FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Combining k-mer and quasi-read-mapping methods to improve the clustering of multiple *de novo* transcriptome assemblies

M A S T E R ' S T H E S I S

to obtain the academic degree
Master of Science (M. Sc.)
in bioinformatics

Friedrich Schiller University Jena
Faculty of mathematics and computer science

submitted by Lasse Faber
born on June 26 1992 in Karlsruhe

Supervisor: Prof. Dr. M. Marz
Jena, November 25, 2019

Abstract

Many *de novo* transcriptome assemblers yield different results due to their distinct implementations. This master's thesis aims to use the advantages of several assemblers and create an improved *de novo* transcriptome assembly. Concatenating multiple assemblies complicates the downstream analysis, such as annotation of the transcripts and the identification of differentially expressed isoforms. Here, a new algorithm based on **K**-mer **A**nd quasi-**R**ead-**MA**ppings (`karma`) is implemented that can be used on paired-end and single-end RNA-Seq data. The method relies on recent k-mer and read-graph-based clustering methods that are combined and enhanced. `karma` is compared to state-of-the-art clustering tools and evaluated in terms of completeness and correctness of the resulting *de novo* transcriptome assembly.

Conclusively, it seems sufficient to rely on a single assembler rather than combining multiple assemblers for the most use cases. However, preprocessing of reads, assembler choice and suitable k-mer size should be carefully considered. Clustering single-end datasets with `cd-hit-est` with a sequence identity of 90 % show significant improvements for the assemblies. Even though `karma` was not able to outperform other clustering methods, a comprehensive evaluation of the underlying algorithm is performed. The read-graphs built by `karma` can be utilized for further investigations of reliable *de novo* reconstructed isoforms from transcriptomic sequencing data.

Zusammenfassung

Viele *de novo* Transkriptom-Assemblierer produzieren aufgrund ihrer unterschiedlichen Implementierungen verschiedene Ergebnisse. Ziel dieser Masterarbeit ist es, die Vorteile mehrerer Assemblierer zu nutzen, um eine verbesserte *de novo* Transkriptom-Assemblierung zu erhalten. Die durch diese Methode entstehende Redundanz soll mittels eines eigens implementierten Algorithmus reduziert werden. Der Algorithmus (`karma`, engl. **K**-mer **A**nd quasi-**R**ead-**MA**ppings) benutzt, kombiniert und verbessert bereits bestehende k-mer und Graphen-basierte Methoden. Die Methode eignet sich außerdem für Single-End und Paired-End Datensätze von RNA-Sequenzierungen. `karma` wird mit aktuellen Clustering-Programmen verglichen und hinsichtlich der Vollständigkeit und Qualität der resultierenden *de novo* Transkriptom-Assemblierung bewertet.

Für die meisten Anwendungsfälle ist es ausreichend, einen einzigen Assemblierer zu verwenden. Allerdings sollte die Vorverarbeitung von Reads, die Wahl des Assemblierers und eine geeignete k-mer-Länge sorgfältig gewählt werden. Für Single-End-Datensätze erzielt `cd-hit-est` mit einer Sequenzidentität von 90 % signifikante Verbesserungen für die Transkriptom-Assemblierung. Die von `karma` erstellten Graphen können für weitere Untersuchungen von zuverlässig *de novo* rekonstruierten Isoformen aus Transkriptomdaten verwendet werden.

Contents

1 Introduction

Understanding biological processes at the molecular level has always been of great interest to life science researchers. Being responsible for cellular mechanisms and responses, the transcriptome is a research topic of significant importance [1]. The transcriptome represents all RNA molecules in a cell, reflecting its gene expression. Consequently, the methods for investigation of the transcriptome have made some remarkable changes over the past decade [2]. In the early days of molecular research, methods were time-consuming and restricted only to a small number of genes. The genes of interest were commonly chosen by hand, based on the function already known in other organisms [3]. This changed drastically with the appearance of microarrays that could be used to examine a large set of RNAs simultaneously, speeding up the research process [4, 5]. With the rise of Next-Generation Sequencing methods such as provided by Illumina, transcriptome analysis became possible on an unprecedented scale and affordable for many research groups. Nowadays, it is possible to get information about the whole transcriptome of an entire organism in a cheap and comprehensive way by sequencing the RNA (RNA-Seq). In particular, RNA-Seq data can also provide information about new organisms where there has no genome been sequenced yet.

1.1 RNA-Sequencing data

Since the design of microarrays relies on already known genes, RNA-Seq is superior for less-studied species lacking an appropriate reference genome [1]. But RNA-Seq has even more advantages compared to microarrays. Over time it is becoming more and more cost-effective, has a higher sensitivity and accuracy and allows for the detection of new transcripts, including non-coding RNA (ncRNA) [7]. The inclusion of RNA-Seq data is now part of the standard repertoire in many studies (Figure 1). For example, Wang *et al.* have used RNA-Seq data for ovarian cancer research [8]. Their findings lead to earlier detection

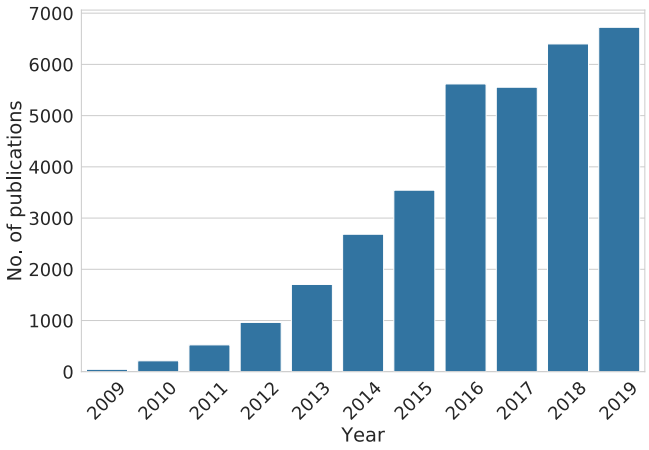

Figure 1: Search results for 'RNA-Seq' on Pubmed over the years [6].

and identification of the pathological origin by determining the responsible genes hence dysregulated molecular pathways [8]. Furthermore, Rai *et al.* describe the supremacy of RNA-Seq data analysis over microarrays in their studies of profiling human ligament tissue [9]. With all the advantages from above, the number of publications that use RNA-Seq data for their analyses has been steadily growing over the past few years (Figure 1). Along with the increase of publications comes a large number of freely available RNA-Seq datasets in the NCBI Sequence Read Archive (SRA) and EMBL-EBI European Nucleotide Archive (ENA).

1.2 Transcriptome reconstruction

Besides the use for differential gene expression analyses, RNA-Seq data can also be applied to assemble the transcripts of the whole transcriptome. This includes all different forms of expressed genes, hence the transcription from the genomic DNA into RNA molecules. These can be categorized either as non-coding RNA or protein-coding RNA. The latter is further processed and translated into a protein that fulfills a certain function in the cell. However, before the translation into a protein, the protein-coding RNA can be altered through post-transcriptional modifications such as alternative splicing. The resulting distinct variants of the same gene represent different isoforms [10]. Contrarily, a non-coding RNA is not translated into a protein, but functions as it is [11].

The widely used Illumina sequencing protocols all rely on fragmentation steps, thus produce reads of lengths between approximately 100-350 base pairs (bp). Consequently, the shorter reads have to be combined (*assembled*) to reconstruct the original transcript sequences. The final transcripts are also called contiguous sequences (*contigs*). However, assembling the reads in a way that they represent their original biological state comes with a few challenges to solve, such as the orientation of reads, repeats, contamination, sequencing errors and chimeras [12].

Additionally, an assembly can be done either with a reference genome or without [13]. Reconstruction, while having no reference available, is called a *de novo* assembly and makes the task more difficult since there is no previous knowledge about the order of the reads. The general aim of a (*de novo*) transcriptome assembly should be complete in isoforms [14], have low redundancy [15] and as few misassembled transcripts as possible [13].

Fortunately, there are a lot of tools that are already dealing with the transcriptome assembly problem such as `SOAPdenovo-Trans` [16], `Trinity` [17], `Trans-ABySS` [18] and `rnaSPAdes` [19]. However, all these tools have different advantages and disadvantages also depending on the input data set [20]. All

those programs are based on k-mer dependent De Bruijn graphs, where the reads are divided into subsequences of length k during the assembly process [21]. The selection of the correct k-mer size can be a challenge itself. Whereas small k-mer sizes tend to recover less abundant transcripts, they produce a lot of contigs that suffer from fragmentation due to lack of overlap and sequencing errors. On the other hand, big k-mer sizes have the advantage of producing a more connected assembly that consists of high coverage transcripts and spliced variants. Yet, the assembly contains fewer contigs, which leads to a lower transcript representation [22]. There are existing tools that try to estimate the best available k-mer size, such as `KmerGenie` [23]. The developers of the program also conclude that combining assemblies from a small and big k-mer size can be beneficial [23], nowadays a common procedure implemented in assembly tools such as `rnaSPAdes` [19].

Despite being based on De Bruijn graphs, all of the assemblers likely yield different assemblies due to the distinct implementations and assumptions of their developers. MacManes implemented the `Oyster River Protocol` [24] to combine multiple assemblies to receive an overall better assembly. The assumption here is that the combination of multiple assemblies will negate weaknesses from single assemblers and results in a better overall assembly. Even though more information from multiple assemblers is gained [20], a lot of redundancy is introduced that has to be taken care of through clustering. This master's thesis takes up this idea and deals more deeply with the problem of transcript clustering. Since many clusterings and their underlying methods are based on the concept of graphs, the following section will give a brief introduction into the topic.

1.3 Basic graph theory

In short, a graph is a data structure to represent a relationship between objects. The formal description of a graph is $G = (V, E, \omega)$, where a set of objects represent the vertices V and their relationship among them as edges E. A distance based on a function ω can be assigned to the edges to show a (dis-) similarity between different vertices. This value is also called the weight of an edge but is not a necessary property for a graph. Without the weights, the graph is called an *unweighted graph* (Figure 2A) or with weights a *weighted graph* (Figure 2B), respectively. Furthermore, the graph can be either *directed* or *undirected*. Assuming a walk through the graph between vertices using the edges as connections, directed edges only allow for a walk in one direction between two vertices (Figure 2A). In an undirected graph, both directions are allowed (Figure 2B) [25]. The representation mentioned above of objects and their similarities offer one possibility to cluster objects.

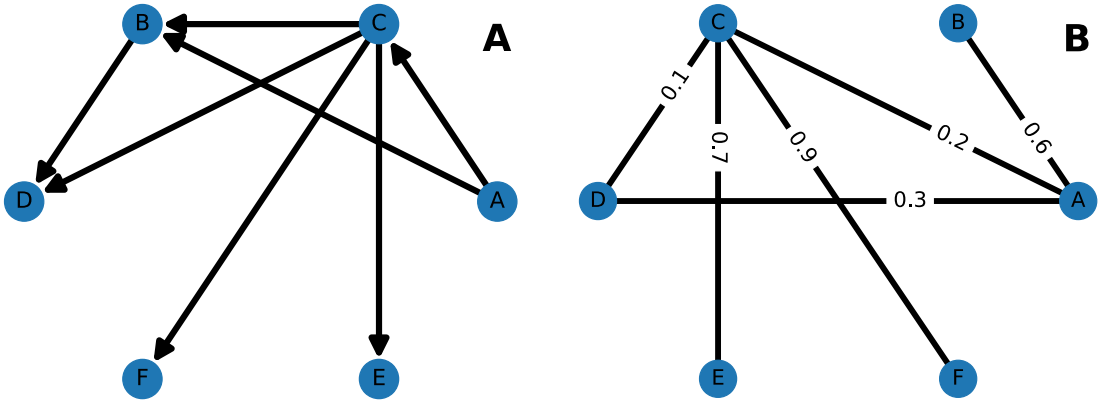

Figure 2: Visualization of different graphs. **(A)** A directed unweighted graph. **(B)** An undirected weighted graph with weights between 0.1 and 0.9.

1.4 Clustering

Simply said, clustering describes the process of grouping a certain dataset into subsets of similar properties. It is a problem that is not only bound to data science or bioinformatics but affects us every day e.g., while sorting laundry. The objects in the datasets usually have a vector of properties, which is a list of features for the data point. This can be seen as a simple representation of a multi-dimensional feature space (Figure 3).

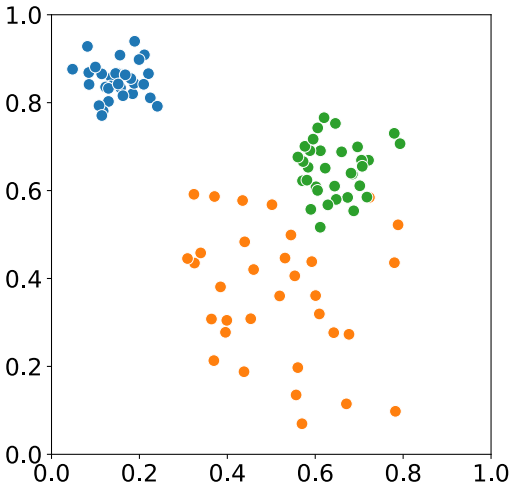

Figure 3: Graphical example of objects in a two-dimensional feature space. Different colored dots represent groups of objects with distinct distributions.

In science, clustering is generally a classification problem. There are two main categories of classifications, unsupervised and supervised. When speaking of the latter, there is an existing set of examples based on which the classification task tries to classify new objects into existing labels. Contrarily, unsupervised classification tries to identify unknown patterns in a dataset without having pre-existing labels available. Therefore, clustering is unsupervised learning, where the resulting groups are based on some similarity among those objects [26]. The resulting subsets from the process are also called clusters. For the clustering task, there are already a vast number of implemented algorithms that can be categorized in at least one of the following four groups.

Hierarchical-based clustering (Connectivity-based clustering)

The hierarchical clustering is mainly based on the distance between objects, which represents a hierarchy (Figure 4). The core idea is that objects are more related to objects closer to them than those further away. From the resulting hierarchy, there are two methods to extract the clusters. The bottom-up approach (agglomerative) starts with each object in its own cluster. Going up the hierarchy, the objects are then merged into clusters. The top-down approach (divisive) starts with all objects in one cluster. Going down the hierarchy, the first cluster is subsequently divided into smaller sub-clusters [27].

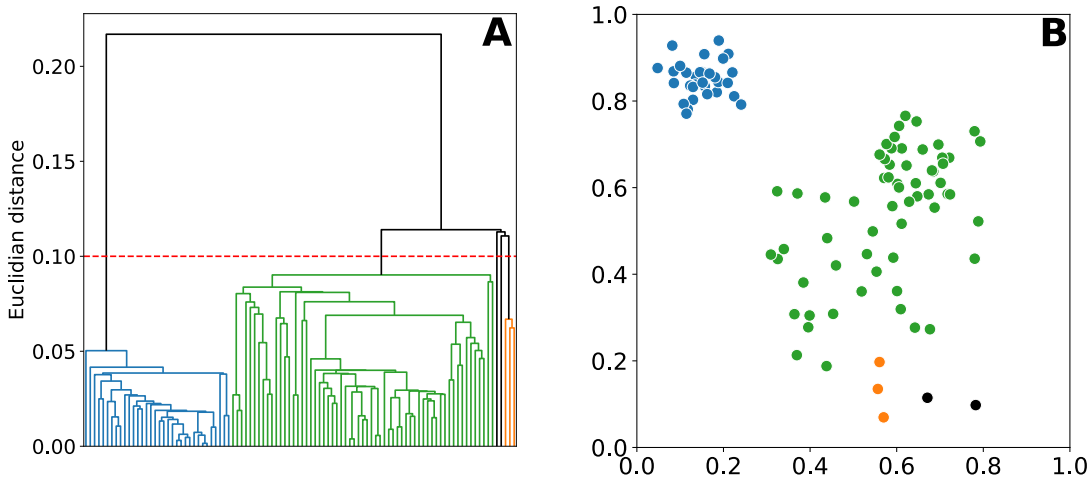

Figure 4: Hierarchical-based clustering using the single-linkage method. The distance between the objects is calculated as Euclidean distance. Clusters are created with a threshold of 0.1 (red dashed line). **(A)** Representation of the hierarchy as a dendrogram with Euclidean distance. **(B)** The resulting clusters. Black dots represent single unclustered objects.

In order to compare the similarity of clusters, the linkage criterion has to be chosen, such as single-linkage clustering (closest objects of two clusters), complete-linkage clustering (farthest away objects) or average-linkage clustering [28]. This is necessary, since a cluster may contain more than one object, and there has to be one candidate per cluster to calculate the distance from. Rather than providing one solution for the clustering problem, this method can provide different clusters based on the chosen distance.

Centroid-based clustering (Partitioning-based clustering)

Generally speaking, this method sets initial cluster centers (centroids) and shifts them iteratively until no further change is observed. One of the most well-known algorithms in this category is the k-means algorithm. The centroids are selected randomly and all other objects are assigned to the closest centroid sequence (Figure 5A). The cluster centers are then updated to be the mean of all objects in the cluster. These two steps are repeated until the clusters don't change anymore (Figure 5B) [29]. Usually, the Euclidean, Minkowski or Manhattan metrics are used for distance computation. During the shifting process, objects also may change their clusters. This method suffers from the disadvantage that it requires the number of clusters in advance and is also sensitive to the initial centroid [30].

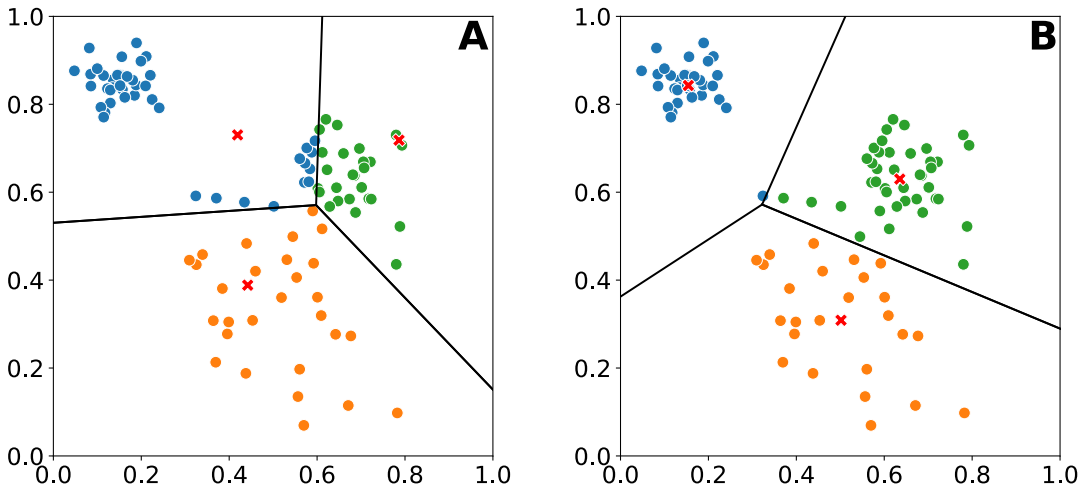

Figure 5: Centroid-based clustering using the k-means algorithm. The number of clusters is set to three. **(A)** The centroids (red cross) are initialized randomly. **(B)** After five iterations the shifting converges and builds the final clusters.

Distribution-based clustering

For this method, a probability model is used to assign the objects to clusters. Those probabilistic models usually have some parameters, which initially have to be guessed to represent the data (Figure 6A). Afterwards, the cluster probabilities can be calculated for each object, allowing for a re-estimation of the probability parameters. This is performed iteratively until the parameters converge (Figure 6B).

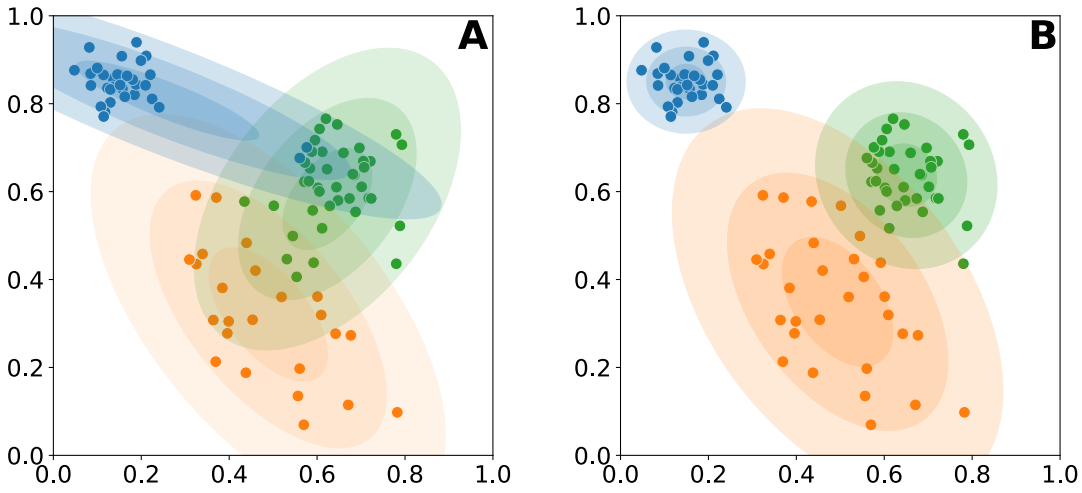

Figure 6: Distribution-based clustering using a Gaussian mixture model. The distribution parameters are initialized randomly and the number of components is set to three. **(A)** Random initialization of parameters of the distribution. **(B)** Convergence after 17 iterations forming the final clusters.

Each object is then assigned to its respective cluster based on the probability it belongs to a certain distribution in the underlying data. The biggest drawback of this method is that it is computationally intensive and may be over-fitting the data [31]. Furthermore, the assumption of the distribution has to fit the data, which is not likely known a priori [32]. One representative for this method are Gaussian mixture models using the expectation-maximization algorithm (Figure 6) [33].

Density-based clustering

Other than the distribution-based clustering, the density-based is a non-parametric approach. The reachability between objects plays a major role in this method (Figure 7). Given an initial core object of the data and a radius r, there is a certain number of other objects that are within this radius. Those objects are also called directly density-reachable.

Only if there are more than n neighbors within this radius, the object is considered a high-density area and is extended as follows. For each of the directly density-reachable neighbors, the same radius is used to identify new neighbor density-reachable objects. The new object is only considered as a part of the cluster, if it has again at least n directly density-reachable objects. It may be the case that new objects don't lie within the direct density-reachability of the core object. However, they can be reached through a neighbor by neighbor connection, which is called a density-reachable connection.

The search for new objects is repeated until no new object fulfills the requirement of minimum objects within the radius. Finally, a cluster is defined by all groups of density-reachable objects of the data [35]. Objects that are not part of density areas, thus have not enough directly density-reachable objects, are considered noise. An advantage of this method is that the number of clusters doesn't have to be specified and the clusters may be arbitrarily shaped in the dimension space [36]. Two of the most common algorithms implementing this approach are the `DBSCAN` [34] and the mean-shift algorithm [37].

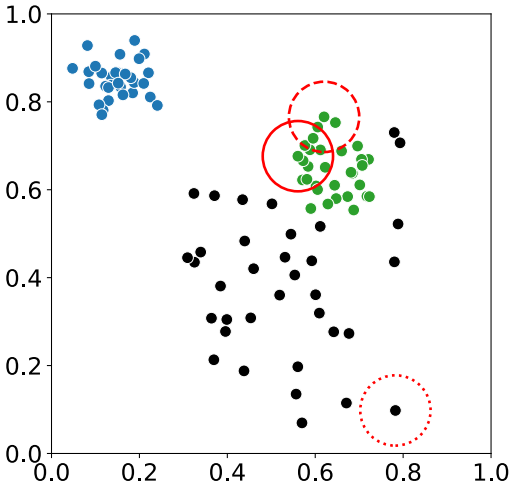

Figure 7: Density-based clustering using `DBSCAN` [34]. The radius is set to 0.07 and the number of samples in a neighborhood set to ten. A core object (red circle) has at least ten other object within its radius. Density-reachable objects (red dashed circle) have fewer objects in their circle, but are in a core objects reachability. Objects that do not have either of these properties are referred to as noise (red dotted circle). The result of the clustering are two clusters (blue and green) and a lot of unclustered objects (black points).

