.88	98.61
	lighter	99.10	99.14	99.08	99.18	99.07	99.18
F-score	quake	94.51	65.56	94.54	65.61	94.51	65.57
	musket	96.06	95.68	96.05	95.69	96.05	95.68
	bless	98.79	97.94	98.78	98.04	98.77	98.04
	lighter	**99.26**	**98.58**	**99.22**	**99.06**	**99.23**	**99.09**
Gain	quake	89.58	48.76	89.64	48.82	89.59	48.78
	musket	92.40	91.70	92.39	91.71	92.39	91.69
	bless	97.58	95.90	97.57	96.11	97.54	96.09
	lighter	**98.52**	**97.17**	**98.44**	**98.12**	**98.46**	**98.18**

Table 1 Accuracy measures for simulated rate(%) for each table for different coverages

Coverage	α	Bloom A	Bloom B
10×	0.35	41.563	19.843
20×	0.175	41.555	32.765
35×	0.1	41.555	33.802
70×	0.05	41.580	33.906
140×	0.025	41.577	33.881
280×	0.0125	41.571	33.903

Table 2 Occupancy rate(%) for each table for different coverages

negative (FN) is an instance where we either fail to detect an error or an incorrect base is substituted. As done in previous studies [17], we report the following summaries: recall = TP/(TP+NP), precision = TP/(TP+FP), F-score = 2×recall×precision/(recall+precision) and gain = (TP-FP)/(TP+FN).

Unlike the other tools, Quake both trims the untrusted tails of the reads, and discards reads that it cannot correct. For a more fair comparison, Quake's result will contain the non-correctable reads through out this paper. And for the trimmed reads, the evaluation is done only on the reported portion. This leads to very high precision relative to other tools, though at the expense of discarded data. Of the remaining tools, Lighter and Musket achieve the highest precision, with Musket achieving slightly higher precision. Lighter achieves the highest recall, F-score and gain in all experiments.

Scaling with depth of simulated sequencing. We also used Mason to generate a series of datasets with 1% error, similar to those used in Table 1, but for 10×, 20×, 35×, 70×, 140× and 280× average coverage. We ran Lighter on each and measured final occupancies (fraction of bits set) for Bloom filters A and B. If our assumptions and scaling arguments are accurate, we expect the final occupancies of the Bloom filters to remain approximately constant for relatively high levels of coverage. As seen in Table 2, this is indeed the case. Note that when coverage is quite low (10×), the occupancy of table B is significantly lower, since distributions of multiplicities of correct and incorrect k-mers become too similar to distinguish clearly.

Cardinality of Bloom filter B. We also measured the number of correct k-mers added to table B. We used the Mason dataset with 70x coverage and 1% error rate. The *E. coli* genome has 4,553,699 distinct k-mers, and 4,553,653 (99.999%) of them are in table B.

We conducted a similar experiment with Mason configured to simulate reads from a diploid version of the *E. coli* genome. Specifically, Mason was configured to introduce heterozygous SNPs at 0.1% of the reference positions. Mason then sampled the same numbers of reads from both haplotypes, making a dataset with 70x average coverage. Of the 159,098 simulated k-mers overlapping a position with a heterozygous SNP, table B held 158,723 (99.764%) of them at the end of the run.

Effect of varying α. In a series of experiments, we measured how different settings for the subsampling fraction α affected Lighter's accuracy (recall, precision, F-score and gain) as well as the occupancies of Bloom filters A and B. We three datasets simulated by Mason with 35×, 70× and 140× coverage. The simulated error rate was 1% in all cases.

As shown in Figures **??** and **??**, only a fraction of the correct k-mers are added to A when α is very small, causing many correct read positions to fail the threshold test. Lighter attempts to "correct" these error-free positions, decreasing accuracy. This also has the effect of reducing the number of consecutive stretches of k trusted positions in the reads, leading to a smaller fraction of correct k-mers added to B, and ultimately to lower accuracy. When α grows too large, the y_x thresholds grow to be greater than k, causing all positions to fail the threshold test, as seen in Figure **??**'s right-hand side. This also leads to a dramatic drop in accuracy as seen in Figure **??**. Between the two extremes, we find a broad range of values for α (from 0.06 to 0.45) that yield high accuracy.

