DISSERTATION TITLE

By

Qingpeng Zhang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2015

ABSTRACT

DISSERTATION TITLE

By

Qingpeng Zhang

Abstract goes here

Dedication goes here.

ACKNOWLEDGMENTS

Acknowledgements go here

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Putting in a citation so that the bibliography file is used[**?**].

1.1 Overview

1.2 Problem Statement

1.3 Significance of Research

1.4 Outline of Dissertation

Chapter 2

Review of Relevant Literature

2.1 Next-generation sequencing and metagenomics

2.1.1 Next-generation sequencing

Sequencing technology is changing really fast. Over the past decade, next-generation sequencing(NGS) has been the overwhelming technology and almost replaced classic Sanger sequencing technology. Illumina and Roche 454 are the two most popular platforms. Illumina can generate reads with shorter length, typically 100 bases for HiSeq and 150 bases for MiSeq [90, 67], but with lower cost compared to Roche 454 sequencing technology, which generates reads 500 bases to 1K bases longer. In fact a recent study comparing Illumina versus Roche 454 for metagenomics shows that both platforms agreed on over 90% of the assembled contigs and 89% of the unassembled reads[61]. Because of the advantage of low cost, there is an obvious trend that Illumina is dominating the sequencing market, which means while designing any tool for metagenomics, the developer should take the relatively short length of Illumina reads into account.

2.1.2 Metagenomics

It is believed that the word ”metagenomics” was coined in 1998 [42], which can be translated as ’beyond the genome’ [34]. At that time, basically it is the technique of cloning environmental DNA randomly and screening for interesting genes, especially 16S rRNA genes. This technique was firstly applied in practice by Schmidt et al. in 1991 [106]. This was a crucial step in the investigation to microbial world. Before that it was standard protocol to culture and isolate microbes from cells or living organisms of any other species and then do analysis. This resulted seriously narrow picture of the diversity of an ecosystem as only a small portion of the microbial species (5% or less) in the biosphere can be cultured with standard culturing techniques [115]. Metagenomics with the concept of cloning DNA directly from the environment without cultivation brings the researchers the ability to explore the genomic DNA from all the genomes of all the organisms in an environmental community, culturable or unculturable.

The improvement of next generation sequencing technology with high throughput and low cost has been accelerating the metagenomics research recently. The number of microbial species in some ecological community is huge. In soil it is estimated that there exist millions of species with most of them in low abundance [33]. Only using high throughput next-generation sequencing strategy can it be possible to sample the contents of those populations deeply enough to cover the rare species.

Currently there are two approaches in metagenomics. One is amplicon metagenomics, to amplify gene of interest like 16S rRNA genes as taxonomic markers and sequence the libraries [115], which is the traditional way dating back to 1991 experiment by Schmidt et al. Many classic methods to access microbial diversity rely on this approach. The other one is whole genome shotgun metagenomics, to sequence the libraries of randomly isolated DNA fragments without screening. Since the whole genomes of organisms in the sample are available rather than the limited genes of interest like 16S/18S rRNA, this whole genome shotgun sequencing approach can provide better taxonomic resolution and more information benefiting other investigation [122] [90]. Now there have been thousands of metagenomic genomes available in online database like MG-RAST [36].

There have been many metagenomics projects focusing on the microbial samples of different kinds of habitat, from extreme environment such as acid mine drainage channels with low complexity [122], and medium complexity samples like human gut [90] and cow rumen [43], to high complexity samples like seawater [124] and soil [35].

Metagenomics studies have expanded our knowledge of microbial world in different habitats. Some of them shed light on the explanation of some serious human deseases. Studies have shown the associations between human gut metagenomes and type II diabetes [92], obesity [121, 51] or crohn's disease [77].

In almost all of these metagenomics projects, diversity analysis plays an important role to supply information about the richness of species, the species abundance distribution in a sample or the similarity and difference between different samples, all of which are crucial to draw insightful and reliable conclusion. Next the challenges in the assessment of microbial diversity will be elaborated.

2.2 Challenges in counting k-mers efficiently

about khmer

2.3 Tackling large and error-prone short- read shotgun data sets

about diginorm

2.4 Challenges in measuring diversity of metagenomics

2.4.1 Concept of Diversity

When we try to characterize an ecological community, diversity measurement is often the first step. It is always desirable to know how many species there are in a sample, which is the concept of richness and how abundant each species is relative to others in the same sample, which is the concept of evenness. They are straightforward conceptually. But in practice, there are a large number of quantities that was suggested to measure species diversity to adapt the different scenarios of sampling individuals.

In a higher level, three diversity indices are well established and used in ecology, α-diversity, β-diversity, and γ-diversity. α-diversity is the diversity in one defined habitat or sample. β-diversity compares species diversity between habitats or samples.γ-diversity is the total diversity over a large region containing multiple ecosystems.

The concept of diversity has two aspects, richness and evenness. Richness is the total number of species identified in a sample, which is the simplest descriptors of community structure. Evenness is a measure of how different the abundance of a species is compared to other species in a community. If all the species in a community has the same abundance, the community has a higher evenness diversity. However all natural communities are highly uneven, which means a large number of species has rare abundance in the community. This raises a question about the effectiveness of using the measurement of richness to represent diversity. Is a community with 1 dominant species and 10 rare species more diverse than a community with 3 dominant species and 2 rare species? So a new metrics taking both richness and evenness into account is suggested. Two popular indice are Shannon diversity [109], which is based on information theory and shows the information in a community as an estimate of diversity, and Simpson diversity index [110], which basically shows the probability that two individuals picked randomly from a community belong to the same species.

Besides these two, Hill [44] proposed a new diversity index to use a weighted counts of species to measure diversity, based on the species abundance distribution. This can be considered as a generalized diversity index, since both Shannon and Simpson index and richness can be seen as special cases of Hill diversity index.

It is necessary to note that we can not tell if any index is better than other. It all depends on the characteristics of the community and the process of sampling and other factors. More often an index is used just because it has been used by many others before in the same scenario, it may be because it is more feasible in this scenario but it is not necessarily because it is better or even provide useful information.

In microbial ecology, richness is simply the most popular index to mesure microbial diversity, partially because of the challenge raised by the different characteristics of microbial community. Lots of methods to estimate richness in classic ecology were borrowed to tackle the problem of estimate microbial diversity, which will be discussed in the next section.

2.4.2 Diversity measurement in Microbial Ecology

There have been numerous mature methods and tools to measure diversity of macroorganisms in decades of development of classic ecology. One would think that we just need to borrow those methods to use in microbial field. Nevertheless in reality it is not that straightforward. The microbial communities are so different from macroorganisms like plant or animal communities, with the number of species many order of magnitude larger[126]. This fact raises serious sampling problems. It is really difficult to cover enough fraction of the microbial community even with impressively large sample size thanks to modern metagenomic approaches [96]. In a word, diversity measurement is a rather big challenge for microbial communities and novel and effective methods are highly demanded [103].

2.4.2.1 Concept of Species and OTU Identification using sequence markers

To borrow the methods of diversity measurement from classic ecology on the use of evaluating microbial diversity, the first problem is that in microbial world, there is no unambiguous way to define "species" [117]. It is impossible to identify a microbial individual as a specific species morphologically. In fact in metagenomics the concept of "species" has been replaced by OTUs(Operational Taxonomic Units). An OTUs are those microbial individuals within a certain evolutional distance. Practically we mainly use 16S rRNA genes as the evolutional marker genes, because 16S rRNA genes exist universally among different microbial species and their sequences change at a rate corresponding with the evolutionary distance. So we can describe microbial individuals with higher than a certain percent(like 97%) 16S rRNA sequence similarity as one OTU, or belonging to one species [103].

2.4.2.2 Binning of Metagenomic Reads into OTUs

In classic ecology dealing with samples from macroorganisms communities, before we can use any statistical method to measure diversity, it is standard procedure to identify the species of each individual in the sample. It is the same for diversity measurement of microbial communities. Difference is that here we need to place the sequences(individuals) into respective "bin" or OTUs(species). There are two strategies to do such binning - Composition-based or intrinsic binning approach and similarity-based or extrinsic binning approach.

2.4.2.2.1 Composition-based approach Lots of efforts have been put to get a comprehensive category of reference microbial genome sequences [47, 128]. Currently there are a large number of finished or high-quality reference sequences of thousands of microbial species available in different databases and this number is still increasing quickly [66, 36, 125]. So the first intrinsic composition-based approach is to use those reference genomes to train a taxonomic classifier and use that classifier to classify the metagenomics reads into bins. Different statistical approaches like Support Vector Machines [82], interpolated Markov models[4],naive Bayesian classifiers, and Growing Self Organizing Maps [97] were used to train the classifier. Without using any reference sequences for the training, it is possible to use signatures like k-mers or codon-usage to develop reference-independent approach. The assumption is that the frequencies distribution of the signatures are similar of the sequences from the same species. TETRA is such a reference-independent tools using Markov models based on k-mer frequencies [119]. There is another tool using both TETRA and codon usage statistics to classify reads [123].

2.4.2.2.2 Similarity-based approach The similarity-based extrinsic approach is to find similarity between the reads sequences and reference sequences and a tree can be built using the similarity distance information. MEGAN [48] is a typical tool using this method, which reads a BLAST file output. Other sequence alignment tools can also be used here like BowTie2 or BWA. Recently, an alternative strategy was developed, which only uses the reference sequences with the most information rather than all the reference sequences to do alignment. Those reference sequences include 16S rRNA genes or some other specific marker genes. The benefit is obvious, it is more time-efficient since there are fewer reference sequences to align to. Also, it can provide better resolution and binning accuracy since the marker genes can be selected carefully with the best distinguishing power. AMPHORA2 [129] and MetaPhlAn [108] are two typical tools using this strategy.

2.4.2.3 Statistics for Diversity Estimation

After the binning of sequences into OTU, we need statistics to help us estimate the diversity. Many statistical methods have been developed and widely used in classic ecology to macroorganisms. However the first difference between diversity measurement of macroorganisms and microbial community is that generally the microbial community diversity is much larger than observed sample diversity, thanks to the high diverse characteristics of microbial community and the limit of metagenomics sampling and sequencing. The first approach which is also considered as classic is rarefaction. Rarefaction curve can be used to compare observed richness among different samples that have been sampled unequally, which is basically the plot of the number of observed species as a function of the sampled individuals. It is worth noting that rarefaction curve shows the observed diversity, not the total diversity. We should never forget those unseen microbial species, which is pretty common for microbial community sampling.

To estimate the total diversity from observed diversity, different estimators are required.

The first one is extrapolation from accumulation curve. The asymptote of this curve is the total diversity, which means the number of species will not increase any more with sampling more individuals. To get the value of that asymptote point, from observed accumulation curve, a function needs to be assumed to fit the curve. Several proposals have been made to use this extrapolation method [19, 38]. The problem is that if the sampling effort only covers a small fraction of the total sample, which means the accumulation curve just starts, it is difficult to find an optimal function to fit the curve. Different functions can fit the curve equally well but will deduct dramatically different asymptote value. So this curve extrapolation method should be used cautiously.

Another one is parametric estimator, which assumes that the relative abundance follows a particular distribution. Then the number of unobserved species in the community can be estimated by fitting observed sample data to such abundance distribution then the total number of species in the community can be estimated. Lognormal abundance distribution is mostly used in different project since most communities of macroorganisms has a log-normal abundance distribution and it is believed that it is also typical for some microbial communities [27, 104, 93]. It is understandable that there is always controversy as to which models fit the communities best since in an ideal world the abundance distribution should be inferred from the data,not be assumed unverifiably. The problem is that we can only infer the abundance distribution accurately when the sample size is large enough. There has been some attempts on this direction recently [33] and more robust methods are still needed.

If the species abundance distribution can not be inferred, we can still use nonparametric estimators to estimate the total diversity without assuming that abundance distribution arbitrarily. These estimators are related to MRR(mark-release-recapture) statistics, which compare the number of species observed more than once and the number of species observed only once. If current sampling only covers a small fraction of a diverse community, the probability that a species is observed more than once will be low and most species will be observed only once. If current sampling is enough to cover most species in the community, the opposite will be the case. A series of estimators invented by Chao are the representative estimators in this category, including Chao1 [12], Chao2 [13], ACE [14] and ICE [57]. For example, Chao1 formular is:

$$S_{Chao1} = S_{obs} + \frac{{n_1}^2}{2n_2}$$

where S_{obs} is the number of species observed, n_1 the number of species observed once(singletons, with only one individule), and n_2 the number of species observed twice(doubletons, with exactly two individuals) in the sample. The ACE uses data from all species rather than just singletons and doubletons. Its formular is:

$$S_{ACE} = S_{abund} + \frac{S_{rare}}{C_{ACE}} + \frac{F_1}{C_{ACE}}{\gamma_{ACE}}^2$$

where S_{rare} is the number of rare species (with few than 10 individuals observed) and S_{abund} is the number of abundant species (with more than 10 individuals).

In past years there are several software packages that have been developed for biodiversity analysis. Out of them, EstimateS [18] is a software that can be used for general purpose diversity analysis, which implement a rich set of diversity analysis algorithms. However it is not designed specifically for microbial diversity analysis. So microbial diversity data should be preprocessed to general population data to be fed into EstimateS. Two other softwares - MOTHUR [105] and QIIME [10] are designed for microbial diversity. So they are more popular in microbial diversity analysis. CatchAll [9] is a relatively newer package, which can estimate the diversity using both nonparametric and parametric estimators including many variants and return the results using different estimators and the respective credibility of the results.

2.4.3 Novel methods required for diversity measurement of large metagenomics sample

As reviewed in previous sections, the topic of microbial diversity measurement has been investigated for a long time with many methods and software packages developed. However there is still lots of room for more work to do.

Basically the mainstream methods to measure microbial diversity are still focusing on the use of 16S rRNA amplicon metagenomics data. Many of the software packages mentioned above are also supposed to accept 16S rRNA data as input. This is understandable that after all the concept of OTU is from the similarity of 16S rRNA sequences. Using 16S rRNA data to measure diversity is popular but is not without problems. 16S rRNAs may not be that reliable to be OTU markers. The reliability is sensitive to potential horizontal gene transfer and the variance of gene copy in bacterium. There is suggestion that alternative marker genes should be used, like single copy housekeeping genes.

There are many projects generating whole genome shotgun metagenomics data sets. However they are mainly used for assembly and annotation purpose. Less attention was paid to diversity measurement using these whole genome metagenomics data sets. One possible reason is that the whole genome metagenomics data sets are often with low depth given the high diversity of metagenomics samples compared to 16S rRNA ampicon metagenomics data set. Assembly and annotation are always challenging with the low depth and lack of reference sequences. It is also true for diversity measurement. On the other hand, although with low depth, some whole genome metagenomics data sets are with large size because of the high diversity. For instance, there may be 4 petabase pairs of DNA in a gram of soil[130]. Many of those methods for sequence binning or diversity estimation do not scale well and will not work for large metagenomics data sets. For instance, many composition-based binning approach involves k-mer/signature frequency distribution calculation, which is rather computationally expensive. Even basic sequence alignment will be impossible for large metagenomics data set. Many of those statistical software packages to estimate diversity using various estimators are not prepared for the large scale of whole genome metagenomics data.

With the development of next generation sequencing technology, the cost of sequencing is dropping rapidly. Whole genome metagenomics sequencing is more popular and large amount of metagenomics data is being generated with increasing speed, which can not be even met by the increase of computational capacity. Novel methods that can scale well are extremely needed to deal with the increasingly large metagenomics data set.

Chapter 3

khmer-efficient online k-mer counting using a probabilistic data structure

3.1 Introduction

The goal of k-mer counting is to determine the number of occurrences for each fixed-length word of length k in a DNA data set [65]. Efficient k-mer counting plays an important role in many bioinformatics approaches, including data preprocessing for de novo assembly, repeat detection, and sequencing coverage estimation [55].

Short-read shotgun sequencing data is both relatively sparse in k-mers and contains many erroneous k-mers. For typical values of k such as 32 these data sets are sparse, as only a small fraction of the total possible number of k-mers (4^{32}) are actually present in any genome or read data sets derived from the genome. The high error rate (e.g. Illumina has a 0.1-1% per-base error rate [73]) generates many unique k-mers. As the total number of generated reads increases, the total number of errors grows with it linearly. This leads to data sets where the erroneous k-mers vastly outnumber the true k-mers [22]. Tracking and counting the resulting large number of k-mers, most of which are erroneous, has become an unavoidable and challenging task in sequence analysis [75].

A variety of k-mer counting approaches, and standalone software packages implementing them, have emerged in recent years; this includes Tallymer, Jellyfish, BFCounter, DSK, KMC, Turtle and KAnalyze [55, 65, 72, 94, 28, 98, 1].

These approaches and implementations each offer different algorithmic trade-offs and enable a non-overlapping set of functionality. Tallymer uses a suffix tree to store k-mer counts in memory and on disk [55]. Jellyfish stores k-mer counts in in-memory hash tables, and makes use of disk storage to scale to larger data sets [65]. BFCounter uses a Bloom filter as a pre-filter to avoid counting unique k-mers, and is the first published probabilistic approach to k-mer counting [72]. DSK adopts an approach to k-mer counting that enables time- and memory-efficient k-mer counting with an explicit trade-off between disk and memory usage [94]. KMC and KAnalyze rely primarily on fast and inexpensive disk access to count k-mers in low memory [28, 1]. Turtle provides several different containers that offer different false positive and false negative tradeoffs when counting k-mers [98].

Our motivation for exploring efficient k-mer counting comes from our work with metagenomic data, where we routinely encounter data sets that contain 300×10^9 bases of DNA and over 50 billion distinct k-mers [45]. To efficiently filter, partition, and assemble these data, we need to store counts for each of these k-mers in main memory, and query and update them in realtime — a set of functionality not readily offered by current packages. Moreover, we wish to enable the use of cloud and desktop computers, which may have poor I/O performance or limited memory. These needs have dictated our exploration of efficient in-memory k-mer counting techniques.

Below, we describe an implementation of a simple probabilistic data structure for k-mer counting. This data structure is based on a Count-Min Sketch [25], a generalized probabilistic data structure for storing the frequency distributions of distinct elements. Our implementation extends an earlier implementation of a Bloom filter [2], which has been previously used in bioinformatics applications, such as sequence matching [64], k-mer counting [72], and de Bruijn graph storage and traversal [83, 50]. Many other variations of Bloom filters have been proposed [6], including counting Bloom filters [31], multistage filters [30], and spectral Bloom filters [17], which are related to the Count-Min Sketch and our khmer implementation.

Probabilistic approaches can be particularly memory efficient for certain problems, with memory usage significantly lower than any exact data structure [83]. However, their use introduces set membership or counting false positives, which have effects that must be analyzed in the context of specific problems. Moreover, unlike existing techniques, the Count-Min Sketch stores only counts; k-mers must be retrieved from the original data set. In exchange, the low memory footprint enabled by this probabilistic approach enables online updating and retrieval of k-mer counts entirely in memory, which in turn supports streaming applications such as digital normalization [8].

We use the Amazon cloud to compare time, memory, and disk usage of our k-mer counting implementation with that of other k-mer counting software packages, for two problems. First, we generate a k-mer abundance distribution for large data sets; and second, we query many individual k-mer counts at random from a previously constructed k-mer count database. We show that khmer is competitive in speed, memory, and disk usage for these problems. We also analyze the effects of counting error on calculations of the k-mer count in sequencing data sets, and in particular on metagenomic data sets. Finally, we discuss khmer's miscount performance in the context of two specific applications: low-abundance k-mer trimming of reads, and digital normalization.

The khmer software [26] is implemented in C++ in a Python wrapper, enabling flexible use and reuse by users with a wide range of computational expertise. The software package is freely available for academic and commercial use and redistribution under the BSD license at github.com/ged-lab/khmer/. khmer comes with substantial documentation and many tutorials, and contains extensive unit tests. Moreover, we have built several applications on top of khmer, including memory-efficient de Bruijn graph partitioning [83] and lossy compression of short-read data sets for assembly [8].

3.2 Results

3.2.1 Implementing a Count-Min Sketch for k-mers

The two basic operations supported by khmer are `increment_count(kmer)` and `c = get_count(kmer)`. Both operate on the data structure in memory, such that neither incrementing a count nor retrieving a count involves disk access.

The implementation details are similar to those of the Bloom filter in [83], but with the use of 8 bit counters instead of 1 bit counters. Briefly, Z hash tables are allocated, each with a different size of approximately H bytes ($H_1, H_2, ..., H_Z$); the sum of these hash table sizes must fit within available main memory. To increment the count for a particular k-mer, a single hash is computed for the k-mer, and the modulus of that hash with each hash table's size H gives the location for each hash table; the associated count in each hash table is then incremented by 1. We use different sizes for each hash table so as to vary the hash function. Even if two k-mers have the same modulus in one hash table (a collision), they are unlikely to collide in the other hash tables. To retrieve the count for a k-mer, the same hash is computed and the minimum count across all hash tables is computed. While different in implementation detail from the standard Bloom filter, which uses a single hash table with many hash functions, the performance details are identical [83]. One particularly important feature of the Count-Min Sketch is that the counting error is *one-sided* [25]. Because counts are only incremented, collisions result in inflated miscounts; if there is no collision for a particular k-mer, the count is correct.

An additional benefit of the Count-Min Sketch is that it is extremely easy to implement correctly, needing only about 3 dozen lines of C++ code for a simple threadsafe implementation. (We have described how khmer scales with multiple threads in [69].)