Additionally, one can differentiate between hard and soft clustering. Hard clustering describes that each object has a cluster it belongs to or not, e.g. centroid-based clustering. In soft clustering, also called fuzzy clustering, each object can belong to more than one cluster, e.g., distribution-based clustering. Each element is a member of a cluster with a certain probability [38].

1.5 Clustering in bioinformatics

The problem of clustering biological sequences is probably as old as bioinformatics itself. For example, already in the year 2007, it was aspired to remove redundant sequences from the Uniprot database to speed up the search of sequences in their database [39]. However, with the rise of Next-Generation Sequencing, it becomes more and more relevant to handle the massively produced data.

Over the last years, there have been various attempts for clustering biological sequences, such as `cd-hit-est` [40, 41], `UCLUST` [42], `Linclust` [43] or `MeShClust` [44]. Based on their application and the input data, all of these tools have their advantages and disadvantages, however their performance on clustering

(*de novo*) transcriptome assemblies remains particularly unknown. As mentioned above, the similarity of objects has to be compared to each other. For biological sequences, current clustering tools use different methods to calculate the sequence similarity, thus showing some kind of relationship between them. `cd-hit-est`, for example, compares sequences by calculating sequence alignments [41]. Another approach is the extraction of k-mer profiles as properties for sequences and performing the clustering based on this information, which was recently performed for metagenomic binning [45]. In comparison, `MeShClust` calculates similarities using alignment-free k-mer statistics such as length difference, Czekanowski similarity or Manhattan distance [44]. Unlike the general purpose of clustering sequences, recent research also focuses on clustering *de novo* transcriptome assemblies, such as `Grouper` [46] and the `Oyster River Protocol`. As mentioned above, the `Oyster River Protocol` uses multiple transcriptome assemblies to enhance the resulting assembly regarding the completeness. Consequently, the tool implements a custom clustering algorithm. Contrarily, `Grouper` presents a method to improve a single transcriptome assembly utilizing the reads from which the assembly was created.

Xu and Tian compare different clustering algorithms, but without biological context [47]. Zou *et al.* compares five different tools for the task of clustering a metagenomic sequencing to identify operational taxonomic units [48]. Despite all of the various clustering methods and the general interest for (*de novo*) transcriptome assemblies, a comprehensive study on clustering methods for *de novo* transcriptome assemblies is missing. The topic of clustering remains a current problem in bioinformatics, especially for metagenomics and (*de novo*) transcriptome assemblies.

In this master's thesis, multiple assemblers are used to improve the completeness of the final assembly. A new algorithm for sequence clustering is implemented to remove the redundancy from the multiple assemblers. To overcome the clustering problem, recent k-mer and read-graph-based methods are combined and enhanced. The proposed method also offers the possibility to extract representative sequences for each cluster. Furthermore, the method is compared to state-of-the-art clustering tools and evaluated in terms of completeness and correctness of the resulting assembly.

2 Material and Methods

All parameters are used as described. If not mentioned otherwise, the corresponding version's default parameters were used.

2.1 Datasets

Reads

The different clustering methods were tested on short-read Illumina datasets for *Escherichia coli* and *Caenorhabditis elegans*. Four datasets with single- and paired-end library layouts were obtained from the European Nucleotide Archive while another one artificially created (Table 1).

Table 1: Datasets used in this master's thesis. All analyses performed are based on these datasets. Paired-end data: PE. Single-end data: SE.

Study accession	**Run accession**	**Sequencer**	**Species**	**Library layout**	**Number of reads (million)**	**Average read length**
PRJNA383382	SRR5456164	Illumina HiSeq 4000	*C. elegans*	PE	29.1	100
PRJEB29657	ERR2886543	Illumina HiSeq 2000	*C. elegans*	SE	22.5	51
PRJEB8751	ERR779269	Illumina HiSeq 2500	*E. coli*	PE	16.4	100
PRJNA343424	SRR4255368	Illumina HiSeq 2000	*E. coli*	SE	12.1	49
-	CEL_FLUX	Simulated dataset	*C. elegans*	PE	1.0	100

Reference datasets

To evaluate the datasets with the programs from Section 2.5 the corresponding reference files for *E. coli* (ASM584, v2; from NCBI) and *C. elegans* (WBCel235, v96; from Ensembl) were used.

Simulated dataset

The artificial dataset was created using the protein-coding and non-coding information of the first chromosome of *C. elegans*. The distribution between the different types of RNA is listed in Table 2. A total of 1,000,000 paired-end reads with a length of 100 bases were created using `Flux Simulator` (Version 1.2.1) [49]. `Flux Simulator` is a tool that simulates the technical processes, such as reverse transcription, fragmentation, adapter ligation, PCR amplification as well as the sequencing to create FASTQ files. For each step a model based on real sequencing data is used, ensuring that the artificial reads have the same features and problems as real reads. The complete parameters and models used can be found in the supplemental data (See Section 9.3). Note that the miRNAs include pre-miRNAs, which are represented as isoforms in Table 2. Furthermore, the existing isoforms of miRNAs, ncRNAs and antisense RNAs usually only differ a few nucleotides, but are listed as single isoforms in the annotation files.

Table 2: Input information for `Flux Simulator`. Distribution of all genes of the first chromosome of *C. elegans*.

	Genes	**Isoforms**
total	4046	6255
protein-coding	2908	5055
ncRNA	809	811
antisense RNA	14	15
lincRNA	23	23
rRNA	5	5
miRNA	33	92
snoRNA	74	74
piRNA	98	98
tRNA	66	66
snRNA	16	16

2.2 *De novo* transcriptome assemblies

The tool `fastp` (Version 0.20) was used to take care of all necessary preprocessing steps [50]. `fastp` is an ultra-fast all-in-one FASTQ preprocessor. The default settings were used to perform a quality filter (Phred score $>$ 15), length filter (Reads $>$ 15 bp) and adapter trimming (automatic detection). The parameter `--low_complexity_filter` (30 %) was enabled to remove low complexity, defined as the percentage of bases that differ from the following base. For paired-end reads, the option `-c` was used for base correction. Additionally, the parameter `--detect_adapter_for_pe` was enabled to automatically detect adapter sequences for paired-end data.

Three different programs were used to perform the *de novo* transcriptome assemblies. The assemblers `rnaSPAdes` [19], `SOAPdenovo-Trans` [16] and `Trinity` [17] were chosen, because they performed comparatively well in a recent studies [20].

`rnaSPAdes` (Version 3.13.1) was used with default settings. By default, the tool incorporates two different k-mer sizes into the assembly process. The k-mer size is based on the average read length. The upper k-mer size is approximately ½ of the read length and the lower approximately ⅓ of the read length.

Since `SOAPdenovo-Trans` (Version 1.04) doesn't perform multiple k-mer assemblies by default, the tool was started with two different k-mers. The k-mer sizes are roughly ½ and ⅔ of the input read length. Finally, the two assemblies were merged using `cd-hit-est` (Version 4.8.1) [40, 41] with a sequence identity of 100 % creating the final `SOAPdenovo-Trans` assembly.

`Trinity` (Version 2.8.4) was also used with default settings. The tool is subdivided into three parts: Inchworm, Chrysalis and Butterfly. Utilizing greedy k-mer based assembly (k = 25), Inchworm produces unique sequences of transcripts, representing a set of alternative variants with shared k-mers. Chrysalis then forms clusters from those contigs that are overlapping and builds a separate De Bruijn graph for each cluster. Subsequently, Butterfly traces the routes that reads can take through the graph and reports all trustworthy transcript sequences, representing alternatively splices isoforms.

The three final assemblies from `rnaSPAdes`, `SOAPdenovo-Trans` and `Trinity`, were then concatenated into one file and served as a basis for the following clustering methods. All steps mentioned in this section were combined in one pipeline using `Snakemake` [76] and can be found on GitHub (See Section 9.1).

2.3 Clustering tools

All clustering methods were combined in a `Snakemake` pipeline as well, which are accessible via GitHub (See Section 9.1).

cd-hit-est

`cd-hit-est` (Version 4.8.1) [40, 41] performs clustering based on sequence similarity. The program uses a short word filter to reduce the number of alignments that have to be calculated for sequence similarity. In brief, this relies on the idea that two sequences with a certain similarity percentage also share certain counts of

different k-mers. Therefore, if those conditions are not satisfied, no alignment has to be computed [51].

The similarity of clustered sequences can be set with the parameter `-c`. Two different runs with two settings of sequence similarity were performed. One approach was quite strict with a sequence similarity of 100 % (c = 1) while the other one was less stringent with a similarity of 90 % (c = 0.9).

Oyster River Protocol

The `Oyster River Protocol` (Version 2.2.6) [24] is a pipeline that also has the aim of combining the strengths of different assemblers to generate a better overall assembly.

The `Oyster River Protocol` removes Illumina sequencing adapters and trims the reads with `Trimmomatic` [52]. Subsequently, they are error corrected using `RCorrector` [53]. MacManes is using three assemblers [24]. `Trinity` was used with default settings (k = 25) and without read normalization. For `rnaSPAdes`, two different k-mer sizes were used (k = 55/75) in two distinct runs. Other than described in their paper and also mentioned on their website, the assembler `Shannon` [54] was replaced by `Trans-ABySS` [18] and used with a k-mer size of 32. Those four assemblies are then merged using the following description.

The merging process is performed using `OrthoFuse`. First, all transcripts are concatenated and then sorted into groups of homologous transcripts using a modified version of `OrthoFinder` [55], which accepts nucleotide sequences. The protocol then makes use of `TransRates` [56] contig scores. For each homologous group, only the best contig (highest contig score) is accepted for the final assembly. Because the `Oyster River Protocol` uses `TransRate` for their merging process, the pipeline is restricted to paired-end data only. The contig score from `TransRate` is, among other things, dependent on information coming from read pairs, making its calculation only possible for paired-end data.

We used the paired-end reads preprocessed with `fastp` (Section 2.2) as input for this pipeline in all of the paired-end datasets. Since the `fastp` preprocessing was chosen to be not very restrictive, only a few reads get filtered out, similar to the quality filtering directly performed by the `Oyster River Protocol`.

Linclust

The algorithm implemented in `Linclust` [43] can reduce the number of pairwise sequence comparisons drastically. Sequences with similar k-mers are grouped and

the longest sequence is set as the center of the group. Each sequence is then compared only with each representative sequence with which it shares a k-mer, and not with all other sequences that it shares a k-mer. For this reason, the runtime scales linear with the size of the input set of sequences and is independent of the number of clusters. The sequence identity scores are then formed into a graph and trimmed with an identity threshold. The representative sequence is selected via greedy incremental clustering, which will almost always yield the longest sequence as a result.

Originally implemented only for protein sequences, an option for nucleotide sequences was added later. `Linclust` (Version 10.6d92c) was used with default parameters according to the manual available in the corresponding GitHub repository.

Grouper

`Grouper` [46] is a program that is specifically designed to cluster *de novo* transcriptome assemblies for further downstream analyses. The tool is based on the equivalence classes provided by `Salmon` [57], which gives information about how many reads are shared between contigs. With this information, a weighted undirected graph is constructed, representing contigs as nodes. The weight of the edges is based on the number of shared reads between contigs compared to the amount of individually mapped reads. The graph is then clustered using `MCL` [58].

If provided with a reference genome of a related species, `Grouper` will also use annotations to improve the clustering. Since, in this master's thesis, no clustering method used information from related species, this option was not used to allow a fair comparison. For the quasi-mapping with `Salmon`, duplicate transcripts were kept while indexing using the parameter `--keepDuplicates`. The mappings were validated with `--validateMappings` and orphan reads were allowed though `--allowOrphans`. To create the equivalence class the option `--dumpEq` was passed. `Grouper` (Version 0.1.1) was then started two different times with the parameters `orphan` and `mincut` set both to true or false. If those flags are enabled, Malik *et al.* state that `Grouper` may produce better results in case of a lower quality assembly [46].

Despite clustering, the tool does not select a representative sequence per cluster. To overcome this limitation, the longest sequence was chosen as representative, similar to `cd-hit-est`.

MeShClust

Most of the tools available for clustering nowadays are working with greedy algorithms, e.g. `cd-hit-est`. For those tools, the user must provide a sequence similarity score to define how similar sequences should be to be grouped into one cluster. James *et al.* describe `MeShClust` as an intelligent tool for clustering DNA, which is independent of the user input [44]. It utilizes an unsupervised machine learning algorithm, respectively, called the mean shift algorithm. In brief, the input sequences are sorted based on their length. The shortest sequence is then picked as a center sequence. Based on this sequence, further similar sequences are detected. Similar sequences are selected using a General Linear Model (GLM) that was trained on the original data using alignment-free measures such as sequence length difference, Pearson coefficient, etc. After that, the mean shift is applied and the sequence closest to the mean is set as the new center sequence. This process is repeated until no new sequences join the cluster. In the next iteration, the shortest sequence is taken again as a new cluster start and the process is repeated until no sequences are left.

`MeShClust` (Version 1.2.0) was used with default parameters and the representative sequences were extracted from the resulting cluster file.

MeShClust^2

`MeShClust`^{2} [59] is the successor of `MeShClust`. As James *et al.* state in their paper, `MeShClust` is optimized for sequences with sequence identity above 60 % and is not scalable to longer sequences [44]. To solve these problems `MeShClust`^{2} was developed. `MeShClust`^{2} (Version 2.3.0) was also used with default parameters.

karma

`karma` (**K**-mer **A**nd quasi-**R**ead-**MA**pping based clustering) is the working name of the program developed during this master's thesis. It combines a k-mer based clustering with a read-graph-based method, similar to `Grouper`. The general workflow of `karma` is shown in Figure 8.

As seen in Figure 8A1, the pipeline starts with a k-mer based clustering. A k-mer profile for each sequence is generated resulting in a high dimensional space matrix. The k-mer frequency was then normalized to the sequence length. Initially, the k-mer size was adapted from Lin and Liao [45] that performed a successful metagenomic binning. In their study, the authors tested different k-mer sizes along with a 5p6-mer, which is a combination of 5-mers and palindromic 6-mers. The results using

5p6-mers are comparatively superior if more than 50 clusters are generated [45]. Considering the target of this pipeline, where each cluster will represent a gene with its isoforms, the threshold of 50 clusters will be exceeded. For example, *E. coli* has more than 3000 genes that are expressed during exponential growth, not accounting for non-protein-coding transcripts [60].

The huge dimensional matrix contains a lot of information, not all of which is needed to perform successful clustering. For this reason the information is usually reduced to features that are relevant throughout all sequences. The Barnes-Hut implementation of `t-SNE` from Lin and Liao [45], until recently commonly used to cluster single cell RNA-Seq data [61], was replaced by 'Uniform Manifold Approximation and Projection for Dimension Reduction' (`UMAP`) [62].

UMAP

Just like `t-SNE`, `UMAP` is a k-neighbor graph-based algorithm, which can be divided into two steps for a quick summary of the algorithm. The first step is the graph construction. For each input data point the set of the k-nearest neighbors are calculated utilizing the nearest neighbor descent algorithm [63], resulting in fuzzy simplicial sets. Afterwards, the weights between all members of those sets are calculated using a weighting function so that the adjacency matrix is symmetric. The result is an undirected weighted graph. In the second step, the actual dimension reduction is performed using a force directed graph layout algorithm, which originates from a more general graph visualization problem. For the initialization of the dimension reduction a spectral layout can be used. Rather than using a random initialization like `t-SNE`, this allows for a faster convergence and better stability within the algorithm. The embedding is then optimized through minimizing the fuzzy set cross entropy [62].

`UMAP` was chosen over `t-SNE`, because it preserves local and global structures better than `t-SNE` and has no restrictions of embedding dimensions [64, 62]. `UMAP` was used in version 0.3.9 and the main parameters were set as follows. The parameter `n_neighbors` controls how `UMAP` balances local versus global structure in the data. It describes how many neighbors should be considered when searching for defined structures. Low values mean that local structures are preferred. In contrast, larger values mean the loss of detailed structures for a broader global view of the data. Here, a value of 2 was chosen. `n_components` determines the dimensionality of the reduced dimensional space in which the data is embedded. The default value was set to 10. `dist` controls how tightly points may be packed together. Since it is possible for two sequences to be identical, a value of 0 is selected. The parameter `metric`

controls how the distance in the space of the input data is calculated. Currently, only the Euclidean distance calculation is used. However, UMAP offers a variety of other distances to use such as Manhattan, Minkowski or Canberra distance. It is also possible to implement custom distance calculations.

Next, the UMAP-reduced k-mer profiles from the contigs were clustered using the tool 'Hierarchical Density-Based Spatial Clustering of Applications with Noise' (HDBSCAN, Version 0.8.22) [65]. Recent viral metagenomics research has also used the combination of UMAP and HDBSCAN to directly recover high-quality viral genomes from environmental samples [66].

HDBSCAN

HDBSCAN converts the original DBSCAN [34] into a hierarchical clustering algorithm, rather than just a density-based algorithm. The main assumption is that real-world data is messy and those outliers need to be separated from the actual data of interest. For this reason, a transformation of the data space is performed, utilizing a k-nearest neighbor approach to measure the core distance for each point. The core distance is defined by the furthest neighbor out of k neighbors and gives an idea of the data density. The data points are then shifted according to the density. High-density points (low core distance) stay near to their original position. However, low-density points (high core distance) are spread to be at least the core distance away from any other point. As a result, the noisy data points spread out, while original clusters nearly stay the same. From this dimensional representation, an undirected weighted graph is constructed, where the weight represents the mutual reachability distance, which is a distance based on the core distance. For this graph, the minimum spanning tree is calculated using Prim's algorithm. Subsequently, a hierarchy of connected components is created using the minimum spanning tree. To identify which edges join two clusters, union-find is used. Afterward, the resulting cluster hierarchy is condensed to find actual clusters. For this purpose, a walk through the split-points of the hierarchy tree is performed. New clusters are only formed if a split-point in the tree creates two clusters that are greater than the user-provided `min_cluster_size` parameter. Otherwise, the walk through the tree continues. At last, the clusters are extracted from the condensed tree. A stability value is calculated for each cluster to identify the true clusters. In order to compare the stability of different clusters, a reverse walk through the tree is performed. For each sub-branch of the tree, the child stabilities are compared to their parent stability. If the parent has a larger stability value, the new stability value is set to the sum of the children's stability. The parent cluster becomes the child and the

stability is compared to its next parent. However, if the stabilities of the children are greater than the stability of the parent, those clusters are selected.

The primary parameter `min_cluster_size` sets the minimum group size that is considered a cluster. Since it is technically possible, that there is only one contig in a single cluster, it would be reasonable to set this option to 1. However, in this case the underlying algorithm may produce a bias towards the root cluster. Therefore, the default value chosen was 2.

This procedure then results in k-mer based clusters, as well as a large group that is called unlabeled. These unlabeled contigs can't be clustered by the previous named algorithms. The k-mer based information is not sufficient to assign the contigs to a specific cluster.

The existing clusters were then complemented with coverage information derived from the reads. To obtain this information `Salmon` (Version 0.14.1) [57] was chosen mainly because of its speed in calculation. The high speed of calculation is a result of the quasi-mapping, which gives a rough overview about the quantity of reads shared by all contigs. This information is stored in an equivalence class file and used in this implementation. In order to perform a quasi-mapping with `Salmon` an index has to be created for the transcripts. The index for the quasi-mapping was built with the parameter `--keepDuplicates`. By default `Salmon` discards sequence-identical duplicate transcripts. The parameter disables this behavior and treats duplicate transcripts as individual sequences. `Salmon` uses the previous built index to perform the fast quasi-mapping. `--libType A` was set to allow `Salmon` to automatically detect the orientation and strandedness of the reads. `--validateMappings` enables a more sensitive and accurate quasi-mapping algorithm resulting in more accurate results. `--dumpEq` will cause `Salmon` to write an equivalence class file with the shared read counts between contigs that were computed during quasi-mapping.

The k-mer based clusters and the coverage information is then used to create a weighted undirected graph $G = (V, E, \omega)$ for each of these clusters. These graphs consist of vertices V (contigs) and the edges E between them store the information from the `Salmon` equivalence class file. The read-graph is based on the program `Grouper` [46]. However, the weighting function ω for the edge weight calculation from `Grouper` (Equation 1) was replaced with a custom weighting function (Equation 2). Both equations describe the weight between two different contigs i and j, where N is the number of assigned reads to the contig. So N_i/N_j describes the amount of reads assigned to contig i/j and $N_{i,j}$ the amount of shared reads between i and j.

$$\omega_{ij} = \frac{N_{ij}}{\min\left(N_i, N_j\right)} \tag{1}$$

$$\omega_{ij} = \frac{\frac{N_{ij}}{N_i} + \frac{N_{ij}}{N_j}}{2} \tag{2}$$

In short, Equation 2 is chosen over the original weighting function, because it puts more value towards both individually assigned reads, resulting in more evenly distributed weights.

Table 3 compares the different results from the two weighting functions for variations of N_i, N_j, N_{ij} that may occur between two contigs, to distinguish the different results for both equations. For example, Equation 2 yields the maximum best value of 1 only if all reads of contig i are shared with contig j (Table 3, row 1). By contrast, it is sufficient for Equation 1 that one contig shares all of its reads with the other contig, to yield the best score (Table 3, row 1-4). Overall, the weight ω from Equation 2 is lower than from Equation 1 and has a lower minimum score (Table 3, row 6).

Table 3: Comparison of the weighting function from `karma` and `Grouper`. The table shows the different results for the calculation of weights.

Row	**Combinations**			**Weight** (ω)	
	N_i	N_j	N_{ij}	`karma`	`Grouper`
1	100	100	100	1	1
2	100	80	80	0.9	1
3	100	70	70	0.85	1
4	100	50	50	0.75	1
5	100	50	25	0.375	0.5
6	100	50	1	0.015	0.02

With this additional information based on read quasi-mappings the next step in the pipeline assigns contigs from the unlabeled k-mer clustering to their correct cluster, according to the coverage information.

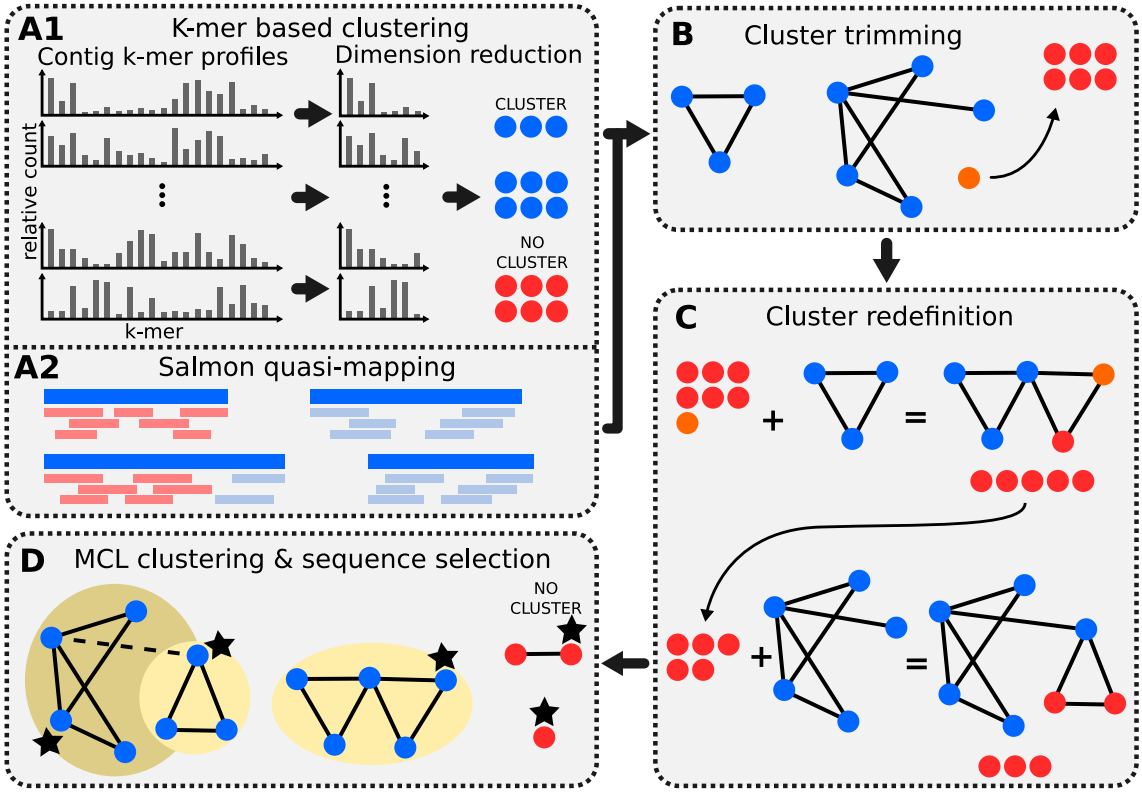

Figure 8: Workflow scheme for `karma`. **(A)** A pre-clustering step using k-mers is done by `UMAP` and `HDBSCAN`. The existing clusters are then supplemented with the information from the equivalence classes from `Salmon`. **(B)** Cluster trimming. Each contig that doesn't share contigs with all other contigs in a k-mer based cluster is moved to a 'unclustered group'(red). **(C)** Cluster redefinition. Contigs from the 'unclustered group' are added to each cluster, if they share reads with the specific cluster. **(D)** `MCL` clustering & sequence selection. The resulting groups are then clustered using `MCL`. Each contig has a score equal to the sum of the weights of the edges to which it is connected. For each subgroup the contig with the highest score is chosen as representative.

