BIOINFORMATICS ORIGINAL PAPER

Vol. 30 no. 24 2014, pages 3541–3547
doi:10.1093/bioinformatics/btu713

Sequence analysis

Advance Access publication October 28, 2014

KmerStream: streaming algorithms for *k*-mer abundance estimation

Páll Melsted[1,2,*] and Bjarni V. Halldórsson[2,3,*]

[1]Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, Reykjavík, Iceland, [2]deCODE Genetics/Amgen, Reykjavík, Iceland and [3]School of Science and Engineering, Reykjavík University, Reykjavík, Iceland

Associate Editor: Gunnar Ratsch

ABSTRACT

Motivation: Several applications in bioinformatics, such as genome assemblers and error corrections methods, rely on counting and keeping track of *k*-mers (substrings of length *k*). Histograms of *k*-mer frequencies can give valuable insight into the underlying distribution and indicate the error rate and genome size sampled in the sequencing experiment.

Results: We present KmerStream, a streaming algorithm for estimating the number of distinct *k*-mers present in high-throughput sequencing data. The algorithm runs in time linear in the size of the input and the space requirement are logarithmic in the size of the input. We derive a simple model that allows us to estimate the error rate of the sequencing experiment, as well as the genome size, using only the aggregate statistics reported by KmerStream.

As an application we show how KmerStream can be used to compute the error rate of a DNA sequencing experiment. We run KmerStream on a set of 2656 whole genome sequenced individuals and compare the error rate to quality values reported by the sequencing equipment. We discover that while the quality values alone are largely reliable as a predictor of error rate, there is considerable variability in the error rates between sequencing runs, even when accounting for reported quality values.

Availability and implementation: The tool KmerStream is written in C++ and is released under a GPL license. It is freely available at https://github.com/pmelsted/KmerStream

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

Contact: pmelsted@hi.is or Bjarni.Halldorsson@decode.is.

Received on April 6, 2014; revised on July 30, 2014; accepted on October 22, 2014

1 INTRODUCTION

k-mers are one of the most fundamental objects used when analyzing DNA sequencing data. Many assembly algorithms (Gnerre *et al.*, 2011; Li *et al.*, 2010; Zerbino and Birney, 2008) start by constructing a *de Bruijn* graph, a graph containing all *k*-mers for some fixed *k*. Some of the most commonly used algorithms for aligning reads to a reference genome start by finding short exact matches of a fixed length *k* (commonly referred to as a seed); an index of all *k*-mers is constructed and from this index an initial alignment of a part of the read is found.

In this work, we focus on the problem of estimating the number of distinct *k*-mers in a set of sequencing reads. This problem is related to *k*-mer counting, in the sense that if we can solve *k*-mer counting we can report the number of distinct *k*-mers. However, the methods we develop are an order of magnitude faster and use only a fraction of the memory compared with current *k*-mer counting software. We develop streaming algorithms to solve this problem, a framework first proposed by Alon *et al.* (1996). A similar approach is used by KmerGenie (Chikhi and Medvedev, 2014) which samples *k*-mers to approximate the frequency histogram of *k*-mer occurrences.

Sequencing errors cause considerable problems for *k*-mer based algorithms for both assembly and read mapping; in assembly these may cause the assembly to become disconnected and increases both the memory usage and computational time. In read mapping it may lead to increased computational overhead and the true location of the read not being found. The removal or correction (Meacham *et al.*, 2011; Schröder *et al.*, 2009) of erroneous bases and erroneous fragments is therefore a common preprocessing step in the analysis of DNA sequences, in particular when doing *de novo* assembly or read mapping to a reference assembly. Several programs (Kelley *et al.*, 2010; Li *et al.*, 2010; Liu *et al.*, 2013) use *k*-mers to explicitly fix or remove sequencing errors in the dataset and it is recommended to use them prior to assembly (Salzberg *et al.*, 2012).

A number of software programs have been written for quality control of DNA sequencing data, including FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/). FastQC has a number of functionalities useful for quality control, including giving a distribution of the quality values assigned by the sequencer, quality distribution by position, N content and GC content, identifying overrepresented *k*-mers and sequence length distribution. However, the *k*-mers identified tend to be short, 6 basepairs by default, and although they are useful for identifying contaminants they are not unique enough in the underlying genome to be useful for assessing the sequencing error rate, independent of what is given by the DNA sequencers.

The problem of *k*-mer counting for high throughput sequencing data sets has been well studied (Kurtz *et al.*, 2008; Marçais and Kingsford, 2011), given a set of reads report the coverage of each *k*-mer, i.e. how many reads contain that *k*-mer. Current methods for obtaining aggregate statistics of *k*-mer data are based on keeping track of all *k*-mers in a set of reads.

*To whom correspondence should be addressed.