Effect of varying k. A key parameter of Lighter is the k-mer length k. Smaller k yields higher probability that a k-mer affected by a sequencing error also appears elsewhere in the genome. For larger k, the fraction of k-mers that are correct decreases, which could lead to fewer correct k-mer in Bloom filter A. We measured how different settings for k affect accuracy using the Li: please say which dataset we used. Results are shown in Figure **??**. Accuracy is high for k-mer lengths ranging from about 18 to 30.

Real datasets

E. coli. Next we benchmarked the same error correction tools using a real sequencing dataset, ERR022075. This is a deep DNA sequencing dataset of the the K-12 strain of the *E. coli* genome. We again used Quake, Musket, BLESS and Lighter to correct errors in the dataset. To obtain a level of coverage more reflective of other projects, we randomly subsampled the reads in the dataset to obtain roughly 75x coverage (3.5M reads) of the *E. coli* K-12 reference genome. The reads are 100 × 102 bp paired-end reads. Because BLESS cannot handle paired-end reads where the ends have different lengths, we truncated the last 2 bases from the 102 bp end before running our experiments.

These data are not simulated, so we cannot measure accuracy directly. But we can measure it indirectly, as other have done [10], by measuring read alignment statistics before and after error correction. We use Bowtie2 [16] with default parameters to align the original reads and the corrected reads to the *E. coli* K-12 reference genome. We then count the total number of the matched positions in all the alignments. Results are shown in Table 3. Lighter yields the greatest improvement in number of reads aligned and in average matched positions per aligned reads. As before, Quake is hard to compare to the other tools because it trims and discards reads. This leads to negative values in the "Increase" columns.

Also, for each tool we examined the alignments for the first read in the pair. We filtered out the alignments with indels or trimmed bases (in the case of Quake), then calculated the fraction of nucleotides at each alignment position that match the reference genome. These are plotted in Figure **??**. "Position" on the x

	Read Level		Base Level	
	Mapped Reads	Increase(%)	Base Match/Read	Increase(%)
Original	3,464,137	-	99.038	-
Quake	3,475,689	0.33	97.982	-1.97
Musket	3,467,875	0.11	99.601	0.57
BLESS	3,472,976	0.26	99.611	0.58
Lighter	3,476,422	0.35	99.611	0.58

Table 3 Alignment statistics for the 75× E. coli data set, before error correction (Original row) and after error correction (Quake, Musket, BLESS and Lighter rows). The first "Increase" column shows percent increase in reads aligned. The second "Increase" column shows percent increase in average number of matching positions per aligned read.

	N50	NG50	Edits / 100kbps	Misassemblies	Coverage(%)
Original	94,879	87,008	3.41	0	97.496
Quake	100,379	91,194	5.6	2	97.515
Musket	86,419	79,481	7.25	1	97.504
BLESS	94,879	90,126	4.72	1	97.419
Lighter	98,555	94,875	4.84	2	97.510

Table 4 De novo assembly of E. coli data set

axis is the offset from the 5' end of the read. An unusual feature of this dataset is that many reads begin with an "N" indicating that the sequencer was unable to make a base call at that position. Nevertheless, error correction significantly improved the fraction of nucleotides matching the reference genome, especially at the ends of the reads.

To further assess accuracy, we assembled the reads before and after error correction and measured relevant assembly statistics using Quast [8]. We used Velvet 1.2.10[30] to assemble. Velvet is a De Bruijn graph-based assembler designed for second-generation sequencing reads. A key parameter of Velvet is the De Bruijn graph's k-mer length. To avoid being overly influenced by choice of k-mer length, for each dataset we ran Velvet with several k-mer lengths and reported statistics for the assembly with the best N50 contig size. For each assembly, we then evaluated the assembly's quality using Quast, which was configured to discard contigs shorter than 100 bp before calculating statistics.

N50 is the length such that the total length of the contigs no shorter than the N50 cover at least half the assembled genome. NG50 is similar, but with the requirement that contigs cover half the reference genome rather than half the assembled genome. Edits per 100kbps is the number of mismatches or indels per 100kbps when aligning the contigs to the reference genome. A misassembly is an instance where two adjacent stretches of bases in the assembly align either to two very distant or to two highly overlapping stretches of the reference genome. The Quast study defines these metrics in more detail [8].

