EVALUATING TRANSCRIPTOME ASSEMBLY POTENTIAL BY DIFFERENT DE NOVO SEQUENCE ASSEMBLER TYPES
Bachelor project in Bioinformatics, 30 credits
2023-01-16 – 2023-05-26
Version 1

Student: Luan Gardenalli
a20luaga@student.his.se

Supervisor: Benjamin Ulfenborg
benjamin.ulfenborg@his.se

Examiner: Zelmina Lubovac
zelmina.lubovac@his.se

School of Bioscience
University of Skövde
Box 408
541 28 Skövde
Abstract

With the rise of NGS technologies, the transcriptomes of non-model organisms can be reconstructed even with the absence of a reference genome, using de novo assembly tools. There is a wide range of de novo assembly tools frequently being developed, however, there is a still a knowledge gap about the different effects and efficiency of different de novo assembly software types for RNA-seq assembly. This study aims to assemble the transcriptome of two different mussel species, *Anodonta anatina* and *Margaritifera margaritifera*, using three different types of genomic assemblers and to evaluate their distinct performances. Here, the transcriptomes have been assembled using whole-genome, single-cell and RNA-seq specific assemblers, and the results have been evaluated and compared using reference-free transcriptome evaluation tools.

Whole-genome assemblers are not designed to handle variable transcript expressions and splice variations, and have thus achieved poor performance at assembling the transcriptomes. Single-cell assemblers, however, are designed to assemble genomes with uneven coverage, which make them able to handle variable transcript expressions and have therefore achieved good efficiency at assembling the transcriptomes. Single-cell assembler SPAdes has matched the performance of the well established RNA-seq assembler Trinity and the single-cell version of IBDA performed just as well as their RNA version. Overall, the top performing assembler in the study was the RNA version of SPAdes.
**Introduction**

The transcriptome sequencing of non-model organisms has become much easier with the rise of NGS technologies (Gordo et al., 2012), which provides efficient and cost effective methods for analyzing the transcriptome of non-model organisms that lack a fully sequenced genome (Fan et al., 2013). This is done via RNA sequencing, a process where RNA molecules are converted to cDNA before sequencing, so that only the coding regions of the DNA are sequenced (Wang et al., 2009). This technology can produce hundreds of millions of short reads (50 – 250bp) which must be assembled to obtain the organism's transcriptome (Zhao et al., 2020).

A very efficient way to assemble a genome or transcriptome is to use a reference based model, where an already known genome or transcriptome of the same or similar organism is used as a template, and reads from an NGS experiment are mapped to this reference template (Liu et al., 2018; Mering et al., 2003). Software tools such as Olego (Wu et al., 2013), MapSplice (Wang et al., 2010), (Liao et al., 2013), TopHat (Trapnell et al., 2009), DeepBound (Shao et al., 2017) and Supersplat (Bryant et al., 2010) all utilize splice and exon/intron junctions, which are strongly conserved sequences that aid the RNA splicing process (Senapathy, 1988), to properly map RNA reads to a reference sequence.

Reference based assembly, however, is not always possible since an existing sequence to be used as reference is not always available. In those cases, assembly is normally performed by a technique called *De novo* assembly. *De novo* assembly merges sequencing reads into larger sequences known as contigs, which are then linked together by gaps to produce *scaffolds*, a process known as *scaffolding* (Paszkiewicz & Studholme, 2010).

When it comes to *de novo* assembly, two main strategies are commonly used: Overlap-layout-consensus (OLC) and de-bruijn-graphs (DBG). OLC works by creating a graph with all overlaps and reads which is used to reach a consensus sequence. This approach is more suitable for earlier sequencing technologies, such as Sanger sequencing derived methods (Pareek et al., 2011), while DBG is more suitable for NGS reads (Li et al., 2012).

The DBG approach involves breaking down the reads into substrings of length k, called k-mers, building a digraph and utilizing the Eulerian cycle concept to find the position of each k-mer. Firstly, each read is split into all possible unique k-mers. Secondly, for each k-mer, its prefix and suffix are defined, by removing the first and last nucleotide of the sequence, respectively. These prefixes and suffixes will make up the nodes of the graph, while k-mers will make up directed edges, connecting nodes that consist of the correct prefix and suffix of the k-mer connecting them. Finally, the Eulerian cycle is found that visits each edge of the graph only once to obtain the assembled sequence (Compeau et al., 2011).