Much work has been done on reducing memory requirements, based on exact or approximately correct methods of keeping track of a large set of *k*-mers, this work includes using succinct set representations (Conway and Bromage, 2011) or probabilistic encodings such as Bloom filters (Chikhi and Rizk, 2012; Melsted and Pritchard, 2011; Pell *et al*., 2012), whereas recent advances have focused on more speed (Deorowicz *et al*., 2013; Roy *et al*., 2014). Although the impact on memory usage is considerable, compared to previous approaches, these methods require storing all *k*-mers, explicitly or implicitly, in memory. Thus the amount of resources will grow linearly with the input size. Many methods also rely on having access to all the reads for multiple passes over the data or require additional disk space for storing intermediate results. Thus all of the above methods suffer from 'the curse of deep sequencing' (Roberts *et al*., 2011) in which more sequencing can overwhelm the program in terms of memory usage and the algorithms simply fail to make use of increased amounts of data.

1.1 Contribution

The main contribution of this article is a streaming algorithm for estimating efficiently the number of *k*-mers that occur exactly once in a data set, taking care of identifying the *k*-mer with its reverse complement. The time requirement for this algorithm is only a constant factor times the time taken to read the data and requires space that is only logarithmic in the size of the dataset. This method can be extended to give an estimate of the *k*-mer abundance histogram. Additionally our algorithm reports frequency moments of the *k*-mer count, which are aggregate statistics of the histogram. For experimental validation we show the results of running KmerStream on reads from a single lane of PhiX control sequences. We show the distribution accuracy of the estimator compared to the accurate *k*-mer counts and that the estimators are almost always within the approximation levels guaranteed.

Additionally we formulate a simple error model, both from the estimates KmerStream produces as well as the true *k*-mer counts. Both error estimates are then compared to the true *k*-mer error rate obtained from mapping *k*-mers to the reference genome of PhiX174. The results in Section 3.2 show that our simple error model underestimates the true error rate, however only by a few percent on average.

As a simple application of our method we use it to estimate the error rate in a sequencing dataset, conditioned on the per base-pair quality value given by the sequencing equipment. Each base-pair in the read is assigned a quality value on PHRED scale, an assessment of the probability that the basecall is correct. It is up to the sequencing equipment, and the basecalling method used, to assess this probability from raw data. Unlike most other methods designed for estimating error of sequencing data, our method does not require the mapping of the data to a reference genome as a preprocessing step. The low computational overhead of our method makes it suitable to identify quality issues early on in the analysis pipeline. The method can therefore be used as a filtering step to determine the quality of a run, quickly determining errors before the read mapping or assembly stage. We ran the method on a set of 2656 whole genome sequenced individuals. Our results suggest that the error probabilities given by the Illumina HiSeq machines are largely accurate. Our results also show that once we have conditioned on an error probability given by the sequencing equipment there is considerable variance in the sequencing error, suggesting that sequencing error needs to be reassessed on a per sample basis.

2 METHOD

We start by giving the background behind our work, we then present our algorithm for estimating the number of unique *k*-mers. Finally we show how this estimate can be used to estimate the error rate in a sequencing experiment, independent of the error rate reported by the sequencing equipment.

2.1 Background

A common task is to generate the abundance histogram for the *k*-mers. Generating the exact histogram requires storing a large number of *k*-mers in memory and can be done with *k*-mer-counting tools such as Jellyfish (Marçais and Kingsford, 2011) or BFCounter (Melsted and Pritchard, 2011). Counting all *k*-mers in high-throughput sequencing datasets requires tens, or even hundreds of gigabytes of memory, whereas the method we propose requires less than 10 MB. Recently, a method was proposed that generates an approximation of the histogram based on sampling *k*-mers (Chikhi and Medvedev, 2014), this method however does not have a guaranteed error rate associated with it.

We propose to not consider the exact histogram itself, but to compute statistics based on the histogram counts, which can be found using less memory and more speed than current methods by using streaming algorithms.

We define f_i to be the number of distinct *k*-mers that appear *i* times in the set of reads. The histogram is then simply a table of the f_i values. One of the key statistics we are interested in is f_1, the number of *singleton* *k*-mers, i.e. *k*-mers that appear exactly once in the set of reads. Previous studies (Melsted and Pritchard, 2011) have shown that when a genome is being sequenced at relatively high coverage the majority of singleton *k*-mers do not come from the genome, and are generated from sequencing errors. Also, the number of singleton *k*-mers grows with increased coverage, whereas the number of *k*-mers from the genome will approach a fixed number as coverage increases.

Given the frequencies f_i of the histogram, we define the *k*-th frequency moment, F_k, as

$$F_k = \sum_{i=1}^{\infty} i^k \cdot f_i \tag{1}$$

In addition to the number of singleton *k*-mers f_1, we are interested in three moments; F_0, F_1 and F_2. F_0 is the number of distinct *k*-mers in the set of reads and F_1 is simply the number of *k*-mers in the reads, counted with repetition. The second moment, F_2, puts a higher weight on the number of *k*-mers that have high abundance values.

For each of the frequency moments streaming algorithms have been developed, that can estimate their value to within a factor of $(1 \pm \varepsilon)$ with high probability using only $O(\varepsilon^{-2} \cdot \log(N))$ memory, where N is the number of distinct *k*-mers. It should be noted that estimating the frequency is solved in the general setting of counting arbitrary items in a stream, but for the remainder of the paper we will focus exclusively on the *k*-mer counting case.