After the graph construction, cluster trimming is performed (Figure 8B). All single contigs are removed that don't share reads with all other contigs in one cluster. Those contigs are then added to the unlabeled group. Subsequently, the clusters are redefined using the unlabeled cluster (Figure 8C). The unlabeled group is added to each existing k-mer based cluster. If a read has a connection with one of the contigs in the original group it stays this group. Similar to step one, all reads that have no connection are removed. This process is then repeated for each k-mer based cluster until all clusters have been paired with and separated from the unlabeled group. In the best case, the resulting clusters should then represent one gene with different isoforms. To extract those isoforms, the Markov Cluster Algorithm (`MCL`, Version 14.137) [58] is used to determine subclusters in the corrected k-mer based clusters (Figure 8D).

`MCL` is a graph clustering algorithm based on the simulation of random walks,

which are calculated through Markov chains. The core idea is, that the simulation of a random flow through a graph reveals natural clusters. The weights of the edges reflect the strength of the flow between two vertices. Respectively, a random walk starting in a dense cluster (high weights, many connections) will most likely stay in that cluster, thus gives information about the structure of the graph.

The main parameter for this is the inflation value `-I`. This value is the main handle for affecting cluster granularity. A bigger value will result in finer clusterings. The allowed values range from 1.2 to 5, while the default value is set to 2. For each subcluster a representative sequence has to be selected. The sum of the edge weights is calculated for every node. For each subcluster the contig with the highest overall weight is chosen as representative sequence. This means that one cluster may have multiple representative sequences, respectively isoforms.

Table 4: Combination of parameters used for the parameter evaluation. All combinations (432) of the listed parameters were tested.

Parameter	**Values**					
k-mers	3	5p6	7			
n_neighbors	2	5	10	20		
n_components	2	5	10	20	30	40
min_cluster_size	1	2	5	10	15	20

The source code of `karma` is freely available on GitHub and can also be easily installed via Anaconda Cloud (See Section 9.2)

Since the pipeline has a lot of parameters that can be tuned, a parameter evaluation was done for the following combinations of parameters with the simulated dataset (Table 4).

2.4 Runtime analysis

The time was measured for the clustering tools on a Linux-based system (Debian 9.0) with 40 cores (AMD Opteron Processor 6238, 2.6 GHz) and 500 GB of available RAM.

2.5 Evaluation of assemblies

The metrics to compare the quality of assemblies are calculated with the following tools.

BLAST

This tool was only used for evaluation of the simulated dataset. A BLAST (Version 2.9.0) [67] search was performed to identify which gene and isoform a certain contig from the *de novo* assemblies corresponds to.

A custom database was created from the original protein and non-protein coding sequences of the first chromosome of *C. elegans*. The assembly was then blasted against the database and the hit with the highest sequence similarity and lowest e-value was chosen as the *true* gene/isoform. With the information from the blast results, the cluster file was analyzed for intra and inter cluster similarity.

The purity of all clusters was measured using the intra cluster similarity, where C denotes the total amount of clusters. For each cluster $i \in C$, the most common gene $N_{i,com}$ was selected to be the 'true' gene and was compared to the total cluster size S_i. By calculating the mean value, we get the overall intra cluster similarity as shown in equation 3. The best value is a 1 and a lower score means a lower intra cluster similarity.

$$\frac{\sum_{i=1}^{C} \frac{N_{i,com}}{S_i}}{C} \tag{3}$$

The outer cluster similarity describes the proportion of dispersal of all genes G between all clusters (Equation 4). For every gene $i \in G$ the amount of clusters in which gene i is present, is denoted as C_i. In theory, each gene should be exactly in one cluster, leading to an optimal score of 1. Lower scores mean higher distributed genes among clusters.

$$\frac{G}{\sum_{i=1}^{G} C_i} \tag{4}$$

DETONATE

DE novo TranscriptOme rNa-seq Assembly with or without the Truth Evaluation (DETONATE, Version 1.11) [68] is an evaluation tool that consists of the two packages RSEM-EVAL and REF-EVAL. RSEM-EVAL produces an evaluation score that is based only on an assembly and the set of reads from which it was constructed. For example, the support of the assembly through the reads and the compactness of the assembly are used among other factors for the calculation of this score. When a reference transcript set is available, REF-EVAL may be used to compute a num-

ber of reference-based measures. As for this evaluation the following metrics were chosen: unweighted contig/nucleotide F1 scores and the k-mer compression score, according to Hölzer and Marz [20]. `DETONATE` was used as proposed on their online vignette.

rnaQUAST

`rnaQUAST` (Version 1.5.1) [69] provides a wide variety of basic statistics like N50 or length distribution, but also reference-based metrics such as the number of genes, isoforms or duplication ratio. For the calculation of the latter `rnaQUAST` needs the reference genome and the corresponding annotation.

TransRate

`TransRate` [56] is specifically designed to evaluate *de novo* transcriptome assemblies without a reference to detect common artifacts such as chimeras, structural errors, an incomplete assembly and base errors. The tool calculates individual scores for each contig from which the entire assembly is then evaluated. These individual scores can not only be used to evaluate assemblies, but also for their optimization. `TransRate` only accepts paired-end reads, which is the reason it is not used to evaluate single-end assemblies. Since Smith-Unna reported bugs affecting the calculated metrics, `TransRate` was used in version 1.0.1 rather than 1.0.3[1].

HISAT2

`HISAT2` (Version 2.1.0) [70] was used to map the reads back to contigs of the assembly to calculate the remapping rate.The percentage of mapped reads is then divided by the total number of bases, setting it into relation to the size of the assembly.

BUSCO

Benchmarking Universal Single-Copy Orthologs (`BUSCO`, Version 3.0.2) [71, 72] evaluates a given assembly in terms of completeness of gene content. The necessary databases were obtained from their website and are also found in the supplemental data.

[1] https://gitter.im/blahah/transrate (Accessed 2019-09-18); Quote from the author: 'Yes - there's a bug in 1.0.3 - trust the scores from 1.0.1.'

Trinity and Salmon

The N50 statistics is one traditional way of measuring assemblies in terms of contiguity. It describes the length of the shortest contig, so that the summed up nucleotides of all contigs equal or longer than that size cover 50 % of the total nucleotides of the assembly. Thus, the N50 statistic can be used as a measurement for assembly contiguity and fragmentation and is frequently used in genomics. The Ex90N50 value is the same metric but only calculated on a subset of the assembly. However, by taking into account the expression of transcripts. The subset for the value contains the top 90 % of the highest expressed transcripts from the total normalized expression data. Thus, low expressed and short transcripts that might dominate a transcriptome assembly do not affect the N50 value that much. The necessary transcript abundance estimation for this metric was calculated using `Salmon` (Version 0.14.1) [57]. The Ex90N50 values were calculated using scripts that are included in `Trinity` (Version 2.8.4) [17].

The evaluation tools except `BLAST` were also combined into a `Snakemake` pipeline and can be accessed via GitHub (See Section 9.1).

2.6 Sequence alignments

All alignments in this thesis were calculated with `MUSCLE` (Verion 3.8.31) [73].

2.7 Visualization

Venn diagrams were created with the `Intervene` web service [74]. The alignments from `MUSCLE` were visualized with `Jalview` (Version 2.11.0) [75].

2.8 Workflow management

Snakemake

`Snakemake` is a workflow manager using python as the underlying programming language. It was created to increase reproducibility and decrease multiple calculations based on certain rules. The workflows above were created using `Snakemake` in version 5.5.0 [76].

3 Results

The following figures and tables in this section use some aliases, which are explained as follows. 'Combined' describes a concatenation of all three assemblies from `rnaSPAdes`, `SOAPdenovo-Trans` and `Trinity`. The percentage of the sequence identity threshold is added as a number behind `cd-hit-est`. `Grouper` has both parameters **`orphan`** and **`mincut`** enabled, whereas `Grouper*` has them disabled. `karma` uses the default parameters described in the Methods Section 2.3 and `karma*` describes the run with the optimized parameters from Section 3.2.

3.1 Different assemblers yield distinct genes and isoforms

For the simulated paired-end dataset, the assemblies from `rnaSPAdes`, `Trinity` and `SOAPdenovo-Trans` were annotated via a `BLAST` search (See Section 2.5). In total, the three assemblers produce of 3,235 genes and 3,925 isoforms together (Figure 9). However, 382 genes (16 %) and 747 isoforms (32 %) are produced solely by individual assemblers. The assembly from `rnaSPAdes` contains the most individual genes (270), whereas `Trinity` (42) and `SOAPdenovo-Trans` (70) contain much less (Figure 9A). Likewise, `rnaSPAdes` produces the most individual isoforms (327), while `SOAPdenovo-Trans` (277) and `Trinity` (143) produce at least 50 isoforms less (Figure 9B).

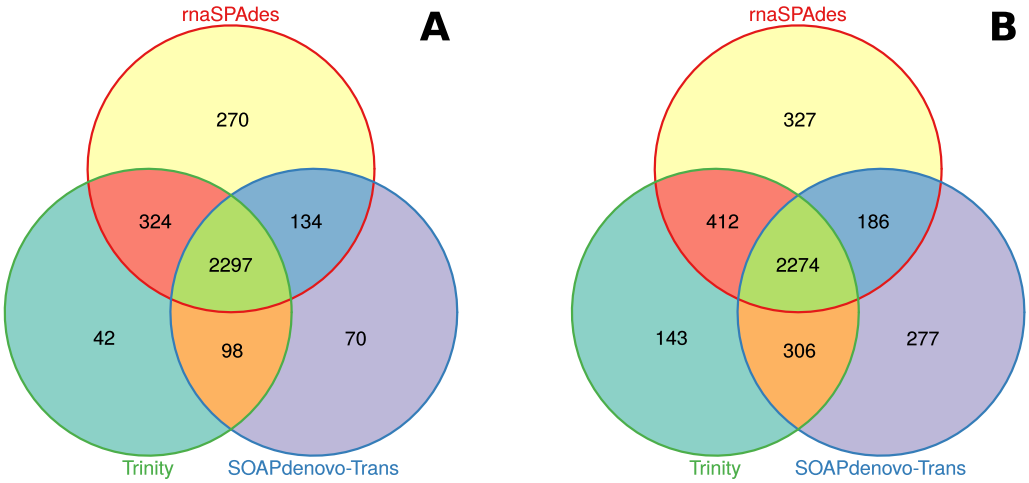

Figure 9: Intersection of annotated transcripts from the different assemblies. The transcripts were produced by `rnaSPAdes`, `SOAPdenovo-Trans` and `Trinity` for the simulated paired-end *C. elegans* dataset. Visualization created with `Intervene` [74]. **(A)** Intersection of distinct unique genes. **(B)** Intersection of distinct unique isoforms.

3.2 Smaller k-mer sizes are beneficial for karma clustering

Besides the default selection of parameters (formated italics), Table 5 lists all parameter combinations that yield the best results for the selected evaluation metrics. HDBSCAN creates a certain *number of groups* as well as *unlabeled* sequences with no assigned groups. Furthermore, the algorithm calculates a probability for each contig that the specific certainty of it belongs to a cluster. The *mean probability* combines all those scores and gives an impression of the clustering's reliability. The inner and outer cluster homogeneity are described in Section 2.5. Additionally, the percentages of genes/isoforms give an impression of the proportion of recovered genes from the original dataset.

Table 5: A selection of the best results for different parameters for karma. The best results are highlighted in bold. The initial parameter combination is listed in italics.

Parameters				**Results**						
k-mer size	**n_neighbors**	**n_components**	**min cluster size**	**Unlabeled sequences**	**No. of groups**	**Mean probability**	**Inner cluster homogeneity**	**Outer cluster homogeneity**	**Genes** [%]	**Isoforms** [%]
3	2	30	2	2,324	**2,846**	0.6752	0.9535	0.5240	41.28	30.06
3	2	40	20	8,043	181	0.4118	0.9962	0.3432	62.18	**47.69**
3	10	10	20	**6**	4	0.9753	0.9989	0.6061	62.90	44.92
5p6	*2*	*10*	*2*	*2688*	*2604*	*0.6617*	*0.9337*	*0.5218*	*43.99*	*32.21*
5p6	10	2	20	128	12	0.9826	0.9975	0.5888	**63.00**	45.20
5p6	20	5	20	74	6	0.9875	**0.9993**	**0.6390**	30.43	20.90
5p6	20	30	20	18	3	**0.9973**	0.9990	0.6387	30.43	20.90

The initial selection doesn't get the highest score in any of the evaluated categories but is listed for comparison purposes. Noticeably, the combination that results in the fewest unlabeled contigs (6) only generates four groups and has a big mean probability (0.9753). Additionally, the parameter combinations that yield the best mean probability (0.99), inner cluster homogeneity (0.9993), outer cluster homogeneity (0.6390), highest recovered genes (63 %) and the highest rate of recovered isoforms (47.69 %) all have fewer than 181 groups produced by HDBSCAN. There are only two combinations that generate approximately ∼2,700 initial groups. Moreover, the inner cluster homogeneity is bigger than 0.95 for every combination and comparatively the outer cluster homogeneity fluctuates between ∼0.34 and ∼0.63.

Since the aim is to achieve a more complete assembly, the parameters that yield the most isoforms are selected for further evaluation even though they have the lowest outer cluster homogeneity (0.34) and lowest mean probability (0.41) among the shown parameter combinations. These parameters are used as a separate clustering for the examined datasets (karma*). Contrarily, the parameter combination with the highest number of recovered genes (63 %) has a higher outer cluster homogene-

ity (0.58) and a better mean probability (0.98). In general, it appears that small k-mer sizes (3-mer and 5p6-mer) are beneficial for the clustering process since the k-mer size of seven isn't among the listed parameter selection. The full parameter evaluation is accessible in the supplemental data (See Section 9).

3.3 Runtime analysis reveals `Linclust` as fastest clustering tool

The captured runtimes for the different clustering tools are listed in Table 6. Since the `Oyster River Protocol` only works for paired-end data, there is no runtime available for single-end data. With a maximum of eight seconds for the single-end *C. elegans* dataset (ERR2886543), `Linclust` yields by far the shortest runtimes for all samples.

The runtimes for both versions of `MeShClust` range from 44 seconds (simulated dataset, `MeShClust`^{2}) up to 49 minutes (ERR779269, `MeShClust`), but generally don't differ too much from each other. For the two distinct runs of `cd-hit-est`, the difference is more noticeable. With a sequence identity of 90 % (c=0.9), `cd-hit-est` usually takes longer than with a sequence identity of 100 %. The longest run (∼12 hours) of `cd-hit-est` (c=0.9) was measured for the single-end *C. elegans* dataset (ERR2886543) and the shortest (∼5 minutes) for the single-end *E. coli* dataset (SRR4255368).

Table 6: Runtime (hh:mm:ss) of the different clustering methods for each dataset. The best runtime per sample is marked bold.

Clustering method	**C. elegans**			**E. coli**	
	SRR5456164 (PE)	**ERR2886543 (SE)**	**FLUX (PE)**	**ERR779269 (PE)**	**SRR4255368 (SE)**
`cd-hit-est-100`	00:22:20	00:34:41	00:01:33	02:58:47	00:00:47
`cd-hit-est-90`	02:44:36	12:03:22	00:23:03	04:57:00	00:04:50
`Linclust`	**00:00:05**	**00:00:08**	**00:00:03**	**00:00:04**	**00:00:03**
`MeShClust`	00:02:31	00:04:24	00:02:07	00:49:10	00:00:44
`MeShClust`^{2}	00:07:34	00:06:19	00:00:43	00:01:04	00:00:42
`Grouper`	21:59:22	00:45:52	00:03:29	00:01:48	01:27:00
`Grouper`*	23:22:52	00:46:13	00:03:36	00:01:37	01:19:42
`karma`	07:41:40	12:46:45	00:26:52	00:08:04	00:28:04
`karma`*	00:48:06	00:39:20	00:03:36	00:05:11	00:06:22
`Oyster River Protocol`	11:39:50	N/A	00:31:33	05:53:03	N/A

`cd-hit-est` (c=1) clusters the single-end *E. coli* dataset (SRR4255368) in

roughly a minute and takes approximately 3 hours for the paired-end *E. coli* dataset (ERR779269). Out of all clustering tools, both `Grouper` runs for the paired-end *C. elegans* dataset (SRR5456164) take longer than 21 hours to complete. However, for all other datasets, the runtime is lower than 1 hour. Besides `cd-hit-est` and `Grouper`, `karma` is also a program that has a runtime in the range of hours for individual datasets. For example, `karma` needs approximately eight hours for the paired-end *C. elegans* dataset and 13 hours for the single-end *C. elegans* dataset. For all other datasets, its runtime lies within the range of minutes. `karma*` reveals a similar pattern. Yet, the runtime for the full *C. elegans* datasets is ∼40 minutes each, the maximum time needed for the other datasets is ∼5 minutes. In general, karma* takes less time than `karma`. The `Oyster River Protocol` takes between ∼30 minutes and ∼12 hours for the paired-end datasets.

3.4 Coverage information complements k-mer based clusters

This section visualizes `karma`'s clustering process of the artificial paired-end *C. elegans* dataset in depth. During the k-mer based clustering, trimming and cluster redefinition, several cases may occur. These cases are exemplarily shown (Figures 10-12) and described in the following three subsections. All other graphs are accessible in the supplemental data (See Section 9.3).

Case 1

After k-mer based clustering, cluster 28 contains seven contigs with six different isoforms (Annotation IDs: F27D4.5.1, T03F1.1.2, Y37E3.13a.1, B0511.8.2, Y37E3.15a.1, C09D4.3.1, see Figure 10A). Note, that even though Y37E3.13a.1 and Y37E3.15a.1 have a similar name, they originate from different genes. The isoform F27D4.5.1 is produced both by `rnaSPAdes` and `Trinity` and connected with a weight of 0.92. All other contigs are present as single vertices (Figure 10A). Consequently, the cluster trimming step removes those and during the cluster redefinition process, another contig is added to the graph. This new contig was produced by `SOAPdenovo-Trans` and is connected to both other contigs with weight ∼0.9 (Figure 10B). `MCL` clustering selects only one contig as a representative sequence. The sequence alignment for the final cluster 28 shows that the transcripts are highly similar, while the contig produced by `SOAPdenovo-Trans` is with 1181 nt about 246 nt shorter than the other two sequences (See Section 9.3, Figure 26).

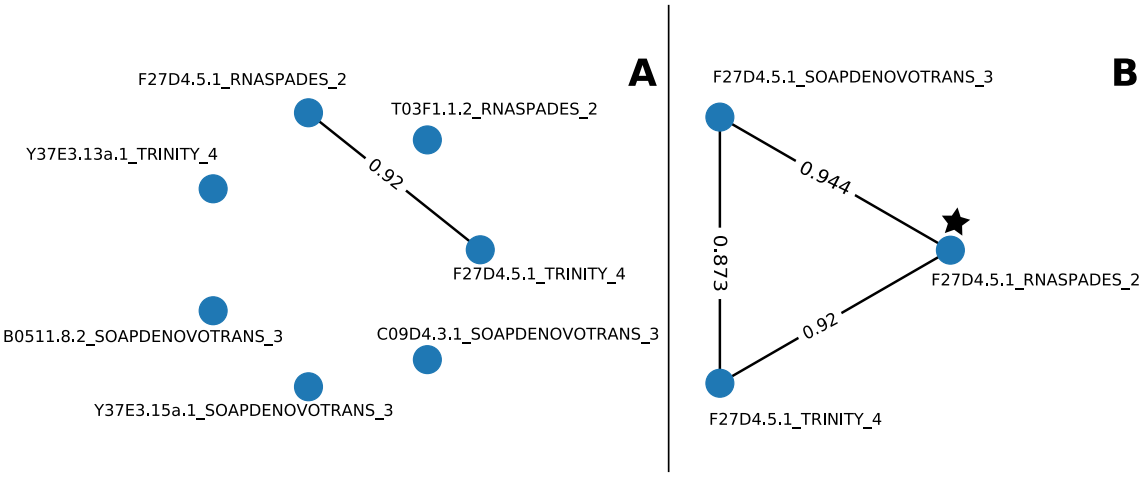

Figure 10: In-depth clustering process of `karma` (Case 1). Cluster 28 of clustering the artificial paired-end dataset. **(A)** K-mer based cluster produced by `HDBSCAN` complemented with the calculated weights from the quasi-mapping information. The cluster contains six different isoforms (Annotation IDs: F27D4.5.1, T03F1.1.2, Y37E3.13.a.1, B0511.8.2, Y37E3.15a.1, C09D4.3.1) of which two are connected with a weight of 0.92. **(B)** Cluster after trimming and cluster redefinition. The asterisk marks the selected representative sequence for this cluster. The single vertices were removed and a new vertice is added via cluster redefinition.

Case 2

Figure 11 shows cluster 455 after trimming and redefinition. It contains two different isoforms (Annotation IDs: C25A1.10a.2 and C25A1.10b.2) that are produced by `Trinity` and `SOAPdenovo-Trans`. Two contigs of isoform C25A1.10a.2 were produced by `SOAPdenovo-Trans` and are connected with a weight of 0.952. `Trinity` Contrarily, `Trinity` created two contigs of two isoforms, C25A1.10a.2 and C25A1.10b.2, that are connected with a weight of 0.529.

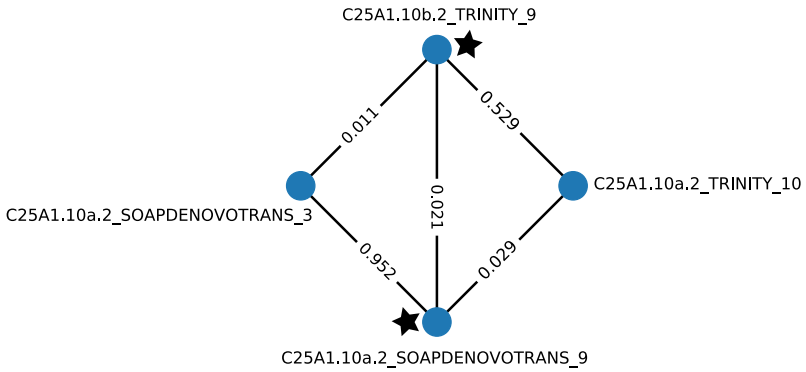

Figure 11: In-depth clustering process of `karma` (Case 2). Cluster 455 of clustering the artificial paired-end dataset after trimming and cluster redefinition. The cluster contains two different isoforms (Annotation IDs: C25A1.10a.2 and C25A1.10b.2). `MCL` clustering selects two distinct representative sequences (asterisk).

In contrast, the edges between the contigs from SOAPdenovo-Trans and Trinity are smaller than 0.03, which leads to two representative sequences during MCL clustering. The contigs from each assembler are quite similar to each other. However, the sequences from Trinity are significantly shorter than those from SOAPdenovo-Trans, thus impairing the quality of the alignment (See Section 9.3, Figure 27).