![Figure 1: De Bruijn graph assembly visualisation.](image-url)
Figure 1 shows a simplified example of a DBG assembly, assembling the sequence ATGGCGTGCAAT, with k set to 3, resulting in the k-mers ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA, CAA, AAT.

Currently, there are many different genomic assembly software available to use. According to the bioinformatics software tools database found on "Bioinformatics home" (Tammi, 2018), there are around 120 sequencing tools available. These assembler are normally designed for specific data types; for example, ABySS (Simpson et al., 2009), Velvet (Zerbino & Birney, 2008), SPAdes (Bankevich et al., 2012) are designed for NGS short read data, CANU (Koren et al., 2017), Miniasm (Li, 2016) FALCON (Chin et al., 2016), HapCol (Pirola et al., 2016) are designed for long reads from PacBio and Oxford nanopore technologies and HiCANU (Nurk et al.), hifiasm (Cheng et al., 2021), rust-mdbg (Ekim et al.) are designed for the new PacBio HiFi sequencing technology (Hon et al., 2020). While whole genome assemblers can also assemble RNA-seq data, there are also RNA specific assembler, such as Trinity (Grabherr et al., 2011), Oases (Schulz et al., 2012), SOAPdenovo-Trans (Xie et al., 2014), BinPacker (Liu et al., 2016).

Due to the wide range of options, selecting an assembler can be challenging. Comparative studies (Kleftogiannis et al., 2013; Lin et al., 2011; Miller et al., 2010; Narzisi & Mishra, 2011; Wang et al., 2020; Zhang et al., 2011) can provide an idea of how each assembler perform compared to others, however these studies are limited, and have only evaluated a few assemblers each. The results can also be dependent on the data used. When it comes to transcriptome assembly, comparative studies tend to use mostly RNA-seq specific assemblers (Amin et al., 2014; Chopra et al., 2015; Honaas et al., 2016; Madritsch et al., 2021; Rana et al., 2016) which makes it unclear how other types of assemblers perform when assembling RNA-seq data, and thus how important it is to use RNA-seq specific models is such cases. A transcriptome assembly comparative study by Hölder & Marz (2019) included RNA-seq specific assemblers, however it also included the single-cell version of SPAdes to investigate whether single-cell mode performs well for bulk RNA-seq data (Bushmanova et al., 2016), and it ended up performing even better than the RNA-seq version of SPAdes.

This study compares transcriptome assembly using different types of assemblers, where in addition to RNA-seq specific ones, it also includes whole genome and single-cell specific assemblers, to investigate how effective they can be for assembling RNA-seq data, and to analyze the importance of using RNA-seq specific assemblers when assembling transcriptomes.

**Aims of the study**

The aim of this study is to evaluate and compare the performance of different de novo sequence assembler types for sequencing a non-model organism transcriptome, including whole genome, single-cell and RNA-seq specific assemblers. This will be achieved by performing these two main objectives:

- Determine the full transcriptome of a non-model organism at the highest possible quality; and
- Perform an in-depth transcriptome assembly quality comparison between the assemblies using methodology and software described below.

The results of the study can contribute to genomic databases by adding assembled transcripts, provide insight on the performance of the included assemblers and illustrate the importance of using RNA-seq specific assemblers when assembling RNA-seq data.
Materials and methods
Assemblies were performed on the RNA-seq data from the non-model organism *Anodonta anatina*, also known as duck mussel, and *Margaritifera margaritifera*, also known as pearl mussel, which as of today have very little genomic data available (NCBI, 2023). Since these organisms do not have an abundance of genomic data available to be used as reference, quality assessment will be performed through reference-free methods.

Since the aim of this study is to compare the performance of different types of assemblers for transcriptome assembly, the selected assemblers include three RNA-seq specific, two single-cell specific, and two whole genome specific assemblers, as listed in Table 1. All computations in this study were enabled by resources in project NAISS 2023/22-426 provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at UPPMAX, funded by the Swedish Research Council through grant agreement no. 2022-06725.

