Algorithms for Multisample Read Binning

Gabriel Sebastian Ioan Ilie

B.S. Computer Science, University of Bucharest, Romania, 2012

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

2013

APPROVAL PAGE

Master of Science Thesis

Algorithms for Multisample Read Binning

Presented by

Gabriel Sebastian Ioan Ilie, B.S. Computer Science

Major Advisor	Ion Măndoiu
Associate Advisor	Sanguthevar Rajasekaran
Associate Advisor	Yufeng Wu

University of Connecticut

2013

ACKNOWLEDGEMENTS

I would like to thank my major advisor Dr. Ion Măndoiu and Dr. Alexander Zelikovsky for their direct contributions to this thesis. I would also like to thank my associate advisors, friends and family for their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Metatranscriptomics is the study of gene expression levels in uncultured microbial communities sampled directly from their environment. In cultured microbes, the sequencing data comes from a single clone, making assembly and annotation tractable. However, in metatranscriptomics, the data comes from communities with diverse microbial compositions, possibly containing thousands of species. The assembly of transcripts from these complex samples represents a serious computational challenge.

The development of *genome-independent methods* is essential because of the lack of trusted reference genomes that can be used to guide the assembly process. The predominant assembly formalism used for short-read datasets is the *de Bruijn graph*. In a *de Bruijn graph*, reads are split into consecutive overlapping words, or *k-mers*, which are then used to build a connectivity graph. In order to reduce the complexity and improve the results of the assembly process, *genome-independent* methods which cluster the reads into bins can be employed. Previous methods for read clustering process single samples only. In this thesis we propose a novel method for binning reads from $N > 1$ metatranscriptomics samples. In the first stage we run a sample-by-sample error removal algorithm to remove as many of the incorrect *k-mers* as possible from the *de Bruijn graph*. In the second stage we use the structure of the de Bruijn graph and the N-dimensional count vectors for each *k-mer* to identify paths in the *de Bruijn graph*

that correspond to maximal substrings which appear in the same transcripts with the same multiplicity (*pseudo-exons*).

In order to validate our algorithm, we simulated RNA-Seq data (both error-free and with errors) using real expression levels of human genes from the *GNFAtlas2* project [8]. We created several different workflows for processing the data and managed to obtain promising results. One of the biggest challenges we faced was dealing with errors. Since our algorithms are based on *k-mers* a single error in a read results in multiple erroneous *k-mers*. For an error rate of 0.1% over two thirds of the *k-mers* are incorrect, however, we managed to achieve a very good balance between how many correct *k-mers* we keep (47,708,616 out of 49,540,693) and how many erroneous ones we keep (29,583 out of 108,528,982).

Chapter 1
Introduction

Microbial organisms are an essential part of life on Earth, as they are the primary source of nutrients and the primary recyclers of dead matter back to available organic form. Not only this, bacteria and archaea live in all environments capable of sustaining any other form of life and in many cases are the sole inhabitants of extreme environments.

Every living animal or plant shares an intimate relationship with these microorganisms. It is estimated that we humans have more bacterial cells (10^{14}) in our bodies than our own cells (10^{13}) [11]. Also, the development of the immune system, Crohns disease, diabetes, obesity, cancer, attractiveness to mosquitoes and perhaps even anxiety seem to correlate with changes in the microorganisms inhabiting our bodies (the *microbiome*) [9]. Therefore, in order to understand our own condition it is insufficient to research just our own genome, we also need to understand the *microbiome*.

Microbial studies focusing on single organisms have their limits. Firstly, we can culture only a very small percentage of the microorganisms found in nature, which means that existent genomic data is highly biased and does not represent a true picture of the diversity of microbial life. Secondly, microbes do not live in single species communities: species interact both with each other and with their habitats, which may include host organisms. Therefore, clonal cultures fail to represent the interactions of organisms found in nature.

Initial studies of the microbiome were mostly surveys of the microbial composition of these communities done by sequencing a variable region of the 16s rRNA gene. Further advances in sequencing technologies have recently enabled moving from sequencing 16S rRNA genes to shotgun metagenome sequencing, in which bulk DNA or RNA is extracted directly from environmental samples and shotgun sequenced.

In metagenomics, shotgun sequencing is done in the same manner as in clonal culture genomics. However, the reads obtained can come from any of the possibly hundreds of unknown species present in the sample. Species in environmental samples have different abundances, therefore the number of reads coming from a genome is proportional with the relative abundance of that genome.

The volume of sequencing data from environmental samples is several orders of magnitude larger than that acquired in single organism genomics, increasing the complexity of analysis. While some species account for 20% of a community others may be present at concentrations below 0.1%, but their physiological contributions can play an important role in the physiology of the entire microbiome. This greatly affects how much sequence information needs to be generated. The sequences can be analyzed using two different approaches, reference-based mapping or de novo assemblies.

The reference-based mapping requires comprehensive reference genomes. In this approach, the reads, which are a few hundred base pairs long at most, are mapped to the closest match from the reference dataset. For most microorganisms there is no reference genome available, since a large number of them cannot be cultivated, so they will not be mapped against. Also, many microorganisms vary dramatically in their gene content, for example three strains of Escherichia Coli may have as little as 40% of their genome in common. So if only one strain was sequenced, 60% of the genes of the other strains would not be part of the reference and would prevent reads from being mapped to it.