Case 3

The last case shows cluster 660 before trimming and redefinition. Figure 12 shows five contigs and three isoforms (Annotation IDs: T20F10.1.1, T19B4.7.1, F25D7.5.1). The trimming process removes the isoform F25D7.5.1 and the redefinition does not add new sequences to the cluster (not shown). However, the two other isoforms are both represented by two contigs from different assemblers. Based on the coverage information, the edges between the two contigs of each isoform have a weight of 1. The clustering with MCL yields two representative sequences for this cluster. The sequences of two transcripts of the same isoform are concurring. However, the sequences between two isoforms is quite different. Furthermore, the sequences for isoform T19B4.7.1 are only approximately 250 nt long and the sequences for isoform T20F10.1.1 are approximately 700 nt long (See Section 9.3, Figure 28).

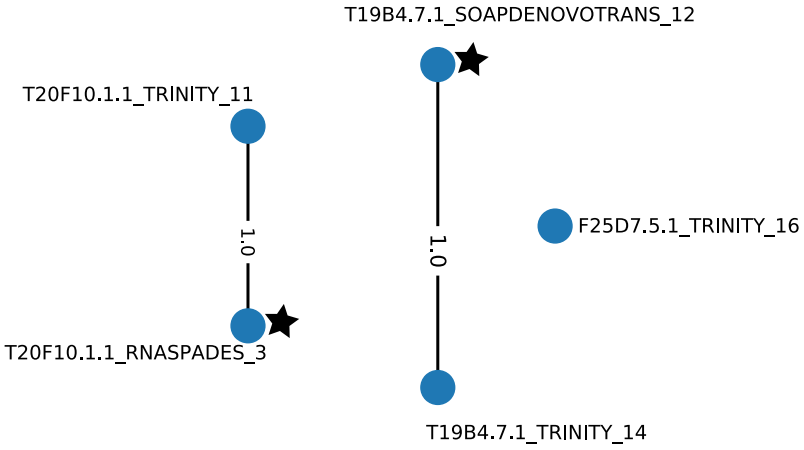

Figure 12: In-depth clustering process of karma (Case 3). Cluster 660 of clustering the artificial paired-end dataset before trimming and cluster redefinition. The cluster contains three different isoforms (Annotation IDs: T20F10.1.1, T19B4.7.1, F25D7.5.1). MCL yields two representative sequences for this cluster (asterisk).

3.5 The size of the clustered assemblies varies greatly

Simulated *C. elegans* dataset

Figure 13 represents the size of all three assemblies, the concatenation, as well as the clusterings for the simulated *C. elegans* dataset. The concatenated assembly

has 15,908 sequences, which originate from the assemblies with sizes between 4,500 and 6,300 sequences in total. The size of the assemblies after clustering varies significantly between two (`Grouper`) and 14,033 (`MeShClust`) sequences and the number of reproduced genes hover around 3,000 except for `karma` and `Grouper`. The number of isoforms is always slightly higher, with a maximum of 3,872 isoforms for `MeShClust`. Of the two sequences extracted from `Grouper`, none of them represents a gene from the original dataset. Both `karma` runs produce around 3,000 to 4,000 sequences, with roughly 1,500 genes recovered, which is only about half the amount compared to other clustering methods.

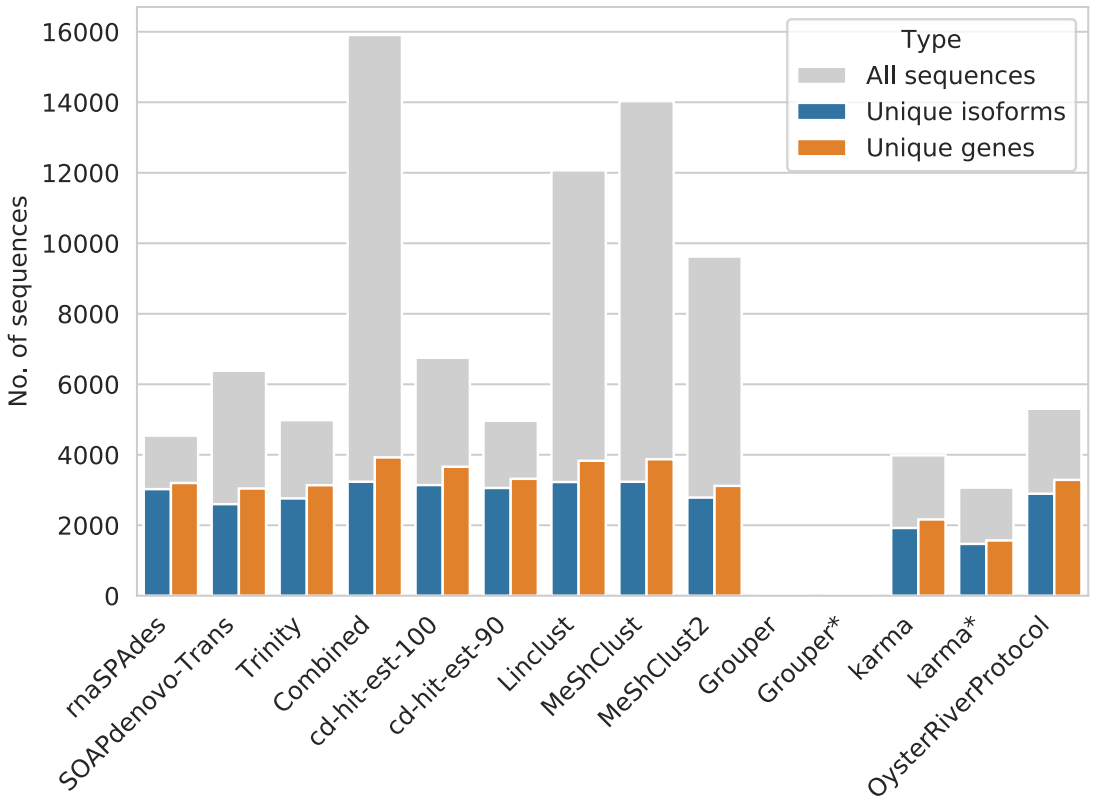

Figure 13: The number of sequences (grey bar) in each (clustered) assembly for the simulated *C. elegans* dataset (CEL_FLUX). The number of genes/isoforms of each assembly were evaluated via a `BLAST` search against the original data. The orange/blue bars represent the number of unique genes/isoforms in each assembly.

When looking at the length distribution of the assemblies and clusterings, it is noticeable that the extracted sequences from `Grouper` have a median length of ∼25,000 nt (Figure 14). This is far above all other assemblies where the medians lie around 200 to 600 nt. Additionally, the longest sequences of all other assemblies are also around 25,000 nt long. Furthermore, the clusterings from `karma` and `MeShClust`^{2} consist of comparatively shorter median sequence lengths (∼220 nt). Interestingly, the shortest sequence of the `Oyster River Protocol` is with ∼100 nt nearly twice as long compared to the shortest sequences from all other clustering methods. Additionally, `cd-hit-est` (c=1) and the `Oyster River Protocol` show a higher proportion of long sequences. Considering only single-tool assemblies, `SOAPdenovo-Trans` produces an assembly of rather small sequence lengths (Median: 255 nt) with the shortest sequence being 51 nt long.

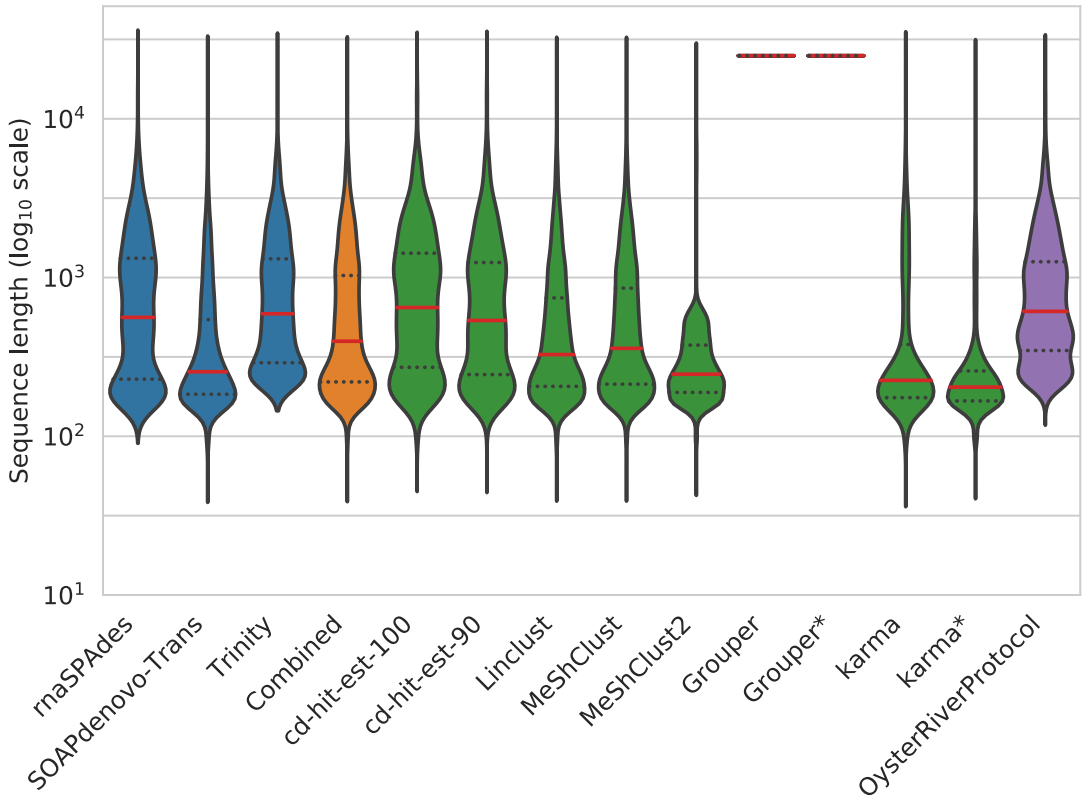

Figure 14: Distribution of transcript lengths in each (clustered) assembly for the simulated *C. elegans* dataset (CEL_FLUX). The $\log_{10}$ scale reaches from 10 nt (10^1) to ∼31,622 nt ($10^{4.5}$). The wideness of the violin indicates the number of sequences with a certain length. Blue: Single assemblers, Orange: Concatenation of the blue assemblies, Green: Different clustering methods based on the 'Combined' set. Purple: The `Oyster River Protocol` that relies on its own internal assemblies. The red line represents the median sequence length and the dotted grey lines mark the quartiles.

C. elegans datasets

Figure 15 shows the sizes of the assemblies from both *C. elegans* datasets. Comparing only single assemblers, `rnaSPAdes` produces the largest assembly for the paired-end dataset (SRR5456164) and `SOAPdenovo-Trans` the largest assembly for the single-end dataset (ERR2886543). Regarding the clusterings, `Grouper` and `karma` produce very few sequences through clustering for both datasets. However, `karma` yields up to 8,056 sequences for the single-end dataset and 7,608 sequences for the paired-end dataset. Additionally, `Grouper` returns three sequences for the single-end dataset and 53 sequences for the paired-end dataset for both runs, respectively. Looking at the numbers from `cd-hit-est`, one can see that the clusterings with a smaller identity threshold are smaller for both datasets compared to those with a higher identity threshold.

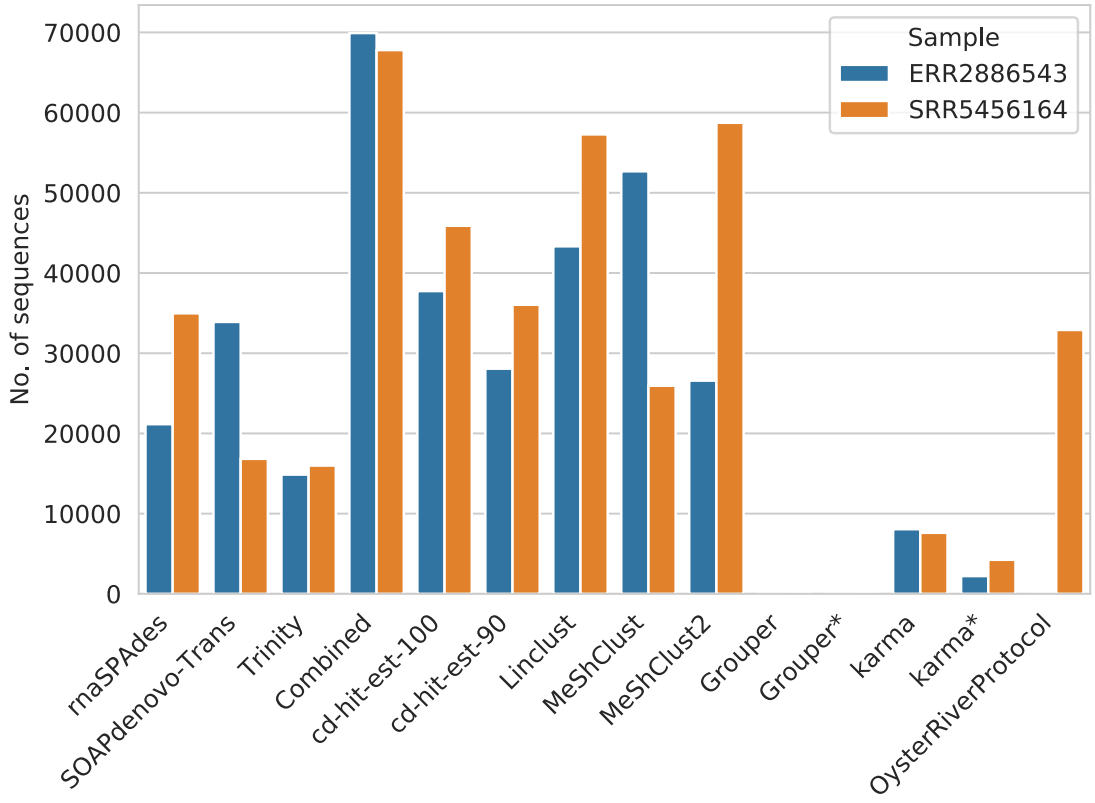

Figure 15: Number of sequences for each (clustered) assembly in the *C. elegans* datasets. Single-end dataset: ERR2886543. Paired-end dataset: SRR5456164. There is no `Oyster River Protocol` clustering available for the single-end dataset.

Interestingly, `MeShClust` and `MeShClust`^{2} produce very distinct results between the two datasets. `MeShClust` returns more sequences (52,678) than `MeShClust`^{2} (26,578) for the single-end dataset. Contrarily, for the paired-end dataset, those

numbers are swapped, thus more sequences are reported from `MeShClust`[2]. As mentioned in Section 1.4, there is no clustering available for the single-end dataset from the `Oyster River Protocol`.

To complement the information from Figure 15, the following Figures show the length distribution for each assembly for the paired-end (Figure 16) and the single-end dataset (Figure 17).

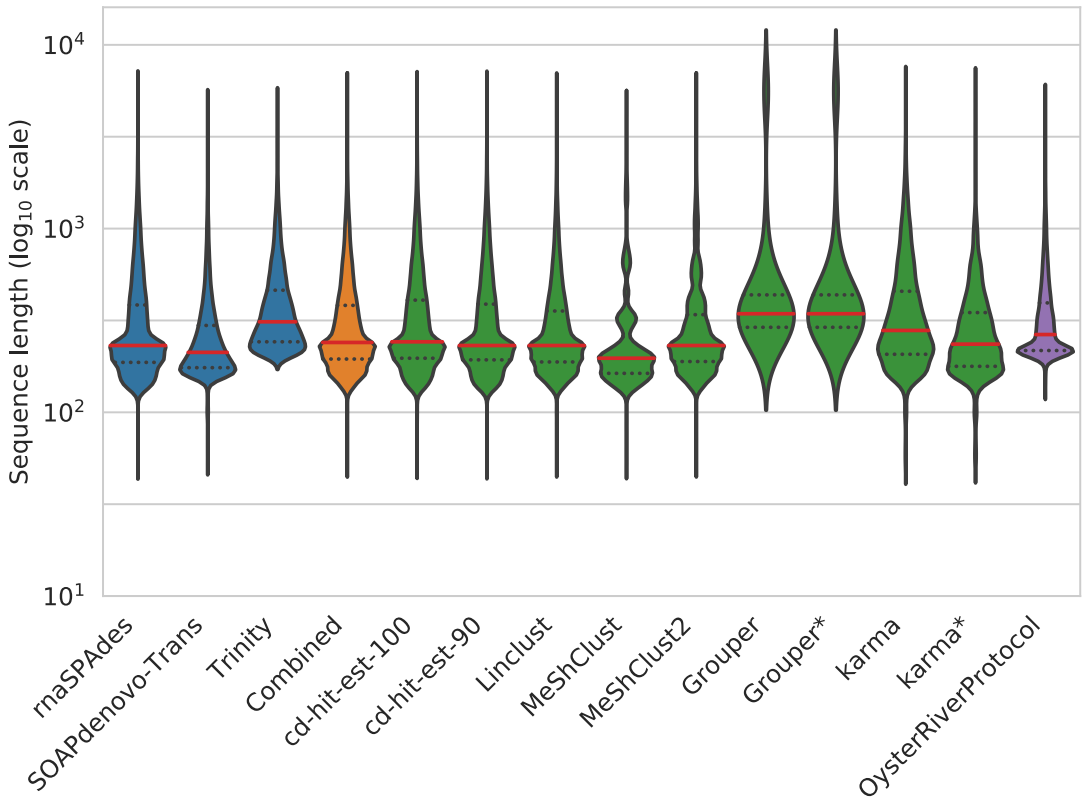

Figure 16: Distribution of transcript lengths in each (clustered) assembly for the paired-end *C. elegans* dataset (SRR5456164). The $\log_{10}$ scale reaches from 10 nt (10^1) to ~10,000 nt (10^4). The wideness of the violin indicates the number of sequences with a certain length. Blue: Assemblers, Orange: Concatenation of the blue assemblies, Green: Different clustering methods based on the 'Combined set'. Purple: The `Oyster River Protocol` that relies on its own internal assemblies. The red line represents the median sequence length and the dotted grey lines mark the quartiles.

For the paired-end dataset, `Trinity` produces fewer small contigs compared to `SOAPdenovo-Trans` and `rnaSPAdes`. `Trinity`'s shortest contig is 197 nt long, whereas `rnaSPAdes` and `SOAPdenovo-Trans` provide the shortest contig length of 50 nt. The median sequence length doesn't have a huge difference throughout all different tools and varies between 197 nt (`MeShClust`) and 344 nt (`Grouper`). Both `Grouper` runs also contain the longest contig with a length of 67,791 nt. Note

that Grouper contains multiple versions of large contigs (∼6200 nt) and therefore, the violin plot seems to reach values of up to 10,000 nt even though there are no actual sequences of that length. This is simply due to the way the data is represented through the violin plot. Additionally, the Oyster River Protocol and both runs from Grouper have a minimum sequence length of ∼100 nt, which is twice as long as the shortest sequence for all other clustering tools.

Looking at the single-end dataset, the median sequence lengths of all tools vary again (Figure 17).

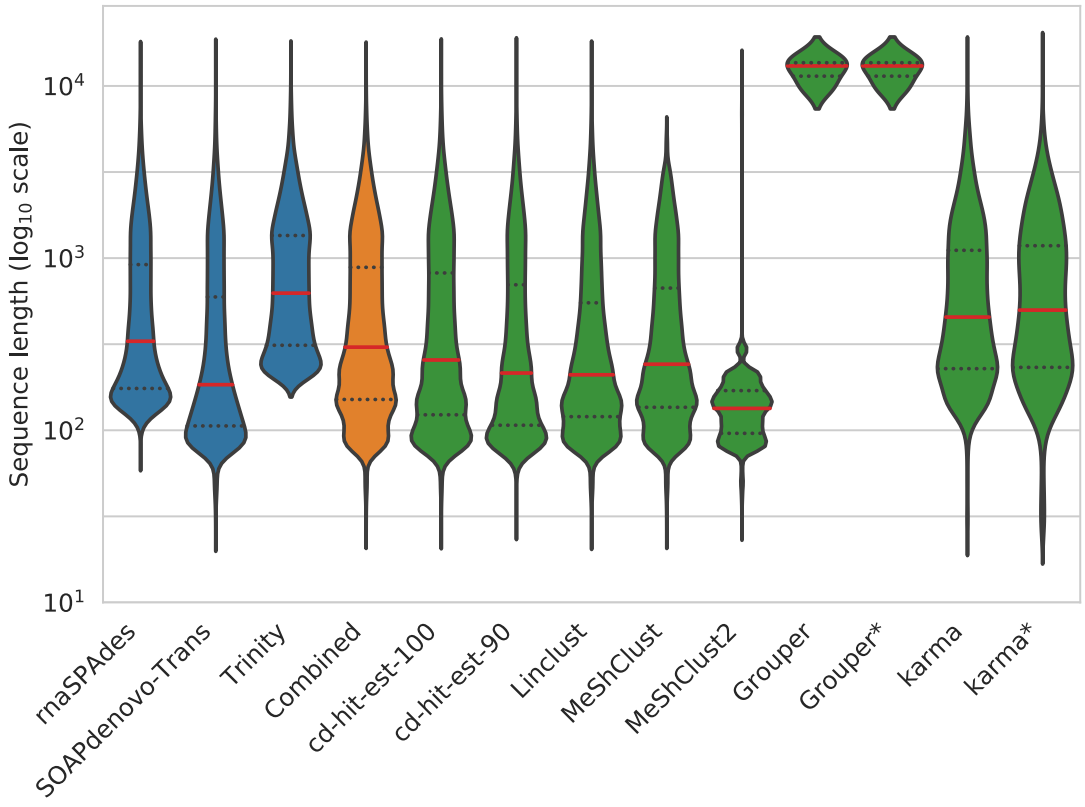

Figure 17: Distribution of transcript lengths in each (clustered) assembly for the single-end *C. elegans* dataset (ERR2886543). The $\log_{10}$ scale reaches from 10 nt (10^1) to ∼10,000 nt (10^4). The wideness of the violin indicates the number of sequences with a certain length. Blue: Assemblers, Orange: Concatenation of the blue assemblies, Green: Different clustering methods based on the 'Combined set'. The red line represents the median sequence length and the dotted grey lines mark the quartiles.

Regarding the assemblers, SOAPdenovo-Trans produces the broadest spectrum of sequence lengths between 26 nt and 14,243 nt with mostly shorter sequences (Median: 184 nt). Contrarily, Trinity has a median sequence length of 625 nt that ranges from 200 nt up to ∼14,000 nt.

cd-hit-est, Linclust and MeShClust have a similar distribution of se-

quence lengths with a median of ∼210 nt. However, the longest sequence from MeShClust is only 5237 nt long, whereas the longest sequences from the other tools are around 14,000 nt long. Again, the three sequences of Grouper are all long sequences with a median of ∼13,000 nt. MeShClust^2 has the lowest median (134 nt) while still having sequences of the length of ∼14,000 nt. The distribution between long and short sequences is balanced the most for karma with a median sequence length of around 450 nt.

E. coli datasets

Equivalent to the *C. elegans* datasets, Figure 18 describes the number of sequences per assembly and the clustered concatenation of the two *E. coli* datasets.

For the single-end dataset, the produced assemblies from rnaSPAdes, Trinity and SOAPdenovo-Trans vary a lot. While SOAPdenovo-Trans produces up to 13,059 sequences, Trinity yields only 2,443 contigs. For the paired-end dataset, rnaSPAdes produces 3,351 contigs, which are about 50 % of the amount that rnaSPAdes produces for the single-end dataset. Also as mentioned in the *C. elegans* section, cd-hit-est with a higher sequence similarity produces more sequences for both *E. coli* datasets. The smallest clusterings are once again produced by Grouper and karma. Grouper yields only two sequences for both runs and karma has 1,058 sequences for the paired-end dataset and 2,414 for the single-end dataset. With alternative parameters (karma*), the clustering only contains around 600 sequences for both datasets. The other clustering tools produce assemblies with 11,780 sequences (cd-hit-est, c=0.9) up to 18,299 sequences (MeShClust^2) for the single-end dataset. Contrarily, for the paired-end dataset, the Oyster River Protocol, cd-hit-est, Linclust, MeShClust and MeShClust^2 produce between 3,323 (cd-hit-est, c=0.9) and 7,014 sequences. Again, the clusterings for the single-end dataset contain way more sequences than those from the paired-end datasets, which is a result of the smaller assemblies from rnaSPAdes, SOAPdenovo-Trans and Trinity.