Table 1: Features of assemblers included in the study. Obtained from related references and webpages.

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Algorithm</th>
<th>Type</th>
<th>Programming Language</th>
<th>Running Platform</th>
<th>Download website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trinity</td>
<td>DBG</td>
<td>RNA-seq</td>
<td>Java, C++</td>
<td>Linux</td>
<td><a href="https://github.com/trinityrnaseq/trinityrnaseq/releases/tag/v2.8.6">https://github.com/trinityrnaseq/trinityrnaseq/releases/tag/v2.8.6</a></td>
</tr>
<tr>
<td>SPAdes</td>
<td>DBG</td>
<td>Single-Cell / RNA-seq</td>
<td>Python</td>
<td>Linux, Mac OS</td>
<td><a href="https://cab.spbu.ru/software/spades/">https://cab.spbu.ru/software/spades/</a></td>
</tr>
<tr>
<td>IDBA-UD</td>
<td>DBG</td>
<td>Single-Cell / RNA-seq</td>
<td>C++</td>
<td>Linux, Mac OS X</td>
<td><a href="https://github.com/loneknightpy/idba">https://github.com/loneknightpy/idba</a></td>
</tr>
<tr>
<td>Velvet</td>
<td>DBG</td>
<td>Whole genome</td>
<td>C</td>
<td>Linux</td>
<td><a href="https://github.com/dzerbino/velvet">https://github.com/dzerbino/velvet</a></td>
</tr>
<tr>
<td>SGA</td>
<td>String Graph</td>
<td>Whole genome</td>
<td>C++</td>
<td>Linux</td>
<td><a href="https://github.com/its/sga">https://github.com/its/sga</a></td>
</tr>
</tbody>
</table>

When performing reference free assembly, it is important to be able to evaluate the quality of the assemblies without any reference. The most basic way is to analyze the assembly size, the proportion of reads that were assembled into contigs, the count and length of contigs and the count of singletons (reads that were not assembled into any contig) (O’Neil & Emrich, 2013).

A popular metric for evaluating the quality of genome assembly is the N50 metric, which is defined as the length of the shortest contig from the set of contigs with equal or larger length, for which the sum of the lengths add up to at least half of the total assembly length, all contigs combined (Figure 2) (Alhakami et al., 2017). Normally, a larger N50 indicates a higher assembly quality since it means that a greater number of reads were overlapped into contigs. However, when it comes to transcriptome assembly, that may not always be the case since transcripts have varying lengths and larger contigs could also represent chimeric contigs (incorrectly assembled contigs) (Behera et al., 2021). Therefore, the N50 metric will not be used in this study.
Figure 2: Calculating the N50 of a set of eight contigs with varying lengths: the total sum of all contigs lengths amount to 3 400, therefore the N50 will be the length of the first contig that brings up the cumulative sum to 1 700 or higher, when adding up from the longest contig to the shortest.

Instead, assembly quality will be assessed using available software tools designed to benchmark transcriptome assemblies: TransRate (Smith-Unna et al., 2016) is a reference free tool that detects common de novo assembly artifacts, such as chimeras, structural errors and incomplete assemblies, making it useful for assembly optimizations and comparisons. BUSCO (Manni, Berkeley, Seppey, Simão, et al., 2021; Manni, Berkeley, Seppey, & Zdobnov, 2021) estimates the completeness of transcriptome assemblies by comparing it to orthologs databases and analyzing the expected gene count.

Datasets
In this study, RNA sequencing data from two different mussel species were used: Anodonta anatina (duck mussel) and Margaritifera margaritifera (fleshwater pear mussel).

The Margaritifera margaritifera dataset, here referred to as MM, is derived from a study by Gomes-dos-Santos et al. (2022) which performed transcriptome assembly on five different European mussels species. The raw dataset is publicly available at the NCBI-SRA database under the identifier SRR19261768.

The Anodonta anatina dataset, here referred to as AA, was obtained from a study by Ekelund Ugge et al. (2022), in which the mussels’ gene expression was measured over different Cu concentrations for biomarkers evaluation.