An alternative approach is de novo assembly where the reads are assembled from scratch. Clearly this is a more computationally intensive task and posses significant challenges. (1) Due to the differences in abundance, organisms present at low numbers may not be sequenced to a sufficient depth while organisms present at a high coverage may be sequenced at too great of a depth leading to the breakage of assemblies due to sequencing errors. (2) The presence of closely related organisms complicates the assembly. The rate of change and consequently the similarity between genes varies greatly. For example, ribosomal proteins evolve very slowly and are thus highly conserved, housekeeping genes evolve at a moderate rate while some surface proteins evolve rapidly. This leads to two different organisms potentially having identical sequences for some genes interspersed with distantly related genes, which in turn can lead to false assemblies. (3) Another great challenge is linking the contigs to specific organisms and this can be achieved using nucleotide usage patterns and differential coverage in case multiple samples are available.

While metagenomes provide important insights, they only provide information about the potential functions of the microorganisms, but having any given gene does not mean that this gene is also expressed inside the host or during a particular condition. In order to understand the physiology of the microbiome we need to use metatranscriptomic studies. In metatranscriptomic studies, RNA is isolated from the microbial community, reverse transcribed, and the resulting RNA-seq libraries are sequenced. The standard approach is to perform reference based-mapping, which is followed by de novo assembly of the reads that did not map to the reference genomes. Thus analysis of metatranscriptomes has to deal with all of the challenges of metagenomic approaches and the following extra challenges: (1) in addition to having a range in the abundance levels of the organisms, genes are expressed at drastically different levels, with elongations factors, ribosomal proteins and short regulatory RNAs being expressed at the highest levels, while regulatory genes are expressed at very low levels. This further increases the amount of sequence data that needs to be obtained. (2) Strand-specificity: regulatory RNAs are short and complementary to the mRNA. If the RNAseq libraries are not prepared in a strand-specific fashion or not analyzed in a manner that keeps track of strand specificity, it is challenging to differentiate regulatory RNAs from mRNA, leading to false expression values. (3) In contrast to eukaryotes, there are typically no splice variants in bacteria.

In this thesis, we propose a novel method for *unsupervised coverage-based multi-sample reads binning*. This new algorithm can be used as a preprocessing step for the *de novo* metatranscriptome assembly of all RNA-seq samples.

Chapter 2
Previous Work

Metagenomic studies have provided valuable insights into the uncultured microbial world. However, because of the vast amounts of sequencing data produced from environmental samples and the complexity of the assembly process, many computational tools have been developed to infer species information from raw short reads directly.

One of the primary goals of metagenomic projects is to identify the organisms present in a sample. Because assembly of metagenomic data is so difficult, many tools have been developed recently to classify reads into different bins (i.e. species) based on various characteristics. These tools can be broadly split into two categories: genome-dependent and genome-independent techniques. Genome-dependent techniques rely on various DNA compositions patterns, e.g. tetra-mer frequencies. The basis of these approaches is that genome G+C content, dinucleotide frequencies, and synonymous codon usage vary among species. However, most of these techniques achieve a reasonable performance only for long reads. For short reads there is too much variation of DNA composition patterns within a single genome.

Genome-independent techniques usually perform the binning based on the abundances of reads or *k-mers*. These methods rely on the following observations: (1) the *k-mer* frequencies from reads of a genome are usually linearly proportional to the genome's abundance, and (2) sufficiently long *k-mers* are usually unique in each genome.

The main drawbacks of these methods are: (1) that they group together reads from different species if they have close abundance levels, and (2) that they do not perform well on species with low abundances.

[12] describes a binning tool, called AbundanceBin, which can be used to classify very short reads sampled from species with different abundance levels. This method relies on the assumption that reads are sampled from genomes following a Poisson distribution [4]. In this context, the reads are viewed as coming from a mixture of Poisson distributions. Their algorithm starts by counting the *k-mers* present in the reads. Under the assumption that the number of bins is known, they use an *Expectation-Maximization* (EM) algorithm to infer the parameters of the Poisson distribution, which reflect the relative abundance levels of each bin. Afterwards, each *k-mer* is assigned to a bin using the fitted Poisson distribution. The *k-mer* bins, are then used to classify the reads into the same number of bins.

In the EM algorithm, the number of bins needs to be given as an input parameter. Because this number is often unknown, the authors have implemented a recursive approach to determine this number automatically. This approach works by initially splitting the dataset into two bins and continuing to recursively split the bins while any of the following conditions are true: (1) the predicted abundance values of the two bins differ significantly; (2) the predicted genome sizes are larger than a certain threshold; and (3) the number of reads associated with a bin is larger than a certain threshold. The main advantage of this method is that it has the ability to perform unsupervised binning of short reads from species with different abundances.

Another approach to reads binning was proposed in [10]. A two rounds unsupervised binning method, called*MetaCluster 5.0*, to identify both low abundance and high abundance species in the presence of noise. To overcome the issue of low coverage for low abundance species, MetaCluster 5.0 separates the high abundance reads from those of low abundance. Multiple values for *k* are used to group the reads: large *k* for high abundance and small *k* for low abundance.

In the first round, MetaCluster 5.0 filters reads from low abundance genomes as well as reads with errors. Based on the observation that *k-mer* frequency from reads of a genome is usually linearly proportional to that genome's abundance, reads with all *16-mers* appearing rarely in the dataset are likely to be sampled from low abundance genomes or be the result of sequencing errors. The reads are then grouped based on common *36-mers*. Each group represents a *virtual contig* of a genome, groups whose virtual contigs are of length smaller than a certain threshold are removed. The *5-mer* distribution of each virtual contig is estimated and the virtual contigs are grouped together using *K-means clustering*.

