UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 734

De novo metagenomic assembly using Bayesian model-based clustering

Mirta Dvorničić

Zagreb, June 2014

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
MASTER THESIS COMMITTEE

Zagreb, March 10th, 2014

MASTER THESIS ASSIGNMENT No. 734

Student:	**Mirta Dvorničić (0036454014)**
Study:	Computing
Profile:	Computer Science

Title: **De novo metagenomic assembly using Bayesian model-based clustering**

Description:

The purpose of this thesis is implementation of a Bayesian model-based approach for clustering to aid in metagenomic assembly. In preliminary studies it has been already shown that this approach can significantly improve results achieved by the state-of-the-art metagenomic assembly programs. This approach should be further refined to handle experimental biases in sequencing data. Explore other potential refinements such as using additional hints for clustering refinement (i.e. gene and k-mer composition) and incorporation of long reads from third-generation sequencing protocols. The code should be extensively commented in order to make its debugging and maintenance easier. Provide detailed instructions for installation and usage.

Thesis submitting date: June 30th, 2014

Mentor:

Mile Šikić

Prof. Mile Šikić, PhD

Committee Chair:

[illegible]

Prof. Siniša Srbljić, PhD

Committee Secretary:

[illegible]

Prof. Tomislav Hrkać, PhD

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 10. ožujka 2014.

DIPLOMSKI ZADATAK br. 734

Pristupnik:	**Mirta Dvorničić (0036454014)**
Studij:	Računarstvo
Profil:	Računarska znanost

Zadatak: **De novo sastavljanje metagenomskih podataka korištenjem grupiranja podataka temeljenih na Bayesovom modelu**

Opis zadatka:

Cilj ovoga rada je implementacija grupiranja podataka temeljenog na Bayesovom modelu kao pomoć pri sastavljanju genoma iz metagenomskih podataka. U prethodnim studijama pokazano je da ovaj pristup može značajno poboljšati rezultate postignute standardnim programima za sastavljanje genoma iz metagenomskih podataka. Ovaj pristup se može dodatno profiniti uzimanjem u obzir eksperimentalne pogreške u sekvenciranim podacima. Istražiti druga moguća profinjenja koristeći informacije kao što su kompozicija gena i frekvencija pojavljivanja pojedinih k-torki te uključujući dugačka očitanja dobivena uređajima treće generacije. Sav kod treba biti iscrpno komentiran u cilju olakšanja ispravljanja pogrešaka i održavanja. Upute za instalaciju i izvođenje moraju se nalaziti zajedno s kodom.

Zadatak uručen pristupniku:	14. ožujka 2014.
Rok za predaju rada:	30. lipnja 2014.

Mentor:
[illegible]
Doc. dr.sc. Mile Šikić

Djelovođa:
[illegible]
Doc. dr.sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
[illegible]
Prof. dr.sc. Siniša Srbljić

CONTENTS

List of Figures

List of Tables

1. Introduction

Microbes were the first forms of life to develop on Earth. Microbial communities thrive in almost all environments, playing an important role in many processes relevant to environment, agriculture, industry, health and disease [1]. Consequently, a lot of effort has been put into improving the understanding of microbes, their interactions and interdependencies as well as relationships with their hosts and habitats.

For many years, it has been recognized that vast majority of microbial life cannot be cultivated in isolation using known methods [2]. This *great plate count anomaly* propelled the development of culture-independent methods for studying microbial communities. Metagenomics is broadly defined as culture-independent analysis of microbial communities from genetic material extracted directly from the environment.

Advances in DNA sequencing technologies resulted in cheaper, faster and high-throughput second generation sequencing which opened the way for many metagenomic studies. Several studies have shown that these technologies can be utilized to characterize complex microbial communities [3] and even reconstruct near-complete genomes [4].

The vast amount of data being generated by metagenomic sequencing projects requires the development of efficient, robust and automated metagenomic assembly tools, especially because the downstream analysis is largely impacted by correctness and contiguity of the assembly [1]. This emphasizes the importance of scaffolding, ordering and orienting contiguous sets of overlapping reads called contigs, into larger, not necessarily contiguous sequences called scaffolds. Throughout the years, many scaffolding modules and stand-alone scaffolding tools were developed for single genome assembly [5]. However, fewer were designed specifically for metagenomic data and additional challenges it introduces [6].

In this thesis, a Bayesian model-based hierarchical clustering approach to aid in de novo metagenomic assembly is presented. This approach, called Sigma, utilizes assembly information in order to decompose metagenome scaffolding problem into independent single genome scaffolding problems. Sigma is combined with optimal single genome scaffolder Opera [7] and the performance of this pipeline, called OperaMS, is evaluated against state of the art single genome (Velvet [8] and SOAPdenovo [9]) and metagenomic (Bambus2 [10] and MetaVelvet [11]) assembly tools.

The rest of this thesis is organized as follows. A brief introduction to metagenomics, genome sequencing and genome assembly is given in Chapter 2. Chapter 3 describes the theoretical background and gives an overview of the method. Key implementation details as well as installation and usage instructions are provided in Chapter 4. The evaluation results are presented and discussed in Chapter 5 and summarized in Chapter 6 together with future work possibilities.

2. Background

2.1. Metagenomics

Bacteria, archaea, viruses, microscopic fungi and microscopic eukaryotes comprise microbial communities. Metagenomics studies those communities from one or more DNA samples obtained directly from their environments. Metagenome is a collection of all the genomes present in the microbial community.

There are two complimentary approaches to metagenomics that together provide insight into largely unknown microbial world: function-based and sequence-based metagenomics [12]. In function-based approaches environmental DNA is cloned into a host bacteria and then screened for specific functions in order to identify new genes and functions they encode. On the other hand, in sequence-based approaches environmental DNA is randomly sequenced and the sequences are then analyzed in order to explore phylogenetic complexity and distribution of functions within the community.

A typical project in shotgun metagenomics includes metadata collection, DNA extraction, library construction, sequencing, read preprocessing, assembly and analysis [13]. Analysis can be performed during several stages of the project and the outcome is highly impacted by complexity of the microbial community being studied, i.e., species richness and species evenness. Species richness refers to the number of different species present in the community, while species evenness refers to the relative species abundance in the community. A species-rich community with relatively even species abundance is considered more complex than a species-poor community with relatively uneven species abundance.

The ultimate goal of metagenomics is to provide a descriptive and eventually predictive metabolic and taxonomic model of an ecosystem [14].

2.2. Genome sequencing

In metagenomic shotgun sequencing, the total DNA extracted from the environmental sample is sheared into many small fragments which are then sequenced to obtain reads. The choice of suitable sequencing platform, library types and amount of sequencing is typically driven by complexity of the microbial community being analyzed as well as research goals and budget of the study [2][13].

First metagenomic studies were mainly conducted by sequencing DNA cloned into a bacterial host using Sanger sequencing. Besides bypassing the requirement for cloning, second generation sequencing provided faster, cheaper and high-throughput sequencing, enabling analysis of complex microbial communities. Initially, Roche 454 sequencing platform was preferred because of longer read length, but substantial improvements in read length and throughput increased the popularity of Illumina sequencing platform for metagenomic studies [6]. The error rate of Illumina reads is low, between 1% and 2%, but the errors are not uniformly distributed. There are also substantial sequencing biases caused by DNA amplification. Third generation sequencing, such as Pacific Biosciences sequencing platform, provides significant improvements with an exponential increase in read length and throughput in the past few years, uniform error distribution and very low sequencing bias. However, since the accuracy of PacBio reads is only at 85%, some claim that it is not usable for metagenomic sequencing in its current form [15].

Some sequencing technologies have the ability to generate paired-end and mate-pair libraries, i.e., pairs of reads with known orientation and approximate distance. Shotgun libraries are typically prepared using a few different distances between those paired reads called insert sizes. Short-insert paired-end reads are generated by sequencing both forward and reverse strands of a DNA fragment. Long-insert mate-pair reads represent the ends of a DNA fragment generated by sequencing a circularized DNA fragment. Mate-pair reads with larger insert sizes have greater likelihood of spanning gaps and repeats but they are technically more demanding to prepare [13]. Each pair of reads puts a constraint on orientation and distance of two contigs, providing useful information for scaffolding.

Sequencing amount is usually expressed in terms of depth of coverage or simply coverage, i.e., the average number of times a particular base in the genome was sequenced. If the goal is to determine gene content and abundance within the community, lower sequencing coverage will suffice. On the other hand, if the goal is to obtain complete or near-complete genomes, substantially higher sequencing coverage is needed, especially to reconstruct genomes of community members with low abundance. Biases in DNA sample preparation, sequencing, and genomic alignment and assembly can result in regions of the genome with unusually low or high coverage [16]. For example, low coverage is usually observed in GC-rich or GC-poor regions of the genome due to amplification bias of second generation sequencing technologies. Mappability issues caused by sequencing errors and repetitive regions are another source of coverage bias. These issues are especially manifested when using short reads from second generation sequencing technologies.

There are two important terms related to coverage. Read count is the total number of reads starting in a particular sequence. Arrival rate is the average number of reads starting in a particular position in the sequence, i.e., read count divided by length of the sequence.

Before the assembly, reads are usually filtered to remove contaminant reads such as reads from the host DNA and trimmed to remove low quality bases at the ends of reads.

2.3. Genome assembly

There are two main approaches to genome assembly. Comparative approaches use a genome of a closely related organism called the reference genome to guide the assembly of the sequenced genome. When the sequenced genome is not similar to any previously assembled genome, de novo approaches are used. Although polymorphisms between the sequenced genome and the reference genome present a challenge for comparative assembly, it remains simpler and computationally less expensive than de novo assembly which falls within a class of NP-hard problems [17].

Some of the first de novo approaches were greedy algorithms. Greedy algorithms simply iteratively merge two sequences with the best overlap if merging them does not contradict the current assembly. Some of them incorporate additional heuristics to guide the merging. These approaches often lead to misassemblies and result in suboptimal solutions. Modern de novo genome assemblers can broadly be divided into two categories: overlap-layout-consensus assemblers and de Brujin graph assemblers.

Overlap-layout-consensus assemblers use an overlap graph whose vertices represent reads and edges represent detected overlaps between the suffix of the first read and the prefix of the second read. The goal is to find a Hamiltonian path in the graph, i.e., a path which traverses through each vertex exactly once.

De Brujin graph assemblers decompose the reads into words of length k called k-mers, where k is usually a user-specified parameter. They use a de Brujin graph whose vertices represent k-mers and edges represent overlaps of length $k - 1$ between the suffix of the first k-mer and the prefix of the second k-mer. The goal is to find an Eulerian path in the graph, i.e., a path which traverses through each edge exactly once.

Finding a Hamiltonian path is known to be NP-hard, while polynomial time algorithms exist for finding an Eulerian path. The size of the overlap graph depends on the number of reads, while the size of the de Brujin graph depends solely on k. Therefore, overlap-layout-consensus assemblers are more suitable for long reads which typically require lower coverage, i.e., a smaller number of reads. Additionally, sequencing errors can introduce new k-mers in the de Brujin graph making de Brujin graph assemblers more suitable for short reads which typically have a lower error rate.