Descriptive statistics for the raw read files for each dataset are shown on Table 2:

Table 2: Basic information of the raw reads data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Total reads</th>
<th>Reads length</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>Paired end</td>
<td>118 270 517</td>
<td>151</td>
</tr>
<tr>
<td>MM</td>
<td>Paired end</td>
<td>65 525 653</td>
<td>151</td>
</tr>
</tbody>
</table>

Workflow
The project consists of 3 phases: pre-processing, assemblies, and evaluations. In the first phase, both datasets have been pre-processed with the all the software described under Pre-processing software, using the same order and parameters. On the second phase, a series of assemblies were performed in each processed dataset with the assembler described under assembly software, using the same parameters for each. On the last phase, each assembly has been evaluated using the tools described under evaluation software. The methods are illustrated step-by-step on Figure 3.
The parameters used for each software are listed on supplementary table 1.

When trimming data with Trimmomatic, read pairs that have one pair member trimmed while the other member survive are output as unpaired reads. Since not every assembler in the study supports combining paired data with unpaired data, the unpaired reads have been discarded.

**Pre-processing software**

**FastQC**

FastQC is a widely used quality control tool for sequencing data files (Andrews, 2010). FastQC provides a range of useful data plots that can be helpful for determining the raw data features, such as phred quality measurements, nucleotides proportions, GC-content measurement, N-content measurement, reads length distribution, read duplication and overrepresentation.
measurements, adapter and k-mer content. These measurements are useful for deciding on the subsequent pre-processing steps, for example, what quality threshold and adapters to use when trimming the sequences.

**Read trimming**

Sequence quality trimming is a pre-processing step that “cuts” reads, partially or entirely, based on read quality thresholds. Suppose a read has its bases on the first half denoted with a quality of 25, and the second half, 20. If you perform a quality trim with a threshold of 23 on that read, the first half will be kept, and the second half would be discarded. However, if a threshold of 30 is used, the entire read would be discarded. Trimming is also often used to remove adapter sequences from the reads, which can be detected using FastQC. In this study quality trimming and adapter removal was performed using Trimmomatic.

Trimmomatic (Bolger et al., 2014) is a fast and efficient sequence trimming tool, which takes an adapter sequence as input and performs an alignment for each read in the data set to this input, and cuts the read from the position that the alignment matches at a certain score (user defined), retaining only the sequence next to aligned region.

Trimmomatic also performs quality trimming using a *sliding window*. The user specifies a number $n$ of bases and a quality threshold, and the trimmer will check the average quality of the first $n$ bases starting from the first base, moving from base to base until the threshold is reached and cutting the read at this point.

**Read correction**

Read correction is a pre-processing step that attempts to identify and correct base errors present in the reads. This can be achieved by comparing nearly identical reads, and statistically determining that a base is most likely an error. There are a number of software tools designed to perform this procedure, such as Quake (Kelley et al., 2010), Reptile (Yang et al., 2010); and ECHO (Kao et al., 2011). Some of the assemblers included in this study also contain base correcting modules, including SGA and SPAdes. Most of the forementioned correctors, however, are designed for genomic data rather than RNA-seq data, which could be a problem for RNA-seq data correction since different isoforms will result in nearly identical reads, in which neither are incorrect. If an isoform has a very low expression compared to a similar isoform, it could be mistakenly detected as base errors by the corrector. RNA-seq specific correctors are optimized to handle this issue. Options are more limited, such as SEECER (Le et al., 2013) or Rcorrector (Song & Florea, 2015). According to a base correction comparative study by Macmanes & Eisen (2013), the SEECER method performed worse than the genomic data correctors even for RNA-seq data, so Rcorrector was chosen instead for this study.

Rcorrector uses a k-mer approach to identify base errors, using the tool Jellyfish (Marçais & Kingsford, 2011) to find all k-mers that occur more than once in the reads, and then builds a De Bruijn Graph (DBG) with those k-mers, while recording the number of occurrences in the reads for each one. A k-mer will be suspected of containing base error when its occurrences fail to reach a certain threshold, and if an alternative path is present that passes the threshold and preserves the rest of the sequence as best as possible, the base is corrected accordingly to that path, as shown in Figure 4.
Figure 4: Rcorrector graph visualisation. Sequence k-mers have been collapsed for better visualization. Suppose a read contains a k-mer in which the count fails the threshold (red oval). In the DBG, two alternate paths are present, one in which the k-mer has very high count (yellow oval) and a second with much lower count but passing the threshold (green oval). Since the yellow path would change two bases (CT) and the green would only change one, the green path will be chosen, and A will be corrected to T.