After the first round, reads sampled from high abundance genomes have been binned. In the second round, the remaining reads, the ones sampled from low abundance organisms are binned. Reads whose *16-mers* appear only once are discarded (they are probably caused by errors or are from extremely low abundance genomes). Just like in the previous round, the reads are grouped into virtual contigs based on the frequency of their *22-mers*. This time the *4-mer* distribution of each virtual contig is used to group together the reads using *K-means clustering*.

The main disadvantage of the previous binning method, AbundanceBin, was that it could not distinguish between different species which had the same relative abundances. Thanks to its two round approach MetaCluster 5.0 manages to overcome this disadvantage and produce better results.

Common to all of the methods presented so far, is the processing of single samples only. In [1], the authors use multiple samples with varying relative abundances of identical target community members to perform the binning of the reads The samples can be produced by using different extraction methods or by spatial or temporal sampling.

Using different extraction methods, they produced two paired-end sequence datasets with different relative abundances of community members. The larger dataset was assembled into scaffolds, and reads from both sets were mapped unto these scaffolds giving two coverage estimates. The different coverage is due to extraction specific efficiencies. The binning of scaffolds is done by plotting the two coverage estimates against each other for all scaffolds. Scaffolds clustering together, represent possible genomes, however, in order to get good results some manual curration is required.

Another read clustering algorithm which operates on multiple samples was proposed in [3]. Just as in the previous binning method, they rely on the assumption that multiple samples contain the same microbial species, possibly in different proportions. The algorithm divides the reads from the N samples, into b bins which correspond to the species from which they were sequenced, b is assumed to be known.

The binning algorithm, called MultiBin, works as follows: (1) The reads from all the samples are pooled together. A graph $G = (V, E)$ is generated where V is the set of reads and E corresponds to pairs of reads with a substantial overlap. (2) A maximal independent set of G is found using a greedy algorithm. The reads in this set are called *tags* and are denoted by T, each read is either a tag or is affiliated with a single tag. For each tag t, let c_{ti} be the number of reads from sample i substantially overlap it. (3) *K-medoids* clustering is performed on the set $\{(c_{t1}, c_{t2}, \ldots, c_{tN}) | t \in T\}$, starting from a random choice of b centers. After convergence the tags will be divided into bins, every non-tag read will be assigned to the same bin as its affiliated (neighboring) tag read.

Single sample binning algorithms struggle to separate species which have the same relative abundance level. Algorithms which operate on multiple samples, can overcome this disadvantage because they can use the abundance differences in any of the samples to tell between such species. Another advantage of multiple sample binning is that older datasets can be combined to improve the results of the analysis of new samples. Unfortunately, MultiBin does not scale well to an increase in the number of samples because the algorithm is quadratic in the number of reads.

Chapter 3
Multisample Read Binning

In this thesis, we propose a novel method for *unsupervised coverage-based multi-sample reads binning*. This new algorithm can be used as a preprocessing step for the *de novo* metatranscriptome assembly of N RNA-seq samples. In brief, our method consists of the following steps: (1) we compute the N-dimensional count vectors for each *k-mer* and *(k+1)-mer* which appears in the reads. The lists of *k-mers* and *(k+1)-mers* present in the reads give us an implicit representation for the *de Bruijn* graph in which the set of vertices is the set of *k-mers* and the set of edges is the set of *(k+1)-mers*. (2) Next, we run different error removal algorithms in order to remove as many of the incorrect *k-mers* and *(k+1)-mers* as possible. We apply the error removal algorithms in different combinations on a sample-by-sample basis, on the union of the samples or on both. (3) In the final step of our method, we try to identify the connected components of the *de Bruijn* graph that correspond to *maximal substrings which are expressed in the same transcripts with the same multiplicity*, which we call *pseudo-exons*. In order to identify these connected components, we use the structure of the graph and the vectors with the counts to remove edges which connect *pseudo-exons*. The identified connected components correspond to paths or cycles in the graph, which can then be trivially assembled into contigs.

In the following sections we detail each of the steps outlined above and also provide some preliminary results.

3.1 K-mer counting

For performing the counting we use a very efficient *k-mer* counter called **Jellyfish** [6]. Jellyfish was designed for shared memory parallel computers with more than one core, it uses several lock-free data structures and multi-threading to count *k-mers* much faster than other tools.

Formally, for a given value of k, we count the number of occurrences of all *k-mers* in each of the N samples. These counts are then normalized to obtain samples-specific relative abundances. Most of the times, we lack information about which strand the sequencing data comes from, therefore the counts of complementary *k-mers* are summed together as they are indistinguishable. The smaller value lexicographically between a *k-mer* and its reverse-complement is called the **canonical representation** of that *k-mer*. We combine the counts over all the samples into a list of N-dimensional vectors.

From the data obtained by Jellyfish we use not only the N-dimensional count vectors, but also the list of *canonical k-mers* present in the reads, this is necessary in order to construct the *de Bruijn* graph. In this graph, vertices are the *canonical k-mers*. If we were to follow the classical definition of a *de Bruijn* graph we would have to put a directed edge from vertex u to vertex v if the $k-1$ length suffix of

u is the same as the $k-1$ prefix of v. However, we want to model only edges (or *(k+1)-mers*) which appear in the reads, therefore we put an edge from vertex u to vertex v only if there exists a *(k+1)-mer* in the reads such that, for itself or its reverse-complement, the following are true: (1) the prefix of length k in canonical form is equal to u, and (2) the suffix of length k in canonical form is equal to v.