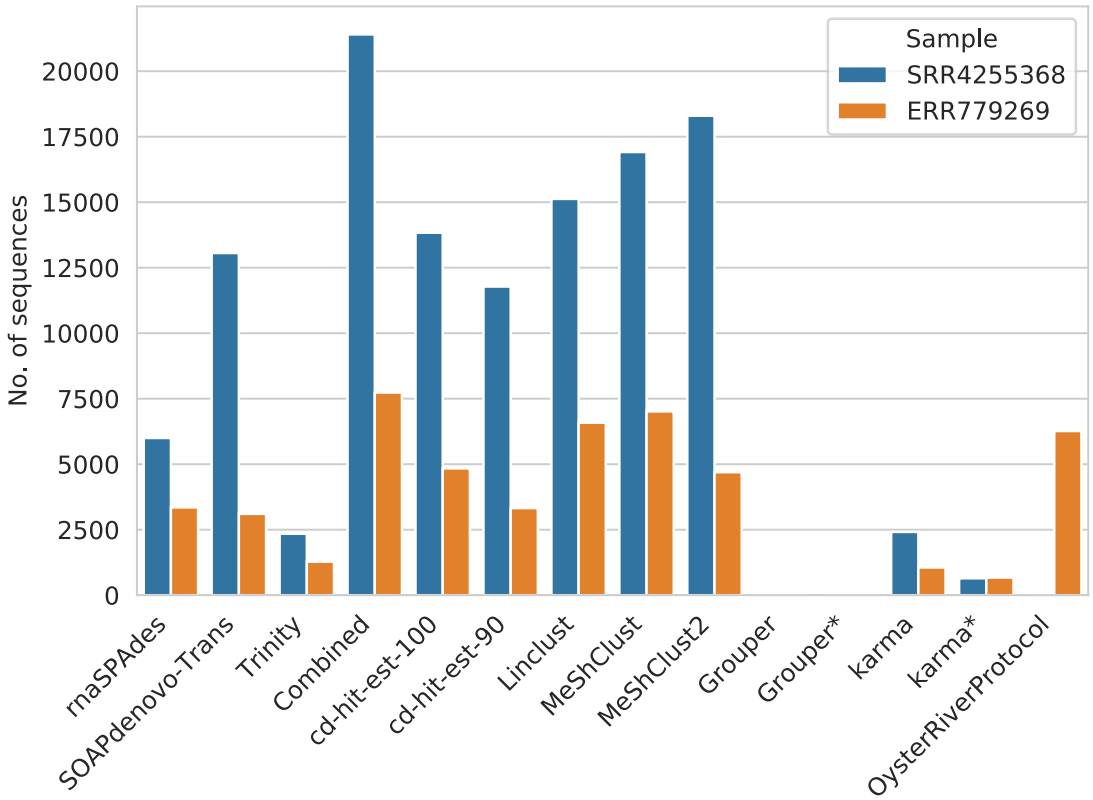

Figure 18: Number of sequences for each (clustered) assembly in the *E. coli* datasets. Single-end dataset: SRR4255368. Paired-end dataset: ERR779269. There is no `Oyster River Protocol` clustering available for the single-end dataset.

Figure 19 shows the contig length distribution for the paired-end *E. coli* dataset. Considering only assemblers, `Trinity` produces overall a bigger amount of long sequences (Median: 593 nt) compared to `SOAPdenovo-Trans` (404 nt) and `rnaSPAdes` (192 nt). From all clusterings, `Grouper` once again has only a few very long sequences with a median sequence length of 112,170 nt. Interestingly, the `Oyster River Protocol` pipeline produced a longer contig (143,831 nt) than all other contigs produced by the individual runs by `rnaSPAdes`, `SOAPdenovo-Trans` and `Trinity`. Nevertheless, the big majority of sequence lengths are around 200 nt. For the remaining tools, the clustered assemblies have an equal distribution of sequence length, while most of the sequences are short contigs (Median ∼150-260 nt).

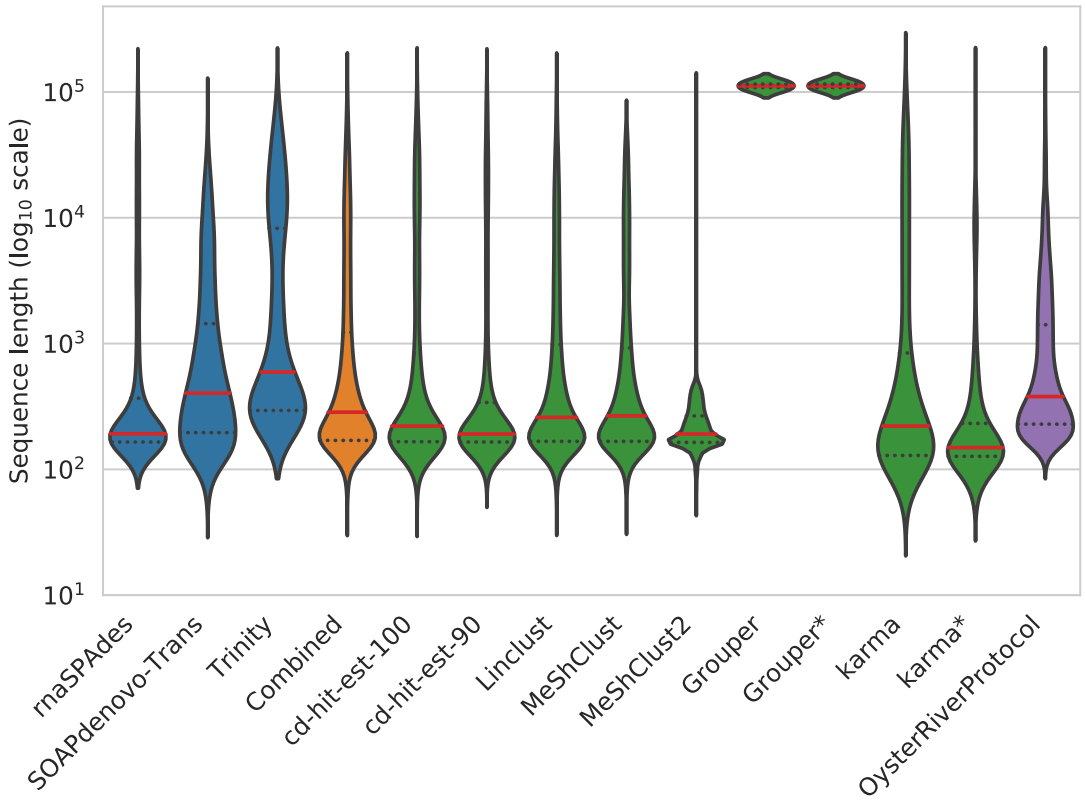

Figure 19: Distribution of transcript lengths in each (clustered) assembly for the paired-end *E. coli* dataset (ERR779269). The $\log_{10}$ scale reaches from 10 nt (10^1) to ~100,000 nt (10^5). The wideness of the violin indicates the number of sequences with a certain length. Blue: Assemblers, Orange: Concatenation of the blue assemblies, Green: Different clustering methods based on the 'Combined set'. Purple: The `Oyster River Protocol` that relies on its own internal assemblies. The red line represents the median sequence length and the dotted grey lines mark the quartiles.

Contrarily to the paired-end dataset, the maximum contig length of all assemblers is only 5,140 nt for the single-end dataset (Figure 20). `SOAPdenovo-Trans` produces contigs with a length between 27 nt up to 5,112 nt. On the other hand, `rnaSPAdes` (Median: 159 nt) and `Trinity` (332 nt) have a higher proportion of long sequences. Once more, `Grouper` has the highest median sequence length of 4,062. `karma`'s median sequence length is slightly higher compared to the other tools (~200 nt). `cd-hit-est`, `Linclust`, `MeShClust` and `MeShClust`^{2} have a median sequence length of approximately 120 nt with a similar length distribution. However, `MeShClust`'s longest sequence is only about ~2400 nt long compared to the other tools (~5100 nt).

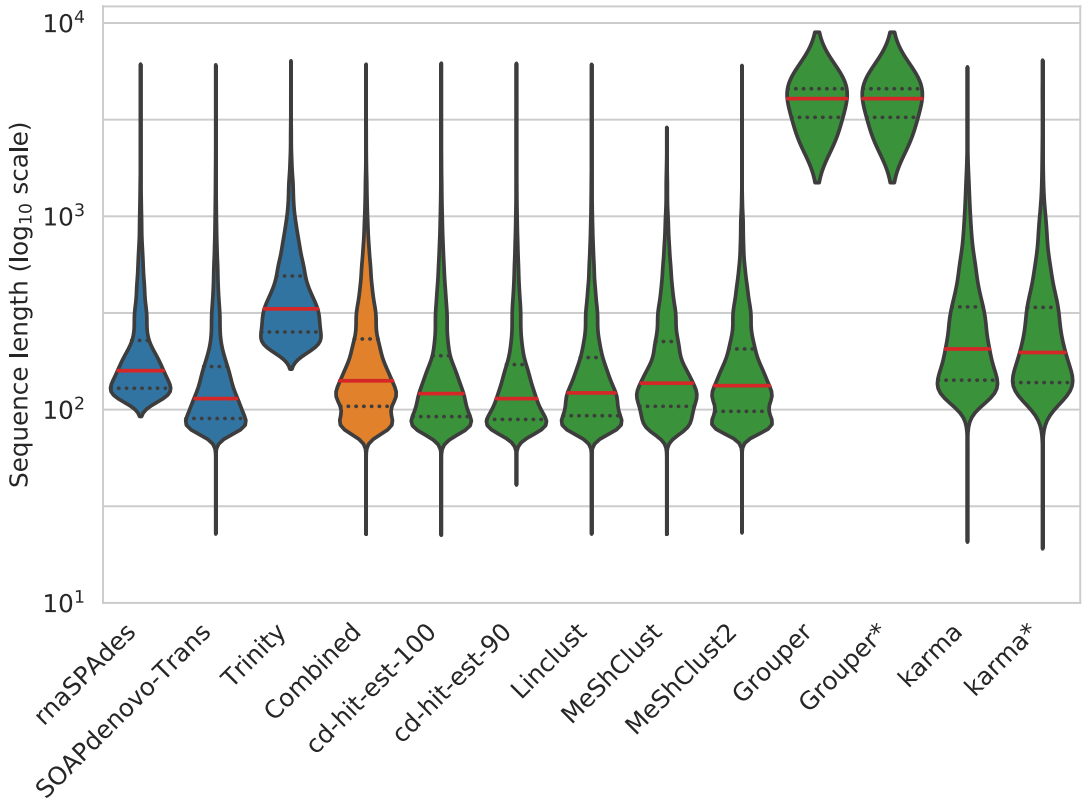

Figure 20: Distribution of transcript lengths in each (clustered) assembly for the single-end *E. coli* dataset (SRR4255368). The $\log_{10}$ scale reaches from 10 nt (10^1) to ~10,000 nt (10^4). The wideness of the violin indicates the number of sequences with a certain length. Blue: Assemblers, Orange: Concatenation of the blue assemblies, Green: Different clustering methods based on the 'Combined set'. The red line represents the median sequence length and the dotted grey lines mark the quartiles.

3.6 `cd-hit-est` outperforms other clustering tools

The metrics used to evaluate the datasets are explained as follows. The arrows behind each metric show if a low (↓) or high (↑) value for this metric is considered good.

- **`rnaQUAST`** [69]
  - **Database coverage (↑):** The total number of bases covered by reads (in all isoforms) divided by the total length of all isoforms.
  - **Duplication ratio (↓):** The total number of aligned bases in assembled transcripts divided by the total number of isoform covered bases. This metric counts neither paralogous genes nor shared exons, only real overlaps of the assembled sequences that are mapped to the same isoform.

  - **95 %-assembled isoforms (↑):** The number of isoforms from the database that have at least 95 % captured by a single assembled transcript.
  - **Misassemblies (↓):** The number of misassembled transcripts calculated from GMAP and BLASTN.

- **`TransRate`** [56]
  - **Mean open reading frame (ORF) percentage (↑):** For all contigs that contain a ORF, the mean percentage of contigs covered by the ORF.
  - **Assembly score (↑):** `TransRates` basic idea is to calculate a contig score for each transcript, representing its accuracy, completeness and non-redundant representation of a transcript that was present in the sequenced sample. The assembly score is calculated by using the contig scores, yielding a score that measures how complete the assembly is and how confident one can be with the assembly.
  - **Uncovered bases percentage (↓):** The proportion of bases that are not covered by any reads.

- **`DETONATE`** [68]
  For some of its calculated metrics, `DETONATE` estimates a 'true' assembly. Briefly said, this would be an assembly constructed with prior knowledge of the position of each read.
  - **K-mer compression score (KC score) (↑):** Reflects the similarity of an assembly compared to `DETONATE`s 'true' assembly.
  - **RSEM EVAL (↑):** A reference-free evaluation measure combining the compactness of an assembly and the support of the assembly through the set of reads it comes from.
  - **Nucleotide/Contig F1 (↑):** The F1 score describes the recovery of all nucleotides/contigs contained in the 'true' assembly with a identity bigger than 90 %.

- **BUSCO** [71, 72]
  BUSCO scans an assembly in regards to detecting benchmarked universal single-copy orthologs (BUSCOs) from their OrthoDB. The database contains genes that are present as single-copy orthologs in at least 90 % of the selected species. HMMER [77] detects genes in an assembly that match to the BUSCOs in their database. The identified BUSCO is considered complete if the

sequence length of the match lies in the 95 % expectation of the mean length of the specific BUSCO group.

  - **Complete and single-copy BUSCOs (↑):** The subset of complete BUSCOs that are present only once.
  - **Complete and duplicated BUSCOs (↓):** The subset of complete BUSCOs that are present multiple times.
  - **Fragmented BUSCOs (↓):** Identified BUSCOs, which are not in the expected range of length to represent a complete BUSCO. These indicate incomplete transcripts in case of a transcriptome.
  - **Missing BUSCOs (↓):** Unidentified or low-scoring BUSCOs. For transcriptomes, this means that the orthologous genes are missing.

- **Trinity and Salmon** [17, 57]

  - **Ex90N50 (↑):** As explained in Section 2.5, the Ex90N50 value describes the N50 value based on the 90 % of the highest expressed transcripts from the total normalized expression data in the assembly.

- **HISAT2** [70]

  - **Remapping rate (↑):** The proportion of mapped reads divided by the number of nucleotides in the assembly.

The heatmaps contain [0,1]-normalized values of the described metrics. The normalization takes into account whether a large or small value for the metric is considered good, thus in the heatmaps, a value of one always represents the best value. The complete metrics from all evaluation programs are accessible in the supplemental data (See Section 9.3).

Figure 21 lists the $[0, 1]$-normalized assembly evaluation scores for the paired-end *C. elegans* dataset (SRR5456164). The assembler `rnaSPAdes` achieves the best overall score of 12.39. Concatenating all three assemblies from `rnaSPAdes`, `SOAPdenovo-Trans` and `Trinity` leads to a score of 9.13. The worst score of 6.00 takes `Grouper`, regardless of the choice of parameters. Considering only clustering tools, `cd-hit-est` (c=0.9) yields the highest sum score (12.32). The self-implemented algorithm `karma` achieves a score of 9.49 and 6.92 (parameters from Section 3.2).

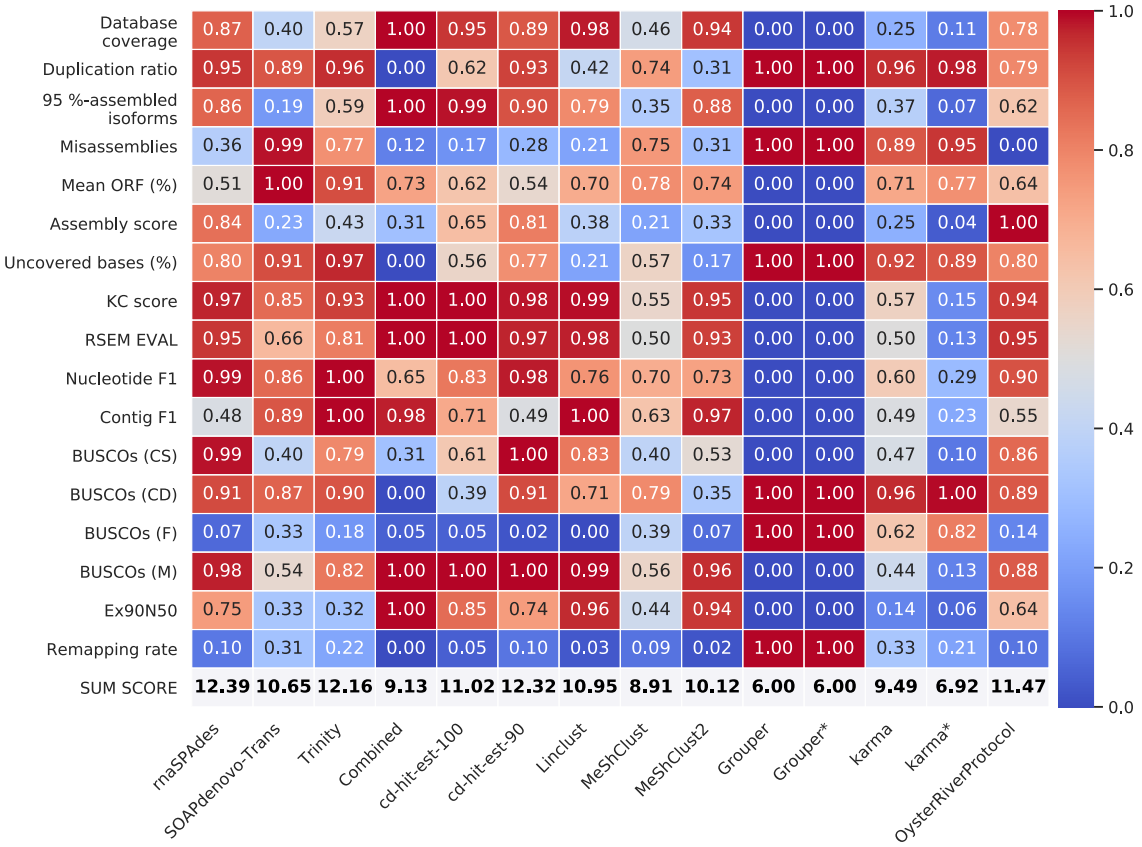

Figure 21: $[0, 1]$-normalized assembly evaluation metrics for the paired-end *C. elegans* dataset (SRR5456164). A score of 1 represents the best score. The last row summarizes all other metrics into a comparative score. 'Combined' is the concatenated version from all three assemblers. The number in `cd-hit-est` represents the percentage of the chosen identity threshold. `Grouper`/`Grouper*`: `Grouper` with parameters **`orphan`** and **`mincut`** disabled/enabled. `karma`: Parameters according to Section 2.3. `karma*`: Parameters chosen in Section 3.2.

The normalized metrics for the single-end *C. elegans* dataset (ERR2886543) are found in the heatmap in Figure 22. From the used three assemblers, `rnaSPAdes` gets the highest score of 11.26. The union of all three assemblies yields a score of 7.36. The best overall score (11.54) achieves `cd-hit-est` (c=0.9), which also makes it the clustering tool with the highest score. `karma` scores a 8.22 with parameters from Section 3.2 and `karma*` gets a score of 5.98. Finally, `Grouper` scores the lowest with a sum score of 4.41. This score is achieved regardless of whether different parameters are used.

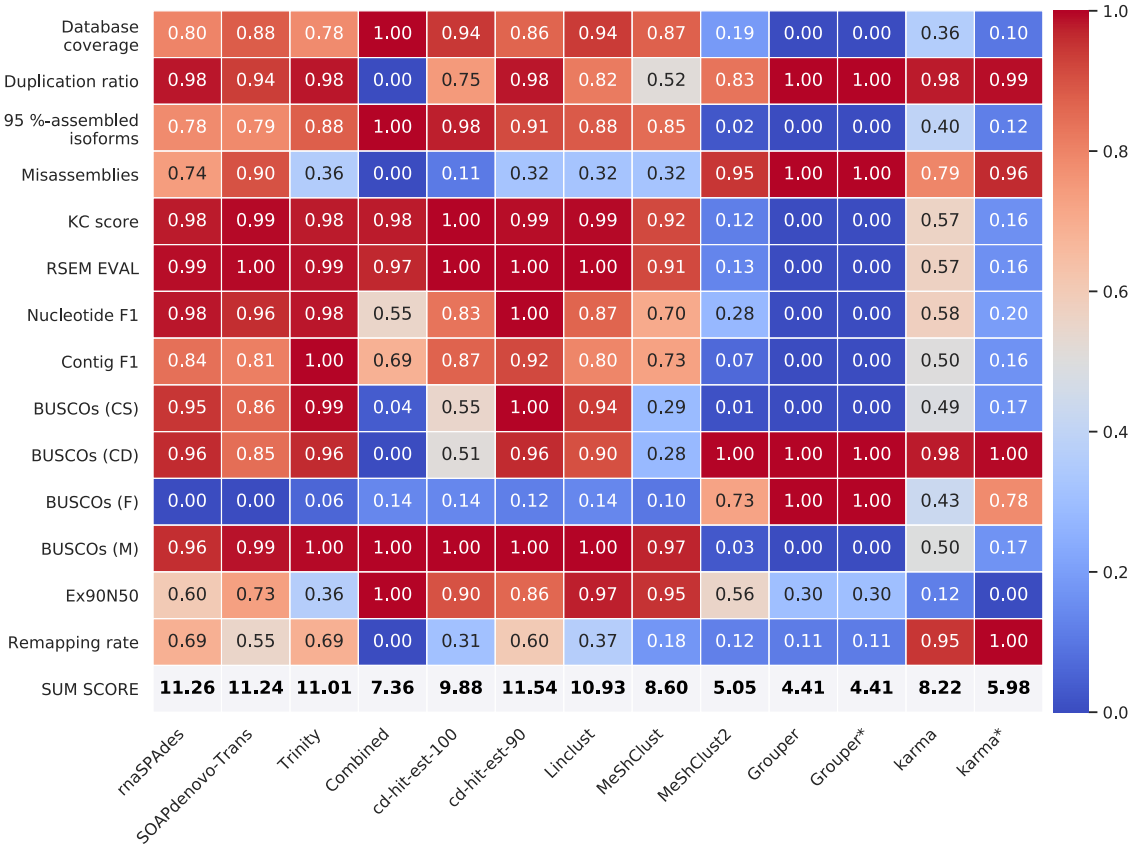

Figure 22: $[0, 1]$-normalized assembly evaluation metrics for the single-end *C. elegans* dataset (ERR2886543). A score of 1 represents the best score. The last row summarizes all other metrics into a comparative score. 'Combined' is the concatenated version from all three assemblers. The number in `cd-hit-est` represents the percentage of the chosen identity threshold. `Grouper`/`Grouper*`: `Grouper` with parameters **`orphan`** and **`mincut`** disabled/enabled. `karma`: Parameters according to Section 2.3. `karma*`: Parameters chosen in Section 3.2.

Figure 23 lists the $[0, 1]$-normalized assembly evaluation scores for the paired-end *E. coli* dataset (ERR779269). `SOAPdenovo-Trans` yields the best overall score of 13.24. The other assemblers `rnaSPAdes` (9.61) and `Trinity` (11.25) have lower scores. The concatenated assembly gets a score of 8.87. From all clustering methods, `cd-hit-est` (c=0.9) has the highest score of 10.30. `Grouper` gets the lowest overall score of 6.20. With the initial parameters (Section 2.3) `karma` achieves score of 9.29 and a score of 6.11 with the parameters from Section 3.2.