What makes this approach tricky, is that fact that when it comes to RNA-seq, the possibility that low count k-mers are denoting a lowly expressed alternative isoform, instead of being base substitution. In that case, correcting the base could cause the assembly to miss transcripts that would otherwise be assembled correctly. In order to avoid this, a local k-mer threshold is computed instead of a global one. The threshold is the lowest from two values, a k-mer level threshold value and a read level threshold value. The k-mer level threshold value $T$ is acquired using Equation 1:

$$T = \alpha t + 6\sqrt{\alpha t} \quad \text{(Equation 1)}$$

where $t$ is the max multiplicity from all possible k-mer successor bases and $\alpha$ is a coefficient acquired by comparing the ratio of possible k-mer successor bases multiplicities with a pre-defined sample of 1 million k-mers from the dataset. The read level threshold is acquired by ordering all k-mers contained in a read by their multiplicity value, descending. The multiplicity at the first sharp drop (> 2-fold) is defined as the threshold.

The datasets MM and AA have been corrected with Rcorrector using default parameters.

**De-contamination**

Normally, there will be a small amount of reads on sequencing datasets that do not actually belong to the organism of interest, but rather contaminants, such as viruses and bacteria. These can be detected and removed using short read taxonomy classifiers. Some options for this process include Kraken2 (Wood et al., 2019), Centrifuge (Kim et al., 2016) and LMAT (Ames et al., 2013). Taxonomy classification software tend to require enormous amounts of memory, easily surpassing that required for assembly. LMAT, while showing great performance in their self-evaluation, can take up to 1TB of RAM, which can be an inconvenience. Both Centrifuge and Kraken greatly reduces the memory requirements to < 200gb, depending on the database used, but Kraken2 also has the advantage of having the fastest computing times, reaching 90 million reads computed per minute, compared to 10 million from Centrifuge. Kraken2 also offers slightly richer contamination databases compared to Centrifuge (*Centrifuge Classifier for metagenomic sequences*), due to the inclusion of plants, and much more recent updates (*Kraken 2, KrakenUniq and Bracken indexes*). For those reasons, decontamination in this study was performed using Kraken2.

Kraken 2 takes a genomic database of sequences that the user wants to filter from the provided dataset, scans through all the reads in the dataset and classifies every read in which a match with the provided database was found. Like many of the tools described in this study, Kraken2 uses k-mers to process the reads. K-mers of a given sequence are mapped to the database using a taxonomy tree, where nodes represent k-mers of a known sequence, and the highest weighted path (weighted by adding each k-mer count along the path) from a root node to a leaf node is used.
to classify the sequence. Sequences in which k-mers do not make up a path for any taxonomy tree in the database are left unclassified.

As mentioned, the challenge with this method is the memory requirements. In order to efficiently search through the database to classify every single read in the sequence data, an index of the database must be build and kept on memory at all times, which could take hundreds of gigabytes (Ames et al., 2013).

To address memory requirements, Kraken2 utilizes a compact hash table to map k-mers to the taxonomies tree’s Least common ancestor (LCA) nodes. The compact hash table utilizes only 32 bits for both key and value pairs, where the keys are the k-mers of the query sequence and the values are the nodes of the taxonomy tree. The database is then scanned to find a match, using linear probing for collision handling. The procedure is illustrated on Figure 5:

```
k-mer = TATAGCCAATACAGGTACATATGA...
Hash-key = 0x854gc123f2t52sq
Hash code = 010011101011101
```

![Hash table diagram](image)

Figure 5: Kraken2 k-mers classification procedure. A hash-key and value pair is computed for each k-mer of the reads, so that they can be effectively searched through a hash table for a corresponding node of the taxonomy tree. In the example to the left, the k-mers hash code corresponded to the LCA node stored on bucket 4, which means that the read containing that k-mer has been successfully classified to the organism from LCA node on bucket 4.

The user can provide his own database, however, and there are public standard databases that can be used to search for common DNA-seq data contaminants, such as archaea, viral, plasmids, vectors, protozoa, fungi and plant. Public standard databases are listed in supplementary Table 2.