In order to achieve this accurate representation of connectivity in the *de Bruijn* graph, knowing the set of *k-mers* present in the reads is not sufficient. Therefore, besides computing the relative abundances of *k-mers* (or vertices), we also get the relative abundances of *(k+1)-mers* (or edges).

Notice that just by knowing which *k-mers* and *(k+1)-mers* are present in the reads we already have an implicit representation of the *de Bruijn* graph, so there is no need to explicitly construct it in memory because we can efficiently check if two vertices are connected.

Due to limitations imposed by Jellyfish, the maximum value for k is 31, we use $k = 30$, so vertices are *30-mers* and edges are *31-mers*. For the rest of this thesis, we will consider the vertices to be synonymous with *k-mers*, and edge to be synonymous with *(k+1)-mers*.

3.2 Error-removal algorithm

A very common coverage-based error removal method applied to *k-mers* obtained from the short sequencing reads produced by Next Generation Sequencing technologies (NGS) is to remove unique *k-mers*. Usually, a *k-mer* which appears only once in the reads is considered to have been caused by a sequencing error. While this assumption holds for the classical approach where just one genome is sequenced with high average coverage, however in the context of metatranscriptomic (and metagenomic) data, we discovered that we lose to much information. Because of the different abundance levels, removing unique *k-mers* compromises the reconstruction of low abundance transcripts (genomes).

Much better approaches are two coverage-independent error-removal techniques called **tip removal** and **bubble removal** [13]. We expect *pseudo-exons* to have a path-like structure in the *de Bruijn* graph. Consider a read which contains exactly one substitution somewhere in the middle. If the read is long enough, the first *k-mers* produced from its 3′ end may not contain any errors, these beginning *k-mers* would be correct, however, once the erroneous *k-mers* are reached, these will create an extra branch (a new path) going out of the ′′last″ correct one. This "wrong" branch will either end in a leaf, creating a **tip**, or if the read is long enough and we reach another "correct" *k-mer* it will reconnect with the path and create a **bubble** see fig. 3.2.1.

In [13] the authors use $2k$ as the maximum length of a tip. This number is a result of the observation that a single sequencing error can create up to $(2k-1)$ erroneous *k-mers*. Therefore, tips or bubbles longer than $2k$ represent either a genuine sequence or an accumulation of errors. Distinguishing between these two cases is almost impossible.

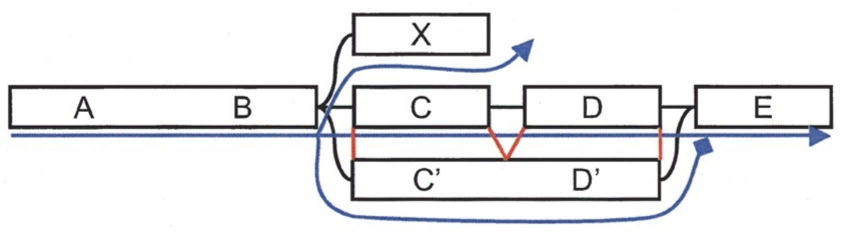

Figure 3.2.1: X represents a tip, while $C'D'$ is an example of a bubble. Figure reproduced from [13].

A "tip" is a chain of nodes that is disconnected on one end. Removing these tips is a straightforward task, discarding this information results in only local effects, as no connectivity is disrupted: 1) In order to find the tips, we first find the leaves of the *de Bruijn* graph. 2) We sort these leaves from the least abundant one (the sum of the coverages in all samples) to the most abundant one. 3) We start from the least abundant tip and process each of them in non-decreasing order of coverage. We start from the leaf and go over the path starting from it (tips have to be paths) to determine or not it is part of a tip. If a tip is detected, then we remove all edges between the vertices which are part of it. By processing tips in this manner we prioritize the removal of low abundance tips. We expect true paths which come from low abundance transcripts to be longer than the length maximum allowed tip length.

For bubble removal, [13] describes an algorithm called *Tour Bus*. Two *paths redundant* if they start and end at the same nodes (forming a "bubble") and contain similar sequences. Detection of redundant paths is done through a

Dijkstra-like breadth-first search. If the sequences of two redundant paths are judged to be similar enough, they are merged. Notice that these error removal techniques are applied directly to the graph, therefore they remove errors in the *k-mers* and *(k+1)-mers* simultaneously.

In our implementation of bubble removal, we require that the two (redundant) paths have the same length. We remove the least abundant one only if the ratio between the abundant one and the sum of the two is higher than some threshold.

An improvement we are considering is to add an extra step, before the error removal process, to try and correct the reads. One such error correction tool, which is designed for RNA-seq data, is called **SEECER** [5]. SEECER is a probabilistic tool, which uses a hidden Markov Model to perform error correction; it can handle not only sequencing errors but also non-uniform abundance levels and alternative splicing. Using error correction algorithms on the reads before performing the counting of *k-mers* might improve our current results.

3.3 Identifying pseudo-exons

Pseudo-exons are defined as connected components of the *de Bruijn* graph that correspond to *maximal substrings which appear in the same transcripts with the same multiplicity*. Note that, if we ignore self-edges, *pseudo-exons* are *paths* or *chordless cycles*. Chordless cycles are cycles such that no two vertices are connected by an edge which does not belong to the cycle.