To determine the expected false positive rate — the average frequency with which a given k-mer count will be incorrect when retrieved — we can look at the hash table load. Suppose N distinct k-mers have been counted using Z hash tables, each with size H. The probability that no collisions happened in a specific entry in one hash table is $(1-1/H)^N$, or approximately $e^{-N/H}$. The individual collision rate in one hash table is then $\approx 1-e^{-N/H}$. The total collision rate, which is the probability that a collision occurred in each entry where a k-mer maps across all Z hash tables, is $\approx (1-e^{-N/H})^Z$, which is also the expected false positive rate.

While the false positive rate can easily be calculated from the hash table load, the average *miscount* — the degree to which the measured count differs from the true count — depends on the k-mer frequency distribution, which must be determined empirically. We analyze the effects of this below.

3.2.2 Choosing number and size of hash tables used for k-mer counting

The false positive rate depends on the number of distinct k-mers N, the number of hash tables Z, and the size of the hash tables H: $f \approx (1-e^{-N/H})^Z$, with an associated memory usage of $M = HZ$. We face two common scenarios: one in which we have a fixed number of k-mers N and fixed memory M and we want to calculate the optimal number of hash tables Z; and one in which we have a desired maximum false positive rate f and a fixed number of k-mers N, and we want to calculate the minimum memory usage required to achieve f.

For fixed memory M and number of distinct k-mers N, the optimal number of hash tables can be found by minimizing f; taking the derivative, df/dZ, with $f \approx \exp(Z \log(1 - e^{-ZN/M}))$ and solving for 0, we find that f is minimized when $Z = \log(2) * (M/N)$ (see [5] for details).

Given a desired false positive rate f and a fixed number of k-mers N, the optimal memory usage can be calculated as follows. First, the optimal number of hash tables is determined by the expected false positive rate alone: $Z = \log_{0.5} f$. Using this Z, the minimum average hash table size H necessary to achieve f can be calculated as $H = (\log_{0.6185}(f) \times N)/Z$ (see [5] for details).

A remaining problem is that the number of distinct k-mers N is typically not known. However, memory- and time-efficient algorithms for calculating N do exist and we plan to implement this in khmer in the future [32].

3.2.3 khmer efficiently calculates k-mer abundance histograms

We measured time and memory required to calculate k-mer abundance histograms in five soil metagenomic read data sets using khmer, Tallymer, Jellyfish, DSK, KMC, Turtle, and KAnalyze (Table 3.1; Figures 3.1 and 3.2). We chose to benchmark abundance histograms because this functionality is common to all the software packages, and is a common analysis approach for determining assembly parameters [15]. We applied each package to increasingly large subsets of a 50m read soil metagenome data set [45]. For the BFCounter, KMC, Turtle and KAnalyze packages, which do not generate k-mer abundance distribution directly, we output the frequency of each k-mer to a file but do no further analysis.

Figure 3.1 shows that the time usage of the khmer approach is comparable to DSK and BFCounter, and, as expected, increases linearly with data set size. Tallymer is the slowest of the four tools in this testing, while KMC, Turtle, and Jellyfish are the fastest.

From Figure 3.2, we see that the memory usage of Jellyfish, Tallymer, BFCounter, and Turtle increases linearly with data set size. Tallymer uses more memory than Jellyfish generally, while BFCounter and Turtle have considerably lower memory usage. DSK, KMC, and KAnalyze use constant memory across the data sets, but at the cost of more limited functionality (discussed below).

The memory usage of khmer also increases linearly with data set size as long as we hold the false positive rate constant. However, the memory usage of khmer varies substantially with the desired false positive rate: we can decrease the memory usage by increasing the false positive rate as shown in Figure 3.2. We also see that with a low false positive of 1%, the memory usage is competitive with Tallymer and Jellyfish; with a higher 5% false positive rate, the memory usage is lower than all but the disk-based DSK; with an false positive rate as high as 20%, the memory usage is further lower, close to DSK, KAnalyze, and KMC.

We also measured disk usage during counting. Figure 3.3 shows that the disk usage also increases linearly with the number of k-mers in the data set. For a high-diversity metagenomic data set of 5 GB, the disk usage of both Jellyfish and Tallymer is around 30 GB. khmer counts k-mers entirely in working memory and does not rely on any on-disk storage to store or retrieve k-mer counts, although for practicality the hash tables can be saved for later reuse; the uncompressed disk usage for khmer in Figure 3.3 is the same as its memory. At the expense of more time, khmer supports saving and loading gzip-compressed hash tables, which are competitive in size to DSK's on-disk database (Figure 3, dashed line).

3.2.4 khmer accesses k-mer counts efficiently

We measured the time it took to access 9.7m 22-mers across five different data sets after the initial databases had been built (Figure 3.4). Note that Tallymer, Jellyfish, and khmer all support random access to k-mer counts, while BFCounter, DSK, KMC, Turtle and KAnalyze do not. Here, khmer performed well, dramatically outperforming Jellyfish and Tallymer. In all three cases, system time dominated the overall time required to retrieve k-mers, suggesting that the primary reason for the increase in retrieval time was due to the increased size of the database on the disk (data not shown). In particular, khmer is independent of the size of the database in retrieval time once the hash tables are loaded into memory.

3.2.5 The measured counting error is low on short-read data

Due to the use of Count-Min Sketch and its lack of collision tracking, khmer will report some incorrect counts for k-mers; these counts are always higher than the true counts, up to the bound of 255 (a limit imposed by our use of 8-bit counters). The frequency with which incorrect counts are reported can be estimated from the hash table load. However, the expected *miscount* — the difference between the true k-mer frequency and the reported k-mer frequency — cannot be calculated without knowing the distribution of k-mer abundances; in general, the average miscount will be small if the data is left-skewed. As noted by Melsted and Pritchard, a large number of k-mers in short-read data are low-abundance, leading to precisely the skew that would yield low miscounts [72]. Here we use both real and simulated data sets to evaluate the counting performance in practice.

Figure 3.5 shows the relationship between average miscount and counting false positive rate for five different test data sets with similar numbers of distinct k-mers: one metagenome data set; a simulated set of random k-mers; a simulated set of reads, chosen with 3x coverage and 1% error; a simulated set of reads (3x) with no error; and a set of *E. coli* reads (Table 3.2). Even when the counting false positive rate is as high as 0.9 — where 90% of k-mers have an incorrect count — the average miscount is still below 4.

We separately analyzed the average *percentage* miscount between true and false k-mers; e.g. an miscount of 4 for a k-mer whose true count is 1 would be 400%. Figure 3.6 shows the relationship between average miscount and counting false positive rate for the same five data sets as in Figure 3.5. For a false positive rate of 0.1 (10% of k-mer counts are incorrect), the average percentage miscount is less than 10% for all five data sets; this will of course generally be true, because the average miscount is bounded by the product of the false positive rate with k-mer abundance.

We see here that for a fixed false positive rate, the simulated reads without error have the highest average miscount, and the randomly generated k-mers have the lowest average miscount. This is because these two abundance distributions have the least and most left-skew, respectively: the simulated reads without error have no abundance-1 k-mers, while the randomly generated k-mers are entirely low abundance.

3.2.6 Sequencing error profiles can be measured with k-mer abundance profiles

One specific use for khmer is detecting random sequencing errors by looking at the k-mer abundance distribution within reads [70]. This approach, known also as “k-mer spectral analysis”, was first proposed in by [87] and further developed in [58]. The essential idea is that low-abundance k-mers contained in a high-coverage data set typically represent random sequencing errors.

A variety of read trimming and error correcting tools use k-mer counting to reduce the error content of the read data set, independent of quality scores or reference genomes [53]. This is an application where the counting error of the Count-Min Sketch approach used by khmer may be particularly tolerable: it will never falsely call a high-abundance k-mer as low-abundance because khmer never underestimates counts.

In Figure 3.7, we use khmer to examine the sequencing error pattern of a 5m-read subset of an Illumina reads data set from single-colony sequencing of *E. coli* [16]. The high rate of occurrence of unique k-mers close to the 3' end of reads is due to the increased sequencing error rate at the 3' end of reads.

3.2.7 khmer can be applied iteratively to read trimming

We next evaluated the effect of false-positive induced miscounts on read trimming, in which reads are truncated at the first low-abundance k-mer. Because the Count-Min Sketch never undercounts k-mers, reads will never be erroneously trimmed at truly high-abundance k-mers; however, reads may not be trimmed correctly when miscounts inflate the count of low-abundance k-mers. In cases where many errors remain, read trimming can potentially be applied multiple times, with each round reducing the total number of k-mers and hence resulting in lower false positive rates for the same memory usage.

We performed six iterations of unique k-mer trimming on 5 million Illumina reads from sequencing of *E. coli*, with memory usage less than 30 MB. For each iteration we measured empirical false positive rate compared with number of bases trimmed as well as the total number of k-mers (Table 3.3). In the first round, the estimated false positive rate was 80.0%, and 13.5% of the total bases were removed by trimming reads at low-abundance k-mers; the second iteration had a false positive rate of 37.7%, and removed only 1.5% additional data; and by the fourth iteration the false positive rate was down to 23.2% with 0.0% of the data removed.

The elimination of so many unique k-mers (column 5) in the first pass was unexpected: the high false positive rate should have resulted in fewer k-mers being identified as unique, were the erroneous k-mers independent of each other. Upon examination, we realized that in Illumina data erroneous k-mers typically come from substitution errors that yield runs of up to k erroneous k-mers in a row [53]. When trimming reads with high false positive rates, these runs are typically trimmed after the first few unique k-mers, leaving unique k-mers at the 3' end. Because of this we hypothesized that high-FP rate trimming would result in the retention of many unique k-mers at the 3' end of the read, and this was confirmed upon measurement (Table 3.3, column 6, pass 1 vs pass 2).

In comparison to quality-based trimming software such as seqtk and FASTX, trimming at unique k-mers performed very well: in this data set, all unique k-mers represent errors, and even with an initial false positive rate of 80%, khmer outperformed all but the most stringent seqtk run (Table 3.3). With a lower false positive rate or multiple passes, khmer eliminates more erroneous k-mers than seqtk or FASTX. The tradeoff here is in memory usage: for larger data sets, seqtk and FASTX will consume the same amount of memory as on smaller data sets, while khmer's memory usage will need to grow with the data set size.

3.2.8 Using khmer for digital normalization, a streaming algorithm

Digital normalization is a lossy compression algorithm that discards short reads based on saturating coverage of a de Bruijn graph [8]. While several non-streaming implementations exist, including Trinity's *in silico* normalization [41, 7], digital normalization can be efficiently implemented as a *streaming* algorithm. In the streaming implementation, if a read is not kept, it is not loaded into the Count-Min Sketch structure, and the false positive rate does not increase. For high coverage data sets, the digital normalization algorithm is sublinear in memory because it does not collect the majority of k-mers in those data sets [8]. This has the advantage of enabling low-memory preprocessing of both high-coverage genomic data sets, as well as mRNAseq or metagenomic data sets with high-coverage components [8, 45].

While digital normalization is already implemented inside khmer, previous work did not explore the lower bound on memory usage for effective digital normalization. In particular, the effects of high false positive rates have not been examined in any prior work.

We applied digital normalization to the *E. coli* data set used above, and chose seven different Count-Min Sketch sizes to yield seven different false positive rates 3.4. The data set was normalized to a k-mer coverage of 20 and the resulting data were evaluated for retention of true and erroneous k-mers, as in [8] (Table 3.4). The results show that digital normalization retains the same set of underlying "true" k-mers until the highest false positive rate of 100% (Table 3.4, column 5), while discarding only about 2% additional reads (Table 3.4, column 6).

To evaluate the effect of digital normalization with high false positive rates on actual genome assembly, we next performed normalization to a coverage of 20 with the same range of false positive rates as above. We then assembled this data with Velvet [132] and compared the resulting assemblies to the known *E. coli* MG1655 genome using QUAST (Table 3.5). To our surprise, we found that even after executing digital normalization with a false positive rate of 83.2%, a nearly complete assembly was generated. No progressive increase in misassemblies (measured against the real genome with QUAST) was seen across the different false positive rates (data not shown). This suggests that below 83.2% FP rate, the false positive rate of digital normalization has little to no effect on assembly quality with Velvet. (Note that the Velvet assembler itself used considerably more memory than digital normalization.)

While these results are specific to Velvet and the coverage parameters used in digital normalization, they do suggest that no significant information loss occurs due to false positive rates below 80%. Further evaluation of assembly quality in response to different normalization parameters and assemblers is beyond the scope of of this paper.

3.3 Discussion

3.3.1 khmer enables fast, memory-efficient online counting

khmer enables memory- and time-efficient online counting (Figures 3.1, 3.2, and 3.4). This is particularly important for the streaming approaches to data analysis needed as data set sizes increase. Because query and updating of k-mer counts can be done directly as data is being loaded, with no need for disk access or an indexing step, khmer can also perform well in situations with poor disk I/O performance. (Note that BFCounter also supports online k-mer counting [72].)

3.3.2 khmer is a generally useful k-mer counting approach

In addition to online counting, khmer offers a general range of useful performance tradeoffs for disk I/O, time and memory. From the performance comparison between khmer and other k-mer counting packages in calculating k-mer abundance distributions, khmer is comparable with existing packages. In time, khmer performs competitively with DSK and BFCounter (Figure 3.1); khmer also provides a way to systematically trade memory for miscounts across a wide range of parameters (Figure 3.2). khmer's uncompressed disk storage is competitive with Jellyfish, and, in situations where disk space is at a premium, khmer can take advantage of gzip compression to provide storage similar to that of DSK (Figure 3.3, purple line with boxes).

KMC, DSK, and KAnalyze perform especially well in memory usage for calculating the abundance distribution of k-mers. However, in exchange for this efficiency, retrieving specific k-mer counts at random is likely to be quite slow, as DSK is optimized for iterating across partition sets of k-mers rather than randomly accessing k-mer counts.

For retrieving the counts of individual k-mers, khmer is significantly faster than both Tallymer and Jellyfish. This is not surprising, since this was a primary motivation for the development of khmer.

3.3.3 khmer memory usage is fixed and low

The memory usage of the basic Count-Min Sketch approach is fixed: khmer's memory usage does not increase as data is loaded. While this means that khmer will never crash due to memory limitations, and all operations can be performed in main memory without recourse to disk storage, the false positive rate may grow too high. Therefore the memory size must be chosen in light of the false positive rate and miscount acceptable for a given application. In practice, we recommend choosing the maximum available memory, because the false positive rate decreases with increasing memory and there are no negative effects to minimizing the false positive rate.

For any given data set, the size and number of hash tables will determine the accuracy of k-mer counting with khmer. Thus, the user can control the memory usage based on the desired level of accuracy (Figure 3.2). The time usage for the first step of k-mer counting, consuming the reads, depends on the total amount of data, since we must traverse every k-mer in every read. The second step, k-mer retrieval, is algorithmically constant for fixed k; however, for practicality, the hash tables are usually saved to and loaded from disk, meaning that k-mer retrieval time depends directly on the size of the database being queried.

The memory usage of khmer is particularly low for sparse data sets, especially since only main memory is used and no disk space is necessary beyond that required for the read data sets. This is no surprise: the information theoretic comparison in [83] shows that, for sparse sequencing data sets, Bloom filters require considerably less memory than any possible exact information storage for a wide range of false positive rates and data set sparseness.

In our implementation we use 1 byte to store the count of each k-mer in the data structure. Thus the maximum count for a k-mer will be 255. In cases where tracking bigger counts is required, khmer also provides an option to use an STL map data structure to store counts above 255, with the trade-off of significantly higher memory usage. In the future, we may extend khmer to counters of arbitrary bit sizes.

3.3.4 False positive rates in k-mer counting are low and predictable

The Count-Min Sketch is a probabilistic data structure with a one-sided error that results in random overestimates of k-mer frequency, but does not generate underestimates.

In the Count-Min Sketch, the total memory usage is fixed; the memory usage, the hash functions, and the total number of distinct objects counted all influence the accuracy of the count. While the probability of an inaccurate count can easily be estimated based on the hash table load, the miscount size is dependent on details of the frequency distribution of k-mers [25].

More specifically, in the analysis of the Count-Min Sketch, the difference between the incorrect count and actual count is related to the total number of k-mers in a data set and the size of each hash table [25]. Further study has shown that the behavior of Count-Min Sketch depends on specific characteristics of the data set under consideration, especially left-skewness [99, 23]. These probabilistic properties suit short reads from next generation sequencing data sets: the miscounts are low because of the highly left-skewed abundance distribution of k-mers in these data sets.

Figures 3.5 and 3.6 demonstrate these properties well. We see more correct counting for error-prone reads from a genome than for error-free reads from a genome, with a normal distribution of k-mer abundance. Thus, this counting approach is especially suitable for high diversity data sets, such as metagenomic data, in which a larger proportion of k-mers are low abundance or unique due to sequencing errors.

3.3.5 Real-world applications for khmer

For many applications, an approximate k-mer count is sufficient. For example, when eliminating reads with low abundance k-mers, we can tolerate a certain number of low-frequency k-mers remaining in the resulting data set falsely. If RAM-limited we can do the filtering iteratively so that at each step we are making more effective use of the available memory.

In practice, we have found that a false positive rate of between 1% and 10% offers acceptable miscount performance for a wide range of tasks, including error profiling, digital normalization and low-abundance read-trimming. Somewhat surprisingly, false positive rates of up to 80% can still be used for both read trimming and digital normalization in memory-limited circumstances, although multiple passes across the data may be needed.

For many applications, the fact that khmer does not break an imposed memory bound is extremely useful, since for many data sets — especially metagenomic data sets — high memory demands constrain analysis [45, 62]. Moreover, because the false positive rate is straightforward to measure, the user can be warned that the results should be invalidated when too little memory is used. When combined with the graceful degradation of performance for both error trimming and digital normalization, khmer readily enables analysis of extremely large and diverse data sets [46]. In an experiment to assemble the reads of a soil metagenomic sample collected from Iowa prairie, the number of reads to assemble drops from 3.3 million to 2.2 million and the size of the data set drops from 245GB to 145GB accordingly after digital normalization [45]. 240GB memory was used in the process. This also shows that khmer works well to analyze large, real-world metagenomic data sets.

3.3.6 Conclusion

K-mer counting is widely used in bioinformatics, and as sequencing data set sizes increase, graceful degradation of data structures in the face of large amounts of data has become important. This is especially true when the theoretical and practical effects of the degradation can be predicted (see e.g. [72, 83, 98]). This is a key property of the Count-Min Sketch approach, and its implementation in khmer.

The khmer software implementation offers good performance, a robust and well-tested Python API, and a number of useful and well-documented scripts. While Jellyfish, DSK, KMC, and Turtle also offer good performance, khmer is competitive, and, because it provides a Python API for online counting, is flexible. In memory-limited situations with poor I/O performance, khmer is particularly useful, because it will not break an imposed memory bound and does not require disk access to store or retrieve k-mer counts. However, in exchange for this memory guarantee, counting becomes increasingly incorrect as less memory is used or as the data set size grows large; in many situations this may be an acceptable tradeoff.

3.3.7 Future considerations

Applying khmer to extremely large data sets with many distinct k-mers requires a large amount of memory: approximately 446 GB of memory is required to achieve an false positive rate of 1% for 50×10^9 k-mers. It is possible to reduce the required memory by dividing k-mer space into multiple partitions and counting k-mers separately for each partition. Partitioning k-mer space into M partitions results in a linear decrease in the number of k-mers under consideration, thus reducing the occupancy by a constant factor M and correspondingly reducing the collision rate. Partitioning k-mer space is a generalization of the systematic prefix filtering approach, where one might first count all k-mers starting with AA, then AC, then AG, AT, CA, etc., which is equivalent to partitioning k-mer space into 16 equal-sized partitions. These partitions can be calculated independently, either across multiple machines or iteratively on a single machine, and the results stored for later comparison or analysis. This is similar to the approach taken by DSK [94], and could easily be implemented in khmer.

Further optimization of khmer on single machines, e.g. for multi-core architectures, is unlikely to achieve significantly greater speed. Past a certain point k-mer counting is fundamentally I/O bound [69].

Perhaps the most interesting future direction for probabilistic k-mer counting is that taken by Turtle [98], in which several data structures are provided, each with different trade-offs, but with a common API. We hope to pursue this direction in the future by integrating such approaches into khmer.

3.4 Methods

3.4.1 Code and data set availability

The version of khmer used to generate the results below is available at http://github.com/ged-lab/khmer.git, tag '2013-khmer-counting'. Scripts specific to this paper are available in the paper repository at

https://github.com/ged-lab/2013-khmer-counting. The IPython[85] notebook file and data analysis to generate the figures are also available in that github repository. Complete instructions to reproduce all of the results in this paper are available in the khmer-counting repository; see README.rst.

3.4.2 Sequence Data

One human gut metagenome reads data set (MH0001) from the MetaHIT (Metagenomics of the Human Intestinal Tract) project [91] was used. It contains approximately 59 million reads, each 44bp long; it was trimmed to remove low quality sequences.

Five soil metagenomics reads data sets with different size were taken from the GPGC project for benchmark purpose (see Table 3.1). These reads are from soil in Iowa region and they are filtered to make sure there are less than 30% Ns in the read and each read is longer than 30 bp. The exact data sets used for the paper are available on Amazon S3 and the instructions to acquire these data sets are available in the paper repository on github.com.

We also generated four short-read data sets to assess the false positive rate and miscount distribution. One is a subset of a real metagenomics data set from the MH0001 data set, above. The second consists of randomly generated reads. The third and fourth contain reads simulated from a random, 1 Mbp long genome. The third has a substitution error rate of 3%, and the fourth contains no errors. The four data sets were chosen to contain identical numbers of distinct 22-mers. The scripts necessary to regenerate these data are available in the paper repository on github.com.