In this study, the MM and AA datasets have been de-contaminated by Kraken2, using the PlusPFP database (see supplementary Table 2) since it is the largest available and includes the widest range of contaminants.

**Assembly software**

**Trinity**

Trinity, developed by Grabherr et al. (2011) at the Broad Institute, is a de novo genomic assembler that is designed specifically to assemble cDNA short reads to reconstruct a transcriptome rather than the whole genome. When assembling a transcriptome, there are additional challenges that the software should account for: differences in the ratio of transcription expression levels; uneven read coverage across a transcripts length; highly expressed transcripts can derive more reads...
with sequencing errors than lowly expressed transcripts can derive error-free reads; adjacent transcripts can accidentally overlap and result in chimeric assembly; due to alternative splicing, the same region of the transcriptome can produce different transcripts; and the repetition of sequences in different genes.

Trinity divides itself into three main components: Inchworm, Chrysalis and Butterfly, referencing the three steps of a butterfly life cycle and deriving the software name, "Trinity". The assembly process starts with Inchworm, which has the purpose of assembling the preliminary transcripts/contigs. It starts by dividing the reads into k-mers of length 25 and removing likely error-containing k-mers, which are identified by grouping equal k-1-mers, (k-mers that vary from each other only at their terminal nucleotides) and then removing all the k-mers variations that are less than 5% as abundant as the most abundant variation. Once likely error-containing k-mers are removed, the most frequent k-mer of the catalogue is selected as a "seed" k-mer, and a greedy tree search algorithm is deployed to assemble contigs by extending the seed k-mer at both terminals, using the most frequent nucleotide from the next aligning k-mers (Figure 6.A). Once this contig cannot be extended any further, the process is repeated for the next most frequent k-mer of the catalogue, and so on until all k-mers in the catalogue are accounted by this procedure. This process alone can already assemble full length transcripts, which will be a part of the next phases.

Since Inchworm assemblies are based on the most frequent k-mers, it will only properly assemble full transcripts which are the most highly expressed isoforms, therefore producing higher k-mer frequency. This means Inchworm can not address the issue of alternative splicing by itself since it is designed to assemble just one most expressed splicing variation transcript/gene. However, since the different regions of lower expressed isoforms will contain different k-mers from the most frequent isoforms, Inchworm will also create separate contigs with those regions alone (Figure 6.B):

These separate contigs will include overlapping zones with the most frequent isoform contigs in their terminals because since both contigs are derived from the same gene, there will be partial k-mer overlap until the point where the isoforms diverge, as show on Figure 6.C. The next step, Chrysalis, uses these overlapping zones to create DBGs representing splice variants. It accomplishes that by first identifying and grouping contigs containing at least one k-1-mer overlap, supported by the presence of reads containing perfect matches for at least half of the length of k-1-mers on each contig, which represent potential junctions. Chrysalis then builds DBGs with “forks” representing the potential splice junctions based on k-1-overlaps for each contigs group (Figure 6.C).

This will create a large number of independent DBGs, theoretically, one for each gene. Subsequently, Chrysalis maps reads to contig groups by selecting the group that shares the most k-1-mers with each read. It also counts how many times each k-mer appears in the read set.

The last step, Butterfly, takes Chrysalis DBGs as input to derive all transcripts, including different splice variants. It starts by trimming edges of the graph that are suspected to be anomalies, such as sequencing errors and very lowly expressed variants. This is done by comparing the ratio of outgoing edges from a node, if any of several outgoing edges has less than 5% support from the total number of outgoing reads, then those edges are removed. Additionally, if any outgoing edge has less than 2% support from the total number of incoming reads, those are also removed as they are suspected of being spurious transcripts (Grabherr et al., 2011).

Once edges are filtered, Butterfly converts the resulting graph to a weighted, compact sequence graph. This is done by assembling k-mers of linear paths so that linear sequences now represent
nodes, which have edges pointing to different possible isoforms. Those edges are weighted by the average weight of the edges in the un-compacted DBG, which was recorded by Chrysalis. The trimming procedure is then repeated.