Regarding the *de Bruijn* graph, notice that for each *k-mer* (vertex) we can define *in and out degrees* relative to its canonical representation. If the canonical representation of a *k-mer* matches the 5′ end of one of its edges or its reverse complement, then we consider this edge to be outgoing, otherwise we say its an incoming edge.

In the error corrected graph, we expect paths to correspond to *pseudo-exons*. We distinguish between the following cases: (1) if a *k-mer* is on a path, meaning it has in-degree and out-degree at most one, we keep it; (2) if a *k-mer* has out degree (in degree) greater then 1, then we remove all edges, but we keep the most abundant one (*(k+1)-mer*): if the ratio between its relative frequency and that of the sum of the other edges going out (coming in) of the current vertex, is greater than a threshold $\epsilon \in (0, 1)$, and if the correlation between the current node's vector of counts and the other ones is above a threshold $\rho \in (0, 1)$, otherwise we discard it. We expect the abundances of true *k-mers* to dominate the abundances of erroneous ones by a very large factor, so ϵ should be close to 1.

Assuming the data did not contain any sequencing errors, then finding the *pseudo-exons* would be easy because all we would need to do is remove all of the (incoming/outgoing) edges for which at least one of its end points has (in/out) degree more than 1. Because real data is not perfect, the edge removal procedure needs to be able to distinguish between real edges and false ones. We expect that if an incorrect edge "survives" the error-removal step, its relative coverage will be much smaller than the correct edges coming out of the same vertex, so, in this case, we would keep only the correct edge and discard all the others because the ratio of the correct one and the sum of the others would be greater than ϵ. However, if there are two correct edges coming out of the same vertex we want to remove both of them. In this case, the ratio between their relative abundances should be smaller than ϵ so both edges are removed.

Chapter 4
Results

In order to validate our method we simulated short RNA-seq reads from 10 different human tissues: amygdala, appendix, bonemarrow, bronchial epithelial cells, heart, kidney, liver, lung, ovary and pancreas. The tissues and their expression levels have been extracted from the **GNF Atlas** dataset [8]. This dataset contains tissue-specific RNA expression levels for $19,371$ genes. The genes were extracted from the **HG18** reference human genome dataset downloaded from the *UCSC Genome Browser*. For these experiments we limited ourselves to only *one isoform per gene*.

For simulating sequencing reads we used an open-source tool called **Grinder** [2]. Grinder was particularly useful because it is able to simulate environmental microbial communities. We used this feature to simulate tissue specific *error free Illumina* RNA-seq datasets, by giving it as input the set of transcripts (instead of genomes) and the expression levels (instead of abundance levels) associated with the tissues. We simulated 30 *million paired-end reads of length 50* for each dataset. However, Grinder has one very big disadvantage and that is the fact that it is very slow. Because of this, we could not use Grinder to simulate sequencing errors, instead we wrote our own program which uses the error free reads to simulate sequencing data with an error rate of 0.1% per base. We simulated only one type of errors. substitutions because these are the most common ones in *Illumina* sequencing datasets.

#transcript 31-mers	errors	#read 31-mers		
		#correct	#wrong	#missing
49,611,691	0%	49,546,279	0	65,412
	0.1%	49,540,693	108,528,982	70,998

Table 4.1: This table shows the number of 31mers which appear in the reads (compared with the transcripts), how many erroneous 31-mers are introduced and how many 31-mers are missing.

Information about the *31-mers* present in the two sequencing multi-samples (error free and with 0.1% error) is summarized in Table 4.1. What stands out immediately is that even with perfect (error free) data, we still don't cover the entire transcriptome because there are missing *31-mers*. This is caused by transcripts which have ultra-low abundances across all tissues. Another interesting observation is that after adding errors, more than two thirds of the *31-mers* are incorrect (i.e. they don't appear in the transcriptome). If a sequencing error is located between positions 20 and 30 of a read then all the *30-mers* (vertices) extracted from that read are incorrect. Because of the massive amount of erroneous *31-mers* produced by sequencing errors, good error-correction algorithms are essential to obtaining good results.

To see how to remove the erroneous 31-mers, we try different combinations of the two error correction algorithms and apply them on a sample by sample basis (and perform the union of the files later). We also try different parameters for the error removal to see which performs the best.

We encode the operations performed on the data in the following way: (1) *union* means that all current samples are merged together; (2) *tips l* means that we apply the tip removal algorithm for a maximum tip length of *l*; (3) *bubbles t* we apply the bubble removal algorithm on the current dataset with a coverage threshold of *t*; (4) *partitioning x* is the graph partitioning algorithm.

For example: sample-by-sample + tips 21 + union + tips60 + partitioning 0.97, means that: we apply the tip removal on a sample-by-sample basis with, then we perform the union of the samples, we apply tip removal again, this time we allow a maximum tip length of 60 and finally we partition the graph. The results for several such processing flows are summarized in Table 4.2.

In Table 4.2 for tip removal we chose 21 and 60 as maximal tip lengths because: (1) the sequencing reads we simulated have a length of 50, therefore 21 is the number of 30-mers (vertices) produced by a read; (2) in [13] the authors of *Velvet* recommend 2*k* as the optimal value for the maximum length of a tip. From our data we can see that when we perform tip removal on a sample-by-sample basis using a value of 60, this gives us worse results than using 21, whereas for the union of the samples the situation is exactly the opposite. Just from the data in this table, the sequence of operations on the 14th row seems to achieve the best balance between missing correct 31-mers and the remaining erroneous 31-mers.