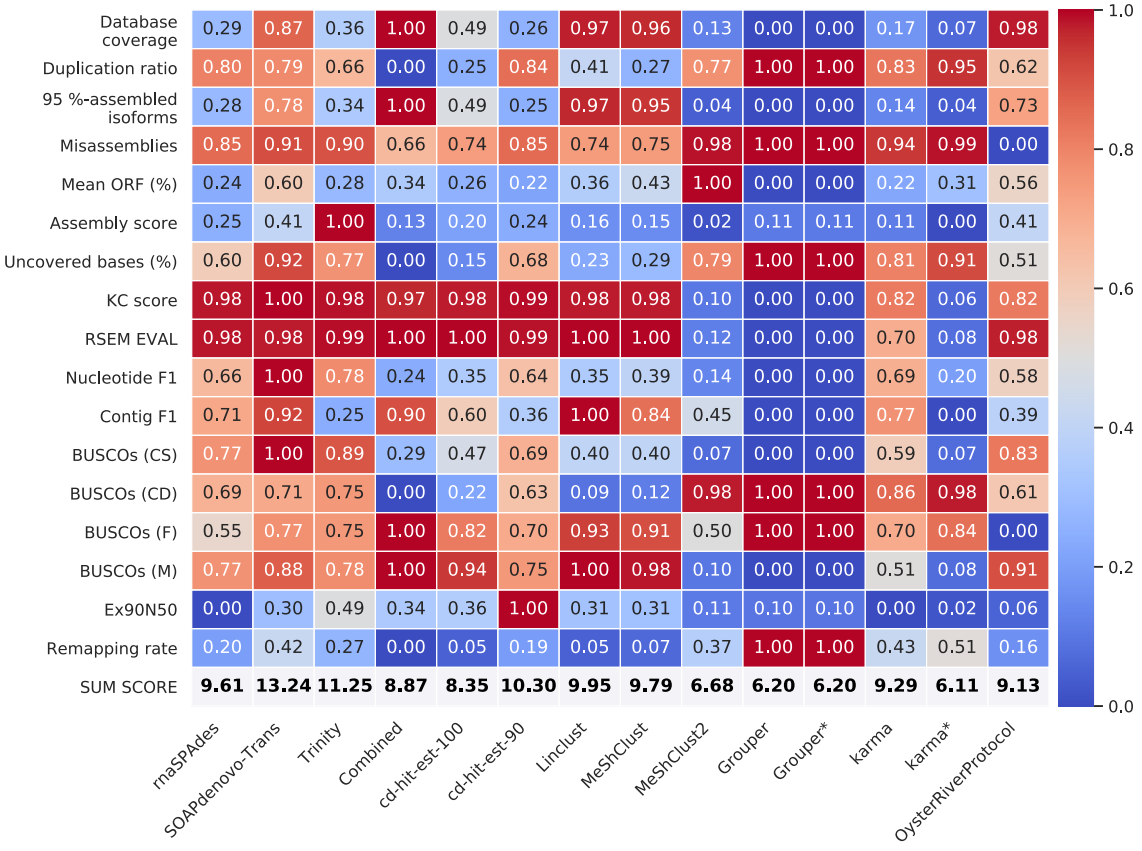

Figure 23: $[0, 1]$-normalized assembly evaluation metrics for the paired-end *E. coli* dataset (ERR779269). A score of 1 represents the best score. The last row summarizes all other metrics into a comparative score. 'Combined' is the concatenated version from all three assemblers. The number in `cd-hit-est` represents the percentage of the chosen identity threshold. `Grouper`/`Grouper*`: `Grouper` with parameters **`orphan`** and **`mincut`** disabled/enabled. `karma`: Parameters according to Section 2.3. `karma*`: Parameters chosen in Section 3.2.

The heatmap in Figure 24 shows the $[0, 1]$-normalized assembly evaluation metrics for the single-end *E. coli* dataset (SRR4255368). `cd-hit-est` (c=0.9) has the highest overall score of 10.80, which also makes it the highest-scoring clustering tool. The lowest score has `Grouper` (5.08). `karma` gets a score of 7.08 and `karma*` obtains a score of 5.23. Looking at the assemblers, `SOAPdenovo-Trans` achieves the highest score (10.74) besides `rnaSPAdes` (9.48) and `Trinity` (8.64). The concatenation of all three assemblers yields a score of 7.55.

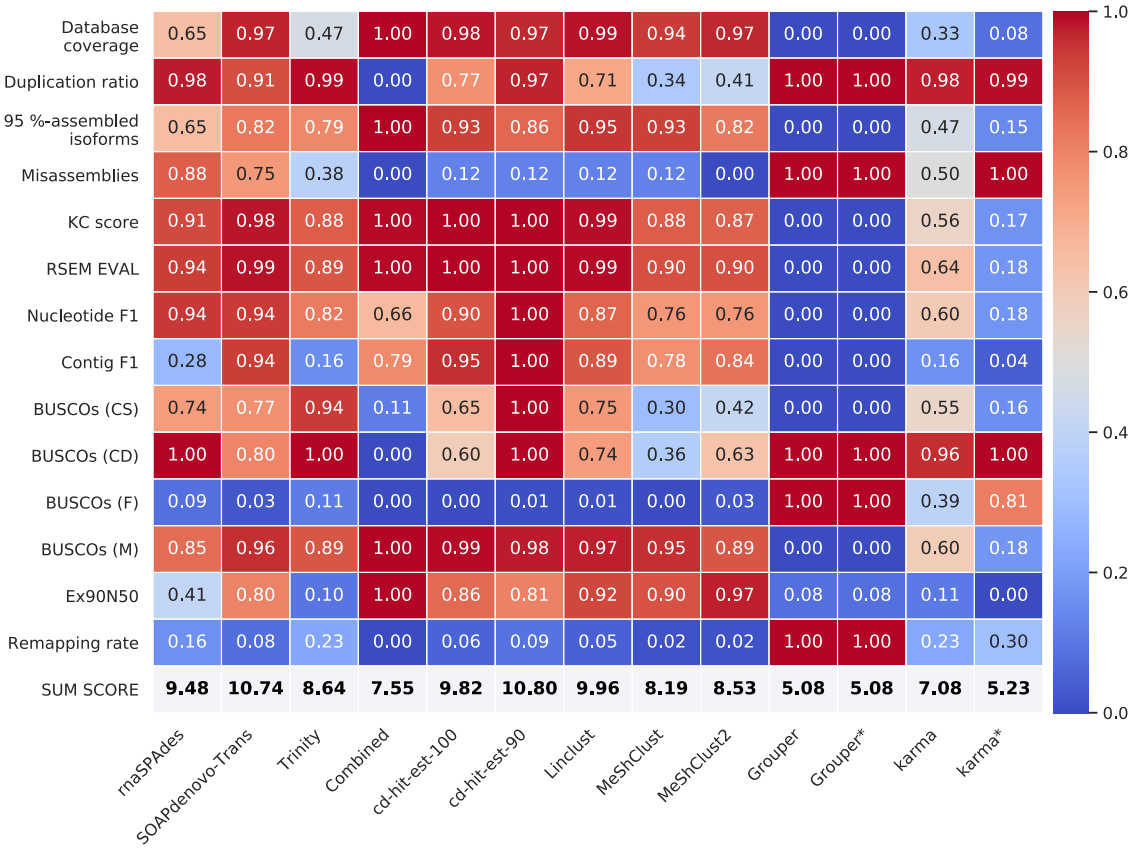

Figure 24: $[0, 1]$-normalized assembly evaluation metrics for the single-end *E. coli* dataset (SRR4255368). A score of 1 represents the best score. The last row summarizes all other metrics into a comparative score. 'Combined' is the concatenated version from all three assemblers. The number in `cd-hit-est` represents the percentage of the chosen identity threshold. `Grouper`/`Grouper*`: `Grouper` with parameters **`orphan`** and **`mincut`** disabled/enabled. `karma`: Parameters according to Section 2.3. `karma*`: Parameters chosen in Section 3.2.

For the artificial paired-end *C. elegans* dataset, the [0,1]-normalized assembly evaluation metrics are shown in Figure 25. Considering only assemblers, `Trinity` achieves the highest score (12.65) compared to `SOAPdenovo-Trans` (11.86) and `rnaSPAdes` (12.49), the concatenation of all three yields a score of 8.24. However, `cd-hit-est` (c=0.9) obtains the best overall score of 12.63. `Grouper` takes the lowest score of 6.28. `karma` gets a score of 9.20 and with alternative parameters (`karma*`), the score is 6.54.

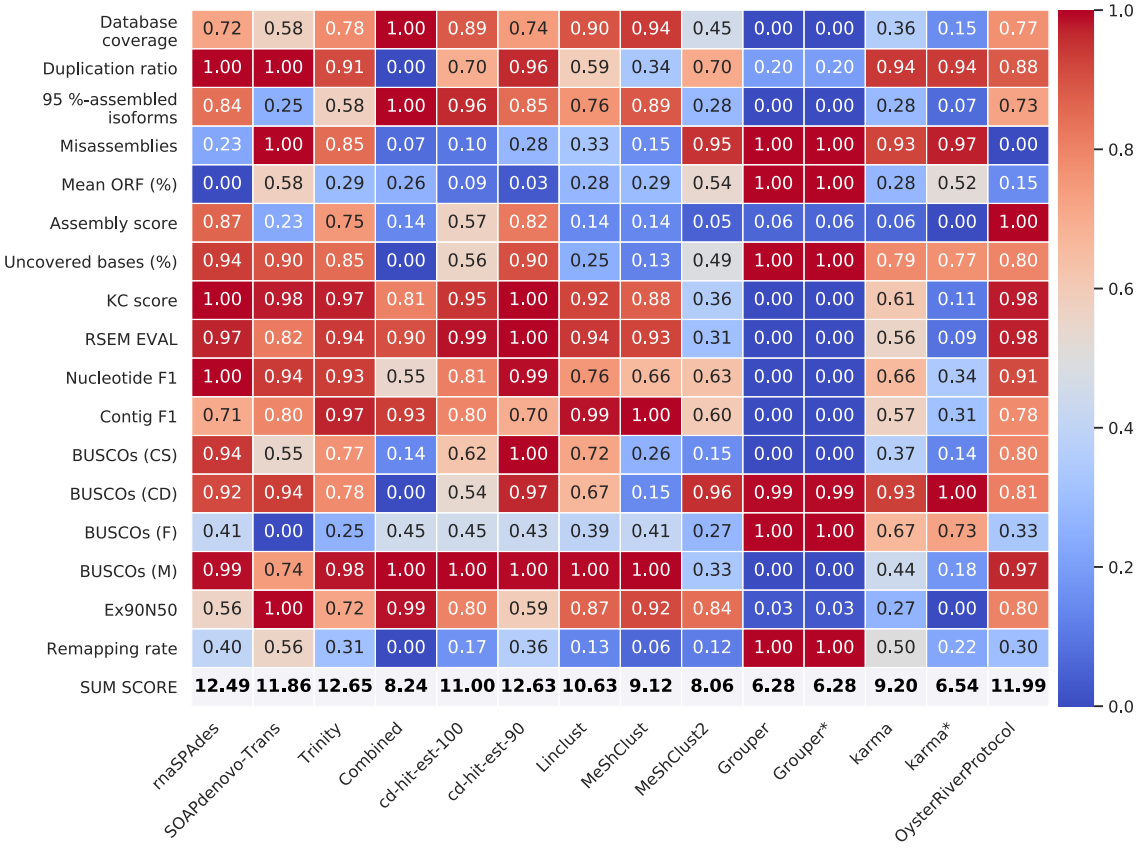

Figure 25: [0, 1]-normalized assembly evaluation metrics for the artificial paired-end *C. elegans* dataset (CEL_FLUX). A score of 1 represents the best score. The last row summarizes all other metrics into a comparative score. 'Combined' is the concatenated version from all three assemblers. The number in `cd-hit-est` represents the percentage of the chosen identity threshold. `Grouper`/`Grouper*`: `Grouper` with parameters **`orphan`** and **`mincut`** disabled/enabled. `karma`: Parameters according to Section 2.3. `karma*`: Parameters chosen in Section 3.2.

4 Discussion

The evaluation metrics from the previous section provide an interesting insight into the world of clustering *de novo* transcriptome assemblies. While the self-implemented method `karma` doesn't perform too well overall, we get a deeper understanding of the underlying algorithm. Furthermore, the comparison of several state-of-the-art clustering methods shows significant differences between single-end and paired-end RNA-Seq data.

4.1 `cd-hit-est` consistently produces good clusterings

Throughout all examined datasets, `cd-hit-est` with a sequence identity of 90 % outperforms the other clustering tools and yields the best results. `cd-hit-est` also manages to surpass the scores of the assemblers in two of the five datasets, thus improving the quality of the assembly based on the given evaluation metrics. Interestingly, those are both single-end datasets (ERR2886543 and SRR4255368), while for the paired-end datasets, either `rnaSPAdes`, `SOAPdenovo-Trans` or `Trinity` produce the best overall assembly. One of the most significant flaws while creating assemblies is the possibility of misassemblies. However, paired-end data possess additional information about the relative location of two paired reads. The utilization of this information reduces the number of misassemblies drastically, thus improving the assembly in general [78]. Assemblers with an active development such as `rnaSPAdes` [19] and `Trinity` [17] also include mechanisms to avoid misassemblies and detect alternatively spliced isoforms, leading to better assembly quality.

Furthermore, `cd-hit-est` with a sequence identity of 100 % still produces well-clustered assemblies. Usually, the size of the assembly with a 100 % sequence identity is larger because the algorithm is more restrictive, resulting in more sequences, which leads to an overall worse score because of more redundancy.

4.2 Large datasets may be disadvantageous for `Grouper`

Although particularly designed for the clustering of *de novo* transcriptome assemblies, `Grouper` consistently scores the worst sum score for all datasets. Nevertheless, the tool gets comparatively good scores for the following evaluation metrics: *Remapping rate*, *BUSCOs (F)*, *BUSCOs (CD)*, *uncovered bases* and *misassemblies*. Those metrics are affected by the size of the *de novo* transcriptome assembly in different ways.

The two `BUSCO` scores (F and CD) and *misassemblies* metrics are pure quantification measurements. Furthermore, they have the property that a smaller value

represents a better score. Naturally, if the transcriptome assembly has a smaller size, the chance of, e.g., a misassembly being present, is inevitably smaller. The same principle applies to *fragmented* and *complete and duplicated BUSCOs*. Consequently, *missing* and *complete and single BUSCOs* always have the lowest score. For example, the clustered assembly from `Grouper` in the single-end *C. elegans* dataset contains only three contigs, while the *Nematoda* dataset from `BUSCO` consists of 987 `BUSCO` groups. With such a difference, it comes to no surprise that those two `BUSCO` scores (M and CS) have consistently bad scores of zero.

Even though the metric *uncovered bases* and *remapping rate* consider the proportion of uncovered bases and mapped reads, the scores might be misleading, in particular, for small-sized transcriptome assemblies. Even if an assembly has a hypothetical mapping rate of 100 %, this doesn't result in a proper assembly if there is only one sequence. In consequence, considering the size of the clustered assemblies from `Grouper` (Figures 13, 15, 18) the evaluation results are rather unsurprising.

Additionally, it seems suspicious that both runs with different parameters for `orphan` and `mincut` produce the same results. The investigation of the clustering pipeline eliminates an error due to wrong parameter settings. Another reason could be that changing the parameters doesn't have that much of an effect on these datasets.

The clusterings from `Grouper` always contain the longest sequences. However, the reason for such a behavior lies in the implementation of the clustering pipeline. Since `Grouper` doesn't provide an option of selecting representative sequences for its clusters, the longest sequence is chosen. Anyway, the amount of sequences gives also information about the number of produced clusters from `Grouper` (2 - 52). The reason for the low amount of groups compared to other clustering methods may be a combination of their graph-based implementation and the huge size of the input assemblies. With the increase of input sequences, the graph also grows bigger, allowing for more connections between the vertices of the graph. Additionally, as shown in Table 3 their weighting function (Equation 1) could complicate the clustering process with `MCL`. Higher edge weights result in a better stochastic flow between vertices and if `MCL` is not adjusted through its parameters, this may lead to fewer clusters.

4.3 Uneven sequence length distribution can be a challenge for `MeShClust`

Overall, `MeShClust` and `MeShClust`^{2} have average evaluation scores for their clustered assemblies (5.05 - 10.12) compared to `cd-hit-est` (8.35 - 12.63), which

scores the best. This difference in clustering quality results could originate from the implementation of the `MeShClust` algorithm. Unlike `cd-hit-est`, which compares sequences through actual sequence similarities, the authors of the tools used alignment-free measurements [44]. Combined with the selection of the shortest sequence for initial clusters, this could lead to a more unprecise clustering process, thus worse scores for their assembly.

The concatenated assemblies from the paired-end *C. elegans* and single-end *E. coli* dataset have the lowest median sequence length (SRR5456164: 240 nt, SRR4255368: 141 nt) and shortest sequence (SRR5456164: ∼6,000 nt, SRR4255368: ∼5,000 nt). All other datasets have a median sequence length above 285 nt and a maximum sequence length up to ∼120,000 nt. `MeShClust`^{2} was designed explicitly for longer sequences with lower similarity [59]. Against the expectations, `MeShClust`^{2} achieves higher evaluation scores than `MeShClust` for datasets with shorter maximum sequence lengths and smaller median sequence lengths. For the paired-end *C. elegans* dataset, `MeShClust`^{2} gets a score of 10.12 and `MeShClust` receives a score of 8.91 (Figure 21). `MeShClust`^{2} scores an 8.53 and `MeShClust` 8.19 on the single-end *E. coli* dataset (Figure 24). A reason for this behavior could be, among other things, the uneven distribution of longer and shorter sequences with a majority of short sequences that neither `MeShClust` and `MeShClust`^{2} can handle. Our results show that `MeShClust` and `MeShClust`^{2} might not be suitable for *de novo* transcriptome clustering.

4.4 `Linclust`'s time-efficient algorithm might reduce clustering quality

Looking at the runtimes of the clustering tools, one has to keep in mind that there was only one time per tool and assembly measured. Even though this makes the results less reliable, it still gives a good relative impression of the time of calculations. Additionally, during the master's thesis, there were also latency problems on the used servers, which may explain some fluctuations of the runtimes. For example, in the single-end *C. elegans* dataset (ERR2886543), `cd-hit-est` (c=0.9) takes about 12 hours to complete the clustering process. For all other datasets, the runtimes are way lower (∼5 minutes up to ∼5 hours). Also, `cd-hit-est` (c=1) finishes the clustering process after ∼3 hours at the latest. This leads to the assumption that `cd-hit-est` (c=1) probably has a generally lower runtime. Still, the latency of the system may be the reason for a longer runtime for the single-end *C. elegans* dataset. For a reliable comparison, one should make sure to eliminate interfering factors beforehand and calculate the mean times from at least three distinct runs.

However, with this information in mind, the runtime results (Table 6) are as expected. `Linclust`, which was specifically developed for time-efficient clustering, performs by far the fastest clusterings, regardless of the number of sequences in the concatenated assembly. Unfortunately, the fast calculation due to reduced sequence alignments comes with its costs. The clustering results usually average around a total sum score of 10-11 and are thus worse than the `cd-hit-est` clustering with 90 % sequence identity. Only in rare cases, `Linclust` can achieve a better score than a single assembler. For example, in Figure 24 `Linclust` gets a 9.96 whereas `rnaSPAdes` only gets 9.45. Similar to `cd-hit-est`, the clusterings tend to have a high *database coverage* and few *missing BUSCOs*. This is most likely because the clustered assemblies from `Linclust` contain comparatively more sequences than those from other tools (Figure 13, 15, 18).

4.5 Clustering with `karma` yields too few transcripts overall

4.5.1 K-mer based clustering is not suitable for transcript clustering

While k-mer based clustering can work for metagenomic binning [45] or 16S rRNA clustering [79], in this case, it seems that it gives rather disenchanting results for clustering transcripts.

Initially, the method seems to be working in some cases (Section3.4, Case 2), where two different isoforms of the same gene share a k-mer based cluster. However, most of the initial k-mer based clusters show a mix of many different genes and isoforms (Section 3.4, Case 1 and 3). The functions of the different genes in a cluster seem to be completely random based on research listed in the UniProt database [80]. For example, in the first case, the isoform F27D4.5.1 (*bckd-1b*) is coding for a protein that positively regulates the larval development of *C. elegans* [81]. Contrariliy, while Y37E3.15a.1 (*npp-1*) is coding for a nuclear pore protein [82], T03F1.1.2 (*uba-5*) plays an essential role in the pathway responsible for environmental stress factors such as heavy metals [83]. This discrepancy of functions applies to both case 1 and case 3. Luckily, if the contigs don't share reads (Case 1), the cluster is cleaned up through coverage information. Contrarily, as seen in the second case, a cluster stays mixed, if two or more sequences for each gene/isoform are in a group. Therefore, the `HDBSCAN` produces low-quality initial clusters. A possible reason could be a poor choice of parameters that may be addressed in future work for `karma`. Likewise, the normalization of k-mer profiles based on the sequence length could be problematic, hence sequences of significant different lengths are falsely grouped.

4.5.2 New weighting function is not beneficial for MCL clustering

In its current implementation, karma seems to perform a quite restrictive clustering, which results in an overall small-sized clustered assembly. When taking a closer look at the clusters built by karma, it appears that there are a few clusters that contain more than 1,000 sequences. The problem with massive clusters is that the MCL clustering usually only selects one representative isoform. It seems that either the parameters for MCL are not suitable for those large clusters or the weights, similar to Grouper, are too high so that no subclusters can be formed. The newly implemented equation for karma (Equation 2) doesn't seem to improve this behavior as initially thought.

As seen in all heatmaps (Figure 21-25), the alternative selection of parameters for karma does not get better results as expected. It decreases the size of the clustered assembly and yields worse scores than the run with initial parameters. Yet, this fits with the previous statement regarding the size of the clusters. The karma* assemblies usually have fewer larger clusters at the end of the process where MCL is not dividing these into subclusters to select multiple representative sequences. This can be seen in the clstr-files in the supplemental data (Section 9.3). Representative sequences are marked with an asterisk. This leads to the assumption that for the current implementation of karma, more k-mer based clusters would result in a better-clustered assembly.

In a similar manner to Grouper, the problems mentioned above are the reason for a comparatively low amount of sequences in the clustered assembly (Figure 13, 15, 18). Hence, the small assembly from karma scores also very well for the evaluation metrics, where Grouper has high scores (See Section 4.2).

In terms of runtime, the initial number of k-mer based clusters seems to have a significant effect on karma. For example, in the simulated dataset, HDBSCAN produces 2,604 clusters for the initial run (karma) and 181 clusters with altered parameters (karma*, Table 5). The time for execution is approximately 23 minutes higher with more initial clusters (Table 6). The increased runtime could originate from the underlying non-optimized library for graph representation. For each graph that exists, there are certain steps to perform, including MCL clustering, which will increase the time of execution.

4.6 Clustering multiple *de novo* transcriptome assemblies is challenging

Even though karma doesn't perform overwhelmingly good, this doesn't make its fundamental concept pointless. The idea of clustering multiple *de novo* transcriptome assemblies is appealing at first glance. It seems like a simple task of combination and clustering redundancy to improve the resulting transcriptome assembly. In 2018, MacManes had a similar idea which resulted in the Oyster River Protocol [24].

However, the Oyster River Protocol, which was also explicitly designed for the same reason as karma, struggles to achieve better scores than the assemblers SOAPdenovo-Trans, rnaSPAdes and Trinity. For example, the tool is not able to produce a better assembly than a single assembler does for the paired-end *E. coli* dataset (Figure 23). For the other paired-end *C. elegans* datasets, the Oyster River Protocol is only able to achieve a higher score than SOAPdenovo-Trans. Still, multiple assemblers have the advantage to produce different sets of transcripts, potentially reducing the amount of missing genes and isoforms (Figure 9). There may be a few reasons that the Oyster River Protocol still gets scores in the range of cd-hit-est.

Firstly, the Oyster River Protocol has its own assembly and preprocessing steps, which may improve the reads even further, compared to the outlined preprocessing steps in Section 2.2, which could lead to better assemblies from the beginning. Secondly, the sequence similarities are calculated through OrthoFinder [55], which performs an all-vs-all BLAST [67] query. Since BLAST as well as cd-hit-est and Linclust works with sequence alignments, this could explain why the clustering process works comparatively well.