Estimating the second moment, F_2, was first solved in the seminal work of Alon, Matias and Szegedy (Alon *et al*., 1996). This article also formalized the framework and popularized the research field. The first rigorous estimator for the number of distinct elements, F_0 is from (Bar-Yossef *et al*., 2002), although earlier work from (Flajolet and Nigel Martin, 1985) applies as well. The first moment, F_1, is easiest to construct as we can maintain a single counter that is incremented once for each *k*-mer.

2.2 Estimating F_0 and f_1

To compute the f_1 estimator we use a hashing approach similar to the approach of Bar-Yossef *et al.* (2002). The high level idea of the algorithm is to sample the stream at different rates, and afterwards select the sampling rate that gives the best result. For each k-mer a we compute the hash value $h(a)$ and find the least significant 1 when the value is written in binary, e.g. if $h(a) = 0110100_2$ then the least significant 1 is in the third position. In the special case when $h(a) = 0$ we define the position to be the number of bits used to represent $h(a)$ (64 in our experiments). If the hash value ends in w zeros, then 2^w divides $h(a)$ and the binary representation of $h(a)$ ends in 1, followed by w zeroes. Assuming the hash values are random half the k-mers will have $w = 0$, a quarter will have $w = 1$, etc.

In order to estimate the number of distinct k-merswe use as our data structure a list of arrays, T, one for each potential value of w, we say that the array T_w is at *level* w, and we say that all k-mers with value w *hash to this level*. Each array has a fixed size R that is only dependent on the error parameters ε, and each element in the array is a 2-bit number storing values from 0 to 3. When processing a k-mer a we look into the array T_w. Conditioned on the value of w, the $w + 1$ least significant bits are no longer random as they must end in a 1, followed by w zeros. We discard the lowest $w + 1$ bits by dividing by 2^{w+1} and use the result of the division, modulo the size of the array, as an index into the array T_w and increase the counter there. As we are only interested in F_0 and f_1, in the application presented here we limit the counter to two bits and if the counter is at 3, we do nothing. An extension of this work would be to allow the counter to count more bits, the algorithm could then be used to estimate f_i for general i.

When the number of k-mers is much larger than R the array in T_1 will almost certainly be full, i.e. all the values will be at 3. This indicates that sampling at a rate of one half is too small, and so we should look at lower sampling rates. Algorithm 1 estimates f_1 by relating it to the probability that a counter in an array has value 0 or value 1, respectively. To get a good estimate for those probabilities we require that the number of distinct k-mers that hash to the array is roughly of the same size as the array itself. Thus after we have processed all the k-mers we can find the most appropriate array T_w to use. In our analysis we ignore the possibility of two distinct k-mers having the same hash value, as we use 64-bit hash values and a pairwise random hash function we expect to have less than 10 collisions for 10 billion distinct k-mers.

If we look at a fixed level w, the probability of a counter being 0 depends only on the number of distinct k-mers that hashed to this level, N_w, and the size of the array, R. This probability is $p_0 = (1 - \frac{1}{R})^{N_w}$, as the probability of each k-mer hashing to this counter is $\frac{1}{R}$. Note that the multiplicity of the k-mer does not factor in here, since each k-mer will always hash to the same counter every time it appears. If we can estimate this p_0 accurately, then we can solve for N_w to get an approximation on the number of distinct k-mers that hash to the level w. Solving

$$p_0 = \left(1 - \frac{1}{R}\right)^{N_w} \tag{2}$$

for N_w yields

$$N_w = \frac{\ln(p_0)}{\ln\left(1 - \frac{1}{R}\right)} \tag{3}$$

Since the sampling rate is $\frac{1}{2^w}$ we can use the estimate for N_w to approximate the number of distinct k-mers. Our estimator for p_0 is $\hat{p}_0 = \frac{t_0}{R}$, where t_0 is the number of counters with value 0 in the array T_w. The estimator of the number of distinct k-mers in the dataset is then

$$\hat{F}_0 = 2^w \frac{\ln\left(\frac{t_0}{R}\right)}{\ln\left(1 - \frac{1}{R}\right)} \tag{4}$$

Although this would work for any value of w, i.e. all values of w have the same expectation, different values of w have different variance. If we use $w = 1$, then nearly all values in T_w will be non-zero, giving a poor point estimate of p_0. Similarly if w is too high, so that only a handful of elements map to T_w the estimate for p_0 will be poor. In general we want our estimates to be bounded away from 0 and 1, to this end we select the level w^* where the fraction of empty counters is as close to 50% as possible. By thinning the stream of hash values we can adaptively sample at different rates and decide on the best rate to use after we have seen all the data when we have to report the answer.

Algorithm 1 Streaming algorithm for counting k-mers

function UPDATE(a)
  $z \leftarrow h(a)$
  $w \leftarrow$ Number of trailing zeros of z
  $i \leftarrow \left\lfloor \frac{z}{2^{w+1}} \right\rfloor \pmod R$
  $T_w[i] \leftarrow \min(T_w[i] + 1, 3)$

function ESTIMATE-STATISTICS
  $w^* \leftarrow \operatorname{argmin}_w \left| \left| \{ i : T_w[i] = 0 \} \right| - \frac{1}{2} \right|$ ▷select level
  $p_0 \leftarrow \frac{|\{i : T_{w^*}[i] = 0\}|}{R}$ ▷Probability estimates
  $p_1 \leftarrow \frac{|\{i : T_{w^*}[i] = 1\}|}{R}$
  $f_1 \leftarrow 2^{w^*}(R-1)\frac{p_1}{p_0}$
  $F_0 \leftarrow 2^{w^*} \frac{\ln(p_0)}{\ln(1-\frac{1}{R})}$
  return (f_1, F_0)

Up to this point we have followed the work of Bar-Yossef *et al.* (2002) for estimating F_0, the estimator here is identical to the one derived in Theorem 2 of (Bar-Yossef *et al.*, 2002). We now show how to extend these results to obtain an accurate estimate of f_1.