Short read de Brujin graph assemblers Velvet [8] and SOAPdenovo [9] were applied to metagenomic datasets in recent studies [3] [4] with satisfactory results. Provided that the strain heterogeneity is limited, they can produce good contig assemblies. However, single genome assemblers are unaware of the nature of metagenomic datasets and some assumptions they make can significantly lower the quality of the assembly. Since microbial communities often contain hundreds or thousands of species and sequencing coverage is limited, many low abundant species will be represented with only a few reads, if at all. Those reads are likely to be perceived as erroneous by traditional assembly algorithms preventing any reconstruction of their genomes. Metagenomic assembly is further complicated by repeats, including repetitive regions within single genomes and regions conserved in multiple related genomes. Traditional assembly algorithms usually label regions with high coverage as repeats, but in metagenomic assembly varying relative species abundance makes the repeat detection harder, since high abundant species can be mistaken for repeats.

Bambus2 [10], proposed in 2011, is a scaffolding module optimized for metagenomic datasets. It aims to distinguish repeats from genomic variation in the assembly graph. Repeats are detected based on the measure of vertex centrality, i.e., the number of times a vertex appears on the shortest path between any two vertices in the graph, and local coverage statistics in connected components of the graph. After filtering out the repeats, contigs are oriented and positioned and the assembly graph is simplified. Finally, subgraphs representing different species are identified by iteratively searching for variation motifs in the graph.

MetaVelvet [11], proposed in 2012, is an extension of single genome assembler Velvet for metagenomic datasets. It aims to decompose the de Bruijn graph into subgraphs that represent different genomes and then build scaffolds for each individual subgraph separately. To achieve that, the empirical histogram of k-mer frequencies is approximated by a mixture of Poisson distributions and multiple peaks in the mixture are detected. Each graph vertex is then classified into one peak or labeled as chimeric. Subgraphs representing different species are distinguished by identifying and disconnecting the chimeric vertices.

Recently, a metagenomic assembly approach utilizing coverage from two samples was presented [18]. In that approach, metagenomic DNA was extracted using two different methods, producing datasets with different relative species abundance. Each dataset was scaffolded independently, and the reads from both datasets were mapped back to the scaffolds assembled from the larger dataset. Initial binning of the scaffolds was done by plotting two coverage estimates against each other and the bins were further refined using principal component analysis of tetranucleotide frequencies. Paired-end reads were used to associate repeats with the appropriate bins. Finally, all reads associated with a specific genome bin were separately re-assembled using parameters optimized for each genome, and the scaffolds were manually corrected. Unfortunately, this approach requires a lot of manual intervention and it cannot easily be modified to incorporate information from more than two samples.

3. Materials and methods

A schematic overview of the pipeline presented in this thesis can be found in Figure 3.1. Broadly speaking, the pipeline can be separated into three main steps:

– assembly graph construction

– hierarchical clustering and model selection

– scaffolding

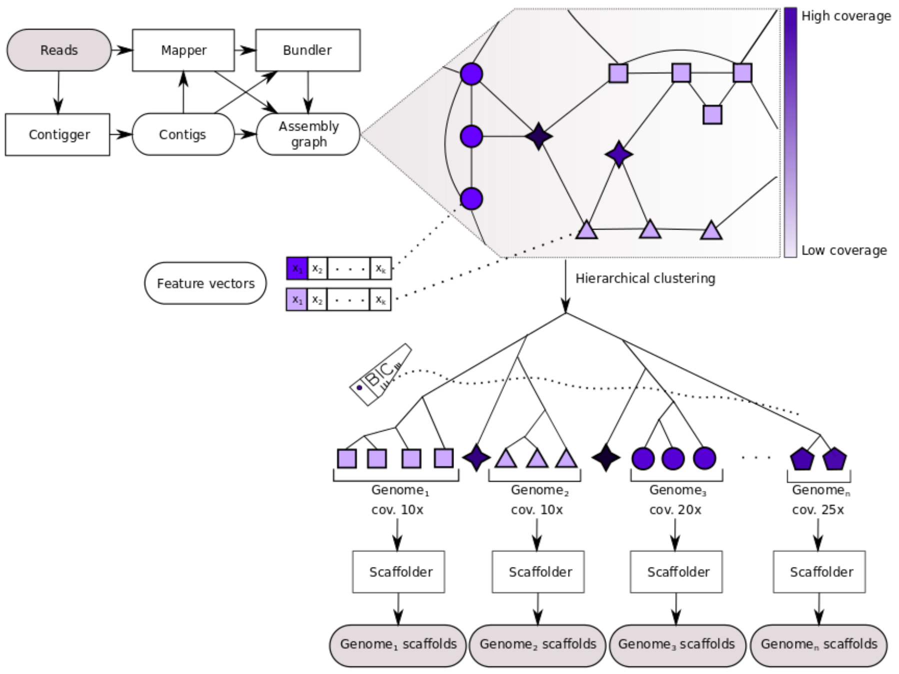

Figure 3.1: Overview of the pipeline.

In the first step, the assembly graph is constructed using contigs and edges generated by contigging, mapping and bundling of one or more paired-end/mate-pair read libraries. Additionally, mapping is used to estimate contig read count and arrival rate.

During the second step, Sigma is used to find the optimal partitioning of the assembly graph based on the Bayesian information criterion (BIC). The partitioning is done on a hierarchical clustering tree built along the assembly graph edges based on contig feature vector similarity. Currently, feature vectors include contig arrival rate from one or more samples, but they are naturally extendible to other sources of information such as k-mer frequencies, GC content and gene content. In the final step, optimal single genome scaffolder Opera is used to produce scaffolds for the partitioned assembly graph.

3.1. Assembly graph construction

Contigs are contiguous sequences which represent a consensus sequence of a set of overlapping reads. They can be assembled by an arbitrary contigger suitable for the used sequencing technology, e.g., Velvet [8] or SOAPdenovo [9] for Illumina reads. Both of these assemblers generate contigs from a de Brujin graph described in Section 2.3. Usually, single-end and short-insert paired-end reads are used for contigging.

After contigging, contig read count, arrival rate and coverage can be estimated by mapping the reads used for contigging back to the assembled contigs. If the sequencing technology provides paired-end and/or mate-pair reads, they can be mapped back to the assembled contigs to find connections between different contigs. Mapping can be done with an arbitrary mapper which supports the error model of the used sequencing technology, e.g., Bowtie [19] or BWA [20] for Illumina reads.

Edges are sets of paired-end/mate-pair reads suggesting the same orientation and similar distance between two contigs. They can then be generated from assembled contigs and mapping by a bundler such as Opera's [7] bundling routine. Let $M = \{m_1, ..., m_k\}$ be a set of paired reads suggesting the same orientation between the same pair of contigs. Each m_i gives an independent estimate (μ_i, σ_i) of the mean and standard deviation of the distance between the two contigs. Standard deviation is usually around 10% of the library insert size. Paired reads in M can be bundled with the following routine [21]:

1. Find the median of distances μ_{med} suggested by $m_{med} \in M$.

2. Select the subset $N \subseteq M$ such that $\mu_i \in [\mu_{med} - 3\sigma_{med}, \mu_{med} + 3\sigma_{med}]$ for all $m_i \in N$.

3. Define $p = \sum_{m_i \in N} \frac{\mu_i}{\sigma_i^2}$ and $q = \sum_{m_i \in N} \frac{1}{\sigma_i^2}$.

4. Add $(\mu, \sigma) = (\frac{p}{q}, \frac{1}{\sqrt{q}})$ to the set of edges with the suggested orientation between the two contigs.

5. Set $M = M \setminus N$.

6. Repeat steps 1.-5. until $M = \emptyset$.

Contigs and their paired-end and/or mate-pair connections can intuitively be represented with an assembly graph $G = (V, E)$ where each vertex represents a contig and each edge represents a bundle of paired-end and/or mate-pair reads connecting two contigs. Each vertex can have various parameters associated with the contig it represents such as length, read count, arrival rate, coverage, k-mer frequencies and GC content. Each edge can be described with orientation, number of supporting reads, and mean and standard deviation of the distance between the incident contigs. The assembly graph is bidirectional, meaning that each edge can be traversed in two ways.

Repetitive regions, sequence similarities between different genomes and mapping errors can introduce additional erroneous edges in the assembly graph and join different genomes together. The goal is to filter out those edges so that each connected component in the graph contains contigs originating from a single genome. By disconnecting the erroneous edges, metagenome scaffolding problem can be reduced to multiple independent single genome scaffolding problems and graph components can be scaffolded independently using existing tools.

3.2. Hierarchical clustering

Clustering or cluster analysis is a method which aims to partition a set of objects in different groups called clusters in a way that objects belonging to the same group are more similar to each other than to objects belonging to other groups according to some criterion. Clustering methods can be broadly divided into hierarchical clustering and partitional clustering. Hierarchical clustering groups objects with a sequence of partitions, while partitional clustering directly divides objects into some prespecified number of clusters without the hierarchical structure [22].

Hierarchical clustering is an unsupervised clustering method, i.e., it tries to model the given data without knowing anything about its true distribution. Furthermore, hierarchical clustering does not require the number of clusters to be specified in advance. These two properties go well with the fact that the number of species and relative species abundance are typically unknown in metagenomic assembly.

There are two main hierarchical clustering strategies. Agglomerative (bottom-up) clustering begins with placing each object into a singleton cluster, i.e., a cluster containing one element, and proceeds with iteratively merging the clusters based on a similarity measure until all objects are merged into one cluster containing all objects. On the other hand, divisive (top-down) clustering begins with placing all objects into one cluster and proceeds with iteratively splitting the clusters based on a splitting criterion until all objects are splitted into singleton clusters. Unlike a splitting criterion for a cluster of contigs, a similarity measure between two contigs can be intuitively defined, making agglomerative clustering a more suitable strategy for metagenomic assembly.

The only input requirement for agglomerative hierarchical clustering is a $n \times n$ matrix of pairwise distances between all objects, where n is the number of objects. Each distance gives a similarity measure between two objects - the smaller the distance the higher the similarity. The distance between two clusters of objects, X and Y, can then be determined based on some commonly used linkage criteria:

- *single-linkage* $d(X, Y) = \min_{x \in X, y \in Y} d(x, y)$
- *complete-linkage* $d(X, Y) = \max_{x \in X, y \in Y} d(x, y)$
- *mean or average linkage* $d(X, Y) = \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} d(x, y)$

After placing each object into a separate cluster and setting the cluster distances to the distances of corresponding objects, clustering proceeds with the following steps:

1. Find clusters X and Y with the smallest distance $d(X, Y)$.

2. Merge clusters X and Y into a single cluster Z.

3. Remove rows and columns corresponding to clusters X and Y in the distance matrix.

4. Compute distances between cluster Z and the remaining clusters and insert the corresponding row and column in the distance matrix.

5. Repeat steps 1.-4. until there is only one cluster left.

A set of clusters in each iteration of hierarchical clustering corresponds to one possible partitioning of the data. The end result of hierarchical clustering is a hierarchical clustering tree where each node represents a cluster containing all objects contained in its child clusters.