Furthermore, MacManes [24] uses the evaluation tool TransRate to optimize the results by choosing only the best contigs from the clusters based on TransRate. For the *C. elegans* datasets, the Oyster River Protocol produces its best scores (Figure 21, 25). Noticeably, the size of the clustered assembly is approximately the same as the cd-hit-est assembly (Figure 13, 15) suggesting that the size of the assembly is somehow relevant for the evaluation scores. The self-created problem of redundancy seems to be a tougher challenge than initially thought.

4.7 Evaluation metrics do not necessarily represent a good assembly

As already seen in the assemblies from `karma` and `Grouper`, some metrics get the highest score, among others, although the assembly contains far too few contigs. If taken out of context, these metrics could lead to false assumptions. Hypothetically considering only the *duplication ratio*, *misassemblies*, percentage of *uncovered bases*, *complete and duplicated BUSCOs* and *fragmented BUSCOs*, `Grouper` would have scored the best overall summed score among all assemblers and clustering tools. Therefore, a broadly and carefully selected repertoire is advised to balance out possible biased metrics.

Furthermore, the evaluation of a single *de novo* transcriptome assembly by itself might be misleading or could lead to wrong conclusions. Depending on the condition of an organism, cells may have a reduced set of expressed genes. For example, the paired-end *E. coli* dataset (ERR779269), which was part of a study to investigate the pathogenicity of ArcA (aerobic respiratory control) [84], has significantly less assembled transcripts than the single-end *E. coli* dataset (Figure 18). ArcA represses a wide variety of aerobic enzymes under anaerobic conditions [85]. Hence, the chosen sample from the study might be the control group, where ArcA is fully functional. Thus the amount of expressed genes is low. It would be critical to state that a single *de novo* transcriptome assembly is bad because of some metrics, without respecting the conditions of the sample. This is especially important to consider for quantitative metrics with a reference, such as *BUSCOs*. However, for comparative analyses, as in this master's thesis or for a comprehensive study on different assemblers [20], such metrics provide an excellent way to measure characteristic differences.

4.8 K-mer size may affect the assembly quality of different species

Comparing the performance of the assemblers throughout the different datasets, it appears that `SOAPdenovo-Trans` outperforms `rnaSPAdes` and `Trinity` for the *E. coli* datasets and vice versa (Figure 21-25). In this study, `SOAPdenovo-Trans` uses significantly larger k-mer sizes than the other assemblers (See Section 2.2), which could be one reason for the better results.

As a prokaryote, *E. coli* doesn't perform alternative splicing, contrarily to *C. elegans* [86]. Hence, each gene has only one transcript. As Surget-Groba and Montoya-Burgos [87] suggest in their study, different k-mer sizes affect the profile of a transcriptome assembly. While shorter k-mer lengths will benefit the transcript diversity,

which leads to more and fragmented transcripts, longer k-mer sizes allow for higher contiguity but with lower transcript diversity [87]. Since *C. elegans* likely has a higher transcript diversity because of alternative splicing, `SOAPdenovo-Trans` will score worse for those datasets due to a large k-mer size. In return, this favors the assemblies from `SOAPdenovo-Trans` for the *E. coli* datasets.

Another reason for this could be the different implementations of the assemblers, leading to beneficial assemblies for `SOAPdenovo-Trans`. In a study Rana *et al.* state that a distinct assembler has a higher impact on the assembly than k-mer sizes. [22]

4.9 Future work leaves room for improvements

Contrarily to `karma`, the `Oyster River Protocol` is limited to paired-end data only. Hence, `karma` is still a respectful competitor to the `Oyster River Protocol`, because as described in Section 4.1, the most significant improvements through clustering were achieved on single-end datasets. Now, one might argue that the vast majority of new RNA-Seq data sets is produced using paired-end protocols. However, single-molecule real-time sequencing approaches such as provided by PacBio [88] or Oxford Nanopore [89] are more frequently used to sequence transcriptomes [90]. Here, much longer reads are achieved, however, without any paired information. As for the future, `karma` could also be used in a modified way to enhance Nanopore RNA-Seq data. A basic idea could be a hybrid concept, that complements the longer reads with short read information from Illumina. The long reads could be initially k-mer clustered, while the quasi-mapping of the short reads might help to resolve isoforms. Such hybrid sequencing methods have been recently used for a *de novo* transcriptome assembly and annotation, which showed an increased diversity of isoforms [91].

Furthermore, a lot of time of this master's thesis was spent evaluating a broad spectrum of clustering methods on *de novo* transcriptome assemblies. Thus the time for implementing the algorithm was limited. However, there are a few options and ideas that may improve the results of `karma`. As already mentioned, the biggest flaw for `karma` seems to be the size of the assembly. To address this issue, one could choose different approaches to increase the size of the clustered assembly.

The main issue seems to be the k-mer based clusters. On the one hand, more parameter tests can be performed, with regards to the number of clusters built or evaluation metrics (Section 3.6). On the other hand, the method could be swapped against different clustering methods that may work better, such as self-organizing maps [92] or Gaussian mixture models [93].

Again, the MCL clustering seems to work for small clusters (Figure 11) but fails for huge clusters (Section 4.5.1). Maybe the inflation parameter for MCL could be increased to get a finer clustering result and consequently more sequences.

Another potential improvement is possible for the cluster redefinition process of karma (Figure 8C). Up to now, the k-mer based clusters, are iteratively combined with the large 'unlabeled' group to remove or add sequences from those clusters. Since a contig can also share reads with several other contigs, it might be better to first assign the 'unlabeled' contigs to all k-mer based groups and see where it fits best. A contig would then be assigned only to the cluster in which it has the highest number of connections or weight of edges.

5 Conclusion

In this master's thesis, the aim was to combine multiple *de novo* transcriptome assemblies, concatenate them and produce a clustering that will outperform the assemblies from single assemblers. Given the facts from the discussion, it seems sufficient to rely on a single assembler rather than combining multiple assemblers for the most use cases. However, the assembly might be slightly less complete compared to a clustered assembly. Eventually, the preprocessing steps, assembler choice and suitable k-mer size should be carefully considered. Furthermore, for RNA-Seq experiments where only single-end reads are available it can be advantageous to perform multiple assemblies and to cluster them with `cd-hit-est` with a 90 % sequence identity for improved results. Even though `karma` was not able to outperform other clustering methods, mainly due to the size of the resulting clustered assembly, it still has great potential. Contrarily to the `Oyster River Protocol`, it is also able to cluster *de novo* transcriptome assemblies utilizing the coverage information from single-end reads which might become more important with the rise of SMRT sequencing technologies producing long single-end reads. With further improvements of the algorithm in the future, `karma` may still be a viable candidate for clustering *de novo* transcriptome assemblies.

6 References

[1] Wang Z., Gerstein M., and Snyder M. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57, 2009.

[2] Hrdlickova R., Toloue M., and Tian B. Rna-seq methods for transcriptome analysis. *Wiley Interdisciplinary Reviews: RNA*, 8(1):e1364, 2017.

[3] Fitzpatrick M. J., Ben-Shahar Y., Smid H. M., Vet L. E., Robinson G. E., and Sokolowski M. B. Candidate genes for behavioural ecology. *Trends in Ecology & Evolution*, 20(2):96–104, 2005.

[4] Bhargava A., Clabaugh I., To J. P., Maxwell B. B., Chiang Y.-H., Schaller G. E., Loraine A., and Kieber J. J. Identification of cytokinin-responsive genes using microarray meta-analysis and rna-seq in arabidopsis. *Plant Physiology*, 162(1):272–294, 2013.

[5] Tarca A. L., Romero R., and Draghici S. Analysis of microarray experiments of gene expression profiling. *American journal of obstetrics and gynecology*, 195 (2):373–388, 2006.

[6] Pubmed . Search results for the term 'rna-seq'. `https://www.ncbi.nlm.nih.gov/pubmed/?term=RNA-Seq`, 2019. [Online; accessed 2019-03-18].

[7] Rodríguez-García A., Sola-Landa A., and Barreiro C. Rna-seq-based comparative transcriptomics: Rna preparation and bioinformatics. In *Microbial Steroids*, pages 59–72. Springer, New York, NY, USA, 2017.

[8] Wang J., Dean D. C., Hornicek F. J., Shi H., and Duan Z. Rna sequencing (rna-seq) and its application in ovarian cancer. *Gynecologic oncology*, 152(1): 194–201, 2019.

[9] Rai M. F., Tycksen E. D., Sandell L. J., and Brophy R. H. Advantages of rna-seq compared to rna microarrays for transcriptome profiling of anterior cruciate ligament tears. *Journal of Orthopaedic Research*, 36(1):484–497, 2018.

[10] Black D. L. Mechanisms of alternative pre-messenger rna splicing. *Annual review of biochemistry*, 72(1):291–336, 2003.

[11] Hüttenhofer A., Schattner P., and Polacek N. Non-coding rnas: hope or hype? *TRENDS in Genetics*, 21(5):289–297, 2005.

[12] Ozsolak F. and Milos P. M. Rna sequencing: advances, challenges and opportunities. *Nature reviews genetics*, 12(2):87, 2011.

[13] Martin J. A. and Wang Z. Next-generation transcriptome assembly. *Nature Reviews Genetics*, 12(10):671, 2011.

[14] Munkley J., Maia T. M., Ibarluzea N., Livermore K. E., Vodak D., Ehrmann I., James K., Rajan P., Barbosa-Morais N. L., and Elliott D. J. Androgen-dependent alternative mrna isoform expression in prostate cancer cells. *F1000Research*, 7, 2018.

[15] Ono H., Ishii K., Kozaki T., Ogiwara I., Kanekatsu M., and Yamada T. Removal of redundant contigs from de novo rna-seq assemblies via homology search improves accurate detection of differentially expressed genes. *BMC genomics*, 16 (1):1031, 2015.

[16] Xie Y., Wu G., Tang J., Luo R., Patterson J., Liu S., Huang W., He G., Gu S., Li S., and others . Soapdenovo-trans: de novo transcriptome assembly with short rna-seq reads. *Bioinformatics*, 30(12):1660–1666, 2014.

[17] Grabherr M. G., Haas B. J., Yassour M., Levin J. Z., Thompson D. A., Amit I., Adiconis X., Fan L., Raychowdhury R., Zeng Q., and others . Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, 29(7):644, 2011.

[18] Robertson G., Schein J., Chiu R., Corbett R., Field M., Jackman S. D., Mungall K., Lee S., Okada H. M., Qian J. Q., and others . De novo assembly and analysis of rna-seq data. *Nature methods*, 7(11):909, 2010.

[19] Bushmanova E., Antipov D., Lapidus A., and Prjibelski A. D. rnaspades: a de novo transcriptome assembler and its application to rna-seq data. *GigaScience*, 8(9):giz100, 2019.

[20] Hölzer M. and Marz M. De novo transcriptome assembly: A comprehensive cross-species comparison of short-read rna-seq assemblers. *GigaScience*, 8(5): giz039, 2019.

[21] Compeau P. E., Pevzner P. A., and Tesler G. Why are de bruijn graphs useful for genome assembly? *Nature biotechnology*, 29(11):987, 2011.

[22] Rana S. B., Zadlock IV F. J., Zhang Z., Murphy W. R., and Bentivegna C. S. Comparison of de novo transcriptome assemblers and k-mer strategies using the killifish, fundulus heteroclitus. *PLoS One*, 11(4):e0153104, 2016.

[23] Chikhi R. and Medvedev P. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2013.

[24] MacManes M. D. The oyster river protocol: a multi-assembler and kmer approach for de novo transcriptome assembly. *PeerJ*, 6:e5428, 2018.

[25] Gross J. L., Yellen J., and Zhang P. *Handbook of graph theory*. Chapman and Hall/CRC, New York, 2nd edition, 2013.

[26] Saxena A., Prasad M., Gupta A., Bharill N., Patel O. P., Tiwari A., Er M. J., Ding W., and Lin C.-T. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

[27] Ferreira L. and Hitchcock D. B. A comparison of hierarchical methods for clustering functional data. *Communications in Statistics-Simulation and Computation*, 38(9):1925–1949, 2009.

[28] Saraçli S., Doğan N., and Doğan İ. Comparison of hierarchical cluster analysis methods by cophenetic correlation. *Journal of Inequalities and Applications*, 2013(1):203, 2013.

[29] Wagstaff K., Cardie C., Rogers S., Schrödl S., and others . Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.

[30] Uppada S. K. Centroid based clustering algorithms - a clarion study. *Int. J. Comput. Sci. Inf. Technol*, 5:7309–7313, 2014.

[31] Kotsiantis S. and Pintelas P. Recent advances in clustering: A brief survey. *WSEAS Transactions on Information Science and Applications*, 1(1):73–81, 2004.

[32] He Z., Xu X., and Deng S. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[33] Maugis C., Celeux G., and Martin-Magniette M.-L. Variable selection for clustering with gaussian mixture models. *Biometrics*, 65(3):701–709, 2009.

[34] Ester M., Kriegel H.-P., Sander J., Xu X., and others . A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[35] Kriegel H.-P. and Pfeifle M. Density-based clustering of uncertain data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 672–677, New York, NY, USA, 2005. ACM.

[36] Kriegel H.-P., Kröger P., Sander J., and Zimek A. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3): 231–240, 2011.

[37] Cheng Y. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[38] Bora D. J. and Gupta D. A. K. A comparative study between fuzzy clustering algorithm and hard clustering algorithm. *International Journal of Computer Trends and Technology*, 10(2):108–113, 2014.

[39] Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. Uniref: comprehensive and non-redundant uniprot reference clusters. *Bioinformatics*, 23(10):1282–1288, 2007.

[40] Li W. and Godzik A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[41] Fu L., Niu B., Zhu Z., Wu S., and Li W. Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[42] Edgar R. C. Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461, 2010.

[43] Steinegger M. and Söding J. Clustering huge protein sequence sets in linear time. *Nature communications*, 9(1):2542, 2018.

[44] James B. T., Luczak B. B., and Girgis H. Z. Meshclust: an intelligent tool for clustering dna sequences. *Nucleic acids research*, 46(14):e83–e83, 2018.

[45] Lin H.-H. and Liao Y.-C. Accurate binning of metagenomic contigs via automated clustering sequences using information of genomic signatures and marker genes. *Scientific reports*, 6:24175, 2016.

[46] Malik L., Almodaresi F., and Patro R. Grouper: graph-based clustering and annotation for improved de novo transcriptome analysis. *Bioinformatics*, 34 (19):3265–3272, 2018.

[47] Xu D. and Tian Y. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[48] Zou Q., Lin G., Jiang X., Liu X., and Zeng X. Sequence clustering in bioinformatics: an empirical study. *Briefings in bioinformatics*, 2018.

[49] Griebel T., Zacher B., Ribeca P., Raineri E., Lacroix V., Guigó R., and Sammeth M. Modelling and simulating generic rna-seq experiments with the flux simulator. *Nucleic acids research*, 40(20):10073–10083, 2012.

[50] Chen S., Zhou Y., Chen Y., and Gu J. fastp: an ultra-fast all-in-one fastq preprocessor. *Bioinformatics*, 34(17):i884–i890, 2018.

[51] Li W., Jaroszewski L., and Godzik A. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17(3): 282–283, 2001.

[52] Bolger A. M., Lohse M., and Usadel B. Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, 30(15):2114–2120, 2014.

[53] Song L. and Florea L. Rcorrector: efficient and accurate error correction for illumina rna-seq reads. *GigaScience*, 4(1):48, 2015.

[54] Kannan S., Hui J., Mazooji K., Pachter L., and Tse D. Shannon: An information-optimal de novo rna-seq assembler. *bioRxiv*, page 039230, 2016.

[55] Emms D. M. and Kelly S. Orthofinder: solving fundamental biases in whole genome comparisons dramatically improves orthogroup inference accuracy. *Genome biology*, 16(1):157, 2015.

[56] Smith-Unna R., Boursnell C., Patro R., Hibberd J. M., and Kelly S. Transrate: reference-free quality assessment of de novo transcriptome assemblies. *Genome research*, 26(8):1134–1144, 2016.

[57] Patro R., Duggal G., Love M. I., Irizarry R. A., and Kingsford C. Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4):417, 2017.

[58] Van Dongen S. Graph clustering via a discrete uncoupling process. *SIAM Journal on Matrix Analysis and Applications*, 30(1):121–141, 2008.

[59] James B. and Girgis H. Z. Meshclust2: Application of alignment-free identity scores in clustering long dna sequences. *bioRxiv*, page 451278, 2018.

[60] Tao H., Bausch C., Richmond C., Blattner F. R., and Conway T. Functional genomics: expression analysis of escherichia coli growing on minimal and rich media. *Journal of bacteriology*, 181(20):6425–6440, 1999.

[61] Wang T., Johnson T., Zhang J., and Huang K. Topological methods for visualization and analysis of high dimensional single-cell rna sequencing data. In *Pacific Symposium on Biocomputing*, volume 24, pages 350–361. World Scientific, 2019.

[62] McInnes L., Healy J., and Melville J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[63] Dong W., Moses C., and Li K. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, New York, NY, USA, 2011. ACM.

[64] Becht E., McInnes L., Healy J., Dutertre C.-A., Kwok I. W., Ng L. G., Ginhoux F., and Newell E. W. Dimensionality reduction for visualizing single-cell data using umap. *Nature biotechnology*, 37(1):38–44, 2019.

[65] McInnes L., Healy J., and Astels S. hdbscan: Hierarchical density based clustering. *J. Open Source Software*, 2(11):205, 2017.

[66] Beaulaurier J., Luo E., Eppley J., Den Uyl P., Dai X., Turner D. J., Pendelton M., Juul S., Harrington E., and DeLong E. F. Assembly-free single-molecule nanopore sequencing recovers complete virus genomes from natural microbial communities. *bioRxiv*, page 619684, 2019.

[67] Altschul S. F., Gish W., Miller W., Myers E. W., and Lipman D. J. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[68] Li B., Fillmore N., Bai Y., Collins M., Thomson J. A., Stewart R., and Dewey C. N. Evaluation of de novo transcriptome assemblies from rna-seq data. *Genome biology*, 15(12):553, 2014.

[69] Bushmanova E., Antipov D., Lapidus A., Suvorov V., and Prjibelski A. D. rnaquast: a quality assessment tool for de novo transcriptome assemblies. *Bioinformatics*, 32(14):2210–2212, 2016.

[70] Kim D., Langmead B., and Salzberg S. L. Hisat: a fast spliced aligner with low memory requirements. *Nature methods*, 12(4):357, 2015.

[71] Simão F. A., Waterhouse R. M., Ioannidis P., Kriventseva E. V., and Zdobnov E. M. Busco: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212, 2015.

[72] Waterhouse R. M., Seppey M., Simão F. A., Manni M., Ioannidis P., Klioutchnikov G., Kriventseva E. V., and Zdobnov E. M. Busco applications from quality assessments to gene prediction and phylogenomics. *Molecular biology and evolution*, 35(3):543–548, 2017.

[73] Edgar R. C. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[74] Khan A. and Mathelier A. Intervene: a tool for intersection and visualization of multiple gene or genomic region sets. *BMC bioinformatics*, 18(1):287, 2017.

[75] Waterhouse A. M., Procter J. B., Martin D. M., Clamp M., and Barton G. J. Jalview version 2—a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, 2009.

[76] Köster J. and Rahmann S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[77] Eddy S. R. Accelerated profile hmm searches. *PLoS computational biology*, 7 (10):e1002195, 2011.

[78] Muggli M. D., Puglisi S. J., Ronen R., and Boucher C. Misassembly detection using paired-end sequence reads and optical mapping data. *Bioinformatics*, 31 (12):i80–i88, 2015.

[79] Hwang K., Oh J., Kim T.-K., Kim B. K., Yu D. S., Hou B. K., Caetano-Anollés G., Hong S. G., and Kim K. M. Clustom: a novel method for clustering 16s rrna next generation sequences by overlap minimization. *PloS one*, 8(5):e62623, 2013.

[80] Consortium U. Uniprot: a worldwide hub of protein knowledge. *Nucleic acids research*, 47(D1):D506–D515, 2018.

[81] Jia F., Cui M., Than M. T., and Han M. Developmental defects of caenorhabditis elegans lacking branched-chain α-ketoacid dehydrogenase are mainly caused by monomethyl branched-chain fatty acid deficiency. *Journal of Biological Chemistry*, 291(6):2967–2973, 2016.

[82] Galy V., Mattaj I. W., and Askjaer P. Caenorhabditis elegans nucleoporins nup93 and nup205 determine the limit of nuclear pore complex size exclusion in vivo. *Molecular biology of the cell*, 14(12):5104–5115, 2003.

[83] Hertel P., Daniel J., Stegehake D., Vaupel H., Kailayangiri S., Gruel C., Woltersdorf C., and Liebau E. The ubiquitin-fold modifier 1 (ufm1) cascade of caenorhabditis elegans. *Journal of Biological Chemistry*, 288(15):10661–10671, 2013.

[84] Jiang F., An C., Bao Y., Zhao X., Jernigan R. L., Lithio A., Nettleton D., Li L., Wurtele E. S., Nolan L. K., and others . Arca controls metabolism, chemotaxis, and motility contributing to the pathogenicity of avian pathogenic escherichia coli. *Infection and immunity*, 83(9):3545–3554, 2015.

[85] Iuchi S. and Lin E. arca (dye), a global regulatory gene in escherichia coli mediating repression of enzymes in aerobic pathways. *Proceedings of the National Academy of Sciences*, 85(6):1888–1892, 1988.

[86] Tan J. H. and Fraser A. G. The combinatorial control of alternative splicing in c. elegans. *PLoS genetics*, 13(11):e1007033, 2017.

[87] Surget-Groba Y. and Montoya-Burgos J. I. Optimization of de novo transcriptome assembly from next-generation sequencing data. *Genome research*, 20 (10):1432–1440, 2010.

[88] Eid J., Fehr A., Gray J., Luong K., Lyle J., Otto G., Peluso P., Rank D., Baybayan P., Bettman B., and others . Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[89] Jain M., Olsen H. E., Paten B., and Akeson M. The oxford nanopore minion: delivery of nanopore sequencing to the genomics community. *Genome biology*, 17(1):239, 2016.

[90] Weirather J. L., Cesare M.de, Wang Y., Piazza P., Sebastiano V., Wang X.-J., Buck D., and Au K. F. Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis. *F1000Research*, 6, 2017.

[91] Fu S., Ma Y., Yao H., Xu Z., Chen S., Song J., and Au K. F. Idp-denovo: de novo transcriptome assembly and isoform annotation by hybrid sequencing. *Bioinformatics*, 34(13):2168–2176, 2018.

[92] Nan F., Li Y., Jia X., Dong L., and Chen Y. Application of improved som network in gene data cluster analysis. *Measurement*, 145:370–378, 2019.

[93] Liu Z., Song Y.-q., Xie C.-h., and Tang Z. A new clustering method of gene expression data based on multivariate gaussian mixture models. *Signal, Image and Video Processing*, 10(2):359–368, 2016.

7 List of figures

8 List of tables

9 Appendix

9.1 Pipelines

- Assembly: https://github.com/lmfaber/multi_assembly
- Clustering: https://github.com/lmfaber/clustering.
- Evaluation: https://github.com/lmfaber/evaluation

9.2 `karma`

- GitHub: https://github.com/lmfaber/karma
- Anaconda Cloud: https://anaconda.org/lmfaber/karma

9.3 Supplemental data

The sequence alignments and the unnormalized evaluation metrics are available as printed version in this chapter. Additionally, everything else can be accessed through the online supplemental data in the following GitHub repository:
https://github.com/lmfaber/master_thesis/tree/master/supplemental_data
Subfolders:

- /assemblies_for_eval: Contains all assemblies from all datasets, on which clusterings were performed.
- /assembly_eval: Contains all evaluation metrics (normalized and unnormalized) of all evaluated assemblies.
- /clusterings: Contains all outputs from clustering tools from all datasets.
- /flux_simulator: Contains the parameter file for the `Flux Simulator` and the simulated reads.
- /graphs: Contains all clustering graphs from `karma` for the simulated *C. elegans* dataset.
- /parameter_evaluation: Complements the information of Table 6. Contains the full list of results of all tested parameters for the simulated *C. elegans* dataset.
- /runtime: Contains the runtimes of all programs.