We define a singleton to be a k-mer that occurs once and let x_1 be the number of singletons that hash to level w. We note that each counter with a value of 1 has to be the result of a singleton k-mer hashing to this counter, since for non-singleton k-mers the counter would have been increased at least twice. On the other hand some singleton k-mers will hash to the same counter as some other k-mer so we must find a way to relate x_1 to some probability we can compute. The probability that a counter contains the value 1 is

$$p_1 = \frac{x_1}{R}\left(1 - \frac{1}{R}\right)^{N_w - 1} \tag{5}$$

This is seen by choosing the one singleton k-mer that should hash to the counter from the x_1 possible candidates. The chosen singleton hashes to the counter with probability $\frac{1}{R}$ and all the other $N_w - 1$ k-mers must not hash to the counter with probability $(1 - \frac{1}{R})^{N_w - 1}$. Note that from (2) we see that the part $(1 - \frac{1}{R})^{N_w - 1}$ is equal to $\frac{p_0}{1 - \frac{1}{R}} = \frac{R}{R-1} p_0$. Thus we have $p_1 = \frac{x_1}{R}\frac{R}{R-1} p_0$, simplifying and solving for x_1 yields

$$x_1 = (R-1)\frac{p_1}{p_0} \tag{6}$$

We estimate p_1 in the same fashion as p_0, namely let t_1 be the number of counters in T_w equal to 1 and set $\hat{p}_1 = \frac{t_1}{R}$.

As t_0 and t_1 follow a binomial distribution with R trials and probabilities p_0 and p_1, respectively, we can expect the error in the estimates of the probabilities, $\hat{p}_0$ and $\hat{p}_1$, to be on the order of $\frac{1}{\sqrt{R}}$. This will then translate to an error rate of similar magnitude for the f_1 estimator.

We formalize the selection of the optimal value of w in Theorem 1, the proof of this theorem can be found in the supplement for the article.

THEOREM 1. *If $f_1 \geq \frac{F_0}{\lambda}$ then Algorithm 1 finds $\hat{f}_1$ such that $(1 - \varepsilon)f_1 \leq \hat{f}_1 \leq (1 + \varepsilon)f_1$ with probability at least $(1 - \delta)$. The algorithm uses $O\left(\frac{\lambda^2 \log(1/\delta)}{\varepsilon^2} \log(N)\right)$ memory and $O(1)$ time per update, where N is the number of distinct elements in the stream.*

We note that this method can be extended to obtain estimates of higher frequencies, f_i for $i \geq 2$. As an example for f_2, we note that if a counter has the value 2, this can only be obtained from two independent singletons mapping to the counter or one k-mer with frequency 2. The first happens with probability $\frac{\binom{x_1}{2}}{R^2}\left(1-\frac{1}{R}\right)^{N-2}$ and the latter with probability $\frac{x_2}{R}\left(1-\frac{1}{R}\right)^{N-1}$, where x_1 and x_2 are the number of k-mers with frequency 1 and 2 that map to a level. Given an estimate for N and x_1 we can solve to obtain an estimate for x_2. This scheme could be generalized for higher values of i to obtain estimates of f_i, although obtaining a guarantee on the error rate is left as an open problem.

2.3 Estimation of k-mer error rates

We can use our results to estimate error rates of a genomic sequencing experiment. For this we will focus solely on the k-mer error rate, i.e. given a k-mer from a read what is the probability that the k-mer did not originate from the DNA sequenced. This rate is higher than the basepair error rate, since we require all of the k basepairs to be intact. If the sequencing errors in a read were independently distributed, it would be easy to convert the k-mer error rate to a basepair error rate. However sequencing errors are not independent, generally the ends of the reads have higher error rates and sequencing errors can come in batches. Thus we will focus on the k-mer error rate in this discussion and note that converting it to a basepair error rate assuming independence will lead to an underestimate of the true basepair error rate.