However, note that just by the data in this table we can not decide which

algorithm	#31mers	#correct	#wrong	#missing	#wrong + #missing
union	158,069,675	49,540,693	108,528,982	5,586	108,534,568
union + tips 21	71,264,038	49,500,472	21,763,566	45,807	21,809,373
union + tips 60	61,345,859	49,222,106	12,123,753	324,173	12,447,926
union + tips 21 + bubbles 0.97	61,483,094	49,435,376	12,047,718	110,903	12,158,621
union + tips 60 + bubbles 0.97	51,564,946	49,157,010	2,407,936	389,269	2,797,205
sample-by-sample + tips 21 + union	62,850,166	49,141,347	13,708,819	404,932	14,113,751
sample-by-sample + tips 60 + union	49,322,032	42,750,085	6,571,947	6,796,194	13,368,141
sample-by-sample + tips 21 + union + tips 21	60,125,921	49,107,772	11,018,149	438,507	11,456,656
sample-by-sample + tips 21 + union + tips 60	53,901,005	48,552,635	5,348,370	993,644	6,342,014
sample-by-sample + tips 60 + union + tips 60	47,556,584	42,511,620	5,044,964	7,034,659	12,079,623
sample-by-sample + tips 21 + bubbles 0.97 + union	59,950,907	49,140,356	10,810,551	405,923	11,216,474
sample-by-sample + tips 60 + bubbles 0.97 + union	45,429,798	42,745,812	2,683,986	6,800,467	9,484,453
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 21 + bubbles 0.97	56,523,523	49,057,208	7,466,315	489,071	7,955,386
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 60 + bubbles 0.97	49,533,347	48,502,571	1,030,776	1,043,708	2,074,484
sample-by-sample + tips 60 + bubbles 0.97 + union + tips 60 + bubbles 0.97	43,485,997	42,471,527	1,014,470	7,074,752	8,089,222

Table 4.2: Results for different combinations of the error removal algorithms run on the samples with 0.1% error rate.

dataset	intra	inter
31-mers partitioning - transcripts	49,441,494	170,197
31-mers partitioning - reads	49,376,188	170,091

Table 4.3: The correct partitioning of 31-mers into *intra-component edges* (if they are part of a component (*pseudo-exon*)) or *inter-component edges* (if they connect two separate components). This numbers are computed for the 31-mers present in the transcripts and for the subset of these which is present in the reads.

sequence is best because this table tells us nothing about the partitioning of the graph.

The partitioning step tries to remove edges from the *de Bruijn* graph such that the remaining connected components are paths (corresponding to *pseudo-exons*). For the data which we simulated, we know the correct partitioning of the edges (see Table 4.3, therefore one way of scoring the output produced by our algorithm is to compute the error statistics based on the classification of the edges: if we keep an edge - intra component edge, or if we discard it - inter component edge. Therefore, true positives are edges which are correctly labelled as intra-component, false positives are inter-component edges which we label as intra, false negatives are intra component edges which we label as inter and true negatives are edges which we correctly label as inter component edges.

After partitioning we also filter out small clusters, this helps us further reduce the number of erroneous 31-mers. We filter out graph components which have strictlyfewer vertices than some threshold. Those thresholds are: 0 (no filtering), 21 and 60. We also compare two different partitioning methods:

(1) if we encounter two or more outgoing(incoming) edges we remove all of them (partitioning with a coverage threshold of 1), and (2) partitioning using a coverage threshold of 0.97 (meaning that we keep one edge if it is more abundant than the others)j. This comparison can be viewed in Table 4.4.

From this data it is easy to see that removing all edges outgoing (incoming) from the graph is not as good as using a slightly tolerant coverage threshold such as 0.97. Also, it is clear that filtering components smaller 60 is better than the alternatives. Therefore, from now on we will only report results obtained by partitioning using a coverage threshold of 0.97 and filtering using 60.

As can be seen from the data in Table 4.4, adding the partitioning and filtering stages helps a lot with removing erroneous 31-mers while still keeping most of the correct edges.

In tables Table 4.5, Table 4.6 and Table 4.7, we look at the performance of the partition and filtering algorithms (coverage threshold 0.97 and filtering by 60) for different combinations of the error removal algorithms, and also compare the results of the same sequence of operation on error free data with the results on data with 0.1% errors. Of note is the sequence "sample-by-sample + tips 21 + bubbles 0.97 + union + tips 21 + bubbles 0.97 + partitioning 0.97" from table Table 4.7, which gets the number of wrong 31-mers down to $29,583$ (out of $108,528,982$) while still maintaining most of the correct 31-mers $47,708,616$ (out of $49,540,693$).