3.4.3 Count-Min Sketch implementation

We implemented the Count-Min Sketch data structure, a simple probabilistic data structure for counting distinct elements [25]. Our implementation uses Z independent hash tables, each containing a prime number of counters H_i. The hashing function for each hash table is fixed, and reversibly converts each DNA k-mer (for $k \leq 32$) into a 64-bit number to which the modulus of the hash table size is applied. This provides Z distinct hash functions.

To increment the count associated with a k-mer, the counter associated with the hashed k-mer in each of the N hash tables is incremented. To retrieve the count associated with a k-mer, the minimum count across all N hash tables is chosen.

In this scheme, collisions are explicitly not handled, so the count associated with a k-mer may not be accurate. Because collisions only falsely *increment* counts, however, the retrieved count for any given k-mer is guaranteed to be no less than the correct count. Thus the counting error is one-sided.

3.4.4 Hash function and khmer implementation

The current khmer hash function works only for $k \leq 32$ and converts DNA strings exactly into 64-bit numbers. However, any hash function would work. For example, a cyclic hash would enable khmer to count k-mers larger in size than 32; this would not change the scaling behavior of khmer at all, and is a planned extension.

By default khmer counts k-mers in DNA, i.e. strandedness is disregarded by having the hash function choose the lower numerical value for the exact hash of both a k-mer and its reverse complement. This behavior is configurable via a compile-time option.

3.4.5 Comparing with other k-mer counting programs

We generated k-mer abundance distribution from five soil metagenomic reads data sets of different sizes using khmer, Tallymer, Jellyfish, DSK, BFCounter, KMC, Turtle and KAnalyze. If the software does not include function to generate k-mer abundance distribution directly, we output the frequency of each k-mer in an output file. We fixed k at 22 unless otherwise noted.

3.4.5.0.1 khmer: For khmer, we set hash table sizes to fix the false positive rate at either 1%, 5% or 20%, and used 8 threads in loading the data.

We did the khmer random-access k-mer counting benchmark with a custom-written Python script `khmer-count-kmers` which loaded the database file and then used the Python API to query each k-mer individually.

3.4.5.0.2 Tallymer: Tallymer is from the genometools package version 1.3.4. For the `suffixerator` subroutine we used: `-dna -pl -tis -suf -lcp`.

We use the `mkindex` subroutine to generate k-mer abundance distribution, we used: `-mersize 22`.

The Tallymer random access k-mer counting benchmark was done using the 'tallymer search' routine against both strands; see the script `tallymer-search.sh`.

3.4.5.0.3 Jellyfish: The Jellyfish version used was 1.1.10 and the multithreading option is set to 8 threads.

Jellyfish uses a hash table to store the k-mers and the size of the hash table can be modified by the user. When the specified hash table fills up, Jellyfish writes it to the hard disk and initializes a new hash table. Here we use a similar strategy as in [72] and chose the minimum size of the hash tables for Jellyfish so that all k-mers were stored in memory.

We ran Jellyfish with the options as below:

`jellyfish count -m 22 -c 2 -C` for k=22.

The Jellyfish random access k-mer counting benchmark was performed using the 'query' routine and querying against both strands; see the script `jelly-search.sh`.

3.4.5.0.4 DSK: We ran DSK with default parameters with `-histo` option to generate k-mer abundance distribution. The DSK version used was 1.5031.

3.4.5.0.5 BFCounter: The BFcounter version used was 1.0 and the multithreading option is set to 8 threads.

We ran BFCounter `count` subroutine with the options as below:

`BFCounter count -k 22 -t 8 -c 100000`. `-n` option representing the estimated number of k-mers is adjusted to the different test data sets.

This subroutine produces the actual count of k-mers in input files.

We ran BFCounter `dump` subroutine with the options as below: `BFCounter dump -k 22`.

This subroutine can write k-mer occurrences into a tab-separated text file.

3.4.5.0.6 KMC: The KMC version used was 0.3. We ran both `kmc` and `kmc_dump` subroutines with default parameters.

3.4.5.0.7 Turtle: The Turtle version used was 0.3. We ran `scTurtle32` with the multithreading option set to 8 threads and `-n` option representing expected number of frequent k-mers is adjusted to different test data sets.

3.4.5.0.8 KAnalyze: The KAnalyze version used was 0.9.3. We ran `count` subroutine with default parameters.

Figure Legends

Figure 3.1: **Comparison of the time it takes for k-mer counting tools to calculate k-mer abundance histograms, with time (y axis, in seconds) against data set size (in number of reads, x axis). All programs executed in time approximately linear with the number of input reads.**

Figure 3.2: **Memory usage of k-mer counting tools when calculating k-mer abundance histograms, with maximum resident program size (y axis, in GB) plotted against the total number of distinct k-mers in the data set (x axis, billions of k-mers).**

Figure 3.3: **Disk storage usage of different k-mer counting tools to calculate k-mer abundance histograms in GB (y axis), plotted against the number of distinct k-mers in the data set (x axis). *Note that khmer does not use the disk during counting or retrieval, although its hash tables can be saved for reuse.**

Figure 3.4: **Time for several k-mer counting tools to retrieve the counts of 9.7m randomly chosen k-mers (y axis), plotted against the number of distinct k-mers in the data set being queried (x axis). BFCounter, DSK, Turtle, KAnalyze, and KMC do not support this functionality.**

Figure 3.5: **Relation between average miscount — amount by which the count for k-mers is incorrect — on the y axis, plotted against false positive rate (x axis), for five data sets. The five data sets were chosen to have the same total number of distinct k-mers: one metagenome data set; a set of randomly generated k-mers; a set of reads, chosen with 3x coverage and 1% error, from a randomly generated genome; a simulated set of error-free reads (3x) chosen from a randomly generated genome and a set of *E. coli* reads.**

Figure 3.6: **Relation between percent miscount — amount by which the count for k-mers is incorrect relative to its true count — on the y axis, plotted against false positive rate (x axis), for five data sets. The five data sets are the same as in Figure 3.5.**

Figure 3.7: **Number of unique k-mers (y axis) by starting position within read (x axis) in an untrimmed *E. coli* 100-bp Illumina shotgun data set, for k=17 and k=32. The increasing numbers of unique k-mers are a sign of the increasing sequencing error towards the 3' end of reads. Note that there are only 69 starting positions for 32-mers in a 100 base read.**

Tables

Table 3.1: **Benchmark soil metagenome data sets for k-mer counting performance, taken from [45].**

Data set	size of file (GB)	number of reads	number of distinct k-mers	total number of k-mers
subset 1	1.90	9,744,399	561,178,082	630,207,985
subset 2	2.17	19,488,798	1,060,354,144	1,259,079,821
subset 3	3.14	29,233,197	1,445,923,389	1,771,614,378
subset 4	4.05	38,977,596	1,770,589,216	2,227,756,662
entire data set	5.00	48,721,995	2,121,474,237	2,743,130,683

Table 3.2: **Data sets used for analyzing miscounts.**

Data set	Size of data set file	Number of total k-mers	Number of distinct k-mers
Real metagenomics reads	7.01M	2,917,200	1,944,996
Totally random reads with randomly generated k-mers	3.53M	2,250,006	1,973,059
Simulated reads from simulated genome with error	5.92M	3,757,479	2,133,592
Simulated reads from simulated genome without error	9.07M	5,714,973	1,989,644
Real *E. coli* reads	4.85M	4,004,911	2,079,302

Table 3.3: **Iterative low-memory k-mer trimming. The results of trimming reads at unique (erroneous) k-mers from a 5m read *E. coli* data set (1.4 GB) in under 30 MB of RAM. After each iteration, we measured the total number of distinct k-mers in the data set, the total number of unique (and likely erroneous) k-mers remaining, and the number of unique k-mers present at the 3' end of reads.**

	FP rate	bases trimmed	distinct k-mers	unique k-mers	unique k-mers at
untrimmed	-	-	41.6m	34.1m	30.4%
khmer iteration 1	80.0%	13.5%	13.3m	6.5m	29.8%
khmer iteration 2	40.2%	1.7%	7.6m	909.9k	12.3%
khmer iteration 3	25.4%	0.3%	6.8m	168.1k	3.1%
khmer iteration 4	23.2%	0.1%	6.7m	35.8k	0.7%
khmer iteration 5	22.8%	0.0%	6.6m	7.9k	0.2%
khmer iteration 6	22.7%	0.0%	6.6m	1.9k	0.0%
filter by FASTX	-	9.1%	26.6m	20.3m	26.3%
filter by seqtk(default)	-	8.9%	17.7m	12.1m	12.3%
filter by seqtk(-q 0.01)	-	15.4%	9.9m	5.1m	5.2%
filter by seqtk(-b 3 -e 5)	-	8.0%	34.5m	27.7m	25.3%

Table 3.4: **Low-memory digital normalization. The results of digitally normalizing a 5m read *E. coli* data set (1.4 GB) to C=20 with k=20 under several memory usage/false positive rates. The false positive rate (column 1) is empirically determined. We measured reads remaining, number of "true" k-mers missing from the data at each step, and the number of total k-mers remaining. Note: at high false positive rates, reads are erroneously removed due to inflation of k-mer counts.**

memory	FP rate	retained reads	retained reads %	true k-mers missing	total k-mers
before diginorm	-	5,000,000	100.0%	170	41.6m
2400 MB	0.0%	1,656,518	33.0%	172	28.1m
240 MB	2.8%	1,655,988	33.0%	172	28.1m
120 MB	18.0%	1,652,273	33.0%	172	28.1m
60 MB	59.1%	1,633,182	32.0%	172	27.9m
40 MB	83.2%	1,602,437	32.0%	172	27.6m
20 MB	98.8%	1,460,936	29.0%	172	25.7m
10 MB	100.0%	1,076,958	21.0%	185	20.9m

Table 3.5: ***E. coli* genome assembly after low-memory digital normalization. A comparison of assembling reads digitally normalized with low memory/high false positive rates. The reads were digitally normalized to C=20 (see [8] for more information) and were assembled using Velvet. We measured total length of assembly, as well as percent of true MG1655 genome covered by the assembly using QUAST.**

memory	FP rate	N contigs	total length(bases)	% of true genome covered
before diginorm	-	106	4,546,051	97.84%
2400 MB	0.0%	617	4,549,235	98.05%
240 MB	2.8%	87	4,549,253	98.04%
120 MB	18.0%	86	4,549,335	98.04%
60 MB	59.1%	90	4,548,619	98.03%
40 MB	83.2%	89	4,550,599	98.11%
20 MB	98.8%	85	4,550,014	98.04%
10 MB	100.0%	97	4,545,871	97.97%

Chapter 4

diginorm: need rewriting, plan: only keep the materilas that are more relevant to IGS

4.1 Introduction

The ongoing improvements in DNA sequencing technologies have led to a new problem: how do we analyze the resulting large sequence data sets quickly and efficiently? These data sets contain millions to billions of short reads with high error rates and substantial sampling biases [73]. The vast quantities of deep sequencing data produced by these new sequencing technologies are driving computational biology to extend and adapt previous approaches to sequence analysis. In particular, the widespread use of deep shotgun sequencing on previously unsequenced genomes, transcriptomes, and metagenomes, has resulted in the development of several new approaches to *de novo* sequence assembly [74].

There are two basic challenges in analyzing short-read sequences from shotgun sequencing. First, deep sequencing is needed for complete sampling. This is because shotgun sequencing samples randomly from a population of molecules; this sampling is biased by sample content and sample preparation, requiring even deeper sequencing. A human genome may require 100x coverage or more for near-complete sampling, leading to shotgun data sets 300 GB or larger in size[37]. Since the lowest abundance molecule determines the depth of coverage required for complete sampling, transcriptomes and metagenomes containing rare population elements can also require similarly deep sequencing.

The second challenge to analyzing short-read shotgun sequencing is the high error rate. For example, the Illumina GAII sequencer has a 1-2% error rate, yielding an average of one base error in every 100 bp of data [73]. The total number of errors grows linearly with the amount of data generated, so these errors usually dominate novelty in large data sets [21]. Tracking this novelty and resolving errors is computationally expensive.

These large data sets and high error rates combine to provide a third challenge: it is now straightforward to generate data sets that cannot easily be analyzed [101]. While hardware approaches to scaling existing algorithms are emerging, sequencing capacity continues to grow faster than computational capacity [118]. Therefore, new algorithmic approaches to analysis are needed.

Many new algorithms and tools have been developed to tackle large and error-prone short-read shotgun data sets. A new class of alignment tools, most relying on the Burrows-Wheeler transform, has been created specifically to do ultra-fast short-read alignment to reference sequence [120]. In cases where a reference sequence does not exist and must be assembled *de novo* from the sequence data, a number of new assemblers have been written, including ABySS, Velvet, SOAPdenovo, ALLPATHS, SGA, and Cortex [113, 131, 59, 37, 112, 49]. These assemblers rely on theoretical advances to store and assemble large amounts of data [20, 111]. As short-read sequencing has been applied to single cell genomes, transcriptomes, and metagenomes, yet another generation of assemblers has emerged to handle reads from abundance-skewed populations of molecules; these tools, including Trinity, Oases, MetaVelvet, Meta-IDBA, and Velvet-SC, adopt local models of sequence coverage to help build assemblies [39, 107, 80, 84, 16]. In addition, several ad hoc strategies have also been applied to reduce variation in sequence content from whole-genome amplification [95, 127]. Because these tools all rely on k-mer approaches and require exact matches to construct overlaps between sequences, their performance is very sensitive to the number of errors present in the underlying data. This sensitivity to errors has led to the development of a number of error removal and correction approaches that preprocess data prior to assembly or mapping [71, 11, 52].

Below, we introduce "digital normalization", a single-pass algorithm for elimination of redundant reads in data sets. Critically, no reference sequence is needed to apply digital normalization. Digital normalization is inspired by experimental normalization techniques developed for cDNA library preparation, in which hybridization kinetics are exploited to reduce the copy number of abundant transcripts prior to sequencing[3, 114]. *Digital* normalization works after sequencing data has been generated, progressively removing high-coverage reads from shotgun data sets. This normalizes average coverage to a specified value, reducing sampling variation while removing reads, and also removing the many errors contained *within* those reads. This data and error reduction results in dramatically decreased computational requirements for *de novo* assembly. Moreover, unlike experimental normalization where abundance information is removed prior to sequencing, in digital normalization this information can be recovered from the unnormalized reads.

We present here a fixed-memory implementation of digital normalization that operates in time linear with the size of the input data. We then demonstrate its effectiveness for reducing compute requirements for *de novo* assembly on several real data sets. These data sets include *E. coli* genomic data, data from two single-cell MD-amplified microbial genomes, and yeast and mouse mRNAseq.

4.2 Results

4.2.1 Estimating sequencing depth without a reference assembly

Short-read assembly requires deep sequencing to systematically sample the source genome, because shotgun sequencing is subject to both random sampling variation and systematic sequencing biases. For example, 100x sampling of a human genome is required for recovery of 90% or more of the genome in contigs $>$ 1kb [37]. In principle much of this high-coverage data is redundant and could be eliminated without consequence to the final assembly, but determining which reads to eliminate requires a per-read estimate of coverage. Traditional approaches estimate coverage by mapping reads to an assembly. This presents a chicken-and-egg problem: to determine which regions are oversampled, we must already have an assembly!

We may calculate a *reference-free* estimate of genome coverage by looking at the k-mer abundance distribution within individual reads. First, observe that k-mers, DNA words of a fixed length k, tend to have similar abundances within a read: this is a well-known property of k-mers that stems from each read originating from a single source molecule of DNA. The more times a region is sequenced, the higher the abundance of k-mers from that region would be. In the absence of errors, average k-mer abundance could be used as an estimate of the depth of coverage for a particular read (Figure 4.1, “no errors” line). However, when reads contain random substitution or indel errors from sequencing, the k-mers overlapping these errors will be of lower abundance; this feature is often used in k-mer based error correction approaches [52]. For example, a single substitution will introduce k low-abundance k-mers within a read. (Figure 4.1, “single substitution error” line). However, for small k and reads of length L where $L > 3k - 1$, a single substitution error will not skew the *median* k-mer abundance. Only when multiple substitution errors are found in a single read will the median k-mer abundance be affected (Figure 4.1, “multiple substitution errors”).

Using a fixed-memory CountMin Sketch data structure to count k-mers (see Methods and [24]), we find that median k-mer abundance correlates well with mapping-based coverage for artificial and real genomic data sets. There is a strong correlation between median k-mer abundance and mapping-based coverage both for simulated 100-base reads generated with 1% error from a 400kb artificial genome sequence ($r^2 = 0.79$; also see Figure 4.2a), as well as for real short-read data from *E. coli* ($r^2 = 0.80$, also see Figure 4.2b). This correlation also holds for simulated and real mRNAseq data: for simulated transcriptome data, $r^2 = 0.93$ (Figure 4.3a), while for real mouse transcriptome data, $r^2 = 0.90$ (Figure 4.3b). Thus the median k-mer abundance of a read correlates well with mapping-based estimates of read coverage.

4.2.2 Eliminating redundant reads reduces variation in sequencing depth

Deeply sequenced genomes contain many highly covered loci. For example, in a human genome sequenced to 100x average coverage, we would expect 50% or more of the reads to have a coverage greater than 100. In practice, we need many fewer of these reads to assemble the source locus.

Using the median k-mer abundance estimator discussed above, we can examine each read in the data set progressively to determine if it is high coverage. At the beginning of a shotgun data set, we would expect many reads to be entirely novel and have a low estimated coverage. As we proceed through the data set, however, average coverage will increase and many reads will be from loci that we have already sampled sufficiently.

Suppose we choose a coverage threshold C past which we no longer wish to collect reads. If we only keep reads whose estimated coverage is less than C, and discard the rest, we will reduce the average coverage of the data set to C. This procedure is algorithmically straightforward to execute: we examine each read's estimated coverage, and retain only those whose coverage is less than C. The following pseudocode provides one approach:

```
for read in dataset:
    if estimated_coverage(read) < C:
        accept(read)
    else:
        discard(read)
```

where accepted reads contribute to the `estimated_coverage` function. Note that for any data set with an average coverage $> 2C$, this has the effect of discarding the majority of reads. Critically, low-coverage reads, especially reads from undersampled regions, will always be retained.

The net effect of this procedure, which we call digital normalization, is to normalize the coverage distribution of data sets. In Figure 4.4a, we display the estimated coverage of an *E. coli* genomic data set, a *S. aureus* single-cell MD-amplified data set, and an MD-amplified data set from an uncultured *Deltaproteobacteria*, calculated by mapping reads to the known or assembled reference genomes (see [16] for the data source). The wide variation in coverage for the two MDA data sets is due to the amplification procedure [116]. After normalizing to a k-mer coverage of 20, the high coverage loci are systematically shifted to an average mapping coverage of 26, while lower-coverage loci remain at their previous coverage. This smooths out coverage of the overall data set.

At what rate are sequences retained? For the *E. coli* data set, Figure 4.5 shows the fraction of sequences retained by digital normalization as a function of the total number of reads examined when normalizing to C=20 at k=20. There is a clear saturation effect showing that as more reads are examined, a smaller fraction of reads is retained; by 5m reads, approximately 50-100x coverage of *E. coli*, under 30% of new reads are kept. This demonstrates that as expected, only a small amount of novelty (in the form of either new information, or the systematic accumulation of errors) is being observed with increasing sequencing depth.

4.2.3 Digital normalization retains information while discarding both data and errors

The 1-2% per-base error rate of next-generation sequencers dramatically affect the total number of k-mers. For example, in the simulated genomic data of 200x, a 1% error rate leads to approximately 20 new k-mers for each error, yielding 20-fold more k-mers in the reads than are truly present in the genome (Table 1, row 1). This in turn dramatically increases the memory requirements for tracking and correcting k-mers [21]. This is a well-known problem with de Bruijn graph approaches, in which erroneous nodes or edges quickly come to dominate deep sequencing data sets.

When we perform digital normalization on such a data set, we eliminate the vast majority of these k-mers (Table 4.1, row 1). This is because we are accepting or rejecting entire reads;

in going from 200x random coverage to 20x systematic coverage, we discard 80% of the reads containing 62% of the errors (Table 4.1, row 1). For reads taken from a skewed abundance distribution, such as with MDA or mRNAseq, we similarly discard many reads, and hence many errors (Table 4.1, row 2). In fact, in most cases the process of sequencing fails to recover more true k-mers (Table 4.1, middle column, parentheses) than digital normalization discards (Table 4.1, fourth column, parentheses).

The net effect of digital normalization is to retain nearly all *real* k-mers, while discarding the majority of erroneous k-mers – in other words, digital normalization is discarding *data* but not *information*. This rather dramatic elimination of erroneous k-mers is a consequence of the high error rate present in reads: with a 1% per-base substitution error rate, each 100-bp read will have an average of one substitution error. Each of these substitution errors will introduce up to k erroneous k-mers. Thus, for each read we discard as redundant, we also eliminate an average of k erroneous k-mers.

We may further eliminate erroneous k-mers by removing k-mers that are rare across the data set; these rare k-mers tend to result from substitution or indel errors [52]. We do this by first counting all the k-mers in the accepted reads during digital normalization. We then execute a second pass across the accepted reads in which we eliminate the 3' ends of reads at low-abundance k-mers. Following this error reduction pass, we execute a second round of digital normalization (a third pass across the data set) that further eliminates redundant data. This three-pass protocol eliminates additional errors and results in a further decrease in data set size, at the cost of very few real k-mers in genomic data sets (Table 4.2).