We use a simple generative model where total number of sampled k-mers with repetition, namely F_1, is given. Each k-mer comes from a random position in the genome of size G, and with probability ε_k it contains a sequencing error and is error-free with probability $1-\varepsilon_k$. To account for repeated errors, we further assume that each erroneous k-mer is produced by sampling a random basepair in the reference k-mer and changing the base to one of the three other nucleotides, so that it does not match the reference. For each true k-mer there are $3k$ possible error k-mers and Hamming distance 1. The number of k-mers sampled at each location follows a Poisson distribution with mean λ and so the coverage of an error-free k-mer is $\mathrm{Poi}(\lambda(1-\varepsilon_k))$. For a fixed erroneous k-mer the coverage follows a $\mathrm{Poi}(\frac{\lambda\varepsilon_k}{3k})$, assuming that there is only one possible position in the genome that could produce this error. Our model has three parameters, G, λ and, ε_k. To obtain estimates for these parameters we require three statistics, namely f_1, F_1 and, F_0. Given the Poisson distributions of the coverages and the number of possible k-mers G for the error-free and $G \cdot 3 \cdot k$ for the single error k-mers, we can derive the following equations

$$F_0 = G \cdot 3k\left(1-e^{-\frac{\lambda\varepsilon_k}{3k}}\right) + G \cdot \left(1-e^{-\lambda(1-\varepsilon_k)}\right) \tag{7}$$

$$F_1 = \lambda \cdot G \tag{8}$$

$$\begin{aligned} f_1 &= G \cdot 3k \cdot \frac{\lambda\varepsilon_k}{3k} e^{-\frac{\lambda\varepsilon_k}{3k}} + G\lambda(1-\varepsilon_k)e^{-\lambda(1-\varepsilon_k)} \\ &= G\lambda\varepsilon_k e^{-\frac{\lambda\varepsilon_k}{3k}} + G\lambda(1-\varepsilon_k)e^{-\lambda(1-\varepsilon_k)} \end{aligned} \tag{9}$$

We can isolate λ from (8) and G from (7) to obtain

$$\lambda = \frac{F_1}{G} = \frac{F_1}{F_0}\left(3k\left(1-e^{-\frac{\lambda\varepsilon_k}{3k}}\right) + \left(1-e^{-\lambda(1-\varepsilon_k)}\right)\right) \tag{10}$$

Similarly we derive by canceling the common factor $G\lambda$ from (9) using (8)

$$\frac{f_1}{F_1} = \varepsilon_k e^{-\frac{\lambda\varepsilon_k}{3k}} + (1-\varepsilon_k)e^{-\lambda(1-\varepsilon_k)} \tag{11}$$

Using Equations (10) and (11) we can solve for λ and ε_k numerically to obtain an estimate of the coverage, k-mer error rate and finally the number of k-mers present in the genome.

We make a couple of observations about this model. First, the assumption that all error k-mers are solely due to single nucleotide differences causes us to slightly underestimate the true error rate. Second, a sequencing error in one k-mer may not be detected as an error because the error k-mer also occurs as a true k-mer in the genome. As we have chosen $k = 31$ then in random DNA the probability that a error k-mer also occurs in the human genome is less than 10^{-9}. Third, the same error k-mer can occur in two distinct reads by chance given high enough coverage, this is modeled in our k-mer distribution model via the Poisson variables. Finally, our model ignores copy numbers in a diploid genome, either single copy from heterozygous SNPs or multi-copy from repetitive regions. In Section 3.3, we give the fraction of k-mers by copy number in the Human Genome.

It has been observed that in practice sequencing reads are not randomly sampled from the genome and the true distribution is not a Poisson distribution. In particular, the coverage of genomic regions is known to be dependent on the GC rate of the basepairs (Minoche *et al.*, 2011). Meacham *et al.* (2011) have shown that the error rate is site specific, i.e. that sequencing error rate varies with the location being considered. Nonetheless we show in Section 3.2 that this simplified model gives a good estimate of the k-mer error rate using only three statistics from the k-mer distribution, rather than the entire histogram.

2.4 Implementation details

The input to KmerStream is a collection of short reads S. For each read $s \in S$ we generate all substrings of length k in s, denoted as k-mers. When reads contain other characters than $ACGT$, such as N, we split the read on the non-$ACGT$ characters and consider all k-mers in the split reads. For each k-mer we also consider the reverse complement of that k-mer, and in general make no distinction between the two when counting.

We considered all k-mers that only contained basepairs that had a q-value above a fixed threshold. Hence, if a read contains a basepair with a q-score less than the given threshold only k-mers to the left and to the right of the basepair were considered. Our computations were run for q-score thresholds of 0, 13, 20 and 30, corresponding to basepair error rates of 1, 0.05, 0.01 and 0.001, respectively. As an example, when using a q-score threshold of 30 then, according to the annotation given by the sequencing equipment, all basepairs considered should have an error rate of less than 0.001 or 0.1%. As many of the basepairs will have even lower annotated error rates, then if the annotation of the manufacturer is correct, the average error rate should be even lower.

The software is implemented in C++ and can read FASTQ, compressed FASTQ and BAM files. The user can select the k-mer size used, the error rate ε and additionally it allows for filtering k-mers based on quality scores. For the quality filtering, all bases in a read that do not meet the cutoff are discarded, thus only k-mers where all the bases in the k-mer have good quality values are kept.

3 RESULTS

We start by showing that our method is both faster and uses less memory than a previously presented methods. In order to validate our method for estimating the sequencing error rate, we first compare our method to an exact method on a known fully sequenced organism. We then estimate the effect that we repeated k-mers in the human genome could have on our error rate estimate. Finally, we apply the method to estimate error rate in a set of 2656 whole genome sequenced individuals. The scripts for the software comparisons and detailed version information are in the supplement.