The basic algorithm described above is not very efficient. The initial step, computing the distances between all objects, takes $O(n^2)$ time. After each iteration the number of clusters is decreased by 1, so the total number of iterations is $n-1$. Identifying the smallest distance takes time proportional to n^2, $(n-1)^2$, $(n-2)^2$, ... and recomputing the distances between the new cluster and the remaining clusters takes time proportional to $n-1$, $n-2$, $n-3$... Thus, the time complexity of the algorithm is $O(n^3)$.

Given the assembly graph described in Section 3.1, precisely:

- a set of contigs $V = (v_1, v_2, ..., v_n)$;
- a set of contig lengths $L = (l_1, l_2, ..., l_n)$, where l_i is the length of v_i;
- a set of contig read counts $R = (r_1, r_2, ..., r_n)$, where r_i is the read count of v_i;
- a set of edges $E = (e_1, e_2, ..., e_k)$, where $E \subseteq V \times V$,

the arrival rate of contig v_i is defined as $\lambda_i = \frac{r_i}{l_i}$ and the similarity measure $d(v_i, v_j)$ between two contigs can be defined with Equation 3.1. A surface plot of this function is shown with Figure 3.2. This function accounts for local arrival rate variation by normalizing the arrival rate difference.

$$d(v_i, v_j) = \begin{cases} \frac{|\lambda_i - \lambda_j|}{\max(\lambda_i, \lambda_j)}, & \text{if } (v_i, v_j) \in E \\ \infty, & \text{if } (v_i, v_j) \notin E \end{cases} \tag{3.1}$$

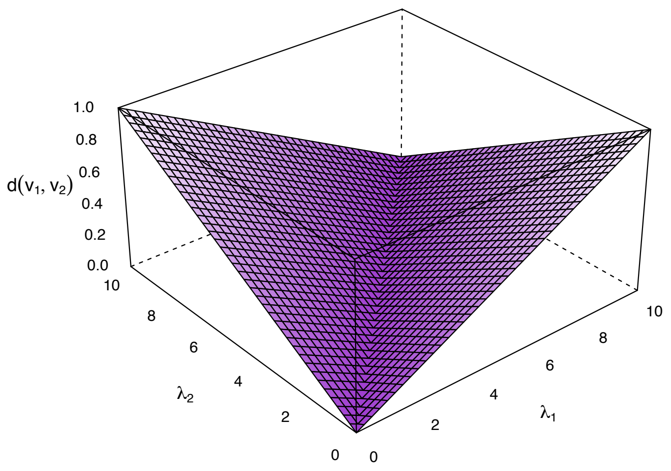

Figure 3.2: A surface plot of contig distance function defined with Equation 3.1.

Single-linkage hierarchical agglomerative clustering of the contigs based on their arrival rates can then be done with the following steps:

1. Select the first edge in the priority queue of edges from E sorted in order of increasing distance $d(v_i, v_j)$.
2. If the contigs incident to this edge do not belong to the same cluster, merge the clusters containing these contigs into a new cluster.
3. Replace the clusters which previously contained these contigs with the new cluster.
4. Repeat steps 1.-3. until all edges are processed.

It is important to notice that the method described above typically produces not one, but multiple hierarchical clustering trees. Prior beliefs on possible partitions of the assembly graph are elegantly represented in the generated clustering trees. Partitions that cluster contigs from different connected components are eliminated by relying on bundles of paired-end/mate-pair connections. Furthermore, the distance between two contigs in a clustering tree increases with distance in arrival rate space and fewer internal nodes in the tree cluster distant contigs together. Finally, by considering edges in the increasing order of the distance, potentially erroneous links are likely to be used closer to the roots.

3.3. Model selection

Model-based clustering assumes that the observed data was generated by a mixture of underlying probability distributions where each mixture component generated a different cluster. The goal is to reconstruct parameters of those probability distributions to find the best-fit model for the observed data.

3.3.1. Lander-Waterman model

According to the model proposed by Lander and Waterman in 1988. [23], the number of reads starting at each position in a genome follows a Poisson distribution. This holds under the assumption that the reads are distributed uniformly across the genome.

Specifically, if N reads were sampled from a genome of length G with the average number of reads starting at each position equal to $\lambda = N/G$, the number of reads starting at each position in the genome can be modeled as $p \sim \text{Poisson}(\lambda)$. Thus, the total number of reads in a non-repetitive contig of length l, i.e. contig read count, can be modeled as $r = \sum_{i=1}^{l} p_i \sim \text{Poisson}\,(l\lambda)$. For a repetitive contig constructed from reads sampled from repetitive regions within a single genome, the total number of reads will be over-represented proportionally to the number of copies of the repetitive region. Similarly, for a repetitive contig constructed from reads sampled from repetitive regions shared between multiple genomes, the total number of reads will follow a Poisson mixture model.

When the assumption that the reads were uniformly sampled from the genome does not hold, which is often the case due to sequencing bias and mappability issues caused by sequencing errors and repetitive regions, the number of reads starting at each position in a genome can instead be modeled with a negative binomial distribution, hence allowing for a larger variance. Negative binomial distribution is used commonly throughout biology as a model for overdispersed count data [24].

Probability mass functions of Poisson distribution and negative binomial distribution are shown with Equation 3.2 and Equation 3.3 respectively. Poission distribution expresses the probability of a given number k of events occurring in a specific interval if these events occur independently with a known average rate λ. Both mean μ and standard deviation σ of the Poisson distribution are equal to λ. Negative binomial distribution expresses the probability of k successes occurring before r failures in a sequence of independent trials with the success probability in each trial equal to p. For modeling the read count distribution it is more natural to parameterize the negative binomial distribution with mean μ and variance over mean ratio $VMR = \frac{\sigma}{\mu}$. Since the mean is equal to $\mu = \frac{pr}{1-p}$ and the variance is equal to $\sigma = \frac{pr}{(1-p)^2}$, we can easily obtain $p = 1 - \frac{1}{VMR}$ and $r = \frac{1-p}{p}\mu$.

$$f(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{3.2}$$

$$f(k; r, p) = \binom{k + r - 1}{k} (1-p)^r p^k \tag{3.3}$$

Given:

- a set of contigs $V = (v_1, v_2, ..., v_n)$;
- a set of contig lengths $L = (l_1, l_2, ..., l_n)$, where l_i is the length of v_i;
- a set of contig read counts $R = (r_1, r_2, ..., r_n)$, where r_i is the read count of v_i;

to partition the assembly graph, the goal is to determine:

– the Poisson components underlying the observed read counts

– the most probable assignment of contigs to those components based on their read counts

If $M = \{m_1, m_2, ..., m_k\}$ is a set of Poisson components underlying the observed set of read counts with a set of respective mean read counts $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_k\}$, the probability of observing R, under the previous assumptions is given with Equation 3.4. The natural logarithm of this equation gives a score for each partition with k clusters in the hierarchical clustering tree, where λ_i is the mean arrival rate of the cluster, estimated as described in the following section.

$$p(R|M) = \prod_{i=1}^{n} p(r_i|M) = \prod_{i=1}^{n} \sum_{j=1}^{k} p(r_i|\lambda_k l_i) p(v_i \in M_k) \tag{3.4}$$

3.3.2. Parameter estimation

Maximum likelihood estimation is a general method for estimating unknown parameters of a probabilistic model. Given a probabilistic model p and a set of observations $\mathbf{X}$, maximum likelihood estimation produces a set of parameters $\hat{\boldsymbol{\theta}}$ called maximum likelihood estimate (MLE) as it maximizes the likelihood of a set of parameters given the observations over all possible sets of parameters $\boldsymbol{\Theta}$. Maximizing the likelihood refers to maximizing the probability of observing the data with a particular set of parameters given the statistical model. For independent and identically distributed observations MLE is defined with Equation 3.5.

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\max}\, \mathcal{L}(\boldsymbol{\theta}|\mathbf{X}) = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\max}\, p(\mathbf{X}|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\max} \prod_{\mathbf{x}\in\mathbf{X}} p(\mathbf{x}|\boldsymbol{\theta}) \tag{3.5}$$

Since natural logarithm is a strictly monotone increasing function, it is often easier to find MLE by maximizing the log-likelihood. For Poisson distribution, MLE for the mean of the distribution of read counts $\hat{\lambda}$ can easily be found analytically by maximizing log-likelihood resulting in $\hat{\lambda} = \frac{\lambda_1+\lambda_2+...+\lambda_n}{n}$. For negative binomial distribution, maximum likelihood estimation does not give a solution in closed form. To avoid complex computations, the mean read count is approximated with the mean of read counts of all contigs in the cluster and the variance over mean ratio is approximated globally for all clusters as the mean variance over mean ratio of read counts in blocks of predefined length for all contigs. In the future, this could be replaced with better and more sophisticated approximations.

3.3.3. Bayesian information criterion

Bayesian information criterion (BIC) was proposed in 1978. [25] as a criterion for selecting one model from a finite set of possible models. It was derived under the assumption that the distribution of the data is in the exponential family, which is true for both Poisson and negative binomial distribution. In BIC a term is subtracted from the log-likelihood of the model to penalize the complexity of the model, i.e. the number of parameters of the model. This is necessary because maximum log-likelihood selection principle would always lead to choosing the most complex model, which would in this case result in clustering each contig separately, since the scores are computed using this principle. BIC score of a model M with k parameters estimated for n observations is shown with Equation 3.6. The goal is to find the model M with the highest BIC score.

$$\text{BIC} = \log \mathcal{L}(M) - \frac{1}{2} k \log n \tag{3.6}$$

In order to efficiently compute the mixture model which maximizes the BIC determined by cutting the hierarchical clustering trees in k clusters, BIC score can be decomposed as shown with Equation 3.7. It is easy to see that for any tree the optimal cut either uses the root node i with the score $\log p(R|\lambda_i) - \frac{1}{2}\log n$ or is made of optimal cuts of the children nodes i_1 and i_1 with the score $\log p(R|\lambda_{i_1}) - \frac{1}{2}\log n + \log p(R|\lambda_{i_2}) - \frac{1}{2}\log n$ and the solution can be found recursively.

$$\ln p(R|\Lambda) - \frac{1}{2} k \log n = \sum_{i=1}^{k} \left(\log p(R|\lambda_i) - \frac{1}{2} \log n \right) \tag{3.7}$$

For the Poisson distribution, the *contig-based* scoring approach described above is used. However, for negative binomial approach, *window-based* scoring approach is used. This approach can be viewed as considering each contig as a cluster of contig windows. Only a few minor changes need to be made: all contigs are splitted into windows of the same predefined length, instead of scoring the contig read count, read counts in each window are scored and n is changed from the number of contigs to the number of windows.

3.4. Scaffolding

Scaffolding, ordering and orienting contigs, typically using paired-end/mate-pair libraries, to obtain larger, non-contiguous sequences is an important step in obtaining a better reconstruction of the genome. Given the assembly graph described in section 3.1, there are two possible orientations, v_i and $-v_i$, for each contig $v_i \in V$. A scaffold is then given by a signed permutation of the contigs and the gap sizes between the adjacent contigs which can be estimated from edges in E. Exhaustive search over all $2^{|V|}|V|!$ possible solutions is clearly not feasible.