Why use this three-pass protocol rather than simply normalizing to the lowest desired coverage in the first pass? We find that removing low-abundance k-mers after a single normalization pass to $C \approx 5$ removes many more *real* k-mers, because there will be many regions in the genome that by chance have yielded 5 reads with errors in them. If these erroneous k-mers are removed in the abundance-trimming step, coverage of the corresponding regions is eliminated. By normalizing to a higher coverage of 20, removing errors, and only then reducing coverage to 5, digital normalization can retain accurate reads for most regions. Note that this three-pass protocol is not considerably more computationally expensive than the single-pass protocol: the first pass discards the majority of data and errors, so later passes are less time and memory intensive than the first pass.

Interestingly, this three-pass protocol removed many more real k-mers from the simulated mRNAseq data than from the simulated genome – 351 of 48,100 (0.7%) real k-mers are lost from the mRNAseq, vs 4 of 399,981 lost (.000001%) from the genome (Table 4.2). While still only a tiny fraction of the total number of real k-mers, the difference is striking – the simulated mRNAseq sample loses k-mers at almost 1000-fold the rate of the simulated genomic sample. Upon further investigation, all but one of the lost k-mers were located within 20 bases of the ends of the source sequences; see Figure 4.6. This is because digital normalization cannot distinguish between erroneous k-mers and k-mers that are undersampled due to edge effects. In the case of the simulated genome, which was generated as one large chromosome, the effect is negligible, but the simulated transcriptome was generated as 100 transcripts of length 500. This added 99 end sequences over the genomic simulation, which in turn led to many more lost k-mers.

While the three-pass protocol is effective at removing erroneous k-mers, for some samples it may be too stringent. For example, the mouse mRNAseq data set contains only 100m reads, which may not be enough to thoroughly sample the rarest molecules; in this case the abundance trimming would remove real k-mers as well as erroneous k-mers. Therefore we used the single-pass digital normalization for the yeast and mouse transcriptomes. For these two samples we can also see that the first-pass digital normalization is extremely effective, eliminating essentially all of the erroneous k-mers (Table 4.1, rows 4 and 5.)

4.2.4 Digital normalization scales assembly of microbial genomes

We applied the three-pass digital normalization and error trimming protocol to three real data sets from Chitsaz et al (2011) [16]. The first pass of digital normalization was performed in 1gb of memory and took about 1 min per million reads. For all three samples, the number of reads remaining after digital normalization was reduced by at least 30-fold, while the memory and time requirements were reduced 10-100x.

Despite this dramatic reduction in data set size and computational requirements for assembly, both the *E. coli* and *S. aureus* assemblies overlapped with the known reference sequence by more than 98%. This confirms that little or no information was lost during the process of digital normalization; moreover, it appears that digital normalization does not significantly affect the assembly results. (Note that we did not perform scaffolding, since the digital normalization algorithm does not take into account paired-end sequences, and could mislead scaffolding approaches. Therefore, these results cannot directly be compared to those in Chitsaz et al. (2011) [16].)

The *Deltaproteobacteria* sequence also assembled well, with 98.8% sequence overlap with the results from Chitsaz et al. Interestingly, only 30kb of the sequence assembled with Velvet-SC in Chitsaz et al. (2011) was missing, while an additional 360kb of sequence was assembled only in the normalized samples. Of the 30kb of missing sequence, only 10% matched via TBLASTX to a nearby *Deltaproteobacteria* assembly, while more than 40% of the additional 360kb matched to the same *Deltaproteobacteria* sample. Therefore these additional contigs likely represents real sequence, suggesting that digital normalization is competitive with Velvet-SC in terms of sensitivity.

4.2.5 Digital normalization scales assembly of transcriptomes

We next applied single-pass digital normalization to published yeast and mouse mRNAseq data sets, reducing them to 20x coverage at k=20 [39]. Digital normalization on these samples used 8gb of memory and took about 1 min per million reads. We then assembled both the original and normalized sequence reads with Oases and Trinity, two *de novo* transcriptome assemblers (Table 4.4) [107, 39].

For both assemblers the computational resources necessary to complete an assembly were reduced (Table 4.4), but normalization had different effects on performance for the different samples. On the yeast data set, time and memory requirements were reduced significantly, as for Oases running on mouse. However, while Trinity's runtime decreased by a factor of three on the normalized mouse data set, the memory requirements did not decrease significantly. This may be because the mouse transcriptome is 5-6 times larger than the yeast transcriptome, and so the mouse mRNAseq is lower coverage overall; in this case we would expect fewer errors to be removed by digital normalization.

The resulting assemblies differed in summary statistics (Table 4.5). For both yeast and mouse, Oases lost 5-10% of total transcripts and total bases when assembling the normalized data. However, Trinity *gained* transcripts when assembling the normalized yeast and mouse data, gaining about 1% of total bases on yeast and losing about 1% of total bases in mouse. Using a local-alignment-based overlap analysis (see Methods) we found little difference in sequence content between the pre- and post- normalization assemblies: for example, the normalized Oases assembly had a 98.5% overlap with the unnormalized Oases assembly, while the normalized Trinity assembly had a 97% overlap with the unnormalized Trinity assembly.

To further investigate the differences between transcriptome assemblies caused by digital normalization, we looked at the sensitivity with which long transcripts were recovered post-normalization. When comparing the normalized assembly to the unnormalized assembly in yeast, Trinity lost only 3% of the sequence content in transcripts greater than 300 bases, but 10% of the sequence content in transcripts greater than 1000 bases. However, Oases lost less than 0.7% of sequence content at 300 and 1000 bases. In mouse, we see the same pattern. This suggests that the change in summary statistics for Trinity is caused by fragmentation of long transcripts into shorter transcripts, while the difference for Oases is caused by loss of splice variants. Indeed, this loss of splice variants should be expected, as there are many low-prevalence splice variants present in deep sequencing data [88]. Interestingly, in yeast we recover *more* transcripts after digital normalization; these transcripts appear to be additional splice variants.

The difference between Oases and Trinity results show that Trinity is more sensitive to digital normalization than Oases: digital normalization seems to cause Trinity to fragment long transcripts. Why? One potential issue is that Trinity only permits k=26 for assembly, while normalization was performed at k=20; digital normalization may be removing 26-mers that are important for Trinity's path finding algorithm. Alternatively, Trinity may be more sensitive than Oases to the change in coverage caused by digital normalization. Regardless, the strong performance of Oases on digitally normalized samples, as well as the high retention of k-mers (Table 4.1) suggests that the primary sequence content for the transcriptome remains present in the normalized reads, although it is recovered with different effectiveness by the two assemblers.

4.3 Discussion

4.3.1 Digital normalization dramatically scales *de novo* assembly

The results from applying digital normalization to read data sets prior to *de novo* assembly are extremely good: digital normalization reduces the computational requirements (time and memory) for assembly considerably, without substantially affecting the assembly results. It does this in two ways: first, by removing the majority of reads without significantly affecting the true k-mer content of the data set. Second, by eliminating these reads, digital normalization also eliminates sequencing errors contained within those reads, which otherwise would add significantly to memory usage in assembly [21].

Digital normalization also lowers computational requirements by eliminating most repetitive sequence in the data set. Compression-based approaches to graph storage have demonstrated that compressing repetitive sequence also effectively reduces memory and compute requirements [89, 112]. Note however that *eliminating* many repeats may also have its negatives (discussed below).

Digital normalization should be an effective preprocessing approach for most assemblers. In particular, the de Bruijn graph approach used in many modern assemblers relies on k-mer content, which is almost entirely preserved by digital normalization (see Tables 4.1 and 4.2) [74].

4.3.2 A general strategy for normalizing coverage

Digital normalization is a general strategy for systematizing coverage in shotgun sequencing data sets by using per-locus downsampling, albeit without any prior knowledge of reference loci. This yields considerable theoretical and practical benefits in the area of *de novo* sequencing and assembly.

In theoretical terms, digital normalization offers a general strategy for changing the scaling behavior of sequence assembly. Assemblers tend to scale poorly with the number of reads: in particular, de Bruijn graph memory requirements scale linearly with the size of the data set due to the accumulation of errors, although others have similarly poor scaling behavior (e.g. quadratic time in the number of reads) [74]. By calculating per-locus coverage in a way that is insensitive to errors, digital normalization converts genome assembly into a problem that scales with the complexity of the underlying sample - i.e. the size of the genome, transcriptome, or metagenome.

Digital normalization also provides a general strategy for applying online or streaming approaches to analysis of *de novo* sequencing data. The basic algorithm presented here is explicitly a single-pass or streaming algorithm, in which the entire data set is never considered as a whole; rather, a partial "sketch" of the data set is retained and used for progressive filtering. Online algorithms and sketch data structures offer significant opportunities in situations where data sets are too large to be conveniently stored, transmitted, or analyzed [78]. This can enable increasingly efficient downstream analyses. Digital normalization can be applied in any situation where the abundance of particular sequence elements is either unimportant or can be recovered more efficiently after other processing, as in assembly.

The construction of a simple, reference-free measure of coverage on a per-read basis offers opportunities to analyze coverage and diversity with an assembly-free approach. Genome and transcriptome sequencing is increasingly being applied to non-model organisms and ecological communities for which there are no reference sequences, and hence no good way to estimate underlying sequence complexity. The reference-free counting technique presented here provides a method for determining community and transcriptome complexity; it can also be used to progressively estimate sequencing depth.

More pragmatically, digital normalization also scales existing assembly techniques dramatically. The reduction in data set size afforded by digital normalization may also enable the application of more computationally expensive algorithms such as overlap-layout-consensus assembly approaches to short-read data. Overall, the reduction in data set size, memory requirements, and time complexity for contig assembly afforded by digital normalization could lead to the application of more complex heuristics to the assembly problem.

4.3.3 Digital normalization drops terminal k-mers and removes isoforms

Our implementation of digital normalization does discard some real information, including terminal k-mers and low-abundance isoforms. Moreover, we predict a number of other failure modes: for example, because k-mer approaches demand strict sequence identity, data sets from highly polymorphic organisms or populations will perform more poorly than data sets from low-variability samples. Digital normalization also discriminates against highly repetitive sequences. We note that these problems traditionally have been challenges for assembly strategies: recovering low-abundance isoforms from mRNAseq, assembling genomes from highly polymorphic organisms, and assembling across repeats are all difficult tasks, and improvements in these areas continue to be active areas of research [29, 81, 40]. Using an alignment-based approach to estimating coverage, rather than a k-mer based approach, could provide an alternative implementation that would improve performance on errors, splice variants, and terminal k-mers. Our current approach also ignores quality scores; a "q-mer" counting approach as in Quake, in which k-mer counts are weighted by quality scores, could easily be adapted [52].

Another concern for normalizing deep sequencing data sets is that, with sufficiently deep sequencing, sequences with many errors will start to accrue. This underlies the continued accumulation of sequence data for *E. coli* observed in Figure 4.5. Assemblers may be unable to distinguish between this false sequence and the error-free sequences, for sufficiently deep data sets. This accumulation of erroneous sequences is again caused by the use of k-mers to detect similarity, and is one reason why exploring local alignment approaches (discussed below) may be a good future direction.

4.3.4 Applying assembly algorithms to digitally normalized data

The assembly problem is challenging for several reasons: many formulations are computationally complex (NP-hard), and practical issues of both genome content and sequencing, such as repetitive sequence, polymorphisms, short reads and high error rates, challenge assembly approaches [79]. This has driven the development of heuristic approaches to resolving complex regions in assemblies. Several of these heuristic approaches use the abundance information present in the reads to detect and resolve repeat regions; others use pairing information from paired-end and mate-pair sequences to resolve complex paths. Digital normalization aggressively removes abundance information, and we have not yet adapted it to paired-end sequencing data sets; this could and should affect the quality of assembly results! Moreover, it is not clear what effect different coverage (C) and k-mer (k) values have on assemblers. In practice, for at least one set of k-mer size k and normalized coverage C parameters, digital normalization seems to have little negative effect on the final assembled contigs. Further investigation of the effects of varying k and C relative to specific assemblers and assembler parameters will likely result in further improvements in assembly quality.

A more intriguing notion than merely using digital normalization as a pre-filter is to specifically adapt assembly algorithms and protocols to digitally normalized data. For example, the reduction in data set size afforded by digital normalization may make overlap-layout-consensus approaches computationally feasible for short-read data [74]. Alternatively, the quick and inexpensive generation of contigs from digitally normalized data could be used prior to a separate scaffolding step, such as those supported by SGA and Bambus2 [111, 54]. Digital normalization offers many future directions for improving assembly.

4.4 Conclusions

Digital normalization is an effective demonstration that much of short-read shotgun sequencing is redundant. Here we have shown this by normalizing samples to 5-20x coverage while recovering complete or nearly complete contig assemblies. Normalization is substantially different from uniform downsampling: by doing downsampling in a locus-specific manner, we retain low coverage data. Previously described approaches to reducing sampling variation rely on *ad hoc* parameter measures and/or an initial round of assembly and have not been shown to be widely applicable [95, 127].

We have implemented digital normalization as a *prefilter* for assembly, so that any assembler may be used on the normalized data. Here we have only benchmarked a limited set of assemblers – Velvet, Oases, and Trinity – but in theory digital normalization should apply to any assembler. De Bruijn and string graph assemblers such as Velvet, SGA, SOAPdenovo, Oases, and Trinity are especially likely to work well with digital normalized data, due to the underlying reliance on k-mer overlaps in these assemblers.

4.4.1 Digital normalization is widely applicable and computationally convenient

Digital normalization can be applied *de novo* to *any* shotgun data set. The approach is extremely computationally convenient: the runtime complexity is linear with respect to the data size, and perhaps more importantly it is *single-pass*: the basic algorithm does not need to look at any read more than once. Moreover, because reads accumulate sub-linearly, errors do not accumulate quickly and overall memory requirements for digital normalization should grow slowly with data set size. Note also that while the algorithm presented here is not perfectly parallelizable, efficient distributed k-mer counting is straightforward and it should be possible to scale digital normalization across multiple machines [102].

The first pass of digital normalization is implemented as an online streaming algorithm in which reads are examined once. Streaming algorithms are useful for solving data analysis problems in which the data are too large to easily be transmitted, processed, or stored. Here, we implement the streaming algorithm using a fixed memory data "sketch" data structure, CountMin Sketch. By combining a single-pass algorithm with a fixed-memory data structure, we provide a data reduction approach for sequence data analysis with both (linear) time and (constant) memory guarantees. Moreover, because the false positive rate of the CountMin Sketch data structure is well understood and easy to predict, we can provide *data quality* guarantees as well. These kinds of guarantees are immensely valuable from an algorithmic perspective, because they provide a robust foundation for further work [78]. The general concept of removing redundant *data* while retaining *information* underlies "lossy compression", an approach used widely in image processing and video compression. The concept of lossy compression has broad applicability in sequence analysis. For example, digital normalization could be applied prior to homology search on unassembled reads, potentially reducing the computational requirements for e.g. BLAST and HMMER without loss of sensitivity. Digital normalization could also help merge multiple different read data sets from different read technologies, by discarding entirely redundant sequences and retaining only sequences containing "new" information. These approaches remain to be explored in the future.

4.5 Methods

All links below are available electronically through ged.msu.edu/papers/2012-diginorm/, in addition to the archival locations provided.

4.5.1 Data sets

The *E. coli*, *S. aureus*, and *Deltaproteobacteria* data sets were taken from Chitsaz et al. [16], and downloaded from bix.ucsd.edu/projects/singlecell/. The mouse data set was published by Grabherr et al. [39] and downloaded from trinityrnaseq.sf.net/. All data sets were used without modification. The complete assemblies, both pre- and post-normalization, for the *E. coli*, *S. aureus*, the uncultured *Deltaproteobacteria*, mouse, and yeast data sets are available from ged.msu.edu/papers/2012-diginorm/.

The simulated genome and transcriptome were generated from a uniform AT/CG distribution. The genome consisted of a single chromosome 400,000 bases in length, while the transcriptome consisted of 100 transcripts of length 500. 100-base reads were generated uniformly from the genome to an estimated coverage of 200x, with a random 1% per-base error. For the transcriptome, 1 million reads of length 100 were generated from the transcriptome at relative expression levels of 10, 100, and 1000, with transcripts assigned randomly with equal probability to each expression group; these reads also had a 1% per-base error.

4.5.2 Scripts and software

All simulated data sets and all analysis summaries were generated by Python scripts, which are available at github.com/ged-lab/2012-paper-diginorm/. Digital normalization and k-mer analyses were performed with the khmer software package, written in C++ and Python, available at github.com/ged-lab/khmer/, tag '2012-paper-diginorm'. khmer also relies on the screed package for loading sequences, available at github.com/ged-lab/screed/, tag '2012-paper-diginorm'. khmer and screed are Copyright (c) 2010 Michigan State University, and are free software available for distribution, modification, and redistribution under the BSD license.

Mapping was performed with bowtie v0.12.7 [56]. Genome assembly was done with velvet 1.2.01 [131]. Transcriptome assembly was done with velvet 1.1.05/oases 0.1.22 and Trinity, head of branch on 2011.10.29. Graphs and correlation coefficients were generated using matplotlib v1.1.0, numpy v1.7, and ipython notebook v0.12 [86]. The ipython notebook file and data analysis scripts necessary to generate the figures are available at github.com/ged-lab/2012-paper-diginorm/.

4.5.3 Analysis parameters

The khmer software uses a CountMin Sketch data structure to count k-mers, which requires a fixed memory allocation [24]. In all cases the memory usage was fixed such that the calculated false positive rate was below 0.01. By default k was set to 20.

Genome and transcriptome coverage was calculated by mapping all reads to the reference with bowtie (`-a --best --strata`) and then computing the per-base coverage in the reference. Read coverage was computed by then averaging the per-base reference coverage for each position in the mapped read; where reads were mapped to multiple locations, a reference location was chosen randomly for computing coverage. Median k-mer counts were computed with khmer as described in the text. Artificially high counts resulting from long stretches of Ns were removed after the analysis. Correlations between median k-mer counts and mapping coverage were computed using numpy.corrcoef; see calc-r2.py script.

4.5.4 Normalization and assembly parameters

For Table 4.3, the assembly k parameter for Velvet was k=45 for *E. coli*; k=41 for *S. aureus* single cell; and k=39 for *Deltaproteobacteria* single cell. Digital normalization on the three bacterial samples was done with `-N 4 -x 2.5e8 -k 20`, consuming 1gb of memory. Post-normalization k parameters for Velvet assemblies were k=37 for *E. coli*, k=27 for *S. aureus*, and k=27 for *Deltaproteobacteria*. For Table 4.4, the assembly k parameter for Oases was k=21 for yeast and k=23 for mouse. Digital normalization on both mRNAseq samples was done with `-N 4 -x 2e9 -k 20`, consuming 8gb of memory. Assembly of the single-pass normalized mRNAseq was done with Oases at k=21 (yeast) and k=19 (mouse).

4.5.5 Assembly overlap and analysis

Assembly overlap was computed by first using NCBI BLASTN to build local alignments for two assemblies, then filtering for matches with bit scores > 200, and finally computing the fraction of bases in each assembly with at least one alignment. Total fractions were normalized to self-by-self BLASTN overlap identity to account for BLAST-specific sequence filtering. TBLASTX comparison of the *Deltaproteobacteria* SAR324 sequence was done against another assembled SAR324 sequence, acc AFIA01000002.1.

4.5.6 Compute requirement estimation

Execution time was measured using real time from Linux bash 'time'. Peak memory usage was estimated either by the 'memusg' script from stackoverflow.com, peak-memory-usage-of-a-linux-unix-process, included in the khmer repository; or by the Torque queuing system monitor, for jobs run on MSU's HPC system. While several different machines were used for analyses, comparisons between unnormalized and normalized data sets were always done on the same machine.