Assemblies produced from reads corrected with the four programs are very similar according to these measures, with Quake and Lighter yielding the longest contigs and the best genome coverage. Surprisingly, the post-correction assemblies have more differences at nucleotide level compared to the pre-correction assemblies, perhaps due to spurious corrections.

Human Chr14

We also evaluated Lighter's effect on alignment and assembly using a dataset from the GAGE project [25]. The dataset consists of real 101 × 101 bp paired-end reads covering human chromosome 14 to 35× average coverage (36.5M reads). We set the k-mer length to 19 for all error correctors for these experiments.

Error correction's effect on Bowtie 2 alignment statistics are shown in Table 5. We used Bowtie 2 with default parameters to align the reads to an index of the human chromosome 14 sequence of the hg19 build of the human genome. Programs had comparable performance, adding between 171,000 - 323,000 aligned

	Read Level		Base Level	
	Mapped Reads	Increase(%)	Base Match/Read	Increase(%)
Original	35,993,146	-	99.492	-
Quake	36,164,028	0.47	92.622	-6.90
Musket	36,316,697	0.90	100.100	0.61
BLESS	36,297,285	0.84	100.192	0.70
Lighter	36,280,347	0.80	100.109	0.62

Table 5 Alignment of chr14 data set

	N50	NG50	Edits / 100kbps	Misassemblies	Coverage(%)
Original	5277	3847	139.16	1257	78.777
Quake	4704	3427	131.98	965	78.55
Musket	5583	4103	131.04	556	79.175
BLESS	5620	4114	128.42	583	79.204
Lighter	5712	4195	128.92	544	79.251

Table 6 De novo assembly of chr14 data set

reads and increasing the average number of matching bases per read by 0.61 - 0.70 bases. As before, Quake produced fewer correct bases per mapped read on average due to trimming.

We also tested error correction's effect on de novo assembly using Velvet for assembly and Quast to evaluate the quality of the assembly. Results are shown in Table 6. Overall, Lighter's accuracy on real data is comparable with other error correction tools, producing the longest contigs and covering the largest portion of the genome with the smallest number of assembly errors.

Speed, space usage, and scalability

We compared Lighter's peak memory usage, disk usage, and running time with Quake, Musket and BLESS. These experiments were run on a computer running Red Hat Linux 4.1.2-52 with 48 2.1GHz AMD Opteron processors and 512G memory. The input datasets are the same simulated *E. coli* datasets with 1% error rate discussed previously, plus the human chromosome 14 data from Gage.

BLESS and Lighter achieve constant memory footprint across sequencing depths. While Musket uses less memory than Quake, it uses more than either BLESS or Lighter. BLESS achieves constant memory footprint across sequencing depths, but consumes more disk space for datasets with deeper sequencing. Note that BLESS can be configured to trade off between peak memory footprint and the number of temporary files it creates. Lighter's algorithm uses no disk space. Lighter's only sizable data structures are the two Bloom filters, which reside in memory.

To assess scalability, we also compared running time for Quake, Musket and Lighter using different number of threads. For these experiments we used the simulated *E. coli* data set with 70× coverage and 1% error. Results are shown in Figure **??**. Note that Musket requires at least 2 threads due to its master-slave design. BLESS can only be run with one thread and its running time is 1475s, which is slower than Quake.

	35×		70×		140×		chr14	
	memory	disk	memory	disk	memory	disk	memory	disk
Quake	2.8G	3.3G	7.1G	6.0G	14G	12G	48G	57G
Musket	139M	0	160M	0	241M	0	1.4G	0
BLESS	10M	661M	11M	1.3G	13M	2.6G	600M	15G
Lighter	31M	0	31M	0	31M	0	510M	0

Table 7 Comparison of four error correction tools based on their memory usage (peak resident memory) and disk usage.