Opera was proposed in 2011 [7] as an exact solution for the scaffolding problem. Unlike previously proposed approaches whose algorithms rely mostly on heuristics, Opera's algorithm is guaranteed to find an optimal solution with the optimality criterion being the maximization of number of concordant edges in the assembly graph. An edge is concordant if the suggested orientation of the paired reads is satisfied and the distance between the paired reads is less than the maximum library size, otherwise the edge is discordant. The main advantages of this approach are that it utilizes as many edges as possible and at the same time avoids overly aggressive scaffolding that could produce longer but erroneous scaffolds. Opera's algorithm can robustly handle errors caused by assembly and mapping errors, and chimeric mate-pair reads.

Given a set of contigs and a mapping of paired reads to contigs, edge bundling is used to generate the assembly graph, as described in section 3.1. Usually, edges supported with less than a prespecified number of paired reads, e.g., 5, are removed from the assembly graph. Repetitive contigs, with the coverage above a prespecified threshold, are also usually removed from the assembly graph.

Partial scaffold S' is defined as a scaffold on a subset of contigs. Dangling set $D(S')$ is defined as a set of edges from S' to $V - S'$. Active region $A(S')$ is defined as the shortest suffix of S' such that all dangling edges are adjacent to a contig in $A(S')$. $X(S')$ is defined as a set of discordant edges in S'. Authors show that by considering two partial scaffolds S'_1 and S'_2 with less than p discordant edges the following property holds: if $(A(S'_1), X(S'_1)) = (A(S'_2), X(S'_2))$ then S'_1 and S'_2 contain the same set of contigs and both or neither of them can be extended to a solution.

They propose a dynamic programming algorithm which searches through all possible (A, X) pairs. Given a minimum contig length l_{min} and an upper-bound on the paired-end/mate-pair library insert size τ, the upper bound on the number of contigs that can be spanned by an edge is given with $w = \frac{\tau}{l_{min}}$. If the maximum allowed number of discordant edges is p, Opera's algorithm runs in $O(|V|^w |E|^{p+1})$ time.

After ordering and orienting contigs, gap sizes are computed from the constraints imposed by paired reads using a maximum likelihood approach.

4. Implementation

Sigma was developed in C++ for time and memory efficiency. It was tested on Linux and Mac OS X operating systems. The implementation is commented in the code itself and the documentation is available in HTML format created using Doxygen, but a brief overview of some important classes and functions as well as installation and usage instructions are given in this chapter.

4.1. Overview

Module *sigma*
This is the main module. It contains class `Sigma` which is used for reading the configuration file described in Section 4.3.1 and storing the method parameters. After configuring the method parameters it runs the method as described with Algorithm 1.

Module *contig_reader*
This module contains interface `ContigReader` and two implementations of that interface `VelvetReader` and `SOAPdenovoReader` that can be used to read the contig files generated by Velvet and SOAPdenovo assemblers respectively.

Module *mapping_reader*
This module contains interface `MappingReader` and one implementation of that interface `SAMReader` that can be used to read the mapping files in SAM file format.

Module *edge_reader*
This module contains interface `EdgeReader` and one implementation of that interface `OperaBundleReader` that can be used to read files with bundled edges generated by Opera's bundling routine.

Algorithm 1 Sigma

```
if contigs_file_type = - then
    read_contigs(sigma_contigs_file, contig_map)
else
    contig_reader.read(contigs_file, contig_map)
    for i:=1 to num_samples do
        mapping_reader.read(mapping_files[i], i, contig_map)
    save_contigs(contig_map, sigma_contigs_file)
for i:=1 to num_edges_files do
    edge_reader.read(edges_files[i], contig_map, edge_set, skipped_edges_files[i])
edge_queue $\leftarrow$ construct_priority_queue(edge_set)
cluster_graph(contig_map, edge_queue)
cluster_graph.compute_scores(probability_distribution)
cluster_graph.compute_models()
cluster_graph.save_clusters()
for i:=1 to num_edges_files do
    edge_reader.filter(edges_files[i], contig_map, filtered_edges_files[i])
```

Module *probability_distribution*

This module contains interface `ProbabilityDistribution` and two implementations of that interface `PoissonDistribution` implementing Poisson distribution and `NegativeBinomialDistribution` implementing negative binomial distribution. Methods computing log of the probability mass function compute it efficiently using Stirling's series approximation for $\log(x!)$

Module *contig*

This module contains classes `Contig` and `ContigIO`. Objects of class `Contig` store contig information such as contig id, contig length, starting position of the first window, ending position of the last window, number of windows, number of reads in each window and current cluster containing the contig. Class `ContigIO` provides methods for reading and saving contigs in Sigma intermediate contig format, so the method can be rerun without the need to read the original contig and mapping files again.

Module *edge*

This module contains classes `Edge`, `EdgeHash` and `EdgeComparator`. Objects of class `Edge` store bundled edges including pointer to contigs incident to the edge and their distance. Distance is computed with Algorithm 2 as the average of normalized distances in all samples for which at least one contig has arrival rate larger than 0. Class `EdgeHash` computes the hash value for the edge. The hash value is computed with inbuilt std::hash function for std::string objects created by a concatenation of ids of contigs incident to this edge separated by space. This class is used when reading edge files to ensure that the connection between the same pair of contigs is stored only once. Class `EdgeComparator` compares two edges based on the distance of the contigs connected by those edges. This class is used for constructing the priority queue of edges where edge with a smaller distance has a higher priority.

Algorithm 2 compute_distance(contig1, contig2)

```
sum_dists := 0
num_non_zero_samples := num_samples
for i:=1 to num_samples do
    arr_rate1 = contig1.arrival_rates[i]
    arr_rate2 = contig2.arrival_rates[i]
    if arr_rate1 < $10^{-6}$ and arr_rate2 < $10^{-6}$ then
        num_non_zero_samples -= 1
    else
        sum_dists += abs(arr_rate1-arr_rate2) / max(arr_rate1, arr_rate2)
return sum_dists / num_non_zero_samples
```

Module *cluster*

This module contains class `Cluster`. Objects of class `Cluster` store a pointer to an array of contigs belonging to the cluster, number of contigs, length, read counts and arrival rates for all samples, pointers to left and right child, score and model score.

Module *cluster_graph*

This module contains class `ClusterGraph` with implementations of the key methods for constructing hierarchical clustering trees with Algorithm 3 and recursively computing model for each cluster with Algorithm 4 starting from the leaves of hierarchical clustering trees.

Algorithm 3 cluster_graph(contig_map, edge_queue)

```
for contig in contig_map do
    roots.insert(new Cluster(contig))
for edge in edge_queue do
    cluster1 = edge.contig1().cluster()
    cluster2 = edge.contig2().cluster()
    if cluster1 != cluster2 then
        roots.insert(new Cluster(cluster1, cluster2))
        roots.erase(cluster1)
        roots.erase(cluster2)
```

Algorithm 4 cluster_graph.compute_cluster_model(cluster)

```
if cluster.num_contigs() = 1 then
    cluster.set_model_score(cluster.score())
    cluster.set_connected(true)
else
    c_score = cluster.score()
    disc_score = cluster.child1().model_score() + cluster.child2().model_score()
    if c_score >= disc_score then
        cluster.set_model_score(c_score)
        cluster.set_connected(true)
    else
        cluster.set_model_score(disc_score)
        cluster.set_connected(false)
```

4.2. Installation

Installing Sigma requires a C++ compiler supporting C++0x/C++11 standard and GNU make utility. To install Sigma go to the source directory and simply type:

```
make
```

After installation, a binary executable file `sigma` will be created in the source directory. To remove the binary files go to the source directory and type:

```
make clean
```

4.3. Usage

4.3.1. Configuration file

This is an example of a configuration file:

```
contigs_file_type = SOAPdenovo
contigs_file = /dataset/contig.fa
mapping_files = /dataset/mapping.sam
edges_files = /dataset/300.dat,/dataset/10k.dat

output_dir = /dataset/SIGMA/

contig_len_thr = 500 # default: 500
contig_edge_len = 80 # default: 0
contig_window_len = 340 # default: 0

pdist_type = NegativeBinomial
```

The configuration file follows the format of Opera's [7] configuration file. Each line contains parameter name and parameter value separated with =. Everything following a # is considered a comment.

Sigma takes the following parameters as the input:

- `contigs_file_type` - name of the assembler used for contigging (either Velvet or SOAPdenovo)
- `contigs_file` - path to file with assembled contigs
- `mapping_files` - comma separated paths to mapping files in SAM format, or − for reading from standard input
- `edges_files` - comma separated paths to files with bundled edges
- `sigma_contigs_file` - path to intermediate file with contigs; used for input when `contigs_file_type` is not specified, otherwise used for output
- `contig_len_thr` - contig length threshold; contigs shorter than this threshold will not be clustered by the method; default: 500
- `contig_edge_len` - length of the contig edge which is disregarded when computing read counts and arrival rates; default: 0

– `contig_window_len` - length of the window used for scoring the clusters; if set to 0 the entire contig is used for scoring; default: 0

– `pdist_type` - probability distribution (either Poisson or NegativeBinomial); default: Poisson

– `vmr` - variance to mean ratio for negative binomial distribution; default: mean vmr estimated from all contigs longer than 10k bp

4.3.2. Running

After successful installation and creation of a configuration file `sigma.config`, you can run Sigma from the directory with the binary executable file by typing:

```
./sigma sigma.config
```

4.3.3. Output files

After running Sigma, the output directory will contain the following files:

– `clusters` - a simple text file containing one line for each contig with the following format: contig id, cluster id, contig read count, cluster arrival rate

– `skipped_bundle.dat` - a text file containing skipped edges for which one or both incident contigs were not found

– `filtered_bundle.dat` - a text file containing filtered edges connecting contigs belonging to the same cluster

Files prefixed with `skipped_` and `filtered_` will be created separately for each file with bundled edges.

5. Results

5.1. Datasets

5.1.1. Simulated datasets

In order to objectively evaluate the correctness of our method, we used four simulated datasets for which the correct solution can be computed.

Three of these datasets, *simLC*, *simMC* and *simHC*, were constructed based on the datasets for standardized benchmarking of methods for processing metagenomic sequences [26]. Each dataset contains the same set of 113 species with varying relative abundance reflecting real metagenomic datasets in terms of complexity and phylogenetic composition:

– *simLC* - low-complexity community with single dominant species

– *simMC* - medium-complexity community with multiple dominant species

– *simHC* - high-complexity community without dominant species

The fourth dataset, *oral biome*, mimics 41 most abundant species in the human salivary microbiome and it was constructed based on the data from a recent study [27]. Figure 5.1 illustrates the distribution of relative species abundance in all simulated datasets.