4.6 Figure Legends

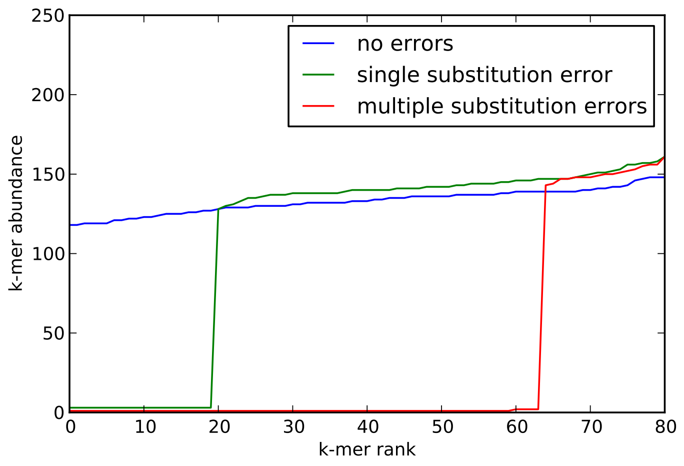
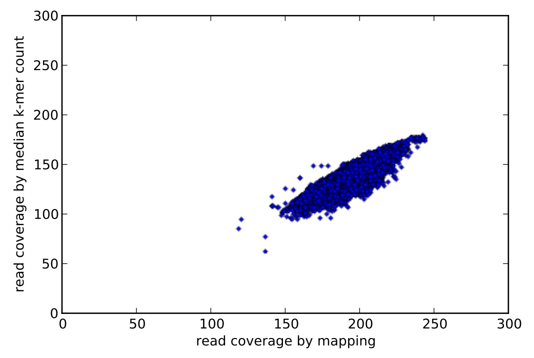


Figure 4.1: **Representative rank-abundance distributions for 20-mers from 100-base reads with no errors, a read with a single substitution error, and a read with multiple substitution errors.**

4.7 Tables

Table 4.1: **Digital normalization to C=20 removes many erroneous k-mers from sequencing data sets. Numbers in parentheses indicate number of true k-mers lost at each step, based on reference.**

Data set	True 20-mers	20-mers in reads	20-mers at C=20	% reads kept
Simulated genome	399,981	8,162,813	3,052,007 (-2)	19%
Simulated mRNAseq	48,100	2,466,638 (-88)	1,087,916 (-9)	4.1%
E. coli genome	4,542,150	175,627,381 (-152)	90,844,428 (-5)	11%
Yeast mRNAseq	10,631,882	224,847,659 (-683)	10,625,416 (-6,469)	9.3%
Mouse mRNAseq	43,830,642	709,662,624 (-23,196)	43,820,319 (-13,400)	26.4%



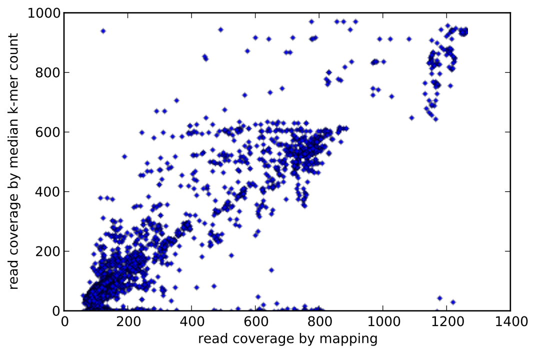


Figure 4.2: **Mapping and k-mer coverage measures correlate for simulated genome data and a real *E. coli* data set (5m reads). Simulated data $r^2 = 0.79$; *E. coli* $r^2 = 0.80$.**

Table 4.2: **Three-pass digital normalization removes most erroneous k-mers. Numbers in parentheses indicate number of true k-mers lost at each step, based on known reference.**

Data set	True 20-mers	20-mers in reads	20-mers remaining	% reads kept
Simulated genome	399,981	8,162,813	453,588 (-4)	5%
Simulated mRNAseq	48,100	2,466,638 (-88)	182,855 (-351)	1.2%
E. coli genome	4,542,150	175,627,381 (-152)	7,638,175 (-23)	2.1%
Yeast mRNAseq	10,631,882	224,847,659 (-683)	10,532,451 (-99,436)	2.1%
Mouse mRNAseq	43,830,642	709,662,624 (-23,196)	42,350,127 (-1,488,380)	7.1%

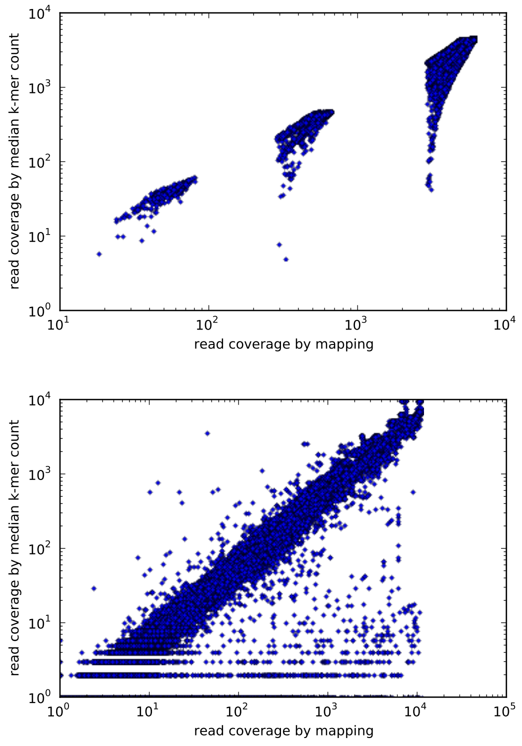


Figure 4.3: **Mapping and k-mer coverage measures correlate for simulated transcriptome data as well as real mouse transcriptome data. Simulated data $r^2 = 0.93$; mouse transcriptome $r^2 = 0.90$.**

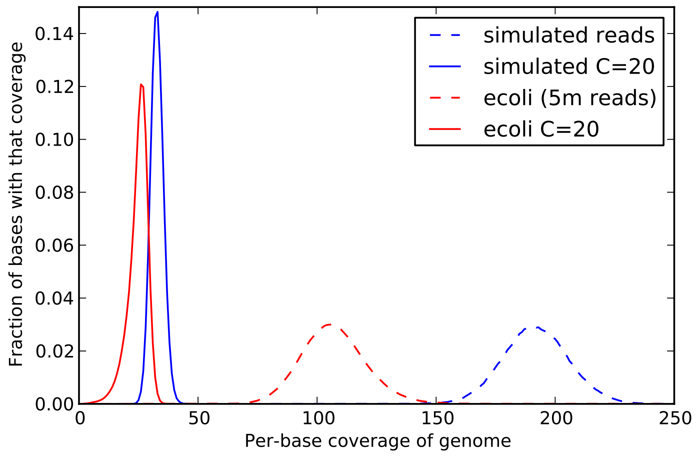


Figure 4.4: **Coverage distribution of three microbial genome samples, calculated from mapped reads (a) before and (b) after digital normalization (k=20, C=20).**

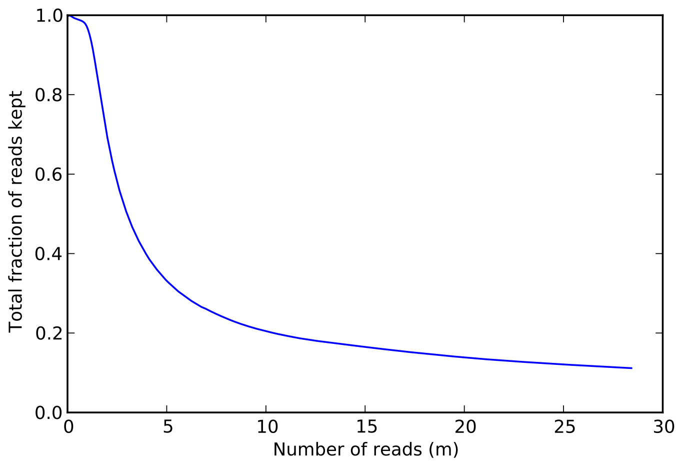


Figure 4.5: **Fraction of reads kept when normalizing the *E. coli* dataset to C=20 at k=20.**

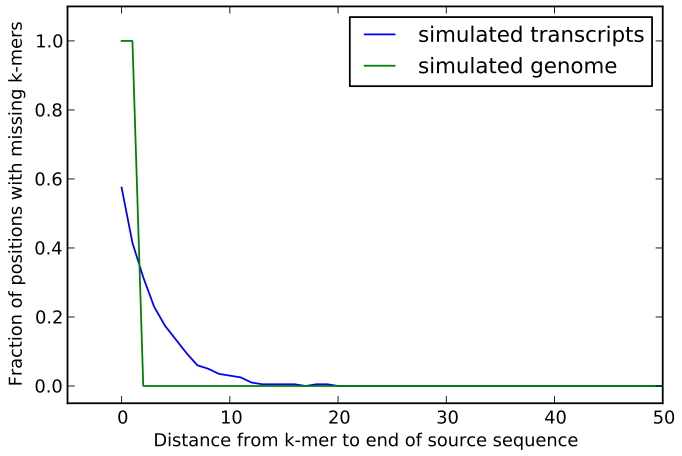


Figure 4.6: **K-mers at the ends of sequences are lost during digital normalization.**

Table 4.3: **Three-pass digital normalization reduces computational requirements for contig assembly of genomic data.**

Data set	N reads pre/post	Assembly time pre/post	Assembly memory pre/pos
E. coli	31m / 0.6m	1040s / 63s (16.5x)	11.2gb / 0.5 gb (22.4x)
S. aureus single-cell	58m / 0.3m	5352s / 35s (153x)	54.4gb / 0.4gb (136x)
Deltaproteobacteria single-cell	67m / 0.4m	4749s / 26s (182.7x)	52.7gb / 0.4gb (131.8x)

Table 4.4: **Single-pass digital normalization to C=20 reduces computational requirements for transcriptome assembly.**

Data set	N reads pre/post	Assembly time pre/post	Assembly memory pre/post
Yeast (Oases)	100m / 9.3m	181 min / 12 min (15.1x)	45.2gb / 8.9gb (5.1x)
Yeast (Trinity)	100m / 9.3m	887 min / 145 min (6.1x)	31.8gb / 10.4gb (3.1x)
Mouse (Oases)	100m / 26.4m	761 min/ 73 min (10.4x)	116.0gb / 34.6gb (3.4x)
Mouse (Trinity)	100m / 26.4m	2297 min / 634 min (3.6x)	42.1gb / 36.4gb (1.2x)

Table 4.5: **Digital normalization has assembler-specific effects on transcriptome assembly.**

Data set	Contigs > 300	Total bp > 300	Contigs > 1000	Total bp > 1000
Yeast (Oases)	12,654 / 9,547	33.2mb / 27.7mb	9,156 / 7,345	31.2mb / 26.4mb
Yeast (Trinity)	10,344 / 12,092	16.2mb / 16.5mb	5,765 / 6,053	13.6 mb / 13.1mb
Mouse (Oases)	57,066 / 49,356	98.1mb / 84.9mb	31,858 / 27,318	83.7mb / 72.4mb
Mouse (Trinity)	50,801 / 61,242	79.6 mb / 78.8mb	23,760 / 24,994	65.7mb / 59.4mb

Chapter 5

IGS-based diversity analysis

5.1 Introduction

In almost all metagenomics projects, diversity analysis plays an important role to supply information about the richness of species, the species abundance distribution in a sample or the similarity and difference between different samples, all of which are crucial to draw insightful and reliable conclusion. Traditionally especially for amplicon metagenomics data set, OTUs(Operational Taxonomic Units) based on 16S rRNA genes are used as the basic units for diversity analysis. OTUs can be good replacement of the concept of "species" in metagenomics. Basically contigs are assembled from reads and are "binned" into OTUs using composition-based or similarity-based approaches. Then the diversity can be estimated by using the abundance information of the OTUs.

Recently there are many more projects generating whole genome shotgun metagenomics data sets. However they are mainly used for assembly and annotation purpose. Less attention was paid to diversity measurement using these whole genome metagenomics data sets. One possible reason is that the whole genome metagenomics data sets are often with low depth given the high diversity of metagenomics samples compared to 16S rRNA ampicon metagenomics data set. Assembly and annotation are always challenging with the low depth and lack of reference sequences. It is also true for diversity measurement. On the other hand, although with low depth, some whole genome metagenomics data sets are with large size because of the high diversity. For instance, there may be 4 petabase pairs of DNA in a gram of soilZarraonaindia:2013aa. Many of those methods for sequence binning or diversity estimation do not scale well and will not work for large metagenomics data sets. For instance, many composition-based binning approach involves k-mer/signature frequency distribution calculation, which is rather computationally expensive. Even basic sequence alignment will be impossible for large metagenomics data set. Many of those statistical software packages to estimate diversity using various estimators are not prepared for the large scale of whole genome metagenomics data.

With the development of next generation sequencing technology, the cost of sequencing is dropping rapidly. Whole genome metagenomics sequencing is more popular and large amount of metagenomics data is being generated with increasing speed, which can not be even met by the increase of computational capacity. Novel methods that can scale well are extremely needed to deal with the increasingly large metagenomics data set.

Here we propose a novel concept - IGS (informative genomic segment) and use IGS as a replacement of OTUs to be the cornerstone for diversity analysis of whole shotgun metagenomics data sets. IGSs represent the unique information in a metagenomics data set and the abundance of IGSs in different samples can be retrieved by the reads coverage through an efficient k-mer counting method. This samples-by-IGS abundance data matrix is a promising replacement of samples-by-OTU data matrix used in 16S rRNA based analysis and all existing statistical methods can be borrowed to work on the samples-by-IGS data matrix to investigate the diversity. We applied the IGS-based method to several simulated data sets and a real data set - Global Ocean Sampling Expedition (GOS) to do beta-diversity analysis and the samples were clustered more accurately than existing alignment-based method. We also tried this novel method to Great Prairie Soil Metagenome Grand Challenge data sets. Furthermore we will show some preliminary results using the IGS-based method for alpha-diversity analysis. Since this method is totally binning-free, assembly-free, annotation-free, reference-free, it is specifically promising to deal with the highly diverse samples, while we are facing large amount of dark matters in it, like soil.

5.2 The concept of IGS(informational genomic segment)

In classic ecology dealing with macroorganisms, diversity measurement is based on the concept of species. For 16S rRNA amplicon metagenomics data set, it is based on the concept of ?OTUs?. When the concept of OTUs does not work for large shotgun metagenomics data set, in the beginning we proposed that the concept of k-mers(a DNA segment with the leng of k) can be used to measure diversity. K-mers can be considered as the atom of information in DNA sequences. One of the composition-based approaches to binning is to use the k-mer as the signature. Suppose the sizes of microbial genomes are similar and the difference between genomic content of microbial genomes is similar, the number of distinct k-mers in the sequence data set is related to the number of species in a sample. However, because of sequencing error, which is unavoidable due to the limit of sequencing technology, this k-mer based analysis doe not work well. One sequencing error on a read will generate at most k erroneous k-mers. In metagenomics data set with low coverage, most of the distinct observed k-mers are from sequencing errors.

Next we turned our gaze to the upper level - reads. A novel method termed as ?digital normalization? was developed to remove abundant reads before assembly. However it also supplies a novel way to distil information from reads by decreasing the bad influence of sequencing errors so that we can use those informative reads to measure the microbial diversity. We term those informative reads as IGS(informative genomic segment), which can be considered as a segment of DNA on a microbial genome. Those IGSs should be different enough to represent the abstract information a genome contains. Suppose microbial genomes contain similar number of those IGSs, as they contain similar number of distinct k-mers, the number of IGSs will be related to the species richness in a sample, and the abundance distribution of IGSs will be related to species evenness in a sample. Many classic diversity estimation methods based on OTUs level described in sections above can be borrowed to estimate the diversity of IGSs and the diversity of actual species subsequently.

IGS may be a good concept in whole genome shotgun metagenomic diversity analysis, especially while facing large amount of "dark matters", unknown species. We don't care about species, we only care about how much information there is in the sample.

For alpha diversity, we can generate a list of IGSs and the respective abundance in a sample. Then existing estimators like Chao's can be used to estimate total number of IGSs in the sample. Rarefaction curve based on number of IGSs can also be generated.

For beta diversity, here we will generate a samples-by-IGS data matrix, as a replacement of samples-by-OTU data matrix in 16s based analysis and samples-by-species data matrix in traditional ecology.

From that samples-by-IGS data matrix, we can use existing methods to calculate similarity/disimilarity/distance between samples and do further analysis like clustering and ordination. QIIME and Mother can do this kind of jobs pretty well.

Using median k-mer frequency can decrease the influence of sequencing error, but can not eliminate the influence of errors. This can cause some problems in the following analysis, which will be discussed in details.

5.2.1 IGS(informative genomic segment) can represent the novel information of a genome

Median k-mer abundance can represent sequencing depth of a read(cite diginorm). For a sequencing reads data set with multiple species, the sequencing depth of a read is related to the abundance of species where the read originates.

The Figure 5.1a shows the abundance distribution of reads from 4 simulated sequencing data sets with different sequencing depth - 3 sequencing data sets generated with different sequencing coverage(1x, 10x, 40x) from 3 simulated random genomes respectively and 1 combined data set with all the previously mentioned data sets. No error is introduced in these simulated data sets. Obviously the reads from the three data sets can be separated by estimated sequencing depth. The combined data set can be considered as a sequencing data set with three species with different abundance.

Each point on the curve shows that there are Y reads with a sequencing depth of X. In other word, for each of those Y reads, there are X-1 other reads that cover the same DNA segment in a genome that single read originates. So we can estimate that there are Y/X distinct DNA segments with reads coverage as X. We term these distinct DNA segments in species genome as IGS(informative genomic segment). We can transform the figure in upper position to show the number of IGSs and their respective reads coverage, as shown in figure in lower position. We sum up the numbers of IGSs with different reads coverage for each data set and get the result as shown in below. The sum numbers of IGSs here essentially are the areas below each curve in the figure.

Even though the datasets have different sequencing depth like 10X and 40X, they have similar numbers of IGSs. Dataset with 1X sequencing depth has fewer IGSs because the depth is not enough to cover all the content of the genome(63.2%) Essentially it is the maximum number of segments with length L on a genome out of which no two segments share any single k-mer. See Figure below. Assume the species genome is totally random, which is the case in the simulated data set, the number of IGSs(N) in a species genome is related to the size of genome(G), read length(L) and k size(k), which can be denoted as

N =G/(L-k+1)

For the simulated genome with size of 1M bps, read length as 80bps, k-mer size as 22bps,expected number of IGSs is

1000000/(80 - 22 + 1) = 16949,

which is pretty close to observed value. See Table 5.1

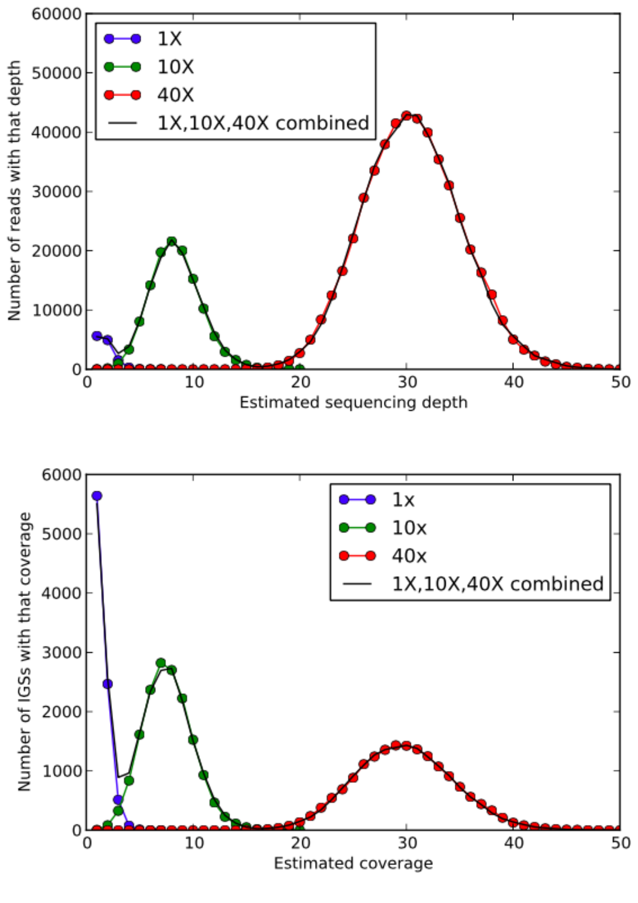


Figure 5.1: **from reads to IGS**

Table 5.1: **Total number of IGSs in different simulated reads data sets.**

Data set	total number of IGSs
1X depth	8714
10X depth	16321
40X depth	16794
1X,10X,40X combined	41742

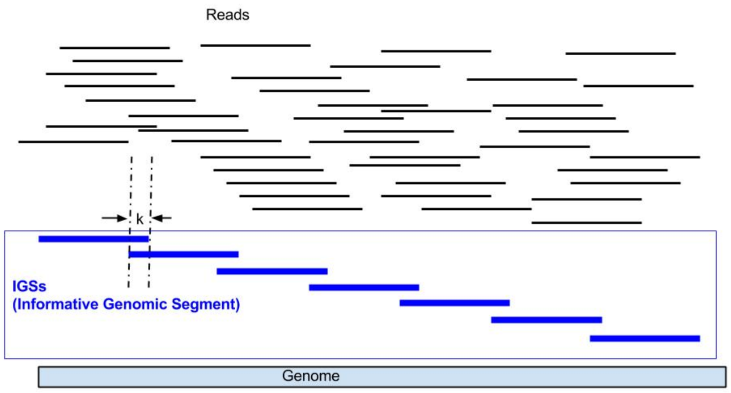


Figure 5.2: **the concept of IGS**

5.2.2 IGS can be used to do alpha diversity analysis

Basically the abundance distribution of IGSs with different coverage in a sample data set is acquired using the method shown above, like:

3 23

4 24

5 25

6 25

...

Here 23 IGSs with coverage as 3, this number is calculated from dividing the total number of reads with coverage as 3, which is 69, by the coverage 3: 69/3. Similarly there are 96/4 = 24 IGSs with coverage as 4.

If we draw an analogy between IGSs and OTUs, this is like there are 23 different OTUs with 3 reads mapped to, and 24 different OTUs with 4 reads mapped to.

Then list all the different IGSs and the corresponding count,and we can get a long list with each IGS and the corresponding coverage.The coverage of an IGS can be considered as the abundance of such IGS in a sample. The list looks like:

IGS_ID	abundance
1	3
2	3
3	3
...	
23	3
24	4
25	4
...	
47	4
48	5
...	
...	

This list is the counterpart of an OTU table in OTU based diversity analysis.

With such table at hand, numerous existing statistical methods and software packages can be used to investigate the alpha diversity.

5.2.3 IGS can be used to do beta diversity analysis

As in alpha diversity analysis, OTU table is also a cornerstone for beta diversity analysis. As long as we get a reliable OTU table, there are existing pipelines to do the beta diversity analysis.

A typical OTU table across different samples is like this, which is also called samples-by-OTU data matrix.

OTU_ID	Sample1_ID	Sample2_ID	Sample3_ID
OTU1	3	4	2
OTU2	2	5	0
OTU3	3	1	4

Like a OTU table, we hope to have the IGS table for the IGSs:

IGS_ID	SampleA	SampleB	SampleC	SampleD
IGS1	5	1	2	1
IGS2	5	1	2	1

So now the problem is how we can generate a sample-by-IGS data matrix as the counterpart of samples-by-OTU data matrix so many of the existing tools/methods used for OTU-based diversity can be borrowed for this kind of IGS-based analysis, just as what is shown above for alpha diversity analysis.