Finally, in Table 4.8 we show some statistics about the components produced by

algorithm	minimum component size	tp	fp	fn	tn	erroneous 31mers
union + tips 60 + bubbles 0.97 + partitioning 0.97	0	48,973,725	14,058	21,734	147,493	2,128,428
	21	48,751,323	7,964	8,502	61,435	1,500,437
	60	48,076,410	5,811	6,454	42,364	40,258
union + tips 60 + bubbles 0.97 + partitioning 1	0	48,891,904	9,547	103,555	152,004	2,134,456
	21	48,552,619	4,864	49,164	64,349	1,505,933
	60	47,822,310	3,309	38,010	45,847	42,044
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 0.97	0	48,328,321	19,746	15,275	139,229	883,369
	21	48,133,212	12,165	6,300	58,488	529,822
	60	47,516,224	9,281	5,080	40,240	46,570
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 1	0	48,284,234	12,480	59,362	146,495	886,211
	21	48,006,697	7,257	23,515	62,258	532,031
	60	47,362,115	5,257	18,231	43,943	46,586

Table 4.4: Size filtering of graph components

the graph partitioning algorithm. From this data we can see that after filtering, most of the ”surviving” components contain only correct 30-mers (vertices). Also, we looked at the components obtained from several of the processing pipelines and the overwhelming majority are (as we expected) paths.

algorithm	errors	tp	fp	fn	tn	erroneous 31mers
union + partitioning 1		48,429,081	1,930	7	50,860	0
union + tips 21 + partitioning 1		48,431,707	2,142	7	50,841	0
union + tips 60 + partitioning 1	0	48,432,565	2,462	7	50,864	0
union + tips 21 + bubbles 0.97 + partitioning 1		48,344,259	2,967	6	49,514	0
union + tips 60 + bubbles 0.97 + partitioning 1		48,345,034	3,283	6	49,528	0
union + partitioning 0.97		34,266,048	4,861	104,185	44,381	1,718,338
union + tips 21 + partitioning 0.97		48,317,028	4,843	7,179	45,453	31,868
union + tips 60 + partitioning 0.97	0.1	48,112,528	5,148	7,051	45,505	50,385
union + tips 21 + bubbles 0.97 + partitioning 0.97		48,281,223	5,482	6,559	42,229	22,209
union + tips 60 + bubbles 0.97 + partitioning 0.97		48,076,410	5,811	6,454	42,364	40,258

Table 4.5: Edge correctness computed by applying the error removal algorithms on the union of the samples.

algorithm	errors	tp	fp	fn	tn	erroneous 31mers
sample-by-sample + tips 21 + union + partitioning 1		47,894,088	2,938	6	48,841	0
sample-by-sample + tips 60 + union + partitioning 1		42,756,499	8,445	6	39,172	0
sample-by-sample + tips 21 + union + tips 21 + partitioning 1	0	47,898,433	3,252	6	48,878	0
sample-by-sample + tips 21 + union + tips 60 + partitioning 1		47,899,481	4,210	6	48,778	0
sample-by-sample + tips 60 + union + tips 60 + partitioning 1		42,787,351	10,828	6	39,157	0
sample-by-sample + tips 21 + union + partitioning 0.97		45,196,271	7,237	92,187	36,778	25,485
sample-by-sample + tips 60 + union + partitioning 0.97		40,601,194	12,580	54,597	31,248	69,688
sample-by-sample + tips 21 + union + tips 21 + partitioning 0.97	0.1	47,733,317	7,884	4,134	42,391	33,894
sample-by-sample + tips 21 + union + tips 60 + partitioning 0.97		47,539,213	8,758	5,471	42,389	50,970
sample-by-sample + tips 60 + union + tips 60 + partitioning 0.97		41,629,503	15,130	6,310	33,096	98,223

Table 4.6: Edge correctness computed by applying the tip removal algorithm on a sample-by-sample basis, then performing the union and applying the error removal once more for some of the sequences.

algorithm	errors	tp	fp	fn	tn	erroneous 31mers
sample-by-sample + tips 21 + bubbles 0.97 + union + partitioning 1	0%	47,894,271	2,992	6	48,613	0
sample-by-sample + tips 60 + bubbles 0.97 + union + partitioning 1		42,757,019	8,544	6	38,928	0
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 21 + bubbles 0.97 + partitioning 1		47,887,279	4,008	6	48,063	0
sample-by-sample + tips 21 + bubbles 0.97 + union + tips60 + bubbles 0.97 + partitioning 1		47,888,604	5,007	6	47,949	0
sample-by-sample + tips 60 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 1		42,786,471	11,738	6	38,269	0
sample-by-sample + tips 21 + bubbles 0.97 + union + partitioning 0.97	0.1%	45,197,325	7,312	92,043	36,510	21,859
sample-by-sample + tips 60 + bubbles 0.97 + union + partitioning 0.97		40,601,189	12,692	54,461	30,969	65,825
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 21 + bubbles 0.97 + partitioning 0.97		47,708,616	8,395	3,729	40,233	29,583
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 0.97		47,516,224	9,281	5,080	40,240	46,570
sample-by-sample + tips 60 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 0.97		41,617,583	15,570	5,955	31,084	94,043

Table 4.7: Edge correctness computed by applying the error removal algorithms on a sample-by-sample basis, then performing the union and applying the error removal once more for some of the sequences.