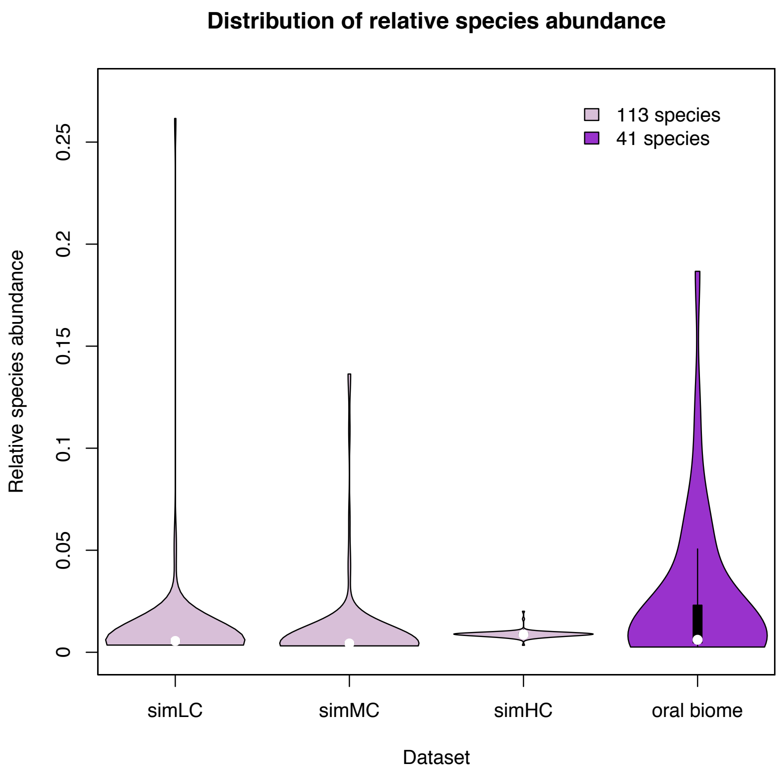

Figure 5.1: Violin plots showing the distribution of relative species abundance in *simLC*, *simMC*, *simHC* and *oral biome* datasets.

MetaSim [28] was used to generate three paired-end and mate-pair libraries for the simulated datasets:

– 2×80 bp, insert size 300 bp, $50\times$ coverage, Illumina 80 bp error model

– 2×50 bp, insert size 2000 bp, $2\times$ coverage, Illumina 50 bp error model

– 2×50 bp, insert size 10000 bp, $2\times$ coverage, Illumina 50 bp error model

5.1.2. Real datasets

Since the simulated datasets do not account for various sequencing biases, we also used three real datasets to evaluate our method.

The first dataset, *cow rumen*, contains metagenomic DNA sequenced from microbes adherent to plant fiber incubated in cow rumen [4]. The following paired-end and mate-pair libraries were downloaded from NCBI Sequence Read Archive under accession number SRA023560:

– 2×125 bp, insert size 200 bp, 17.3 Gb, Illumina sequencing

- 2×101 bp, insert size 300 bp, 191.3 Gb, Illumina sequencing
- 2×75 bp, insert size 3000 bp, 31.7 Gb, Illumina sequencing
- 2×75 bp, insert size 5000 bp, 26.8 Gb, Illumina sequencing

The second two datasets, *kern rei 35 PCB* and *kern rei DWDNA PCB*, contain metagenomic DNA sequenced from two enrichment cultures that could possibly be used to degrade toxic environmental pollutants PCBs. These datasets contain only one paired-end library:

- 2×76 bp, insert size 300 bp, Illumina sequencing

5.2. Contigging, read mapping and edge bundling

All four simulated datasets were assembled using Velvet [8] and SOAPdenovo [9]. Reads were mapped back to the assembled contigs using Bowtie [19] and the mapping was used to generate edges with Opera's [7] bundling routine.

Assembled contigs were aligned back to the reference genomes using MUMmer [29]. Based on all alignments of length ≥ 400 bp and identity $\geq 95\%$, each contig was labeled as one of the following types:

- *unique* - all regions were aligned to the same genome and the number of indels between the alignments was $< 5\%$ of total alignment length
- *repeat* - the sum of non-overlapping regions of the alignments was $> 150\%$ of contig length
- *misassembly* - regions were uniquely aligned to different genomes

Figure 5.2 shows the distribution of contig lengths and Figure 5.3 shows the number of *unique*, *repeat* and *misassembly* contigs for both contig assemblies. We can see that Velvet manages to assemble larger contigs, resulting in a smaller number of contigs than SOAPdenovo, but a lot more contigs assembled by Velvet are labeled as misassemblies.

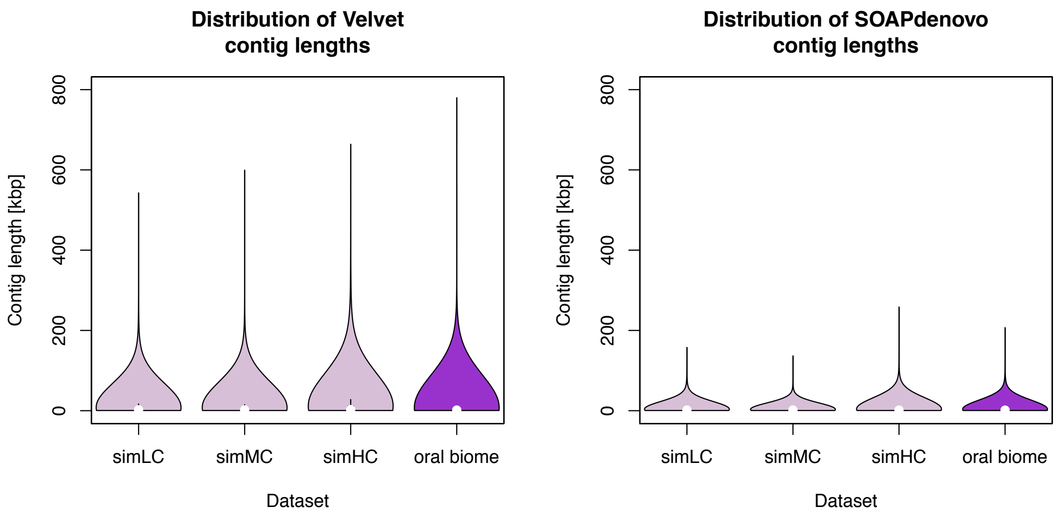

Figure 5.2: Violin plots showing the distribution of contig lengths for Velvet and SOAPdenovo assembly.

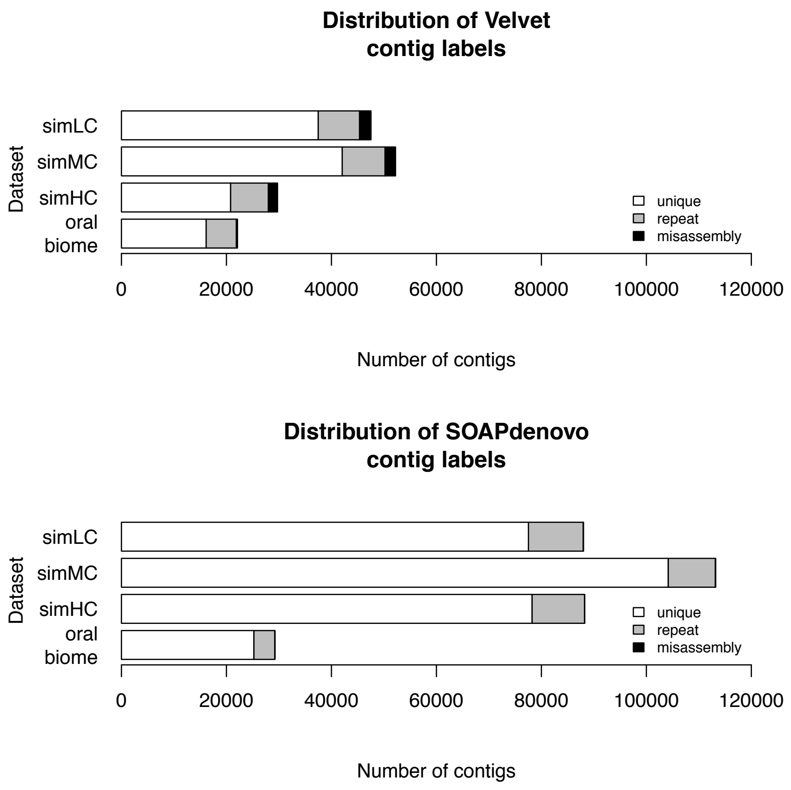

Figure 5.3: Bar plots showing the number of contigs labeled as *unique*, *repeat* and *misassembly* for Velvet and SOAPdenovo assembly.

5.3. Assembly graph partitioning and scaffolding

The main goal of Sigma is to correctly partition the assembly graph. Therefore, we evaluated its sensitivity and specificity in detecting erroneous edges in the assembly graph.

Based on the type of incident contigs, each edge was labeled as one of the following types:

- *erroneous* - connects *unique* contigs aligned to different genomes or *unique* contig with *repeat* contig
- *non-erroneous* - connects *unique* contigs aligned to the same genome

Based on the output of Sigma, each edge was classified as one of the following types:

- *erroneous* - connects contigs belonging to different clusters
- *non-erroneous* - connects contigs belonging to the same cluster

To measure the performance of Sigma, usual measures used in binary classification were defined as:

- *true positive (TP)* - number of edges labeled as *erroneous* and classified as *erroneous*
- *true negative (TN)* - number of edges labeled as *non-erroneous* and classified as *non-erroneous*
- *false positive (FP)* - number of edges labeled as *non-erroneous* and classified as *erroneous*
- *false negative (FN)* - number of edges labeled as *erroneous* and classified as *non-erroneous*

Sensitivity and specificity were defined as:

- *sensitivity* $= \frac{TP}{TP+FN}$ - a fraction of erroneous edges removed by the approach
- *specificity* $= \frac{TN}{TN+FP}$ - a fraction of non-erroneous edges preserved by the approach

Besides correctness, we evaluated contiguity of scaffold assembly produced by Opera after partitioning the assembly graph with Sigma using the following measures:

- *N50* - the length of the smallest scaffold such that at least 50% of the total scaffold length is contained in scaffolds of that size or longer

– *corrected N50 (cN50)* - computed in the same way as N50, but after breaking scaffolds at all misjoins between two contigs

Contig assemblies were labeled with assembler name and insert sizes of used paired-end/mate-pair libraries. Contig-based approach using Poisson distribution was labeled with SigmaP and window-based approach using negative binomial distribution was labeled with SigmaNBW. These two approaches were compared in terms of sensitivity and specificity shown in Table 5.1 as well as N50 and corrected N50 shown in Table 5.2.

We can see that both approaches achieve similar sensitivity and specificity. SigmaP consistently has a bit higher sensitivity, while SigmaNBW consistently has a bit higher specificity. This is expected, since the negative binomial distribution allows for a more flexible model than the Poisson distribution. Another interesting observation can be made when comparing the results on Velvet and SOAPdenovo contig assemblies. We can see that both approaches achieve much better results on SOAPdenovo contig assemblies. This can be explained by the fact that Velvet seems to output longer, but more erroneous contigs, as shown in Section 5.2. Specificity seems to be more influenced by the quality of assembled contigs than sensitivity. For some datasets specificity is almost 40% lower for Velvet contig assembly. The only dataset for which this behaviour is not apparent is the least complex one, *oral biome*, for which the achieved sensitivity and specificity are similar for both contig assemblies. We can conclude that a good contig assembly will typically result in a much better scaffold assembly. Opera achieves similar scaffold N50 and corrected N50 after partitioning the assembly graph with both approaches.