Firstly, as how we get the coverage of a read from a sample dataset in this sample dataset, we can get the coverage of a read from a sample A dataset in another sample B dataset. We can still use the median k-mer count to represent the coverage. The basic idea is the same.

Because a read must derive from a segment in the genome of some species in a sample, if a read R from sample A with a coverage C_A in sample A has a coverage as C_B in sample B, that means that segment of genome in sample A from which read R derive also exists in sample B. That genomic segment has a coverage as C_A in sample A and has a coverage as C_B in sample B. Roughly there should be about C_A reads (read R should be one of them) in sampleA covering that genomic segment and C_B reads in sampleB covering that genomic segment. Meanwhile, the C_A reads in sampleA should all have a coverage as C_B in sampleB, just like read R as one of them. Similarly, the C_B reads in sampleB should all have a coverage as C_A in sampleA.

Ok, now let's make an example.

Suppose there are 6 reads in sample A, all have a coverage as 3 in sampleA, and have a coverage as 2 in sampleB.

According to the discussion about IGS in previous section, the 6 reads cover 2 IGSs with a coverage as 3 for each IGSs. There should be 4 reads in sampleB covering the exact same 2 IGSs, with a coverage as 2 in sampleB.

So now we have 2 distinct IGSs with redundancy as 3 and 2 in the two samples respectively.

Note: small number is used in the analysis above as example, but it should be emphasized that the analysis is based on large number statistically.

Let's expand this example from 2 samples to 4 samples(A,B,C,D), as shown in figure above.

Let's say we find 10 reads in sampleA, with coverage as 5-1-2-1 in samples A-B-C-D respectively. (We call "5-1-2-1" "coverage spectrum" across samples.) So there should be **about** 2 reads in sampleB, 4 reads in sampleC, 2 reads in sampleD, all of which have a "coverage spectrum" as "5-1-2-1". Basically these 18 reads altogether cover 2 distinct IGSs, which apparently exist in all the 4 samples. The 2 distinct IGSs has a redundancy as 5,1,2,1 in the 4 samples respectively.

If we draw an analogy between IGSs and OTUs, this is like there are 2 OTUs, both with 5,1,2,1 reads mapped to in sample A,B,C,D respectively.

Like a OTU table, here we can have the IGS table for the two IGSs:

IGS_ID	SampleA	SampleB	SampleC	SampleD
IGS1	5	1	2	1
IGS2	5	1	2	1

5.3 Evaluating IGS method using simulated data sets

5.3.1 An experiment using a simple simulated data sets

For this experiment, firstly we create 6 synthetic samples (Sample 1-6) based on 9 synthetic 10K genomes (genome A-I), with different composition of species and diversity.

The species composition for each synthetic sample is as below:

sample1: AAAB

sample2: AABC

sample3: ABCD

sample4: ABCE

sample5: AFGH

sample6: IFGH

For sample1, there are two species - A and B, with abundance distribution as 3:1.

The sequencing depth of all the synthetic data sets is 10X. So the species abundance in each sample is as below:

sample1: genomeA - 30, genomeB - 10

sample2: genomeA - 20, genomeB - 10, genomeC - 10

sample3: genomeA - 10, genomeB - 10, genomeC - 10, genomeD - 10

sample4: genomeA - 10, genomeB - 10, genomeC - 10, genomeE - 10

sample5: genomeA - 10, genomeF - 10, genomeG - 10, genomeH - 10

sample6: genomeI - 10, genomeF - 10, genomeG - 10, genomeH - 10

An a simple experiment, there is no sequencing errors introduced in the synthetic reads data sets.

Figure 5.3 and Figure 5.4 show that IGS method can yield the information about the difference of samples correctly. Sample 5 and sample 6 and very close to each other on the figure, which is true if we check the species composition of the two samples shown above.

Figure 5.5 shows the method can yield the richness information correctly. From the figure, samples with 4 different species have the richness almost twice as large as the sample with 2 different species.

From these results, we show the IGS method can work well to a simplest scenario, with high sequencing depth (10X) and no sequencing error. Next we will check the influence to the analysis accuracy of variable sequencing depth and sequencing error and introduce new ways to preprocess the data to decrease the influence of sequencing error.

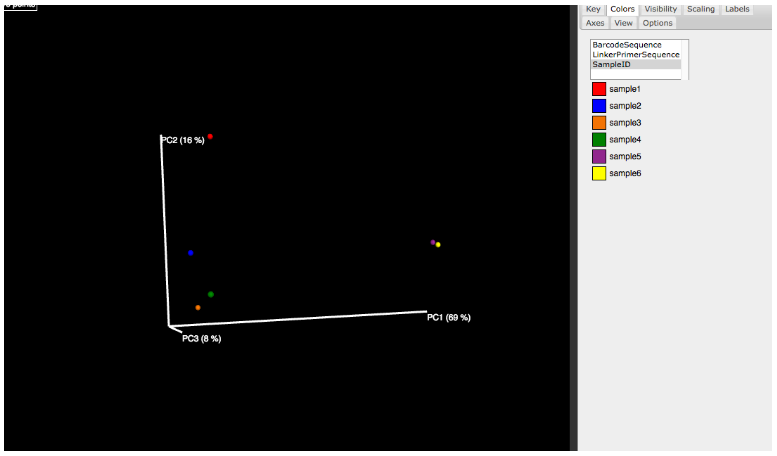


Figure 5.3: **Ordination of the 6 synthetic samples using IGS method**

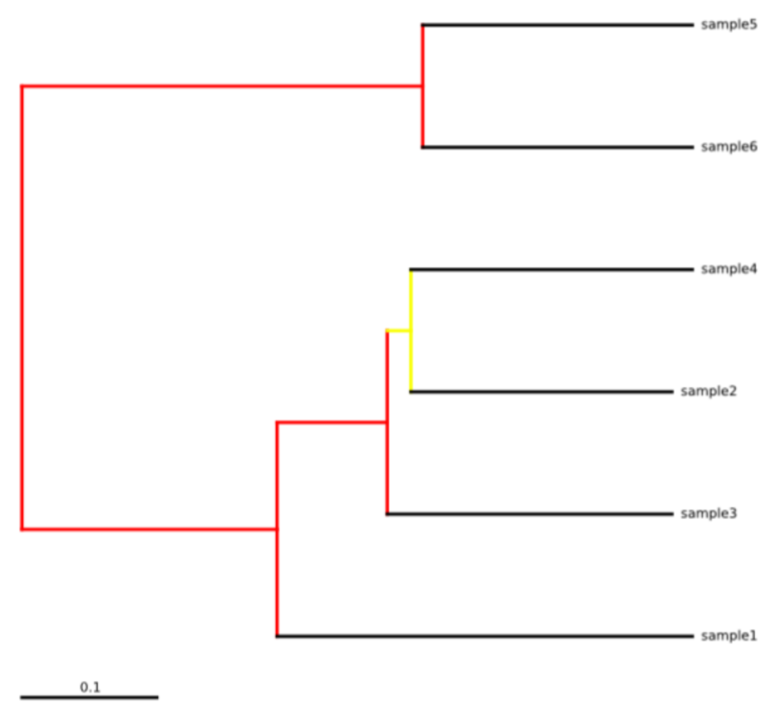


Figure 5.4: **Clustering of the 6 synthetic samples using IGS method**

5.3.2 Evaluating the influence of variable sequencing depth and sequencing error

Previously we have shown the IGS method generally works to a simple simulated data sets, with high sequencing depth and no sequencing error. Since in many situations we deal with

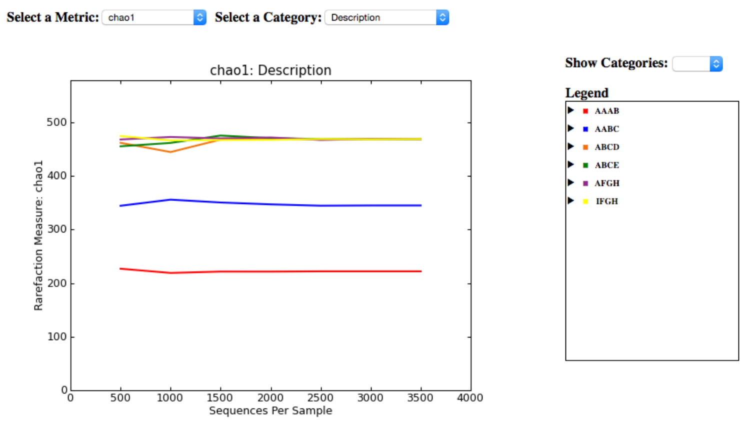


Figure 5.5: **Richness estimation of the 6 synthetic samples using IGS method**

metagenomic data sets with low sequencing depth, like soil or sea water samples, we want to know if we can still get reliable insights from the low coverage sequencing data. Also it is a fact that all sequencing technology will generate some errors. As discussed in the background section, one of the reasons we develop the IGS method is that based on the abundance counting of reads rather than k-mers, it is expected that the IGS method is more robust to sequencing error. We also wonder how error correction will help improving the results of IGS method.

6 synthetic samples are generated with the same species composition as in last section. For each sample, sequencing reads data sets with different sequencing depth are simulated. A series of sequencing reads data sets are also generated with sequencing error as 3%, besides a series of error-free data sets. Here we plan to use these simulated data sets to check the effectiveness of the IGS based method(generally and the effectiveness of error correction/filtering) and the influence of variable sequencing depth.

5.3.2.1 comparison of dissimilarity matrix from beta diversity analysis

To evaluate the effectiveness of beta diversity analysis using IGS based method, we compare the dissimilarity matrix generated by IGS based method with that generated from another metagenomics comparison tool - Commet(Compareads), and the true matrix,since we know exactly the species composition of the simulated data sets.

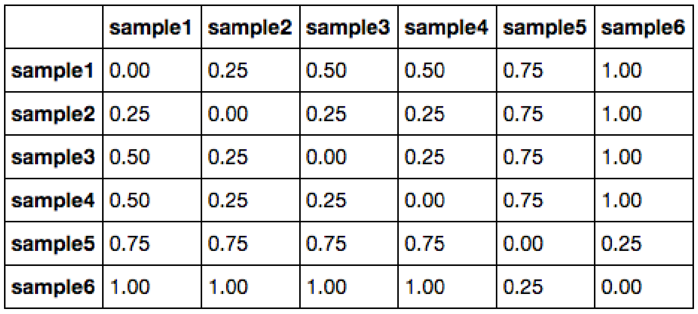
The clustering and ordination are all from the dissimilarity matrix. We think comparing matrix directly makes more sense than comparing the clustering and ordination plot. So we will not show the clustering and ordination figure in this evaluation. If the matrix can reflect the real relationship between samples reliably, the clustering and ordination will only be routine job.

The true dissimilarity matrix of the 6 simulated samples using bray-cutis metric from species composition directly is shown in Figure 5.6. For a simulated data set with 10x coverage and no error introduced (which will tell us the optimal performance of IGS method), the dissimilarity matrix can be calculated by using IGS method, as shown in Figure 5.7. We can see the absolute values in the matrix are not very close to that in the real matrix.But the relative values correspond to that in the real matrix well to show the relative distance between each pair of samples. To get a objective metric, we use Mantel (citing)test to calculate the correlation value between the two matrixes. The correlation is 0.9714, which means a very positive correlation between the two matrices. We are very confident that the matrix from IGS method can reflect the true relationship between samples pretty well.

Next we test how well the matrix calculated by various methods can reflect the real relationship between samples. The simulated data sets with sequencing depth as 1 and 10, with sequencing error as 3% and without sequencing error are used in this experiment. For the data sets with sequencing error, we use a HMM based error correction tool to preprocess the reads to check the effectiveness of error correction. We also compare the performance of IGS based method and another metagenome comparing tool - Comet.

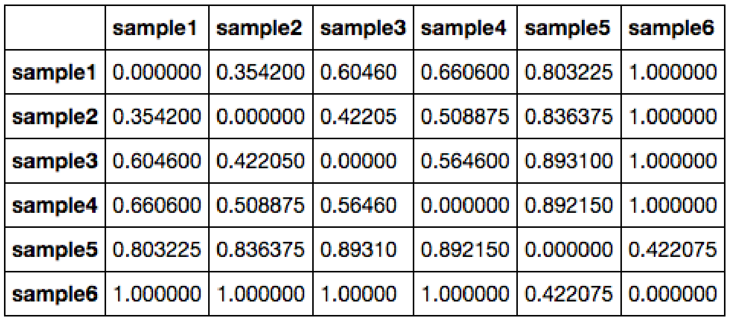
As shown in Figure 5.8, firstly, for all data sets, the matrix from IGS method has a higher correlation to golden standard than that from Comet. As expected, the matrix from data sets with sequencing error has a lower correlation than that from error-free data sets. Also Comet is more sensitive to sequencing error rate, compared to IGS method. However, error correction can increase the correlation significantly. Also higher coverage will yield more accurate matrix, which is not surprising.

Figure 5.9 shows how well the matrix calculated from data set with variable coverage can reflect the real relationship between samples. It is as expected that higher coverage data will yield better/more accurate distance matrix. Note even with a coverage as low as 0.1, the correlation is 0.89. This can give us the hint how reliable the result will be if we only use a small proportion of data from a large metagenomic data set.



	sample1	**sample2**	**sample3**	**sample4**	**sample5**	**sample6**
sample1	0.00	0.25	0.50	0.50	0.75	1.00
sample2	0.25	0.00	0.25	0.25	0.75	1.00
sample3	0.50	0.25	0.00	0.25	0.75	1.00
sample4	0.50	0.25	0.25	0.00	0.75	1.00
sample5	0.75	0.75	0.75	0.75	0.00	0.25
sample6	1.00	1.00	1.00	1.00	0.25	0.00

Figure 5.6: **Dissimilarity matrix between synthetic samples using Bray-cutis from species composition directly**



	sample1	sample2	sample3	sample4	sample5	sample6
sample1	0.000000	0.354200	0.60460	0.660600	0.803225	1.000000
sample2	0.354200	0.000000	0.42205	0.508875	0.836375	1.000000
sample3	0.604600	0.422050	0.00000	0.564600	0.893100	1.000000
sample4	0.660600	0.508875	0.56460	0.000000	0.892150	1.000000
sample5	0.803225	0.836375	0.89310	0.892150	0.000000	0.422075
sample6	1.000000	1.000000	1.00000	1.000000	0.422075	0.000000

Figure 5.7: **Dissimilarity matrix between synthetic samples using Bray-cutis from sequencing reads using IGS method**

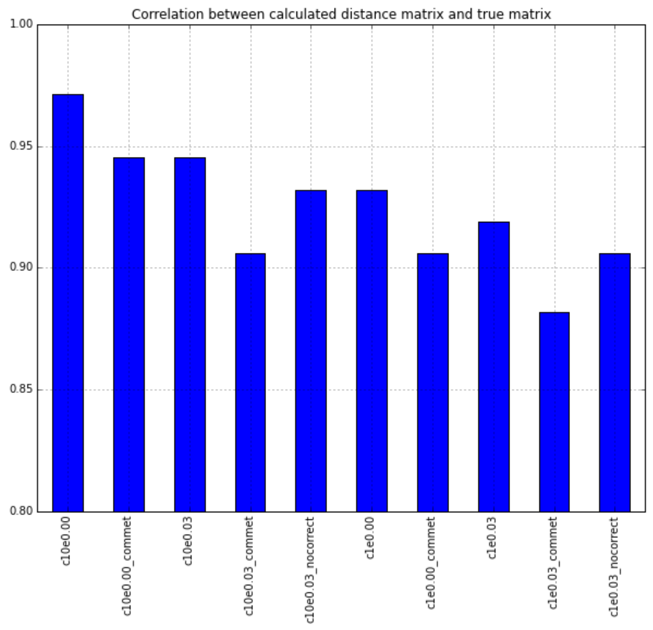


Figure 5.8: **Correlation between calculated distance matrix and true matrix from different data sets and using different methods**

5.3.2.2 Evaluate alpha diversity analysis by estimating size of metagenome

We can use statistic metric to estimate the total number of IGSs in a sample, which can be used to calculate the estimated genome size of a sample using the formula below:

size of genome = number of IGS x (reads_length - k-size +1)

Here we check how accurate the estimated size of genome and coverage is using different

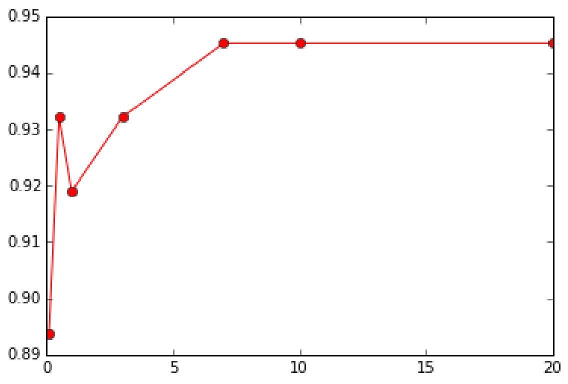


Figure 5.9: **Correlation between calculated distance matrix and true matrix from different dat sets with different sequencing depth**

data sets with variable coverage/sequencing depth.

The genome size of the 6 samples should be:

sample1: AAAB 2x100K = 200K bp

sample2: AABC 3x100K = 300K bp

sample3: ABCD 4x100K = 400K bp

sample4: ABCE 4x100K = 400K bp

sample5: AFGH 4x100K = 400K bp

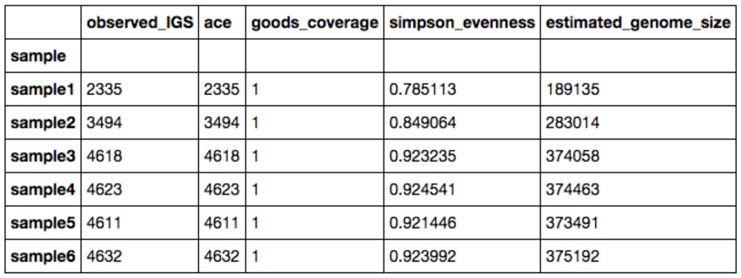
sample6: IFGH 4x100K = 400K bp

In this experiment, we use ACE metric since we find it is more accurate than Chao1, since it uses more abundance information.

Table 5.10 shows the estimated genome size of samples using error-free simulated sequencing data sets is close to true size shown above. If the sequencing error is introduced, the estimated genome size is inflated dramatically as shown in Table 5.11, which is not surprising. However after applying error correction, we can still get good estimation of genome size, as shown in Table 5.12. This proves again error correction can improve the effectiveness of IGS method.

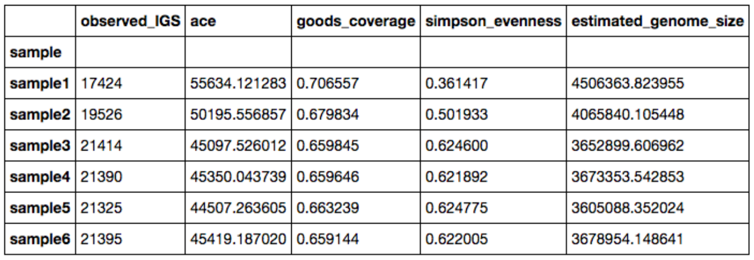
Figure 5.12 shows the estimated genome size from data sets with variable coverage. The estimated genome size keeps increasing as we use more reads, with higher coverage,which is higher than actual genome size. It's interesting that the estimated genome size is very high with low coverage, (probably due to more unique k-mers, which makes error correction more difficult. Then the estimated genome size drops a little bit as coverage from 0.1X to 1-3X, then starts to climb again as coverage increases, probably because there are always more erroneous k-mers that cannot be corrected. The good thing is that the climbing rate is not that high, this is believed to be due to the effectiveness of our error correction algorithms.

It is important to point that even though the absolute value of estimated genome size may be overestimated. The relative relationship between samples are reliable, as shown in the figure. Sample3,4,5,6 all have 4 species, while sample 2 has 3 species, and sample 1 has 2 species. They can be separately pretty well.