Discussion

At Lighter's core is a method for obtaining a set of correct k-mers from a large collection of sequencing reads. Unlike previous methods, Lighter does this without counting k-mers. By setting its parameters appropriately, its memory usage and accuracy can be held almost constant with respect to depth of sequencing. It is also quite fast and memory-efficient, and requires no temporary disk space.

Though we demonstrate Lighter in the context of sequencing error correction, Lighter's counting-free approach could be applied in other situation where a collection of solid k-mers is desired. For example, one tool for scaling metagenome sequence assembly uses of a Bloom filter populated with solid k-mers as a memory-efficient, probabilistic representation of a De Bruijn graph [22]. Other tools use counting Bloom filters [6, 2] or the related CountMin sketch [5] to represent De Bruijn graphs for compression [13] or digital normalization and related tasks [31]. We expect Ideas from Lighter could be useful in reducing the memory footprint of these and other tools.

Lighter has three parameters the user must specify: the k-mer length k, the genome length G, and the subsampling fraction α. While the performance of Lighter seems not to be overly sensitive to these parameters (see Figures **??** and **??**), it is not desirable to leave these settings to the user. In the future, we plan to extend Lighter to estimate G, along with appropriate values for k, and α, from the input reads. This could be accomplished with methods proposed in the KmerGenie [4] and KmerStream [20] studies.

Lighter is free open source software released under the GNU GPL license, and has been compiled and tested on Linux, Mac OS X and Windows computers. The software and its source are available from `https://github.com/mourisl/Lighter/`.

Acknowledgements

The authors thank Jeff Leek for helpful discussions.
Funding: National Science Foundation grant ABI-1159078 to LF and a Sloan Research Fellowship to BL.
Conflicts of Interest: none declared.

Author details

References

1. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
2. F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *Algorithms–ESA 2006*, pages 684–695. Springer, 2006.
3. M. Chaisson, P. Pevzner, and H. Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, 2004.
4. R. Chikhi and P. Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.
5. G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
6. L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, 2000.
7. T. C. Glenn. Field guide to next-generation dna sequencers. *Molecular Ecology Resources*, 11(5):759–769, 2011.
8. A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.
9. E. C. Hayden. Is the $1,000 genome for real? *Nature News*, 2014.
10. Y. Heo, X.-L. Wu, D. Chen, J. Ma, and W.-M. Hwu. Bless: Bloom-filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, page btu030, 2014.
11. M. Holtgrewe. Mason–a read simulator for second generation sequencing data. *Technical Report FU Berlin*, 2010.
12. L. Ilie, F. Fazayeli, and S. Ilie. Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, 2011.
13. D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic acids research*, 40(22):e171–e171, 2012.
14. W.-C. Kao, A. H. Chan, and Y. S. Song. Echo: a reference-free short-read error correction algorithm. *Genome research*, 21(7):1181–1192, 2011.
15. D. R. Kelley, M. C. Schatz, S. L. Salzberg, et al. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.
16. B. Langmead and S. L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.
17. Y. Liu, J. Schröder, and B. Schmidt. Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315, 2013.
18. G. Marçais and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

19. P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, 27(13):i137–i141, 2011.
20. P. Melsted and B. V. Halldórsson. Kmerstream: Streaming algorithms for k-mer abundance estimation. *bioRxiv*, 2014.
21. P. Melsted and J. K. Pritchard. Efficient counting of k-mers in dna sequences using a bloom filter. *BMC bioinformatics*, 12(1):333, 2011.
22. J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje, and C. T. Brown. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*, 109(33):13272–13277, 2012.
23. P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.
24. L. Salmela and J. Schröder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, 2011.
25. S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, et al. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.
26. J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt. Shrec: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.
27. H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig. A parallel algorithm for error correction in high-throughput short-read data on cuda-enabled graphics hardware. *Journal of Computational Biology*, 17(4):603–615, 2010.
28. S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of bloom filters for distributed systems. *Communications Surveys & Tutorials, IEEE*, 14(1):131–155, 2012.
29. X. Yang, K. S. Dorman, and S. Aluru. Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533, 2010.
30. D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.
31. Q. Zhang, J. Pell, R. Canino-Koning, A. C. Howe, and C. T. Brown. These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure. *arXiv preprint arXiv:1309.2975*, 2013.