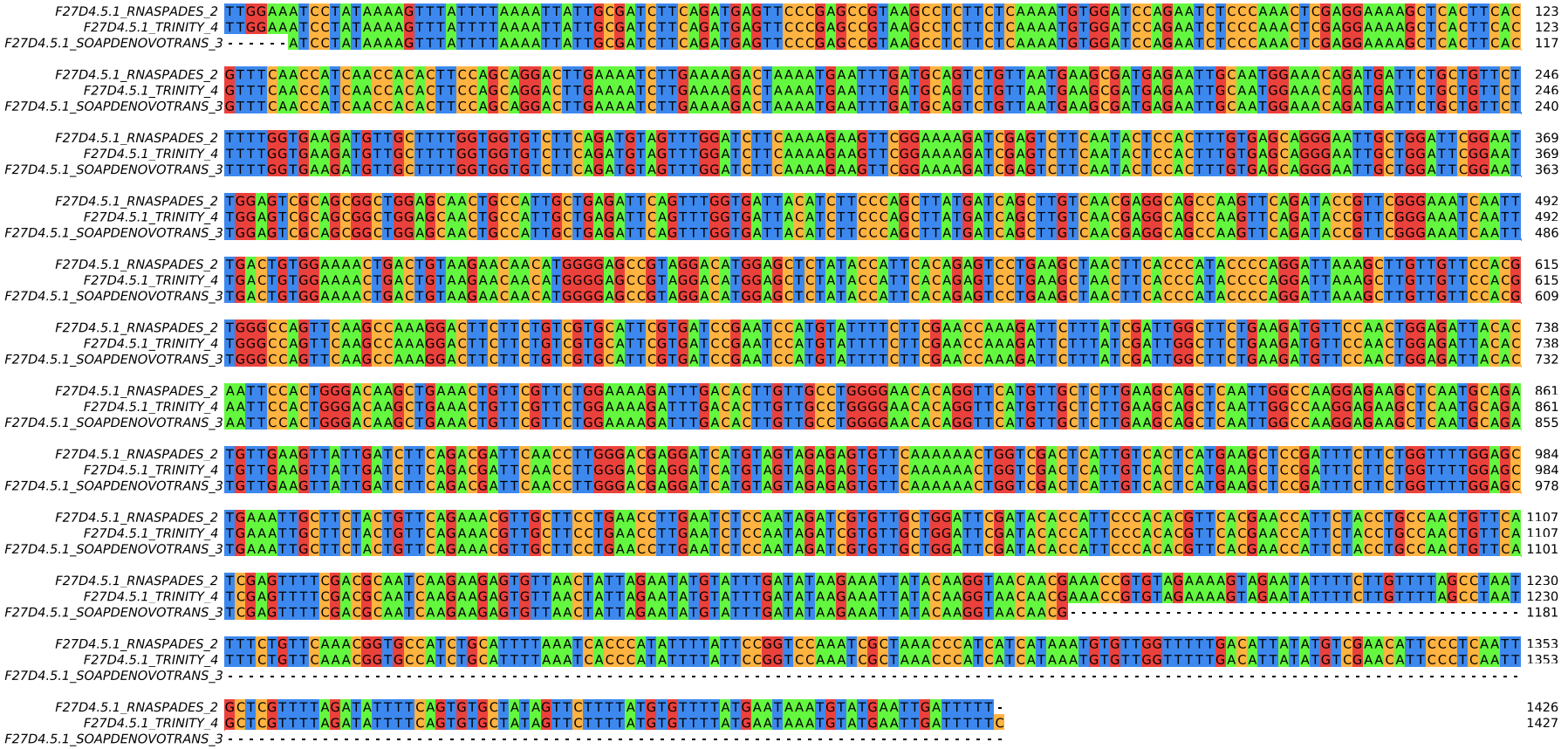

Figure 26: Sequence alignment of the transcripts from Figure 10B. The sequences from the final cluster 28 were aligned with `MUSCLE` [73] and visualized with `Jalview` [75]. Hyphens represent mismatches between sequences.

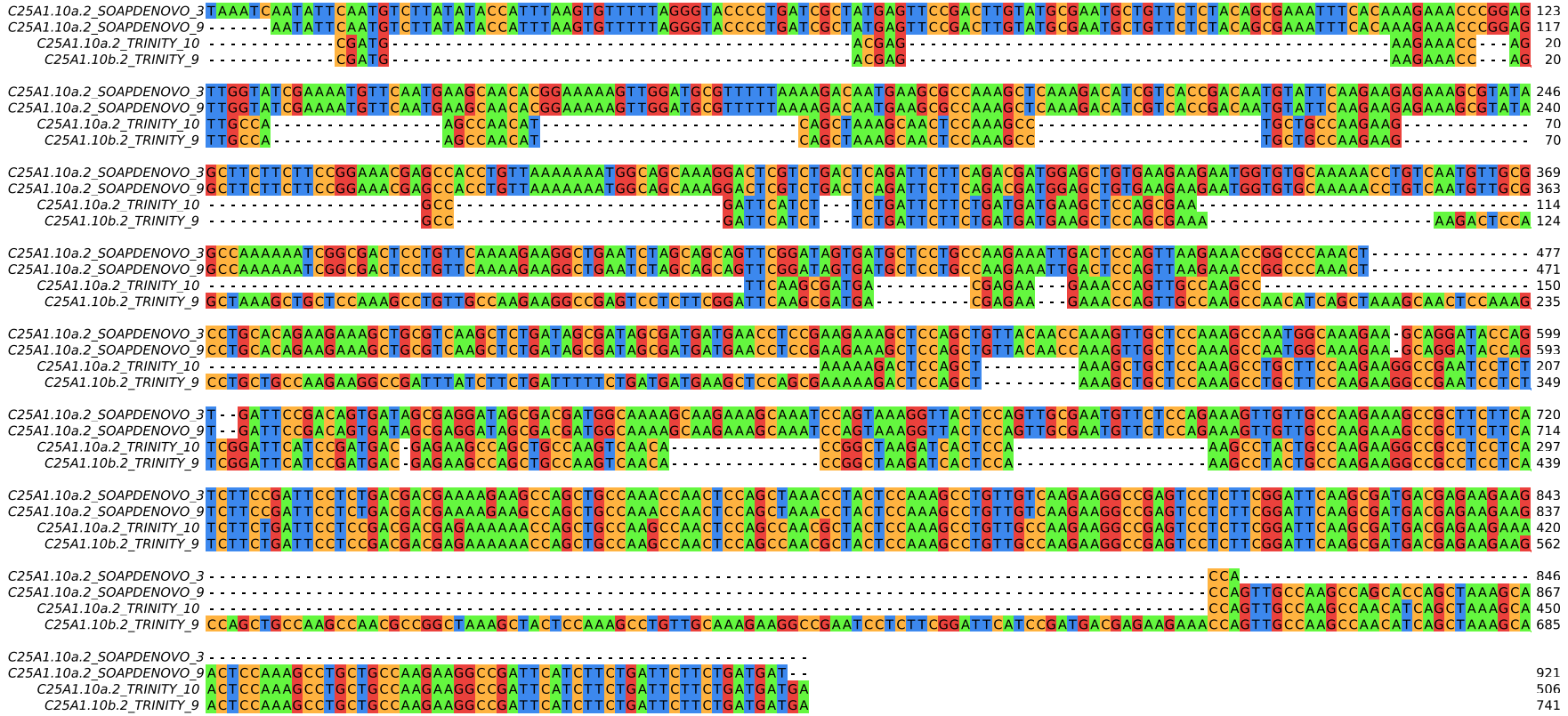

Figure 27: Sequence alignment of the transcripts from Figure 11. The sequences from the final cluster 455 were aligned with `MUSCLE` [73] and visualized with `Jalview` [75]. Hyphens represent mismatches between sequences.

Figure 28: Sequence alignment of the transcripts from Figure 12. The sequences from the final cluster 660 were aligned with `MUSCLE` [73] and visualized with `Jalview` [75]. Hyphens represent mismatches between sequences.

Table 7: Assembly evaluation metrics for the paired-end *C. elegans* dataset (SRR5456164). BUSCOs (CS): Complete and single BUSCOs, BUSCOs (CD): Complete and duplicated BUSCOs, BUSCOs (F): Fragmented BUSCOs, BUSCOs (M): Missing BUSCOs, ORF: Open reading frame

	rnaSPAdes	SOAPdenovo-Trans	Trinity	Combined	cd-hit-est (c=1)	cd-hit-est (c=0.9)	Linclust
Database coverage	0.151	0.069	0.098	0.173	0.164	0.154	0.169
Duplication ratio	1.05	1.101	1.038	1.949	1.365	1.07	1.554
95 %-assembled isoforms	417	90	283	482	475	434	379
Misassemblies	688	15	248	951	904	782	851
Mean ORF percentage	81.30865	92.09741	90.10208	86.02784	83.73223	81.88296	85.47737
Assembly score	0.36624	0.10638	0.1889	0.13718	0.28287	0.35542	0.17099
Uncovered bases percentage	0.06144	0.03603	0.02127	0.25763	0.1214	0.06866	0.20535
KC score	0.950132	0.835138	0.912559	0.975307	0.977535	0.959043	0.96687
RSEM EVAL	-1770045340.73	-3613516656.57	-2632714766.87	-1430506439.82	-1423168410.06	-1604350521.67	-1579390663.72
Nucleotide F1	0.654158	0.567724	0.662377	0.432028	0.549393	0.649173	0.505907
Contig F1	0.0374303	0.0690062	0.0770611	0.0753873	0.0547396	0.0379768	0.0772868
BUSCOs (CS)	284	116	228	90	176	288	238
BUSCOs (CD)	19	29	23	221	135	19	63
BUSCOs (F)	137	98	121	140	140	144	147
BUSCOs (M)	542	739	610	531	531	531	534
Ex90N50	24895	11088	10579	33393	28380	24861	32160
Remapping rate	0.0795	0.1571	0.1236	0.0419	0.0589	0.0769	0.05234

	MeShClust	MeShClust^2	Grouper	Grouper*	karma	karma*	Oyster River Protocol
Database coverage	0.08	0.163	0	0	0.044	0.019	0.135
Duplication ratio	1.245	1.656	1	1	1.04	1.02	1.199
95 %-assembled isoforms	171	422	1	1	181	34	299
Misassemblies	270	751	0	0	117	59	1083
Mean ORF percentage	87.17259	86.42228	69.92872	69.92872	85.70663	87.05276	84.1247
Assembly score	0.09723	0.14648	0.00617	0.00617	0.11471	0.02165	0.43475
Uncovered bases percentage	0.11949	0.21675	0.01356	0.01356	0.03385	0.03957	0.06177
KC score	0.544173	0.925971	0.0159385	0.0159385	0.566657	0.16227	0.917622
RSEM EVAL	-4665266225.96	-1875165073.13	-7881078430.36	-7881078430.36	-4644800060.3	-7046835191.71	-1749138759.43
Nucleotide F1	0.464802	0.483952	0.00565165	0.00565165	0.400357	0.193878	0.597937
Contig F1	0.0485081	0.0750382	4.89E-05	4.89E-05	0.0377662	0.0180084	0.0421688
BUSCOs (CS)	115	152	1	1	137	31	247
BUSCOs (CD)	47	144	0	0	8	1	25
BUSCOs (F)	89	136	0	0	56	27	127
BUSCOs (M)	731	550	981	981	781	923	583
Ex90N50	14827	31540	34	34	4557	2131	21464
Remapping rate	0.0756	0.0494	0.4094	0.4094	0.1626	0.1195	0.0775

Table 8: Assembly evaluation metrics for the single-end *C. elegans* dataset (ERR2886543). BUSCOs (CS): Complete and single BUSCOs, BUSCOs (CD): Complete and duplicated BUSCOs, BUSCOs (F): Fragmented BUSCOs, BUSCOs (M): Missing BUSCOs, ORF: Open reading frame

	rnaSPAdes	SOAPdenovo-Trans	Trinity	Combined	cd-hit-est (c=1)	cd-hit-est (c=0.9)	Linclust
Database coverage	0.249	0.274	0.244	0.311	0.293	0.269	0.292
Duplication ratio	1.026	1.1	1.027	2.599	1.401	1.027	1.288
95 %-assembled isoforms	3845	3864	4304	4901	4783	4478	4334
Misassemblies	29	11	72	112	100	76	76
KC score	0.929661	0.942305	0.929062	0.929203	0.949659	0.940393	0.944621
RSEM EVAL	-489761901.22	-485577948.13	-495633718.12	-514513375.08	-482131738.91	-480112050.53	-484019914.15
Nucleotide F1	0.773771	0.756324	0.774879	0.437381	0.657823	0.78824	0.683277
Contig F1	0.113996	0.109491	0.135287	0.0928126	0.117546	0.124116	0.107675
BUSCOs (CS)	720	654	756	28	418	761	714
BUSCOs (CD)	29	118	27	762	373	27	74
BUSCOs (F)	51	51	48	44	44	45	44
BUSCOs (M)	182	159	151	148	147	149	150
Ex90N50	12206	14730	7522	20003	18022	17253	19470
Remapping rate	0.6211	0.5376	0.6216	0.2005	0.3899	0.5703	0.4246

	MeShClust	MeShClust^2	Grouper	Grouper*	karma	karma*
Database coverage	0.272	0.06	0.001	0.001	0.113	0.032
Duplication ratio	1.775	1.266	1	1	1.035	1.014
95 %-assembled isoforms	4173	121	1	1	1938	589
Misassemblies	76	6	0	0	23	4
KC score	0.869096	0.113817	0.000958855	0.000958855	0.537434	0.155587
RSEM EVAL	-574208935.19	-1431812964.55	-1579133183.78	-1579133183.78	-948072484.18	-1399343882.81
Nucleotide F1	0.556362	0.222673	0.00350515	0.00350515	0.459621	0.159376
Contig F1	0.0987958	0.00922742	5.89E-05	5.89E-05	0.0677556	0.0222713
BUSCOs (CS)	219	11	0	0	372	127
BUSCOs (CD)	546	1	0	0	15	3
BUSCOs (F)	46	14	0	0	29	11
BUSCOs (M)	171	956	982	982	566	841
Ex90N50	19023	11292	6277	6277	2707	358
Remapping rate	0.3101	0.2768	0.2684	0.2684	0.7802	0.8119

Table 9: Assembly evaluation metrics for the paired-end *E. coli* dataset (ERR779269). BUSCOs (CS): Complete and single BUSCOs, BUSCOs (CD): Complete and duplicated BUSCOs, BUSCOs (F): Fragmented BUSCOs, BUSCOs (M): Missing BUSCOs, ORF: Open reading frame

	rnaSPAdes	SOAPdenovo-Trans	Trinity	Combined	cd-hit-est (c=1)	cd-hit-est (c=0.9)	Linclust
Database coverage	0.065	0.194	0.082	0.224	0.111	0.059	0.218
Duplication ratio	1.161	1.177	1.277	1.825	1.621	1.128	1.485
95 %-assembled isoforms	130	357	158	459	224	117	444
Misassemblies	233	146	157	536	414	234	408
Mean ORF percentage	12.86926	28.65736	14.79746	17.48951	13.59771	11.93826	18.13838
Assembly score	0.13693	0.22281	0.5385	0.07283	0.11412	0.13419	0.0897
Uncovered bases percentage	0.14179	0.02989	0.08143	0.34989	0.29899	0.11163	0.2692
KC score	0.34177	0.347566	0.342539	0.336395	0.340589	0.344052	0.340153
RSEM EVAL	-783567110.5	-790600958.27	-758163030.28	-723526184.49	-716761173.43	-765690474.66	-720900920.72
Nucleotide F1	0.429352	0.611536	0.49515	0.203084	0.263362	0.417534	0.26169
Contig F1	0.00105694	0.00136593	0.000363769	0.00133869	0.000883197	0.000530434	0.00148134
BUSCOs (CS)	286	371	332	113	177	258	153
BUSCOs (CD)	155	144	124	484	379	182	440
BUSCOs (F)	20	10	11	0	8	13	3
BUSCOs (M)	320	256	314	184	217	328	185
Ex90N50	61	222	327	246	254	601	230
Remapping rate	0.0877	0.1447	0.1041	0.0358	0.0484	0.0857	0.0487

	MeShClust	MeShClust^2	Grouper	Grouper*	karma	karma*	Oyster River Protocol
Database coverage	0.216	0.029	0.001	0.001	0.04	0.016	0.219
Duplication ratio	1.606	1.187	1	1	1.139	1.041	1.312
95 %-assembled isoforms	435	21	1	1	63	20	335
Misassemblies	404	25	1	1	93	23	1586
Mean ORF percentage	21.2563	46.24479	2.36694	2.36694	12.18414	15.7588	26.77923
Assembly score	0.08534	0.01459	0.06134	0.06134	0.06614	0.00506	0.22179
Uncovered bases percentage	0.25007	0.07262	0.00037	0.00037	0.06844	0.03047	0.17172
KC score	0.34116	0.05506	0.0216994	0.0216994	0.288863	0.0424645	0.2901
RSEM EVAL	-723686090.15	-3860650563.98	-4279875760.58	-4279875760.58	-1790575524.2	-3991884547.23	-801951086.83
Nucleotide F1	0.284619	0.147295	0.0747459	0.0747459	0.44664	0.181951	0.38483
Contig F1	0.00124644	0.00067325	0	0	0.00113723	0	0.000572355
BUSCOs (CS)	154	34	8	8	221	35	308
BUSCOs (CD)	428	15	5	5	74	15	190
BUSCOs (F)	4	22	0	0	13	7	44
BUSCOs (M)	195	710	768	768	473	724	239
Ex90N50	228	120	113	113	62	72	92
Remapping rate	0.0536	0.1304	0.2928	0.2928	0.1452	0.1675	0.0776

Table 10: Assembly evaluation metrics for the single-end *E. coli* dataset (SRR4255368). BUSCOs (CS): Complete and single BUSCOs, BUSCOs (CD): Complete and duplicated BUSCOs, BUSCOs (F): Fragmented BUSCOs, BUSCOs (M): Missing BUSCOs, ORF: Open reading frame

	rnaSPAdes	SOAPdenovo-Trans	Trinity	Combined	cd-hit-est (c=1)	cd-hit-est (c=0.9)	Linclust
Database coverage	0.271	0.406	0.197	0.418	0.41	0.404	0.413
Duplication ratio	1.026	1.111	1.008	2.219	1.277	1.035	1.351
95 %-assembled isoforms	188	238	228	288	268	248	273
Misassemblies	1	2	5	8	7	7	7
KC score	0.798691	0.855741	0.769881	0.865486	0.868888	0.867401	0.861875
RSEM EVAL	-306072610.64	-280456633.52	-328451127.12	-275469820.25	-273036723.67	-274948642.84	-275714130.72
Nucleotide F1	0.500731	0.499008	0.435121	0.351697	0.476455	0.529708	0.459695
Contig F1	0.0186955	0.061932	0.0102523	0.0520025	0.0630086	0.0661045	0.0589524
BUSCOS (CS)	107	111	133	24	95	141	108
BUSCOS (CD)	0	25	0	124	50	0	32
BUSCOS (F)	270	287	262	294	294	292	291
BUSCOS (M)	404	358	386	339	342	348	350
Ex90N50	3933	7388	1193	9123	7884	7434	8410
Remapping rate	0.0584	0.03761	0.0738	0.01843	0.0320	0.0404	0.0306

	MeShClust	MeShClust^2	Grouper	Grouper*	karma	karma*
Database coverage	0.393	0.404	0.001	0.001	0.139	0.036
Duplication ratio	1.801	1.714	1	1	1.029	1.011
95 %-assembled isoforms	269	237	3	3	137	45
Misassemblies	7	8	0	0	4	0
KC score	0.77146	0.765025	0.0475044	0.0475044	0.508967	0.185939
RSEM EVAL	-321998360.25	-322589311.48	-784332546.24	-784332546.24	-457645275.72	-694202626.05
Nucleotide F1	0.402484	0.40176	0.00742467	0.00742467	0.321919	0.0988829
Contig F1	0.0512347	0.0553151	0	0	0.0107127	0.00282716
BUSCOS (CS)	48	64	9	9	82	30
BUSCOS (CD)	79	46	0	0	5	0
BUSCOS (F)	295	286	1	1	181	56
BUSCOS (M)	359	385	771	771	513	695
Ex90N50	8253	8892	1076	1076	1308	351
Remapping rate	0.02236	0.0228	0.2634	0.2634	0.0747	0.0922

Table 11: Assembly evaluation metrics for the artificial paired-end *C. elegans* dataset (CEL_FLUX). BUSCOs (CS): Complete and single BUSCOs, BUSCOs (CD): Complete and duplicated BUSCOs, BUSCOs (F): Fragmented BUSCOs, BUSCOs (M): Missing BUSCOs, ORF: Open reading frame

	rnaSPAdes	SOAPdenovo-Trans	Trinity	Combined	cd-hit-est (c=1)	cd-hit-est (c=0.9)	Linclust
Database coverage	0.07	0.056	0.076	0.097	0.086	0.072	0.087
Duplication ratio	1.041	1.037	1.141	2.237	1.4	1.083	1.529
95 %-assembled isoforms	1816	540	1255	2160	2083	1839	1648
Misassemblies	31	0	6	37	36	29	27
Mean ORF percentage	78.95958	89.51723	84.23147	83.64537	80.64075	79.58343	83.98252
Assembly score	0.25561	0.06978	0.22252	0.0425	0.16848	0.24241	0.04352
Uncovered bases percentage	0.02938	0.04726	0.06874	0.45727	0.20047	0.04366	0.34469
KC score	0.874929	0.856403	0.85325	0.709287	0.830489	0.873406	0.810111
RSEM EVAL	-33440249	-47944582.15	-36491081.99	-39794629.7	-31751191.58	-30412076.44	-35984585.37
Nucleotide F1	0.822899	0.773512	0.769122	0.457153	0.668369	0.813667	0.625944
Contig F1	0.08026	0.0905651	0.110345	0.106274	0.0908302	0.0794115	0.112156
BUSCOs (CS)	166	96	135	25	110	176	127
BUSCOs (CD)	13	10	34	158	73	5	52
BUSCOs (F)	30	51	38	28	28	29	31
BUSCOs (M)	773	825	775	771	771	772	772
Ex90N50	2775	3803	3147	3770	3340	2844	3493
Remapping rate	0.1935	0.2432	0.1644	0.0685	0.1212	0.1821	0.1103

	MeShClust	MeShClust^2	Grouper	Grouper*	karma	karma*	Oyster River Protocol
Database coverage	0.091	0.044	0	0	0.035	0.015	0.075
Duplication ratio	1.835	1.391	2	2	1.105	1.108	1.176
95 %-assembled isoforms	1919	611	1	1	612	154	1569
Misassemblies	34	2	0	0	3	1	40
Mean ORF percentage	84.23501	88.77639	97.11379	97.11379	83.97072	88.33136	81.67814
Assembly score	0.0428	0.01642	0.01929	0.01957	0.01925	0.00199	0.29506
% bases uncovered	0.39997	0.23236	0.00056	0.00016	0.09821	0.10377	0.09251
KC score	0.768498	0.329479	0.023261	0.023261	0.545569	0.116105	0.856374
RSEM EVAL	-37692169.22	-97870947.76	-127943113.61	-127943113.61	-73650870.05	-119216677.57	-32710610.97
Nucleotide F1	0.545201	0.526054	0.0119664	0.0119664	0.546058	0.287783	0.751547
Contig F1	0.113653	0.0687602	0.000189843	0.000189843	0.0649272	0.0354412	0.0889001
BUSCOs (CS)	46	27	0	0	65	24	140
BUSCOs (CD)	135	7	1	1	11	0	30
BUSCOs (F)	30	37	0	0	17	14	34
BUSCOs (M)	771	911	981	981	889	944	778
Ex90N50	3603	3436	1514	1514	2068	1441	3336
Remapping rate	0.0886	0.10479	0.3805	0.3805	0.2249	0.1383	0.1618

Declaration of authorship

I declare that I have done this work independently and only using the sources and resources provided. There are no objections on the part of the author to make the present master thesis available for public use in the university archives.

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Seitens des Verfassers bestehen keine Einwände die vorliegende Masterarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Jena, November 25, 2019