3.1 Comparison to KmerGenie

We compare our software, KmerStream, to KmerGenie in terms of running time and memory usage. As input we used the H. Sapiens chr 14 (36M reads) and B. Impatiens (303 M reads), datasets from GAGE (Salzberg *et al.*, 2012). Both software were run for 4 values of k simultaneously, $k = 31, 47, 63, 79$ using four threads for parallelism. As a comparison running a fast k-mer-counter, scTurtle (Roy *et al.*, 2014) took 3594s using eight cores and 26 G of memory, for a single value of $k = 31$ for B. Impatiens.

Our experiments show that KmerStream is at least 3.5 times as fast as KmerGenie and the memory usage is an order of magnitude better, see Table 1. It should be noted that the time to read the H. Sapiens dataset was 63 s, and 1085 s for the compressed B. Impatiens dataset. Our program is therefore only about two times slower than the time it takes to read the data. This comparison also shows that KmerStream is an order of magnitude faster than full blown k-mer counting, when it comes to exploratory analysis.

3.2 PhiX174 Validation

To validate the accuracy of the k-mer error rate model proposed in Section 2.3, we used sequencing reads obtained from a PhiX control lane. We chose to use PhiX for this experiment as its genome is completely known and short enough for easily classifying k-mers as belonging to the genome or not. The sequencing data from the control has 363M reads, while the genome size of PhiX174 is only 5386 nucleotides. This high coverage allows us to partition the data and repeat the estimation process on a real dataset, rather than working with simulated data. Due to the small genome size, reads were sampled so that the per basepair coverage would be 30-fold. This was necessary to scale down the coverage to values that better correspond to a normal sequencing coverage used for whole genome sequencing. This was repeated 1000 times and each computational experiment run independently. For each input we used $k = 31$ and classified all k-mers in reads as true or false, depending on whether they appeared in the reference sequence or not. Separate frequency histograms were generated for both classes of k-mers. We then ran KmerStream for each input obtaining estimates for f_1, F_1 and F_0. The top row of Figure 1 shows the distribution of relative accuracy for f_1 and F_0 respectively. The program was run with a default error guarantee of 2% for F_0, since for this input f_1/F_0 is about 0.5 this results in a 4% error guarantee of f_1. From the distribution of Figure 1 (top row), we see that the vast majority of the experiments have estimates within the confidence interval specified.

The error rate model was fitted using both the true values for the statistics as well as the estimated values from KmerStream.

Table 1. Software comparison for running time and memory usage

Software	Parameter (k)	Dataset	Time	Memory
KmerStream	31,47,63,79	H. Sapiens, chr14	**158 s**	**47.7M**
KmerGenie	31,47,63,79	H. Sapiens, chr14	609 s	487M
KmerStream	31,47,63,79	B. Impatiens	**1835 s**	**52.4M**
KmerGenie	31,47,63,79	B. Impatiens	6424 s	495M

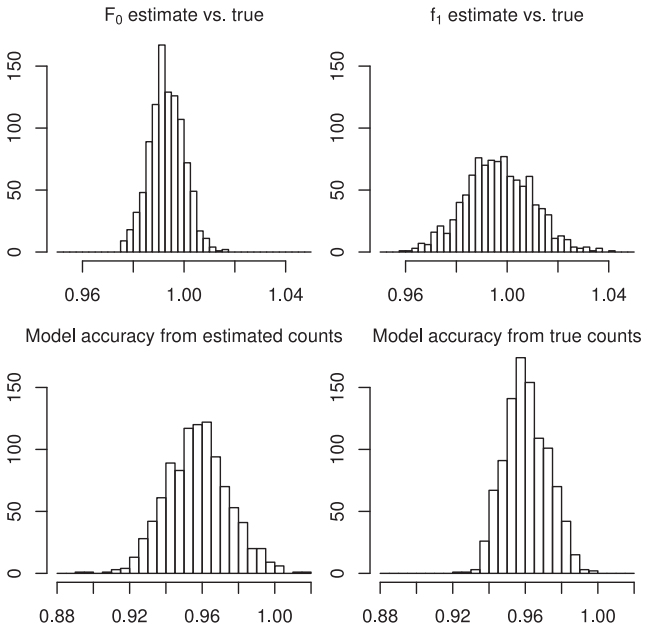

Fig. 1. Relative accuracy of KmerStream and the error model for 1000 PhiX samples. Top row: accuracy for F_0 and f_1, respectively. Bottom row: relative accuracy of 31-mer error rate based on 31-mer estimates (left) and true 31-mer numbers (right)

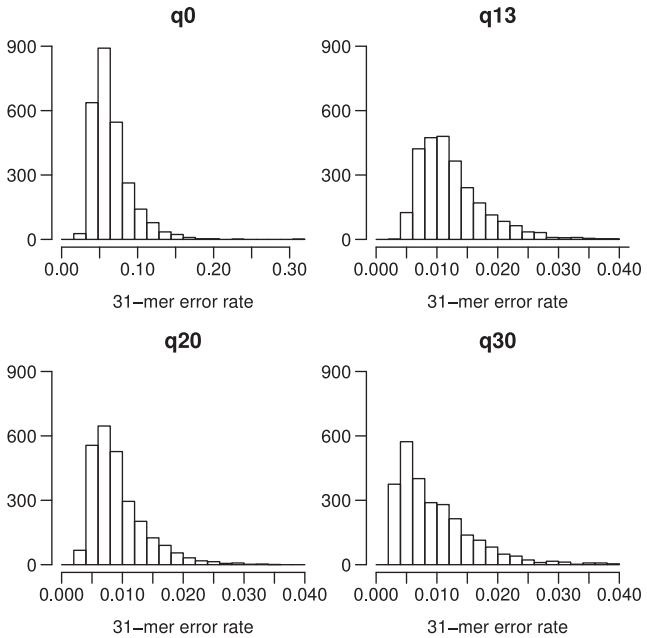

Fig. 2. Distribution of 31-mer error rates for different quality filters for 2656 individuals

The true *k*-mer error rate was obtained by matching *k*-mers to the PhiX174 reference genome.