The read count distribution seems to slightly deviate from the Poisson distribution due to mappability issues even in simulated datasets. In real datasets, it is expected that the distribution of read counts will deviate even more from the Poisson distribution due to sequencing biases. Table 5.3 shows sensitivity and specificity in detecting erroneous edges incident to at least one contig originating from the previously assembled genome of the most abundant species in *kern rei* datasets. We can observe a more significant decrease in specificity when using SigmaP approach compared to SigmaNBW approach. Since Opera can deal with a fraction of incorrect edges, we chose SigmaNBW as the default approach as it achieves higher specificity.

Table 5.1: Sensitivity and specificity in detecting erroneous edges in the assembly graph of different Sigma approaches.

Dataset	Assembly	SigmaP		SigmaNBW	
		Sensitivity	Specificity	Sensitivity	Specificity
simLC	Velvet 300	**0.813**	**0.826**	**0.758**	0.880
	Velvet 10k	0.986	**0.728**	0.982	**0.775**
	Velvet 300 10k	0.924	**0.668**	0.912	**0.695**
	SOAPdenovo 300	0.960	0.988	0.934	0.998
	SOAPdenovo 10k	0.999	0.993	0.998	0.997
	SOAPdenovo 300 10k	0.984	0.989	0.977	0.992
simMC	Velvet 300	**0.795**	0.883	**0.733**	0.932
	Velvet 10k	0.983	**0.815**	0.980	0.851
	Velvet 300 10k	0.906	**0.765**	0.891	**0.793**
	SOAPdenovo 300	0.957	0.991	0.914	0.998
	SOAPdenovo 10k	0.992	0.994	0.987	0.996
	SOAPdenovo 300 10k	0.976	0.986	0.965	0.988
simHC	Velvet 300	**0.803**	**0.742**	**0.823**	0.888
	Velvet 10k	0.980	**0.657**	0.975	**0.710**
	Velvet 300 10k	0.910	**0.568**	0.891	**0.583**
	SOAPdenovo 300	0.914	0.986	0.901	0.997
	SOAPdenovo 10k	0.999	0.992	0.998	0.998
	SOAPdenovo 300 10k	0.974	0.945	0.966	0.975
oral biome	Velvet 300	0.968	0.943	0.938	0.973
	Velvet 10k	0.985	0.963	0.984	0.962
	Velvet 300 10k	0.981	0.917	0.978	0.940
	SOAPdenovo 300	0.983	0.985	0.971	0.997
	SOAPdenovo 10k	0.988	0.994	0.982	0.999
	SOAPdenovo 300 10k	0.988	0.986	0.983	0.992

Table 5.2: N50 and corrected N50 of scaffolds produced by Opera after partitioning the assembly graph with different Sigma approaches.

Dataset	Assembly	SigmaP		SigmaNBW	
		N50 [kbp]	cN50 [kbp]	N50 [kbp]	cN50 [kbp]
simLC	Velvet 300	22	22	23	23
	Velvet 10k	152	126	145	120
	Velvet 300 10k	122	103	100	90
	SOAPdenovo 300	28	28	28	28
	SOAPdenovo 10k	1081	248	1084	248
	SOAPdenovo 300 10k	1729	1472	1746	1523
simMC	Velvet 300	24	24	25	25
	Velvet 10k	167	145	159	137
	Velvet 300 10k	152	134	140	127
	SOAPdenovo 300	21	21	22	22
	SOAPdenovo 10k	408	111	404	112
	SOAPdenovo 300 10k	1198	997	1125	939
simHC	Velvet 300	27	27	29	28
	Velvet 10k	147	144	168	160
	Velvet 300 10k	115	111	102	97
	SOAPdenovo 300	29	29	29	29
	SOAPdenovo 10k	1490	302	1492	296
	SOAPdenovo 300 10k	1754	1234	1859	1357
oral biome	Velvet 300	19	19	20	19
	Velvet 10k	152	95	173	107
	Velvet 300 10k	336	244	453	294
	SOAPdenovo 300	21	21	21	21
	SOAPdenovo 10k	323	131	324	131
	SOAPdenovo 300 10k	1118	639	1121	650

Table 5.3: Sensitivity and specificity of different Sigma approaches in detecting erroneous edges in the assembly graph for the most abundant species in *kern rei* datasets.

Dataset	SigmaP		SigmaNBW	
	Sensitivity	Specificity	Sensitivity	Specificity
kern rei 35 PCB	1.000	**0.816**	1.000	0.994
kern rei DWDNA PCB	0.997	**0.785**	0.993	0.973

Figure 5.4 shows sensitivity and specificity in detecting erroneous edges in the assembly graph achieved by Sigma and Bambus2 with Velvet as the contig assembly. We can see that Sigma achieves slightly higher specificity and substantially higher sensitivity than Bambus2.

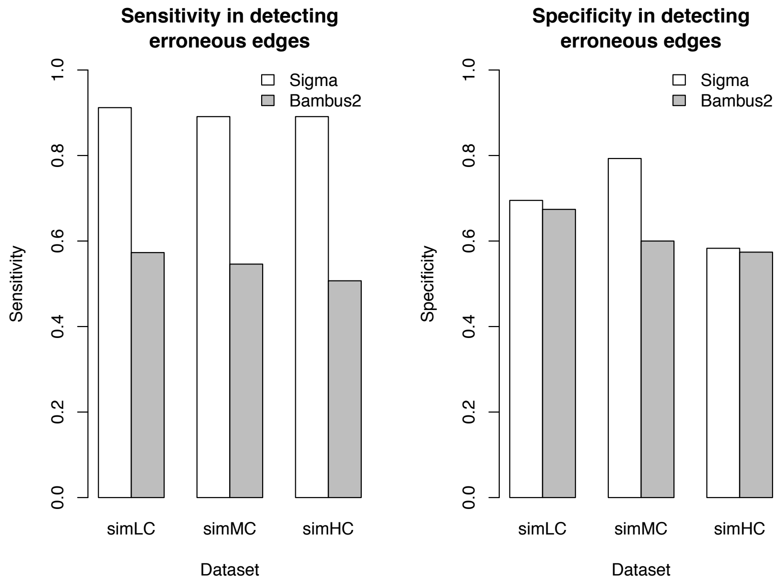

Figure 5.4: Comparison between sensitivity and specificity in detecting erroneous edges in the assembly graph achieved by Sigma and Bambus2 with Velvet as the contig assembly.

Figure 5.5 illustrates the importance of combining paired-end/mate-pair libraries with different insert sizes and their influence on correctness and contiguity of the final scaffolds. Using only a 300 bp or only a 2k bp insert size produces correct, but short scaffolds. Using only a 10k bp insert size produces much longer, but very erroneous scaffolds, since small contigs can be rearranged within a 10k bp constraint in many ways. Using a combination of 300 bp and 10k bp insert sizes utilizes the advantage of combining short-insert library with long-insert library the most. Adding a 2k bp insert size does not significantly improve the final scaffolds.

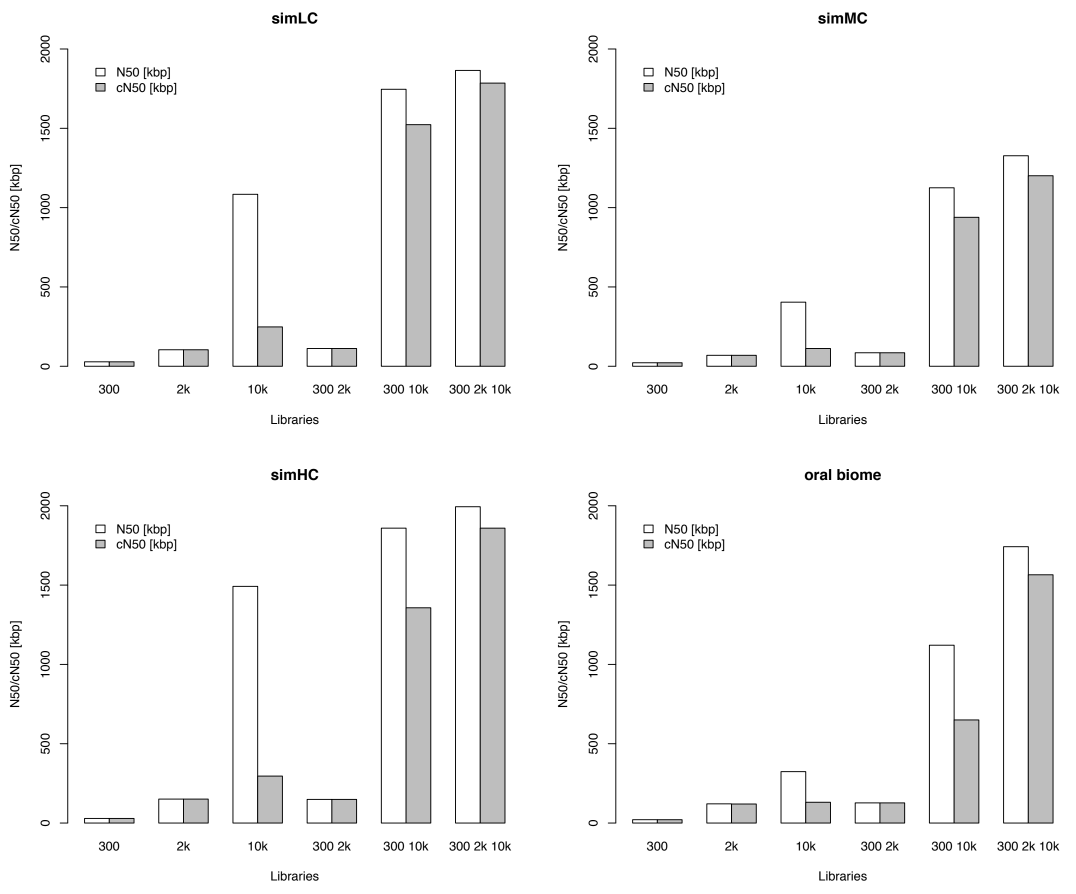

Figure 5.5: N50 and corrected N50 of the final scaffolds produced by OperaMS using different paired-end/mate-pair libraries and their combinations.

Tables 5.4 to 5.7 show scaffold statistics for the simulated datasets for OperaMS with Velvet as the contig assembly (VO), OperaMS with SOAPdenovo as the contig assembly (SO), Velvet (V), SOAPdenovo (S), Bambus2 with Velvet as the contig assembly (B) and MetaVelvet (MV). Besides N50 and corrected N50, we evaluated the error rate as the fraction of incorrect edges present in the scaffolds as well as the number of misjoins between species belonging to different genus, between species belonging to same genus and between different chromosomes of the same species and number of inverse relocations, i.e., incorrect orientations between two contigs.

OperaMS significantly outperforms all other methods in terms of N50 and corrected N50. OperaMS makes less misassemblies than Velvet and has a lower error rate on the same contig assembly. OperaMS makes similar or sometimes a bit higher number of misassemblies than SOAPdenovo and has a bit higher error rate on the same contig assembly, but the contiguity achieved by OperaMS is much higher. Overall, error rate is lower than 5% and the number of misjoins is quite low for all datasets when they are scaffolded with OperaMS. It is interesting to see that Bambus2 and MetaVelvet perform worse than single genome assemblers on these datasets. MetaVelvet did not produce scaffolds for any of the datasets and Bambus2 crashed on *oral biome* dataset.