algorithm	errors	#components	#components which do not contain incorrect 31-mers	SENSITIVITY	PPV	F-score
union + partitioning 1	0	43,285	43,285	97.78%	100%	98.87%
union + partitioning 0.97		229,649	226,207	67.89%	98.50%	80.38%
union + tips 21 + partitioning 0.97		45,955	44,060	93.75%	95.88%	94.80%
union + tips 60 + partitioning 0.97		43,910	40,281	87.61%	91.74%	89.62%
union + tips 21 + bubbles 0.97 + partitioning 0.97		45,548	43,809	93.66%	96.18%	94.90%
union + tips 60 + bubbles 0.97 + partitioning 0.97		43,497	40,041	87.52%	92.05%	89.73%
sample-by-sample + tips 21 + union + partitioning 0.97	0.001	182,799	181,228	90.87%	99.14%	94.82%
sample-by-sample + tips 60 + union + partitioning 0.97		172,316	166,196	80.20%	96.45%	87.58%
sample-by-sample + tips 21 + union + tips 21 + partitioning 0.97		60,919	58,205	92.97%	95.54%	94.24%
sample-by-sample + tips 21 + union + tips 60 + partitioning 0.97		59,866	55,447	89.49%	92.62%	91.03%
sample-by-sample + tips 60 + union + tips 60 + partitioning 0.97		108,192	99,244	77.55%	91.73%	84.05%
sample-by-sample + tips 21 + bubbles 0.97 + union + partitioning 0.97		182,748	181,265	90.90%	99.19%	94.86%
sample-by-sample + tips 60 + bubbles 0.97 + union + partitioning 0.97		172,246	166,230	80.23%	96.51%	87.62%
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 21 + bubbles 0.97 + partitioning 0.97		60,677	58,034	92.90%	95.64%	94.25%
sample-by-sample + tips 21 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 0.97		59,613	55,256	89.44%	92.69%	91.04%
sample-by-sample + tips 60 + bubbles 0.97 + union + tips 60 + bubbles 0.97 + partitioning 0.97		107,987	99,125	77.57%	91.79%	84.08%

Table 4.8: Correctness of the components obtained after partitioning and filtering. The sensitivity is the percentage of the 30-mers found in the transcriptome which are present in the components which are counted in column 4. The PPV is the percentage of clusters which do not contain any wrong 30-mers.

Chapter 5
Conclusions and Ongoing Work

In ongoing work we are planning to assemble the components obtained from the processing pipelines into sequences and align them unto the (meta)transcriptome. We expect that using this information and the reads, we will be able to perform the reconstruction of the (meta)transcriptome.

We validate our algorithm using simulated RNA-Seq data (both error-free and with errors). We created several dierent workflows for processing the data and managed to obtain promising results. One of the biggest challenges we faced was dealing with errors. Since our algorithms are based on *k-mers* a single error in a read results in multiple erroneous *k-mers*. For an error rate of 0.1% over two thirds of the k-mers are incorrect, however, we managed to achieve a good balance between how many correct *k-mers* we keep (47,708,616 out of 49,540,693) and how many erroneous ones we keep (29,583 out of 108,528,982).

The microbiome is an essential part of life on Earth. Studies of environmental samples which have focused on single organism fail to capture the true picture of the diversity of microbial life. In order to better understand these microbial communities and the interactions between their members and their habitat we need to develop more efficient and more accurate methods for analyzing metatranscriptomic data. We hope that the work presented in this thesis shows the potential of genome-independent multi-sample approaches. In particular, we hope that these methods will become a standard part of the metatranscriptomics toolkit and help to further improve our understanding of the microbiome.

Bibliography

[1] Mads Albertsen, Philip Hugenholtz, Adam Skarshewski, Kåre L Nielsen, Gene W Tyson, and Per H Nielsen. Genome sequences of rare, uncultured bacteria obtained by differential coverage binning of multiple metagenomes. *Nature biotechnology*, 31(6):533–538, 2013.

[2] Florent E Angly, Dana Willner, Forest Rohwer, Philip Hugenholtz, and Gene W Tyson. Grinder: a versatile amplicon and shotgun sequence simulator. *Nucleic acids research*, 40(12):e94–e94, 2012.

[3] Yael Baran and Eran Halperin. Joint analysis of multiple metagenomic samples. *PLoS computational biology*, 8(2):e1002373, 2012.

[4] Eric S Lander and Michael S Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2(3):231–239, 1988.

[5] Hai-Son Le, Marcel H Schulz, Brenna M McCauley, Veronica F Hinman, and Ziv Bar-Joseph. Probabilistic error correction for rna sequencing. *Nucleic acids research*, 41(10):e109–e109, 2013.

[6] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

[7] Jason Pell, Arend Hintze, Rosangela Canino-Koning, Adina Howe, James M Tiedje, and C Titus Brown. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*, 109(33):13272–13277, 2012.

[8] Andrew I Su, Tim Wiltshire, Serge Batalov, Hilmar Lapp, Keith A Ching, David Block, Jie Zhang, Richard Soden, Mimi Hayakawa, Gabriel Kreiman, et al. A gene atlas of the mouse and human protein-encoding transcriptomes. *Proceedings of the National Academy of Sciences of the United States of America*, 101(16):6062–6067, 2004.

[9] Peter J Turnbaugh, Ruth E Ley, Micah Hamady, Claire M Fraser-Liggett, Rob Knight, and Jeffrey I Gordon. The human microbiome project. *Nature*, 449(7164):804–810, 2007.

[10] Yi Wang, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. Metacluster 5.0: a two-round binning approach for metagenomic data for low-abundance species in a noisy sample. *Bioinformatics*, 28(18):i356–i362, 2012.

[11] John C Wooley, Adam Godzik, and Iddo Friedberg. A primer on metagenomics. *PLoS computational biology*, 6(2):e1000667, 2010.

[12] Yu-Wei Wu and Yuzhen Ye. A novel abundance-based algorithm for binning metagenomic sequences using l-tuples. *Journal of Computational Biology*, 18(3):523–534, 2011.

[13] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.