Given the number of caveats listed when deriving our simplified model, as well as obtaining estimates from only three key statistics, we are pleasantly surprised to see how well the model fits the actual results. Based on the true values for the *k*-mer statistics we see from Figure 1 (bottom row, right) that our model underestimates the *k*-mer error rate by 4% on average. When using the estimated values from KmerStream, Figure 1 (bottom row, left) shows that we underestimate the error rate by 4% on average, same as using the true *k*-mer statistics, but we observe an increase in the variance of the accuracy of our estimate. This increase in variance is primarily because of the underlying variance in our estimate of the *k*-mer statistics. Of course the KmerStream method is much more efficient than obtaining the accurate counts, both in terms of runtime and memory. Based on these results we can safely interpret the results of the *k*-mer error model, at least up to a factor of 0.9 to 1.0.

3.3 Unique *k*-mers in the human genome

The model for estimating the *k*-mer error rate does not take into account *k*-mers that have a high repeat count, due to repetitive regions, systematic errors in sequencing or heterozygous SNPs. To detemine the number of *k*-mers having high repeat count, we constructed the diploid genome of a single individual. As input we used Human reference genome build 36, genotypes called in that individual as determined using the GATK (DePristo *et al*., 2011; McKenna *et al*., 2010) genotype caller and the assignment of these genotypes to haplotypes using long range phasing (Kong *et al*., 2008). From this information we constructed two copies of each chromosome.

Using Jellyfish (Marçais and Kingsford, 2011) we counted a total of 2.552G 31-mers in this diploid genome and observe that 4.9% of the 31-mers only occur once in the diploid genome, indicating that they overlap a heterozygous polymorphic region. Additionally, 91.7% of the 31-mers occur twice, indicating that the region is non-polymorphic, 0.1% occur three times, 1.9% of the 31-mers occur four times and 1.4% occur more than four times. In total only 3.4% of *k*-mers are repeated in the genome, suggesting that our assumptions that most *k*-mers occur uniquely in the genome is largely correct.

3.4 Error rates

To estimate the *k*-mer error rates we ran our algorithm on a data set of 2656 whole-genome sequenced individuals, using Illumina HiSeq sequencers. These individuals had an average coverage of 15.9× and a minimum coverage of 6×. All of these data are sequenced under the same conditions at the same laboratory and have already undergone a number of quality control procedures (Styrkarsdottir *et al*., 2013; Gudbjartsson *et al*., 2014). We expect these data to have comparable error characteristics.

Figure 2 shows a histogram of the average per *k*-mer error rate for different q-value thresholds and in Table 2 we give critical values of the distribution of the read error rate. We observe that without any filtering on q-values, on average 5.9% of the *k*-mers are estimated to contain errors. This number varies considerably between samples and 5% of the samples have an estimated error rate of 11.5% or higher, while 5% of the samples have an estimated error rate of 3.7% or lower.

We observe that the error rate decreases with a higher q-value threshold. We observe a marked uniform decline in the fraction of error *k*-mers when the threshold is increased from 0 to 13, an average reduction of 80%. A smaller, but clear, decrease is observed when the q-value threshold is increased from 13 to 20, 25% on average. However when comparing increasing the threshold from 20 to 30 we observed an average change of –0.9%, but a median of 9%. Additionally this decrease in the number of error *k*-mers is not uniform and in 44% of all individuals the estimated error rate increases when the q-value threshold is increased from 20 to 30, i.e. using q-value thresholds larger than 20 does not necessarily give one lower error rate *k*-mers. Furthermore the number of *k*-mers, i.e. the coverage of the data set was reduced by 25% on average by increasing the q-value threshold from 20 to 30.

In Table 2, we consider only the *k*-mers that are removed when the error threshold is increased We observe that when increasing the q-value threshold from 0 to 13, on average an estimated 96% of the *k*-mers removed are singleton *k*-mers. This suggests that unless the algorithm being used for analysis is highly robust to errors, then using a q-value threshold of 13 will increase the signal-to-noise ratio of the data, without losing coverage. On average of 85% of all *k*-mers in our dataset have q-value at all bases greater than 13, indicating that not considering *k*-mers with a q-value threshold less than 13 has limited impact on the number of *k*-mers being considered while it has a large impact on the fraction of *k*-mers that are error free. When we increase the q-value threshold from 13 to 20 an average of 91% of the *k*-mers removed are singleton *k*-mers and when increasing the threshold from 20 to 30 an average of 54% of the *k*-mers removed are singleton *k*-mers.