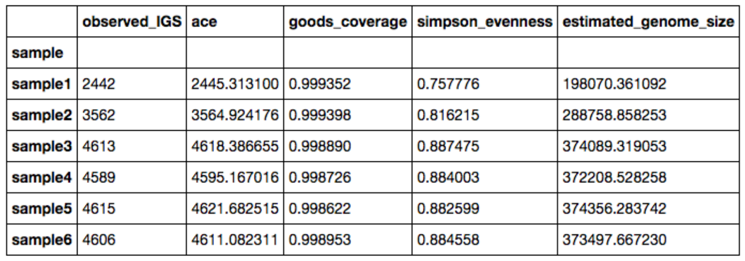
	observed_IGS	ace	goods_coverage	simpson_evenness	estimated_genome_size
sample					
sample1	2335	2335	1	0.785113	189135
sample2	3494	3494	1	0.849064	283014
sample3	4618	4618	1	0.923235	374058
sample4	4623	4623	1	0.924541	374463
sample5	4611	4611	1	0.921446	373491
sample6	4632	4632	1	0.923992	375192

Figure 5.10: **Coverage = 10x, No error**



	observed_IGS	ace	goods_coverage	simpson_evenness	estimated_genome_size
sample					
sample1	17424	55634.121283	0.706557	0.361417	4506363.823955
sample2	19526	50195.556857	0.679834	0.501933	4065840.105448
sample3	21414	45097.526012	0.659845	0.624600	3652899.606962
sample4	21390	45350.043739	0.659646	0.621892	3673353.542853
sample5	21325	44507.263605	0.663239	0.624775	3605088.352024
sample6	21395	45419.187020	0.659144	0.622005	3678954.148641

Figure 5.11: **Coverage = 10x, error = 3%, no error correction**



	observed_IGS	ace	goods_coverage	simpson_evenness	estimated_genome_size
sample					
sample1	2442	2445.313100	0.999352	0.757776	198070.361092
sample2	3562	3564.924176	0.999398	0.816215	288758.858253
sample3	4613	4618.386655	0.998890	0.887475	374089.319053
sample4	4589	4595.167016	0.998726	0.884003	372208.528258
sample5	4615	4621.682515	0.998622	0.882599	374356.283742
sample6	4606	4611.082311	0.998953	0.884558	373497.667230

Figure 5.12: **Coverage = 10x, error = 3%, with error correction**

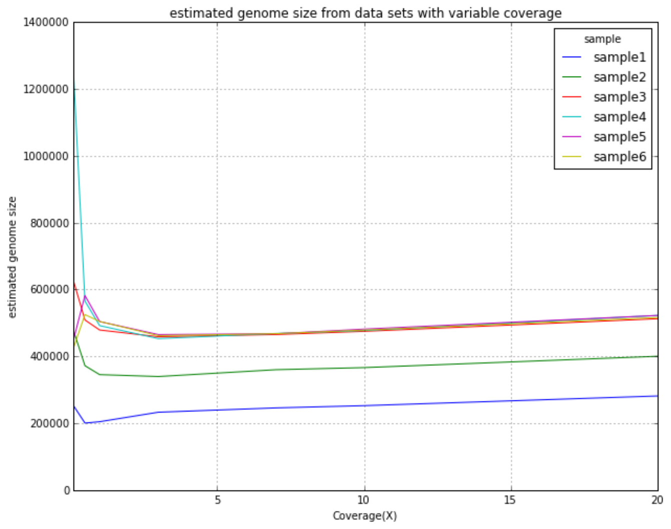


Figure 5.13: **estimated genome size from data sets with variable coverage**

5.3.3 The IGS method can provide a whole framework to do alpha or beta diversity, with good versatility.

From the testing using simulated data sets shown here, we are confident that our IGS method works well and can give reliable results from data sets with error and low sequencing depth.

The IGS method can provide a whole framework to do alpha or beta diversity. Here we tested beta diversity using only bray-curtis metric and alpha diversity on richness only. Actually any metric can be applied to the IGS-by-samples table, abundance-based or incidence-baserd, richness or evenness.

Compareads(Commet) based on reads overlap between samples can get a matrix reflecting the real relationship between samples pretty good but it is stuck with one metric, which is based on the percentage of overlap reads between samples. This metric is like bray curtis , but not exactly the same.

5.4 Applying IGS method to real metagenome data sets

5.4.1 GOS data sets: Sorcerer II Global Ocean Sampling Expedition

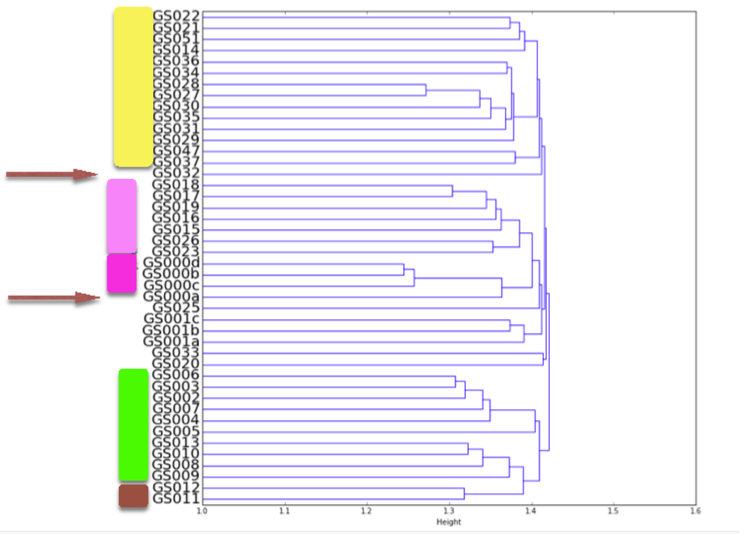


Figure 5.14: **cluster of GOS samples using IGS method**

5.4.2 HMP metagenomics data set

5.4.3 GPGC soil sample - new and old

Using 1m and 2m randomly selected subsets can yield pretty good results.

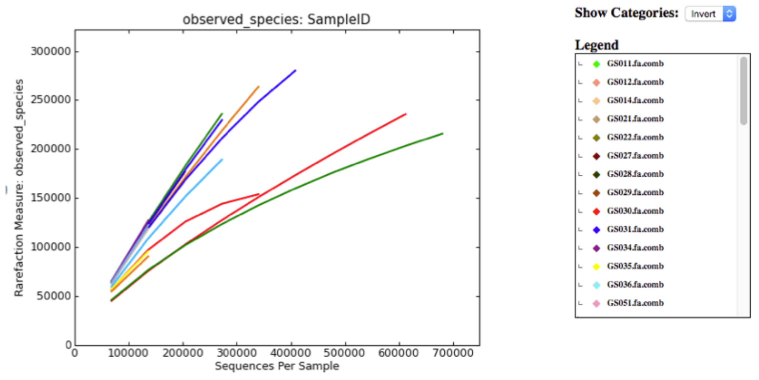


Figure 5.15: **cluster of GOS samples using IGS method**

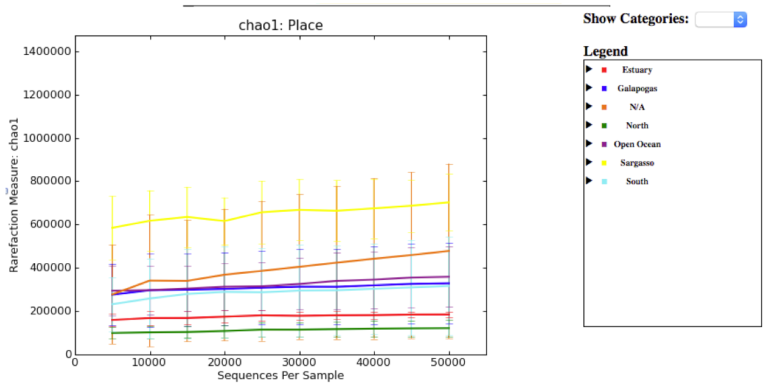


Figure 5.16: **cluster of GOS samples using IGS method**

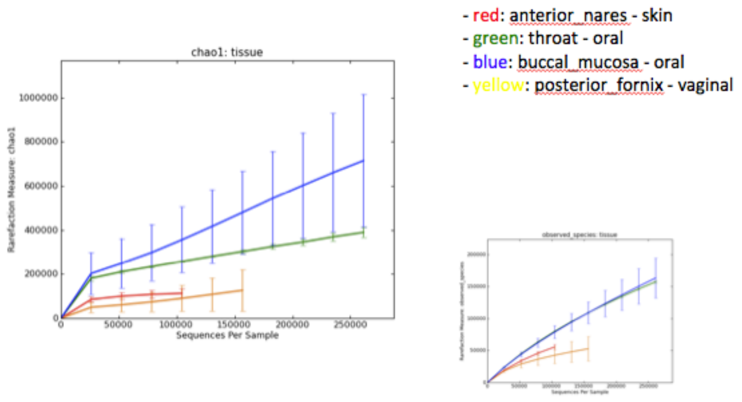
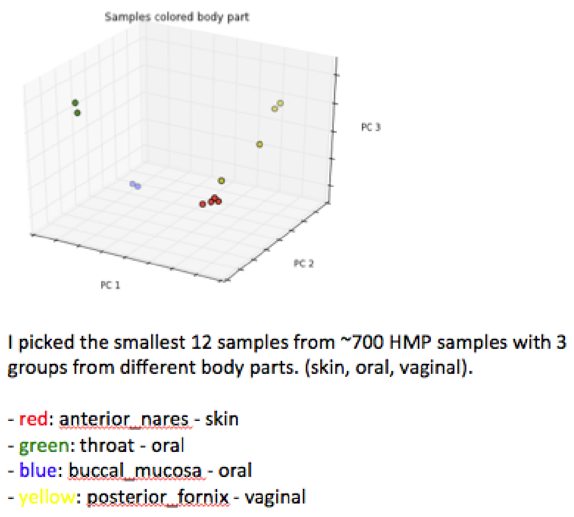


Figure 5.17: **rarefaction of HMP**

5.5 Conclusion

Advantage:



I picked the smallest 12 samples from ~700 HMP samples with 3 groups from different body parts. (skin, oral, vaginal).

- red: anterior_nares - skin
- green: throat - oral
- blue: buccal_mucosa - oral
- yellow: posterior_fornix - vaginal

Figure 5.18: **PCoA of HMP**

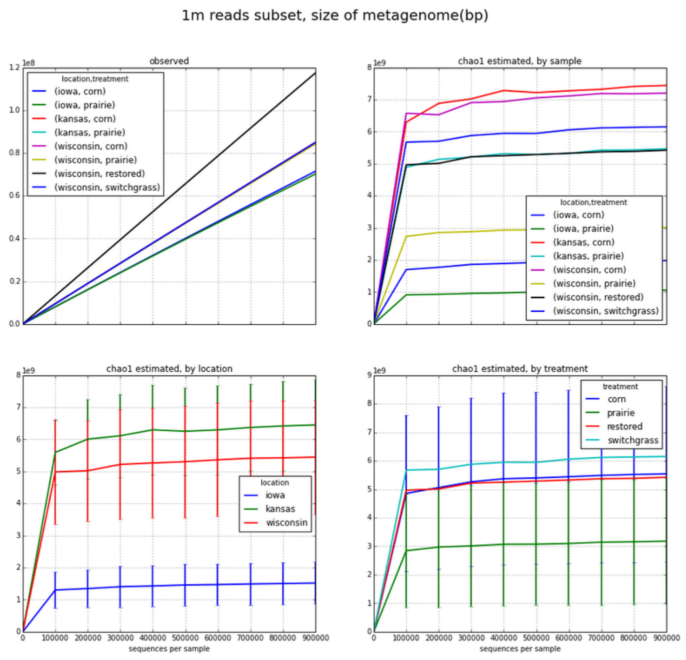


Figure 5.19: **alpha of GPGC**

not only HMP, high depth, data but also low depth data, like soil metagenomics, which is impossible to use traditional method

IGS is promising to more problems

1. alpha-diversity analysis (1 sample) - richness/evenness - rarefaction curve - sequencing

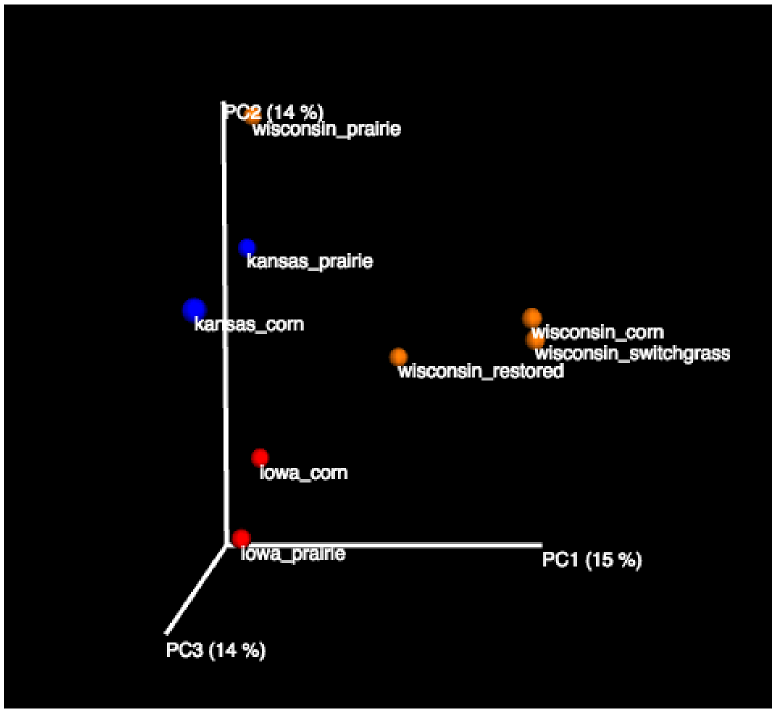


Figure 5.20: **beta of GPGC**

depth evaluation - genome size estimation - better choosing diginorm parameters(size of hashtables, etc.)

2. beta-diversity (multiple samples) - sample by sample comparison, clustering, ordination after getting segment-count table

3. other potential applications: - reads binning/classification (after clustering)(if number of samples is small, may not be effective) - extract IGS(reads) according different filters (shared by all samples, or some specific samples,) - co assembly (by extracting the reads with total coverage across samples ¿ 10, for example)

5.6 Data

5.6.1 Four simulated reads data sets with different species abundance distribution

5.6.2 Simulated sequencing reads of e.coli

Here we simulated 4 sequencing reads data sets with read length as 100bp of e.coli with different sequencing depth(50x and 150x) and different sequencing error rate(1%,2% and 0%). Table 5.2

Table 5.2: **Simulated sequencing reads data sets of e.coli**

sample	coverage	error rate
A	150	0.01
B	50	0.01
C	50	0.01
D	50	0.02

Table 5.3: GPGC Data sets

sample	# of reads	size of .gz file	# of bps	ave. length
iowa corn	1514290825	46G	144202427079	95.2
iowa prairie	2597093273	74G	226815059143	87.3
kansas_corn	2029883371	66G	206933829048	101.9
kansas_prairie	0	145G	0	0
wisconsin_corn	1616440116	51G	162257698471	100.4
wisconsin_prairie	1653557590	53G	166467901724	100.7
wisconsin_restored	226830595	11G	34241520930	151.0
wisconsin_switchgrass	310966735	13G	40259619921	129.5

Chapter 6

Conclusions

APPENDIX

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] P. Audano and F. Vannberg. Kanalyze: A fast versatile pipelined k-mer toolkit. *Bioinformatics: Advance Access published March 18, 2014*, page doi: 10.1093/bioinformatics/btu152, Mar 2014.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] M. Bonaldo, G. Lennon, and M. Soares. Normalization and subtraction: two approaches to facilitate gene discovery. *Genome Res*, 6(9):791–806, 1996.

[4] A. Brady and S. Salzberg. Phymmbl expanded: confidence scores, custom databases, parallelization and more. *Nat Methods*, 8(5):367, May 2011.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[6] A. Z. Broder and M. Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003.

[7] C. T. Brown. What does trinity's in silico normalization do?, 2012.

[8] C. T. Brown, A. Howe, Q. Zhang, A. B. Pyrkosz, and T. H. Brom. A reference-free algorithm for computational normalization of shotgun sequencing data. *arXiv preprint*, 03 2012.

[9] J. Bunge. Estimating the number of species with catchall. *Pac Symp Biocomput*, pages 121–30, 2011.

[10] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Peña, J. K. Goodrich, J. I. Gordon, G. A. Huttley, S. T. Kelley, D. Knights, J. E. Koenig, R. E. Ley, C. A. Lozupone, D. McDonald, B. D. Muegge, M. Pirrung, J. Reeder, J. R. Sevinsky, P. J. Turnbaugh, W. A. Walters, J. Widmann, T. Yatsunenko, J. Zaneveld, and R. Knight. Qiime allows analysis of high-throughput community sequencing data. *Nat Methods*, 7(5):335–6, May 2010.

[11] M. Chaisson, P. Pevzner, and H. Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–74, 2004.

[12] A. Chao. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics*, pages 265–270, 1984.

[13] A. Chao. Estimating the population size for capture-recapture data with unequal catchability. *Biometrics*, 43(4):783–91, Dec 1987.

[14] A. Chao and M. C. Yang. Stopping rules and estimation for recapture debugging with unequal failure rates. *Biometrika*, 80(1):193–201, 1993.

[15] R. Chikhi and P. Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–7, Jan 2014.

[16] H. Chitsaz, J. Yee-Greenbaum, G. Tesler, M. Lombardo, C. Dupont, J. Badger, M. Novotny, D. Rusch, L. Fraser, N. Gormley, O. Schulz-Trieglaff, G. Smith, D. Evers, P. Pevzner, and R. Lasken. Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. *Nat Biotechnol*, 29(10):915–21, 2011.

[17] S. Cohen and Y. Matias. Spectral bloom filters. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *SIGMOD Conference*, pages 241–252. ACM, 2003.

[18] R. Colwell. Estimates: statistical estimation of species richness and shared species from samples. 2004. *Consultado en: http://viceroy. eeb. uconn. edu/estimates.*

[19] R. K. Colwell, C. X. Mao, and J. Chang. Interpolating, extrapolating, and comparing incidence-based species accumulation curves. *Ecology*, 85(10):2717–2727, 2004.

[20] P. Compeau, P. Pevzner, and G. Tesler. How to apply de bruijn graphs to genome assembly. *Nat Biotechnol*, 29(11):987–91, 2011.

[21] T. Conway and A. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–86, 2011.

[22] T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–86, Feb 2011.

[23] G. Cormode. Summarizing and mining skewed data streams. In *Proc. of the 2005 SIAM International Conference on Data Mining*, pages 44–55. 2005.

[24] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch, and its applications. *Journal of Algorithms*, 2004.

[25] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, Apr. 2005.

[26] M. Crusoe, G. Edvenson, J. Fish, A. Howe, E. McDonald, J. Nahum, K. Nanlohy, H. Ortiz-Zuazaga, J. Pell, J. Simpson, C. Scott, R. Srinivasan, Q. Zhang, and C. T. Brown. *The khmer software package: enabling efficient sequence analysis*, 2014.

[27] T. P. Curtis, W. T. Sloan, and J. W. Scannell. Estimating prokaryotic diversity and its limits. *Proc Natl Acad Sci U S A*, 99(16):10494–9, Aug 2002.

[28] S. Deorowicz, A. Debudaj-Grabysz, and S. Grabowski. Disk-based k-mer counting on a pc. *BMC Bioinformatics*, 14(1):160, May 2013.

[29] H. Do, K. Choi, F. Preparata, W. Sung, and L. Zhang. Spectrum-based de novo repeat detection in genomic sequences. *J Comput Biol*, 15(5):469–87, 2008.

[30] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, pages 323–336. ACM, 2002.

[31] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[32] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS Proceedings*, (1), 2008.

[33] J. Gans, M. Wolinsky, and J. Dunbar. Computational improvements reveal great bacterial diversity and high metal toxicity in soil. *Science*, 309(5739):1387–90, Aug 2005.

[34] J. A. Gilbert and C. L. Dupont. Microbial metagenomics: beyond the genome. *Ann Rev Mar Sci*, 3:347–71, 2011.

[35] J. A. Gilbert, F. Meyer, D. Antonopoulos, P. Balaji, C. T. Brown, C. T. Brown, N. Desai, J. A. Eisen, D. Evers, D. Field, W. Feng, D. Huson, J. Jansson, R. Knight, J. Knight, E. Kolker, K. Konstantindis, J. Kostka, N. Kyrpides, R. Mackelprang, A. McHardy, C. Quince, J. Raes, A. Sczyrba, A. Shade, and R. Stevens. Meeting report: the terabase metagenomics workshop and the vision of an earth microbiome project. *Stand Genomic Sci*, 3(3):243–8, 2010.

[36] E. M. Glass, J. Wilkening, A. Wilke, D. Antonopoulos, and F. Meyer. Using the metagenomics rast server (mg-rast) for analyzing shotgun metagenomes. *Cold Spring Harb Protoc*, 2010(1):pdb.prot5368, Jan 2010.

[37] S. Gnerre, I. Maccallum, D. Przybylski, F. Ribeiro, J. Burton, B. Walker, T. Sharpe, G. Hall, T. Shea, S. Sykes, A. Berlin, D. Aird, M. Costello, R. Daza, L. Williams, R. Nicol, A. Gnirke, C. Nusbaum, E. Lander, and D. Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci U S A*, 108(4):1513–8, 2011.

[38] N. J. Gotelli and R. K. Colwell. Quantifying biodiversity: procedures and pitfalls in the measurement and comparison of species richness. *Ecology letters*, 4(4):379–391, 2001.

[39] M. Grabherr, B. Haas, M. Yassour, J. Levin, D. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, Z. Chen, E. Mauceli, N. Hacohen, A. Gnirke, N. Rhind, F. di Palma, B. Birren, C. Nusbaum, K. Lindblad-Toh, N. Friedman, and A. Regev. Full-length transcriptome assembly from rna-seq data without a reference genome. *Nat Biotechnol*, 29(7):644–52, 2011.

[40] W. Gu, T. Castoe, D. Hedges, M. Batzer, and D. Pollock. Identification of repeat structure in large genomes using repeat probability clouds. *Anal Biochem*, 380(1):77–83, 2008.

[41] B. J. Haas, A. Papanicolaou, M. Yassour, M. Grabherr, P. D. Blood, J. Bowden, M. B. Couger, D. Eccles, B. Li, M. Lieber, M. D. Macmanes, M. Ott, J. Orvis, N. Pochet, F. Strozzi, N. Weeks, R. Westerman, T. William, C. N. Dewey, R. Henschel, R. D. Leduc, N. Friedman, and A. Regev. De novo transcript sequence reconstruction from rna-seq using the trinity platform for reference generation and analysis. *Nat Protoc*, 8(8):1494–512, Aug 2013.

[42] J. Handelsman, M. R. Rondon, S. F. Brady, J. Clardy, and R. M. Goodman. Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chem Biol*, 5(10):R245–9, Oct 1998.