Table 5.4: Scaffold statistics for *simLC* dataset.

Method	N50 [kbp]	cN50 [kbp]	error rate	Misjoins diff. genus	Misjoins same genus	Misjoins diff. chr.	inv. reloc.
VO 300 10k	100	90	0.047	72	214	38	748
SO 300 10k	1746	1523	0.021	1	12	2	1372
V 300 10k	345	82	0.119	456	1215	90	3732
S 300 10k	582	414	0.015	2	10	0	711
B 300 10k	21	19	0.275	481	2178	131	240
MV 300 10k	2.65	2.65	NA	NA	NA	NA	NA

Table 5.5: Scaffold statistics for *simMC* dataset.

Method	N50 [kbp]	cN50 [kbp]	error rate	Misjoins diff. genus	Misjoins same genus	Misjoins diff. chr.	inv. reloc.
VO 300 10k	140	127	0.055	55	293	22	1138
SO 300 10k	1125	939	0.022	1	12	6	1933
V 300 10k	363	88	0.106	283	1465	74	2917
S 300 10k	266	217	0.010	2	3	5	566
B 300 10k	16	15	0.205	404	346	33	120
MV 300 10k	2.89	2.89	NA	NA	NA	NA	NA

Table 5.6: Scaffold statistics for *simHC* dataset.

Method	N50 [kbp]	cN50 [kbp]	error rate	Misjoins diff. genus	Misjoins same genus	Misjoins diff. chr.	inv. reloc.
VO 300 10k	102	97	0.029	23	144	30	126
SO 300 10k	1859	1357	0.038	4	24	1	2543
V 300 10k	376	84	0.137	359	890	62	5329
S 300 10k	629	431	0.014	0	5	0	660
B 300 10k	28	26	0.398	204	1835	72	142
MV 300 10k	1.86	1.86	NA	NA	NA	NA	NA

Table 5.7: Scaffold statistics for *oral biome* dataset.

Method	N50 [kbp]	cN50 [kbp]	error rate	Misjoins diff. genus	Misjoins same genus	Misjoins diff. chr.	inv. reloc.
VO 300 10k	453	294	0.034	5	1	3	450
SO 300 10k	1121	650	0.042	0	0	0	953
V 300 10k	11	10.91	0.127	125	0	2	456
S 300 10k	30	30	0.013	0	9	0	101
B 300 10k	7.89	7.89	NA	NA	NA	NA	NA
MV 300 10k	0.97	0.97	NA	NA	NA	NA	NA

Figure 5.6 illustrates the distribution of genome contig and scaffold N50 for OperaMS with SOAPdenovo as the contig assembly for the simulated datasets. We can see that the final scaffolds have much higher contiguity than the starting contigs, on average increased by a factor of 100.

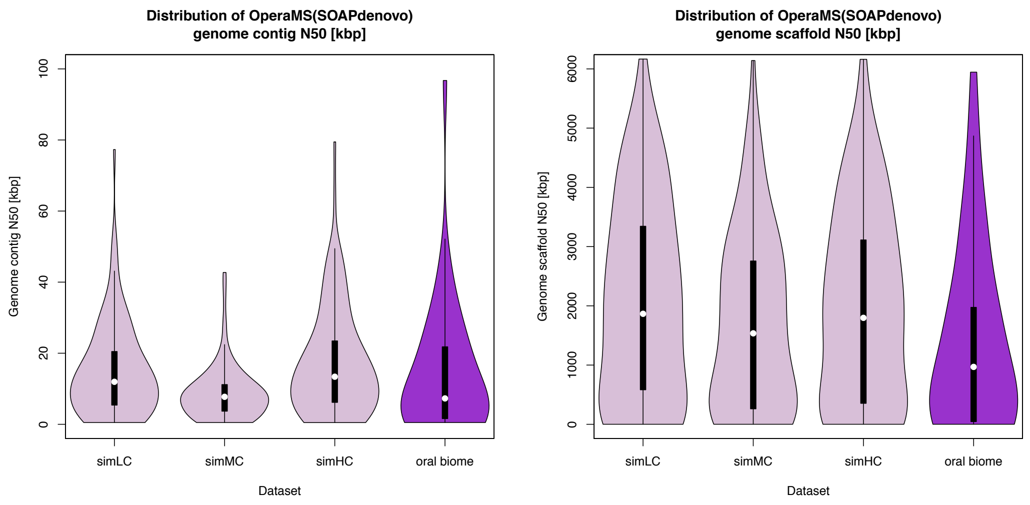

Figure 5.6: Violin plots showing the distribution of genome contig and scaffold N50 for OperaMS with SOAPdenovo as the contig assembly.

Figure 5.7 shows the comparison of OperaMS and SOAPdenovo genome scaffold corrected N50 on the same contig assembly. We can see that OperaMS produces a higher or similar scaffold corrected N50 for majority of the genomes and a lower scaffold corrected N50 for only a few genomes.

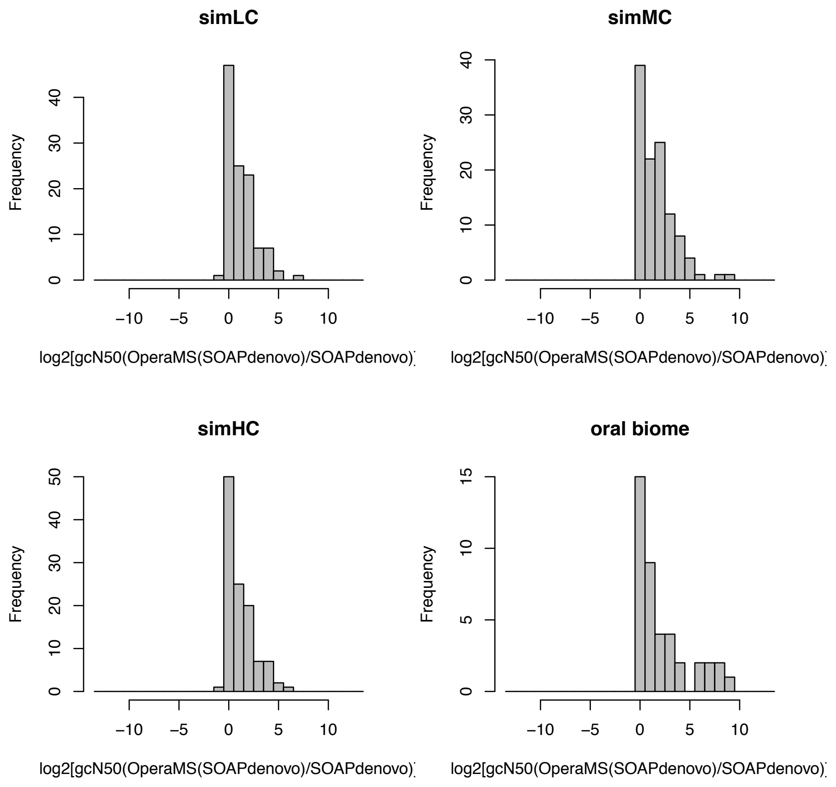

Figure 5.7: Bar plots showing the log of genome scaffold corrected N50 ratio between OperaMS with SOAPdenovo as contig assembly and SOAPdenovo.

Figure 5.8 shows the comparison of the fraction of genome reconstructed in the longest scaffold for Velvet and SOAPdenovo scaffold assembly and OperaMS scaffold assembly on the same sets of contigs. OperaMS manages to reconstruct a significantly higher number of near-complete genomes than Velvet and SOAPdenovo. OperaMS with SOAPdenovo as the contig assembly almost completely reconstructs $20 - 40\%$ of all genomes.

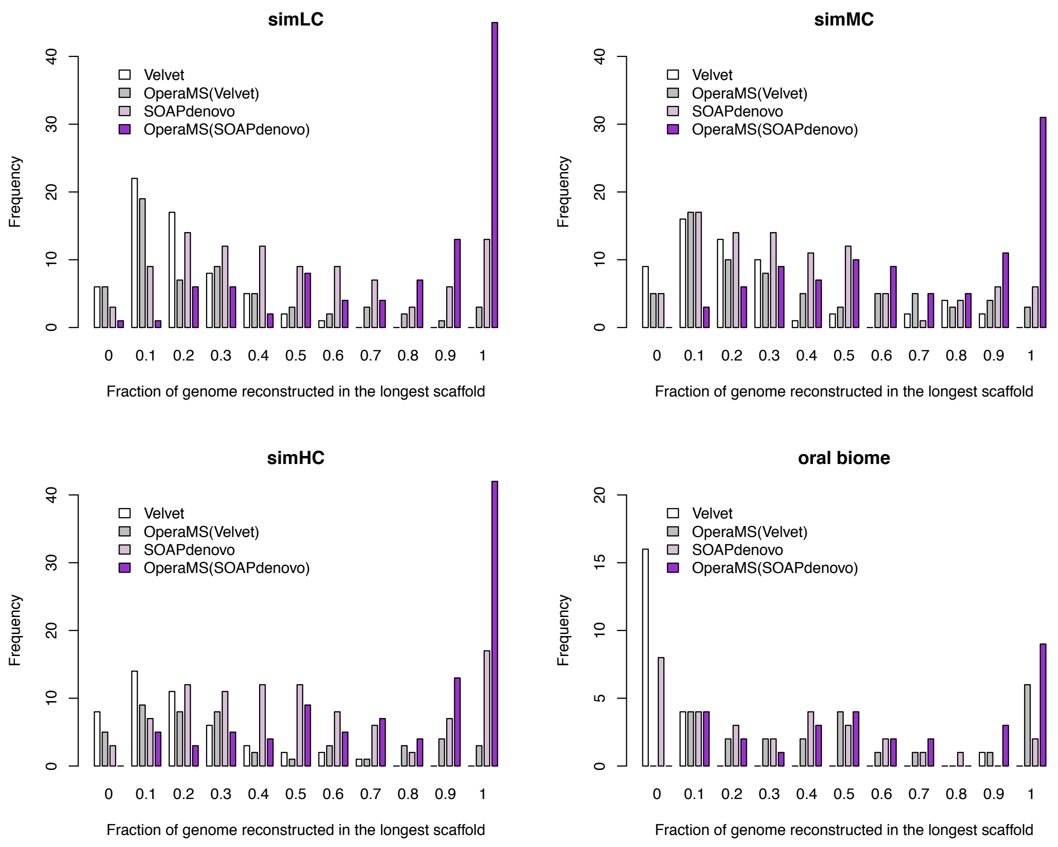

Figure 5.8: A bar plot showing the fraction of genome reconstructed in the longest scaffold by Velvet, OperaMS with Velvet as the contig assembly, SOAPdenovo and OperaMS with SOAPdenovo as the contig assembly.

Table 5.8 shows the number of scaffolds and scaffold N50 for *cow rumen* dataset. OperaMS outperforms SOAPdenovo with more than two times higher N50. OperaMS also outperforms Velvet with a bit higher N50, but Velvet manages to assemble more long scaffolds. However, as shown on the simulated datasets, Velvet tends to assemble large scaffolds at the expense of more errors.

Table 5.8: Scaffold statistics for *cow rumen* dataset.