4 DISCUSSION

The amount of data being gathered with modern sequencing methods continues to grow at a faster rate than our ability to analyze and store the data. An alternative view to the current state of the art is to consider technologies that sequence DNA 'on the fly'. In this case, the sequencing machine does not store all the results, but rather transmits the sequence reads as they are

Table 2. Percentiles of 31-mer error rates at different q-value thresholds for 2656 individuals, changes in the number of 31-mers and the change in number of repeated 31-mers from previous threshold

Threshold	Percentile			Median change in number of	
	5	50	95	31-mers	repeated 31-mers
q0	3.7%	5.9%	11.4%		
q13	0.6%	1.1%	2.2%	$-23.0 \cdot 10^8$	$-0.981 \cdot 10^8$
q20	0.4%	0.8%	1.8%	$-1.50 \cdot 10^8$	$-0.142 \cdot 10^8$
q30	0.3%	0.8%	2.2%	$-1.69 \cdot 10^8$	$-0.696 \cdot 10^8$

generated. Technologies that fit this framework have been proposed (Branton *et al.*, 2008; Clarke *et al.*, 2009) and are currently in development, such as technologies from Oxford Nanopore, but technical details are limited at this point in time. Regardless of the exact technologies used, this new sequencing paradigm opens up new opportunities for online or streaming analysis of the data, where we bypass the storage requirements, and simply plug the sequencing directly into the analysis.

One benefit of the algorithms we have developed is as follows; $F_0 - f_1$ is a crude estimate of the number of k-mers that have been sequenced at least twice, when this number goes above a certain fraction of the genome size we can decide to stop sequencing. Another benefit is that when the error rate goes above some threshold we can decide to stop the experiment immediately, not wasting our time on failed experiments. The method presented here can be particularly useful when used for a species that has not been previously sequenced, allowing us to get an estimate the coverage of this genome while sequencing prior to assembly.

When we condition on the error rate given by Illumina we see considerable variability in the error rate between individuals. Hence, it is not advisable to use the error rates in a model without considering differences between individuals.

Our results show that although the base pair quality values given by the sequencing equipment are largely correct, there appears to be a considerable sample-dependent difference in the error rate conditioned on the base pair quality rate reported by the manufacturer. Our recommendation based on the results of sequencing 2656 individuals is to estimate both the number of k-mers F_0 as well as the coverage and k-mer error rate for multiple q-value thresholds and decide on a case by case basis.

Conflict of interest: None declared.

REFERENCES

Alon,N. *et al.* (1996) The space complexity of approximating the frequency moments. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, pp. 20–29.

Bar-Yossef,Z. *et al.* (2002) Counting distinct elements in a data stream. In: *Randomization and Approximation Techniques in Computer Science*. Springer, Berlin Heidelberg, pp. 1–10.

Branton,D. *et al.* (2008) The potential and challenges of nanopore sequencing. *Nature biotechnology*, **26**, 1146–1153.

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

Chikhi,R. and Rizk,G. (2012) Space-efficient and exact de Bruijn graph representation based on a Bloom filter. In: *Algorithms in Bioinformatics*. Springer, pp. 236–248.

Clarke,J. *et al.* (2009) Continuous base identification for single-molecule nanopore DNA sequencing. *Nat. Nanotechnol.*, **4**, 265–270.

Conway,T.C. and Bromage,A.J. (2011) Succinct data structures for assembling large genomes. *Bioinformatics*, **27**, 479–486.

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

DePristo,M. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genetics*, **43**, 491–8.

Flajolet,P. and Nigel Martin,G. (1985) Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, **31**, 182–209.

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

Gudbjartsson,D.F. *et al.* (2014) Large-scale whole-genome sequencing of the Icelandic population. *Nat. Genetics*. In Press.

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

Kong,A. *et al.* (2008) Detection of sharing by descent, long-range phasing and haplotype imputation. *Nat. Genetics*, **40**, 1068–1075.

Kurtz,S. *et al.* (2008) A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, **9**, 517.

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

Marçais,G. and Kingsford,C. (2011) A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, **27**, 764–770.

McKenna,A. *et al.* (2010) The Genome Analysis Toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.

Meacham,F. *et al.* (2011) Identification and correction of systematic error in high-throughput sequence data. *BMC Bioinformatics*, **12**, 1–11.

Melsted,P. and Pritchard,J. (2011) Efficient counting of k-mers in DNA sequences using a Bloom filter. *BMC Bioinformatics*, **12**, 333.

Minoche,A.E. *et al.* (2011) Evaluation of genomic high-throughput sequencing data generated on illumina hiseq and genome analyzer systems. *Genome Biol.* **12**, R112.

Pell,J. *et al.* (2012) Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proc. Natl Acad. Sci.*, **109**, 13272–13277.

Roberts,A. *et al.* (2011) RNA-Seq and find: entering the RNA deep field. *Genome Med.*, **3**, 74.

Roy,R.S. *et al.* (2014) Turtle: Identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics*, **30**, 1950–1957.

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

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

Styrkarsdottir,U. *et al.* (2013) Nonsense mutation in the LGR4 gene is associated with several human diseases and other traits. *Nature*, **497**, 517–520.

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