[43] M. Hess, A. Sczyrba, R. Egan, T.-W. Kim, H. Chokhawala, G. Schroth, S. Luo, D. S. Clark, F. Chen, T. Zhang, R. I. Mackie, L. A. Pennacchio, S. G. Tringe, A. Visel, T. Woyke, Z. Wang, and E. M. Rubin. Metagenomic discovery of biomass-degrading genes and genomes from cow rumen. *Science*, 331(6016):463–7, Jan 2011.

[44] M. O. Hill. Diversity and evenness: a unifying notation and its consequences. *Ecology*, 54(2):427–432, 1973.

[45] A. C. Howe, J. Jansson, S. A. Malfatti, S. G. Tringe, J. M. Tiedje, and C. T. Brown. Assembling large, complex environmental metagenomes. *arXiv preprint*, 12 2012.

[46] A. C. Howe, J. Pell, R. Canino-Koning, R. Mackelprang, S. Tringe, J. Jansson, J. M. Tiedje, and C. T. Brown. Illumina sequencing artifacts revealed by connectivity analysis of metagenomic datasets. *PLoS ONE*, -.

[47] Human Microbiome Jumpstart Reference Strains Consortium, K. E. Nelson, G. M. Weinstock, S. K. Highlander, K. C. Worley, H. H. Creasy, J. R. Wortman, D. B. Rusch, M. Mitreva, E. Sodergren, A. T. Chinwalla, M. Feldgarden, D. Gevers, B. J. Haas, R. Madupu, D. V. Ward, B. W. Birren, R. A. Gibbs, B. Methe, J. F. Petrosino, R. L. Strausberg, G. G. Sutton, O. R. White, R. K. Wilson, S. Durkin, M. G. Giglio, S. Gujja, C. Howarth, C. D. Kodira, N. Kyrpides, T. Mehta, D. M. Muzny, M. Pearson, K. Pepin, A. Pati, X. Qin, C. Yandava, Q. Zeng, L. Zhang, A. M. Berlin, L. Chen, T. A. Hepburn, J. Johnson, J. McCorrison, J. Miller, P. Minx, C. Nusbaum, C. Russ, S. M. Sykes, C. M. Tomlinson, S. Young, W. C. Warren, J. Badger, J. Crabtree, V. M. Markowitz, J. Orvis, A. Cree, S. Ferriera, L. L. Fulton, R. S. Fulton, M. Gillis, L. D. Hemphill, V. Joshi, C. Kovar, M. Torralba, K. A. Wetterstrand, A. Abouellleil, A. M. Wollam, C. J. Buhay, Y. Ding, S. Dugan, M. G. FitzGerald, M. Holder, J. Hostetler, S. W. Clifton, E. Allen-Vercoe, A. M. Earl, C. N. Farmer, K. Liolios, M. G. Surette, Q. Xu, C. Pohl, K. Wilczek-Boney, and D. Zhu. A catalog of reference genomes from the human microbiome. *Science*, 328(5981):994–9, May 2010.

[48] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster. Megan analysis of metagenomic data. *Genome Res*, 17(3):377–86, Mar 2007.

[49] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nat Genet*, 44(2):226–32, 2012.

[50] 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 Res*, 40(22):e171, Dec 2012.

[51] A. L. Kau, P. P. Ahern, N. W. Griffin, A. L. Goodman, and J. I. Gordon. Human nutrition, the gut microbiome and the immune system. *Nature*, 474(7351):327–36, Jun 2011.

[52] D. Kelley, M. Schatz, and S. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.

[53] D. R. Kelley, M. C. Schatz, and S. L. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.

[54] S. Koren, T. Treangen, and M. Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–71, 2011.

[55] S. Kurtz, A. Narechania, J. C. Stein, and D. Ware. A new method to compute K-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, 9(1):517, 2008.

[56] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[57] S.-M. Lee and A. Chao. Estimating population size via sample coverage for closed capture-recapture models. *Biometrics*, pages 88–97, 1994.

[58] X. Li and M. S. Waterman. Estimating the repeat structure and length of dna sequences using l-tuples. *Genome Res*, 13(8):1916–22, Aug 2003.

[59] Y. Li, Y. Hu, L. Bolund, and J. Wang. State of the art de novo assembly of human genomes from massively parallel sequencing data. *Hum Genomics*, 4(4):271–7, 2010.

[60] M. Loreau, S. Naeem, P. Inchausti, J. Bengtsson, J. P. Grime, A. Hector, D. U. Hooper, M. A. Huston, D. Raffaelli, B. Schmid, D. Tilman, and D. A. Wardle. Biodiversity and ecosystem functioning: current knowledge and future challenges. *Science*, 294(5543):804–8, Oct 2001.

[61] C. Luo, D. Tsementzi, N. Kyrpides, T. Read, and K. T. Konstantinidis. Direct comparisons of illumina vs. roche 454 sequencing technologies on the same microbial community dna sample. *PLoS One*, 7(2):e30087, 2012.

[62] W. Luo, M. S. Friedman, K. Shedden, K. D. Hankenson, and P. J. Woolf. Gage: generally applicable gene set enrichment for pathway analysis. *BMC Bioinformatics*, 10:161, 2009.

[63] A. E. Magurran and B. J. McGill. *Biological diversity: frontiers in measurement and assessment*, volume 12. Oxford University Press Oxford, 2011.

[64] K. Malde and B. O'Sullivan. Using bloom filters for large scale gene sequence analysis in haskell. In A. Gill and T. Swift, editors, *PADL*, volume 5418 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2009.

[65] 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.

[66] V. M. Markowitz, I.-M. A. Chen, K. Chu, E. Szeto, K. Palaniappan, Y. Grechkin, A. Ratner, B. Jacob, A. Pati, M. Huntemann, K. Liolios, I. Pagani, I. Anderson, K. Mavromatis, N. N. Ivanova, and N. C. Kyrpides. Img/m: the integrated metagenome data management and comparative analysis system. *Nucleic Acids Res*, 40(Database issue):D123–9, Jan 2012.

[67] O. U. Mason, T. C. Hazen, S. Borglin, P. S. G. Chain, E. A. Dubinsky, J. L. Fortney, J. Han, H.-Y. N. Holman, J. Hultman, R. Lamendella, R. Mackelprang, S. Malfatti, L. M. Tom, S. G. Tringe, T. Woyke, J. Zhou, E. M. Rubin, and J. K. Jansson. Metagenome, metatranscriptome and single-cell sequencing reveal microbial response to deepwater horizon oil spill. *ISME J*, 6(9):1715–27, Sep 2012.

[68] R. M. May. How many species are there on earth? *Science*, 241(4872):1441–9, Sep 1988.

[69] E. McDonald and C. T. Brown. khmer: Working with big data in bioinformatics. *arXiv preprint*, 03 2013.

[70] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, 27(13):i137–41, Jul 2011.

[71] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, 27(13):i137–41, 2011.

[72] P. Melsted and J. K. Pritchard. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC bioinformatics*, 12:333, Jan. 2011.

[73] M. Metzker. Sequencing technologies - the next generation. *Nat Rev Genet*, 11(1):31–46, 2010.

[74] J. Miller, S. Koren, and G. Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–27, 2010.

[75] A. E. Minoche, J. C. Dohm, and H. Himmelbauer. Evaluation of genomic high-throughput sequencing data generated on illumina hiseq and genome analyzer systems. *Genome Biol*, 12(11):R112, 2011.

[76] C. Mora, D. P. Tittensor, S. Adl, A. G. B. Simpson, and B. Worm. How many species are there on earth and in the ocean? *PLoS Biol*, 9(8):e1001127, Aug 2011.

[77] X. C. Morgan, T. L. Tickle, H. Sokol, D. Gevers, K. L. Devaney, D. V. Ward, J. A. Reyes, S. A. Shah, N. LeLeiko, S. B. Snapper, A. Bousvaros, J. Korzenik, B. E. Sands, R. J. Xavier, and C. Huttenhower. Dysfunction of the intestinal microbiome in inflammatory bowel disease and treatment. *Genome Biol*, 13(9):R79, 2012.

[78] S. Muthukrishnan. *Data streams: algorithms and applications*. Foundations and trends in theoretical computer science. Now Publishers, 2005.

[79] N. Nagarajan and M. Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J Comput Biol*, 16(7):897–908, 2009.

[80] T. Namiki, T. Hachiya, H. Tanaka, and Y. Sakakibara. MetaVelvet: An extension of Velvet assembler to de novo metagenome assembly from short sequence reads. *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2011.

[81] P. Novak, P. Neumann, and J. Macas. Graph-based clustering and characterization of repetitive sequences in next-generation sequencing data. *BMC Bioinformatics*, 11:378, 2010.

[82] K. R. Patil, L. Roune, and A. C. McHardy. The phylopythias web server for taxonomic assignment of metagenome sequences. *PLoS One*, 7(6):e38581, 2012.

[83] 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. *Proc Natl Acad Sci U S A*, 109(33):13272–7, Aug 2012.

[84] Y. Peng, H. Leung, S. Yiu, and F. Chin. Meta-idba: a de novo assembler for metagenomic data. *Bioinformatics*, 27(13):i94–101, 2011.

[85] F. Pérez and B. Granger. Ipython: A system for interactive scientific computing. *Computing in Science Engineering*, 9(3):21–29, 2007.

[86] F. Pérez and B. E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[87] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proc Natl Acad Sci U S A*, 98(17):9748–53, Aug 2001.

[88] J. Pickrell, A. Pai, Y. Gilad, and J. Pritchard. Noisy splicing drives mrna isoform diversity in human cells. *PLoS Genet*, 6(12):e1001236, 2010.

[89] A. Pinho, D. Pratas, and S. Garcia. Green: a tool for efficient compression of genome resequencing data. *Nucleic Acids Res*, 40(4):e27, 2012.

[90] J. Qin, R. Li, J. Raes, M. Arumugam, K. S. Burgdorf, C. Manichanh, T. Nielsen, N. Pons, F. Levenez, T. Yamada, D. R. Mende, J. Li, J. Xu, S. Li, D. Li, J. Cao, B. Wang, H. Liang, H. Zheng, Y. Xie, J. Tap, P. Lepage, M. Bertalan, J.-M. Batto, T. Hansen, D. Le Paslier, A. Linneberg, H. B. Nielsen, E. Pelletier, P. Renault, T. Sicheritz-Ponten, K. Turner, H. Zhu, C. Yu, S. Li, M. Jian, Y. Zhou, Y. Li, X. Zhang, S. Li, N. Qin, H. Yang, J. Wang, S. Brunak, J. Doré, F. Guarner, K. Kristiansen, O. Pedersen, J. Parkhill, J. Weissenbach, MetaHIT Consortium, P. Bork, S. D. Ehrlich, and J. Wang. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, Mar 2010.

[91] J. Qin, R. Li, J. Raes, M. Arumugam, K. S. Burgdorf, C. Manichanh, T. Nielsen, N. Pons, F. Levenez, T. Yamada, D. R. Mende, J. Li, J. Xu, S. S. S. Li, D. Li, J. Cao, B. Wang, H. Liang, H. Zheng, Y. Xie, J. Tap, P. Lepage, M. Bertalan, J.-M. Batto, T. Hansen, D. Le Paslier, A. Linneberg, H. B. r. Nielsen, E. Pelletier, P. Renault, T. Sicheritz-Ponten, K. Turner, H. Zhu, C. Yu, M. Jian, Y. Zhou, Y. Li, X. Zhang, N. Qin, H. Yang, J. J. Wang, S. r. Brunak, J. Doré, F. Guarner, K. Kristiansen, O. Pedersen, J. Parkhill, J. Weissenbach, P. Bork, and S. D. Ehrlich. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, 2010.

[92] J. Qin, Y. Li, Z. Cai, S. Li, J. Zhu, F. Zhang, S. Liang, W. Zhang, Y. Guan, D. Shen, Y. Peng, D. Zhang, Z. Jie, W. Wu, Y. Qin, W. Xue, J. Li, L. Han, D. Lu, P. Wu, Y. Dai, X. Sun, Z. Li, A. Tang, S. Zhong, X. Li, W. Chen, R. Xu, M. Wang, Q. Feng, M. Gong, J. Yu, Y. Zhang, M. Zhang, T. Hansen, G. Sanchez, J. Raes, G. Falony, S. Okuda, M. Almeida, E. LeChatelier, P. Renault, N. Pons, J.-M. Batto, Z. Zhang, H. Chen, R. Yang, W. Zheng, S. Li, H. Yang, J. Wang, S. D. Ehrlich, R. Nielsen, O. Pedersen, K. Kristiansen, and J. Wang. A metagenome-wide association study of gut microbiota in type 2 diabetes. *Nature*, 490(7418):55–60, Oct 2012.

[93] C. Quince, T. P. Curtis, and W. T. Sloan. The rational exploration of microbial diversity. *ISME J*, 2(10):997–1006, Oct 2008.

[94] G. Rizk, D. Lavenier, and R. Chikhi. Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–3, Mar 2013.

[95] S. Rodrigue, R. Malmstrom, A. Berlin, B. Birren, M. Henn, and S. Chisholm. Whole genome amplification and de novo assembly of single bacterial cells. *PLoS One*, 4(9):e6864, 2009.

[96] L. F. W. Roesch, R. R. Fulthorpe, A. Riva, G. Casella, A. K. M. Hadwin, A. D. Kent, S. H. Daroub, F. A. O. Camargo, W. G. Farmerie, and E. W. Triplett. Pyrosequencing enumerates and contrasts soil microbial diversity. *ISME J*, 1(4):283–90, Aug 2007.

[97] G. L. Rosen, E. R. Reichenberger, and A. M. Rosenfeld. Nbc: the naive bayes classification tool webserver for taxonomic classification of metagenomic reads. *Bioinformatics*, 27(1):127–9, Jan 2011.

[98] R. S. Roy, D. Bhattacharya, and A. Schliep. Turtle: Identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics: Advance Access published March 10, 2014*, page doi: 10.1093/bioinformatics/btu132, Apr 2014.

[99] F. Rusu and A. Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems*, 33(3):1–46, Aug. 2008.

[100] D. C. Savage. Microbial ecology of the gastrointestinal tract. *Annu Rev Microbiol*, 31:107–33, 1977.

[101] A. Sboner, X. Mu, D. Greenbaum, R. Auerbach, and M. Gerstein. The real cost of sequencing: higher than you think! *Genome Biol*, 12(8):125, 2011.

[102] M. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–9, 2009.

[103] P. D. Schloss and J. Handelsman. Introducing dotur, a computer program for defining operational taxonomic units and estimating species richness. *Appl Environ Microbiol*, 71(3):1501–6, Mar 2005.

[104] P. D. Schloss and J. Handelsman. Toward a census of bacteria in soil. *PLoS Comput Biol*, 2(7):e92, Jul 2006.

[105] P. D. Schloss, S. L. Westcott, T. Ryabin, J. R. Hall, M. Hartmann, E. B. Hollister, R. A. Lesniewski, B. B. Oakley, D. H. Parks, C. J. Robinson, J. W. Sahl, B. Stres, G. G. Thallinger, D. J. Van Horn, and C. F. Weber. Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Appl Environ Microbiol*, 75(23):7537–41, Dec 2009.

[106] T. M. Schmidt, E. F. DeLong, and N. R. Pace. Analysis of a marine picoplankton community by 16s rrna gene cloning and sequencing. *J Bacteriol*, 173(14):4371–8, Jul 1991.

[107] M. Schulz, D. Zerbino, M. Vingron, and E. Birney. Oases: robust de novo rna-seq assembly across the dynamic range of expression levels. *Bioinformatics*, 28(8):1086–92, 2012.

[108] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, and C. Huttenhower. Metagenomic microbial community profiling using unique clade-specific marker genes. *Nat Methods*, 9(8):811–4, Aug 2012.

[109] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[110] E. H. Simpson. Measurement of diversity. *Nature*, 1949.

[111] J. Simpson and R. Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–73, 2010.

[112] J. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*, 22(3):549–56, 2012.

[113] J. Simpson, K. Wong, S. Jackman, J. Schein, S. Jones, and I. Birol. Abyss: a parallel assembler for short read sequence data. *Genome Res*, 19(6):1117–23, 2009.

[114] M. Soares, M. Bonaldo, P. Jelene, L. Su, L. Lawton, and A. Efstratiadis. Construction and characterization of a normalized cdna library. *Proc Natl Acad Sci U S A*, 91(20):9228–32, 1994.

[115] M. L. Sogin, H. G. Morrison, J. A. Huber, D. Mark Welch, S. M. Huse, P. R. Neal, J. M. Arrieta, and G. J. Herndl. Microbial diversity in the deep sea and the underexplored "rare biosphere". *Proc Natl Acad Sci U S A*, 103(32):12115–20, Aug 2006.

[116] C. Spits, C. L. Caignec, M. D. Rycke, L. V. Haute, A. V. Steirteghem, I. Liebaers, and K. Sermon. Whole-genome multiple displacement amplification from single cells. *Nat Protoc*, 1(4):1965–70, 2006.

[117] E. Stackebrandt, W. Frederiksen, G. M. Garrity, P. A. D. Grimont, P. Kämpfer, M. C. J. Maiden, X. Nesme, R. Rosselló-Mora, J. Swings, H. G. Trüper, L. Vauterin, A. C. Ward, and W. B. Whitman. Report of the ad hoc committee for the re-evaluation of the species definition in bacteriology. *Int J Syst Evol Microbiol*, 52(Pt 3):1043–7, May 2002.

[118] L. Stein. The case for cloud computing in genome informatics. *Genome Biol*, 11(5):207, 2010.

[119] H. Teeling, J. Waldmann, T. Lombardot, M. Bauer, and F. O. Glöckner. Tetra: a web-service and a stand-alone program for the analysis and comparison of tetranucleotide usage patterns in dna sequences. *BMC Bioinformatics*, 5:163, Oct 2004.

[120] C. Trapnell and S. Salzberg. How to map billions of short reads onto genomes. *Nat Biotechnol*, 27(5):455–7, 2009.

[121] P. J. Turnbaugh, M. Hamady, T. Yatsunenko, B. L. Cantarel, A. Duncan, R. E. Ley, M. L. Sogin, W. J. Jones, B. A. Roe, J. P. Affourtit, M. Egholm, B. Henrissat, A. C. Heath, R. Knight, and J. I. Gordon. A core gut microbiome in obese and lean twins. *Nature*, 457(7228):480–4, Jan 2009.

[122] G. W. Tyson, J. Chapman, P. Hugenholtz, E. E. Allen, R. J. Ram, P. M. Richardson, V. V. Solovyev, E. M. Rubin, D. S. Rokhsar, and J. F. Banfield. Community structure and metabolism through reconstruction of microbial genomes from the environment. *Nature*, 428(6978):37–43, Mar 2004.

[123] S. Tzahor, D. Man-Aharonovich, B. C. Kirkup, T. Yogev, I. Berman-Frank, M. F. Polz, O. Béjà, and Y. Mandel-Gutfreund. A supervised learning approach for taxonomic classification of core-photosystem-ii genes and transcripts in the marine environment. *BMC Genomics*, 10:229, 2009.

[124] J. C. Venter, K. Remington, J. F. Heidelberg, A. L. Halpern, D. Rusch, J. A. Eisen, D. Wu, I. Paulsen, K. E. Nelson, W. Nelson, D. E. Fouts, S. Levy, A. H. Knap, M. W. Lomas, K. Nealson, O. White, J. Peterson, J. Hoffman, R. Parsons, H. Baden-Tillson, C. Pfannkoch, Y.-H. Rogers, and H. O. Smith. Environmental genome shotgun sequencing of the sargasso sea. *Science*, 304(5667):66–74, Apr 2004.

[125] Q. Wang, G. M. Garrity, J. M. Tiedje, and J. R. Cole. Naive bayesian classifier for rapid assignment of rrna sequences into the new bacterial taxonomy. *Appl Environ Microbiol*, 73(16):5261–7, Aug 2007.

[126] W. B. Whitman, D. C. Coleman, and W. J. Wiebe. Prokaryotes: the unseen majority. *Proc Natl Acad Sci U S A*, 95(12):6578–83, Jun 1998.

[127] T. Woyke, A. Sczyrba, J. Lee, C. Rinke, D. Tighe, S. Clingenpeel, R. Malmstrom, R. Stepanauskas, and J. Cheng. Decontamination of mda reagents for single cell whole genome amplification. *PLoS One*, 6(10):e26161, 2011.

[128] D. Wu, P. Hugenholtz, K. Mavromatis, R. Pukall, E. Dalin, N. N. Ivanova, V. Kunin, L. Goodwin, M. Wu, B. J. Tindall, S. D. Hooper, A. Pati, A. Lykidis, S. Spring, I. J. Anderson, P. D'haeseleer, A. Zemla, M. Singer, A. Lapidus, M. Nolan, A. Copeland,

C. Han, F. Chen, J.-F. Cheng, S. Lucas, C. Kerfeld, E. Lang, S. Gronow, P. Chain, D. Bruce, E. M. Rubin, N. C. Kyrpides, H.-P. Klenk, and J. A. Eisen. A phylogeny-driven genomic encyclopaedia of bacteria and archaea. *Nature*, 462(7276):1056–60, Dec 2009.

[129] M. Wu and A. J. Scott. Phylogenomic analysis of bacterial and archaeal sequences with amphora2. *Bioinformatics*, 28(7):1033–4, Apr 2012.

[130] I. Zarraonaindia, D. P. Smith, and J. A. Gilbert. Beyond the genome: community-level analysis of the microbial world. *Biol Philos*, 28(2):261–282, Mar 2013.

[131] D. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18(5):821–9, 2008.

[132] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18(5):821–9, May 2008.