Statistic	Method	Scaffolds $\geq$ 1kbp	Scaffolds $\geq$ 50kbp	Scaffolds $\geq$ 100kbp	Scaffolds $\geq$ 500kbp	Scaffolds $\geq$ 1Mbp
Number of scaffolds	OperaMS	96450	5507	1995	61	5
	SOAPdenovo	283580	4190	1166	35	4
	Velvet [4]	179092	7126	2441	65	8
N50 [bp]	OperaMS	39030	123149	198460	613122	1070968
	SOAPdenovo	14673	105724	195228	673006	1204035
	Velvet [4]	34338	117023	187333	667799	1085061

Tables 5.9 and 5.10 show the number of scaffolds and scaffold N50 for *kern rei* datasets. SOAPdenovo outperforms OperaMS on these datasets. This can be explained by the fact that these datasets only have a library with a 300 bp insert size and $85-90\%$ of the contigs are shorter than 500 bp. Unlike OperaMS which has a threshold on the contig length and does not scaffold contigs shorter than 500 bp, SOAPdenovo manages to scaffold those small contigs and achieve higher N50.

Table 5.9: Scaffold statistics for *kern rei 35 PCB* dataset.

Statistic	Method	Scaffolds $\geq$ 500bp	Scaffolds $\geq$ 1kbp	Scaffolds $\geq$ 2kbp	Scaffolds $\geq$ 50kbp	Scaffolds $\geq$ 100kbp
Number of scaffolds	OperaMS	3191	1812	1144	6	3
	SOAPdenovo	4741	2361	998	15	3
N50 [bp]	OperaMS	5600	6403	7419	101704	118731
	SOAPdenovo	9550	13520	18891	83544	153266

Table 5.10: Scaffold statistics for *kern rei DWDNA PCB* dataset.

Statistic	Method	Scaffolds $\geq$ 500bp	Scaffolds $\geq$ 1kbp	Scaffolds $\geq$ 2kbp	Scaffolds $\geq$ 50kbp	Scaffolds $\geq$ 100kbp
Number of scaffolds	OperaMS	4873	2996	1922	13	2
	SOAPdenovo	8685	3393	1418	53	19
N50 [bp]	OperaMS	7253	8255	9432	62752	109809
	SOAPdenovo	10311	17429	24441	105793	150262

5.4. Running time and memory requirements

Testing was performed on the following configuration:

- CentOS Linux 6.1
- Seagate® Enterprise Capacity 2.5 HDD
- Intel® Xeon® CPU X7550 @ 2.00GHz
- 512GB of RAM

Running time was measured with GNU time:

```
time ./sigma sigma.config
```

Memory requirements were measured with Valgrind [30]:

```
valgrind --tool=massif ./sigma sigma.config
```

Reading a mapping file can take from several minutes up to a few hours depending on the file size. For example, reading a 32 GB .bam file for *cow rumen* dataset takes 36.8 minutes.

The initial contig-based approach has lower memory requirements and significantly lower running time than the window-based approach, since it stores only the total read count for each contig and the time required for scoring does not depend on contig lengths. Peak memory usage and total user time for the window-based approach are shown in Table 5.11. Sigma has low memory requirements ranging from only a couple of MB up to 350 MB. The memory requirements are proportional to the total number of windows in all contigs. Running time is below 2 minutes for smaller datasets, but for larger datasets it can get up to 1.5 hours with scoring of the clusters consuming more than 90% of the time.

The scoring of the clusters is mutually independent and could be parallelized relatively easy, but since the focus of this thesis was mainly on the method itself and the running time is still reasonably low, this was left for future work.

Opera's running time is below 25 minutes for all datasets.

Table 5.11: Sigma's running time and memory requirements.

Dataset	Number of contigs	Number of edges	Running time [s]	Memory requirements [MB]
simLC	85248	138003	41.422	40.666
simMC	110698	186531	49.447	48.176
simHC	85502	155683	84.529	40.740
oral biome	28436	70651	12.982	14.644
kern rei 35 PCB	4920	3246	0.159	1.760
kern rei DWDNA PCB	9212	5810	0.354	3.323
cow rumen	828190	1187725	4915	343.032

6. Conclusion

An approach to aid in metagenomic assembly called Sigma was presented in this thesis. Sigma, a Bayesian model-based hierarchical clustering approach, serves as a preprocessing tool for partitioning the assembly graph. Sigma was combined with optimal single genome scaffolder Opera in a pipeline called OperaMS to show that metagenomic assembly problem can be accurately and automatically reduced to single genome assembly problem by systematically exploiting assembly information.

Sigma and OperaMS were tested on 4 simulated datasets and 3 real datasets. It was shown that Sigma partitions the assembly graph with high sensitivity and specificity. Provided that the quality of the initial contig assembly is high, sensitivity and specificity are above 95%. In addition, Sigma has low running time and memory requirements. It was also shown that OperaMS produces correct and contiguous scaffold assemblies, especially when using a combination of a variety of mate-pair/paired-end libraries. Provided that the quality of the initial contig assembly is high, the error rate is below 5% and genome scaffold N50 increases contig scaffold N50 by a factor of 100. In addition, OperaMS manages to almost completely reconstruct $20 - 40\%$ genomes. The performance of OperaMS seems to degrade in terms of contiguity when only a short-insert paired-end library is provided. In majority of other cases, OperaMS outperforms state-of-the-art single genome (Velvet, SOAPdenovo) and metagenomic (MetaVelvet, Bambus2) assembly tools.

Besides small implementation improvements such as parallelizing the scoring of the clusters, there are also a few method improvements left to explore such as more sophisticated parameter estimation for negative binomial distribution. Since this approach is naturally extendible to incorporate additional information such as k-mer frequencies, GC content and gene content it should be further refined to exploit that information in order to distinguish between species with similar abundance and consequently decrease the number of misjoins. The improvements in third generation sequencing technologies could possibly be used to further improve the method in the future.

Bibliography

[1] J.C. Wooley, A. Godzik, and I. Friedberg. A primer on metagenomics. *PLOS Computational Biology*, 6(2):e1000667, 2010.

[2] R. Knight et al. Unlocking the potential of metagenomics through replicated experimental design. *Nature Biotechnology*, 30(6):513–520, 2012.

[3] J. Qin et al. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, 2010.

[4] M. Hess et al. Metagenomic discovery of biomass-degrading genes and genomes from cow rumen. *Science*, 331(6016):463–467, 2011.

[5] M. Hunt et al. A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, 15(3):R42, 2014.

[6] L. Bragg and G.W. Tyson. Metagenomics using next-generation sequencing. In *Environmental Microbiology*, pages 183–201. Springer, 2014.

[7] S. Gao, W. Sung, and N. Nagarajan. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, 18(11):1681–1691, 2011.

[8] D.R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[9] R. Li et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2010.

[10] S. Koren, T.J. Treangen, and M. Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.

[11] T. Namiki et al. MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic Acids Research*, 40(20):e155, 2012.

[12] J. Handelsman. Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and Molecular Biology Reviews*, 68(4):669–685, 2004.

[13] V. Kunin et al. A bioinformatician's guide to metagenomics. *Microbiology and Molecular Biology Reviews*, 72(4):557–578, 2008.

[14] J.A. Gilbert and C.L. Dupont. Microbial metagenomics: beyond the genome. *Annual Review of Marine Science*, 3:347–371, 2011.

[15] T. Thomas, J. Gilbert, and F. Meyer. Metagenomics - a guide from sampling to data analysis. *Microbial Informatics and Experimentation*, 2(3), 2012.

[16] D. Sims et al. Sequencing depth and coverage: key considerations in genomic analyses. *Nature Reviews Genetics*, 15(2):121–132, 2014.

[17] M. Pop. Genome assembly reborn: recent computational challenges. *Briefings in Bioinformatics*, 10(4):354–366, 2009.

[18] M. Albertsen et al. Genome sequences of rare, uncultured bacteria obtained by differential coverage binning of multiple metagenomes. *Nature Biotechnology*, 31(6):533–538, 2013.

[19] B. Langmead et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[20] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[21] D.H. Huson, K. Reinert, and E.W. Myers. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM (JACM)*, 49(5):603–615, 2002.

[22] R. Xu et al. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

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

[24] J.O. Lloyd-Smith. Maximum likelihood estimation of the negative binomial dispersion parameter for highly overdispersed data, with applications to infectious diseases. *PLOS ONE*, 2(2):e180, 2007.

[25] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[26] K. Mavromatis et al. Use of simulated data sets to evaluate the fidelity of metagenomic processing methods. *Nature Methods*, 4(6):495–500, 2007.

[27] I. Nasidze et al. Global diversity in the human salivary microbiome. *Genome Research*, 19(4):636–643, 2009.

[28] D.C. Richter et al. MetaSim-A sequencing simulator for genomics and metagenomics. *PLOS ONE*, 3(10):e3373, 2008.

[29] S. Kurtz et al. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[30] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan Notices*, volume 42, pages 89–100, 2007.

De novo metagenomic assembly using Bayesian model-based clustering

Abstract

Microbial communities influence almost every aspect of our lives. Metagenomics aims to expand our knowledge of those communities by analyzing DNA samples extracted directly from their environments. Metagenomic studies still rely mostly on manual interventions and single genome assembly tools which are unaware of the nature of metagenomic data. In this thesis, a Bayesian model-based hierarchical clustering approach to aid in metagenomic assembly called Sigma is presented. Sigma is combined with an optimal single genome scaffolder Opera to show that metagenomic assembly problem can be accurately and automatically reduced to single genome assembly problem by systematically exploiting assembly information. Comparisons on simulated and real datasets show that this pipeline (OperaMS) outperforms state-of-the-art single genome (Velvet, SOAPdenovo) and metagenomic (MetaVelvet, Bambus2) assembly tools.

Keywords: metagenomics, de novo assembly, hierarchical clustering, Bayesian information criterion

De novo sastavljanje metagenomskih podataka korištenjem grupiranja podataka temeljenih na Bayesovom modelu

Sažetak

Mikrobne zajednice utječu na gotovo svaki aspekt našeg života. Metagenomika nastoji otkriti nove spoznaje o tim zajednicama analizom DNA uzoraka izuzetih direktno iz njihovih okoliša. Metagenomska istraživanja se i dalje uglavnom oslanjaju na ručne intervencije i alate namijenjene za sastavljanje genomskih podataka koji ne uzimaju u obzir specifičnosti metagenomskih podataka. U ovom radu je predstavljena metoda za de novo sastavljanje metagenomskih podataka korištenjem grupiranja podataka temeljenih na Bayesovom modelu (Sigma). Pokazano je da se kombiniranjem te metode s optimalnim skafolderom genomskih podataka (Opera) problem sastavljanja metagenomskih podataka može točno i automatski svesti na problem sastavljanja genomskih podataka iskorištavanjem informacija dostupnih tijekom sastavljanja genoma. Usporedbe na simuliranim i stvarnim metagenomskim podacima pokazuju da kombinacija ovih dvaju metoda (OperaMS) daje bolje rezultate od često korištenih alata za sastavljanje genomskih (Velvet, SOAPdenovo) i metagenomskih (MetaVelvet, Bambus2) podataka.

Ključne riječi: metagenomika, de novo sastavljanje genoma, hijerarhijsko grupiranje, Bayesov informacijski kriterij