The best paths through the graph are then algorithmically selected by comparing the graph paths to reads, read pairings and Chrysalis read mappings. The paths found represent all the transcripts successfully assembled by Trinity (Figure 6.D).

Since Trinity is a very well established RNA-seq specific assembler, it will be used in this project as a proper transcriptome assembly reference.

**Velvet**

Velvet is a whole-genome assembler build upon DBG manipulation algorithms. It is designed to assemble small genomes, such as bacterial genomes, as well as large mammalian genomes. Velvet allows the user to chose k-mer size ranging from 29 to 64, builds a DBG based on the chosen k-mer value, and applies graph simplification and corrections algorithms, such as tip and bubble removal, to derive contigs. The process is illustrated on Figure 7.

**SGA**

String Graph Assembler (SGA), developed by Simpson & Durbin (2012), as the name suggests uses string graphs (SG) instead of De Bruijn graphs to assembler reads. The main difference being that in a string graph, the reads are not broken down into k-mers but are instead kept intact as nodes and are connected in a graph by edges if they have overlapping segments. To build an SG a read overlap graph is built first, where edges are labelled with the reads non-overlapping parts. Since the graph should traverse every node, edges that "skip" a read can be safely ignored, as shown in Figure 8. These skippable edges are referred to as transitive edges, while the remaining are referred to as irreducible edges. An SG is then built from the overlap graph by removing duplicate reads, contained reads (reads that are fully contained by an other read) and transitive edges (Figure 8).

Using a Burrows-Wheeler transform-derived string compression algorithm, known as Ferragina-Manzini index (FM-index), which allows the searching through a compressed representation of text, (Ferragina & Manzini, 2001), SGA effectively finds irreducible edges by algorithmically searching through a suffix array (Simpson & Durbin, 2010)

Since the program starts by creating an FM-index, which is a compressed representation of the dataset, the memory requirements for running SGA are theoretically lower than DBG based assemblers. However, the computation time for creating this index is very high (Simpson & Durbin, 2012).

Since the program starts by creating an FM-index, which is a compressed representation of the dataset, the memory requirements for running SGA are theoretically lower than DBG based assemblers. However, the computation time for creating this index is very high (Simpson & Durbin, 2012).

SGA is designed to assemble full genomes rather than transcriptomes, but it's effectiveness at assembling RNA-seq has never been evaluated, to the best of my knowledge.
Figure 6: (A) Inchworms greedy tree search algorithm. The most frequent k-mer is selected as a seed (ATTGCAAT in the example), which is then extended by picking the most frequent base from the next aligning k-mers. Each level of the tree in the example represents the last base distribution from the next aligning k-mers, in which only the last nucleotide differs. For each end of the seed sequence, a path (highlighted in red in the example) is greedily searched by selecting the nucleotides with the highest counts. In case of ties, the path with the most cumulative coverage is selected. In this example, the seed k-mer would be extended in the right side with the bases A-C-T-C. (B) Results of alternative splicing from Inchworms assembly. Considering a gene with one highly expressed (contig 1) and one lowly expressed isoform (contig 2), only the highly expressed will be assembled as a full transcript while the lowly expressed form will be assembled as a separate contig, containing small overlapping segments with the forementioned transcript at their terminals. (C) Chrysalis DBG visualisation. Chrysalis will identify k-1-mers overlaps which are supported by reads, denoting splicing sites. Once these sites are identified, DBGs are constructed including both isoforms k-mer paths. In the example above, the splicing sites of contig 1 and 2 was identified, and a DBG including two different k-mer paths, one for each isoform, was constructed. (D) Butterfly compact graph example. In this example, the two different isoforms are now represented as compact sequences, rather than k-mers. The graph shows that from the sequence marked in blue, there are two different possible continuations, going straight to the sequence marked in green, or going through the sequence marked in orange first. Both versions will be reported in the final assembly as two different contigs.
Figure 7: Velvet assembly procedure representation:
(A) A DBG is constructed from the set of k-mers. (B) The graph is simplified by collapsing paths with only one outgoing connection into nodes containing the resulting sequence from the k-mers in the path. (C) “Tips”, short node chains which are disconnected in one end, are removed. (D) “Bubbles”, two paths that start and end at the same nodes while containing similar sequences are corrected. This is done by applying an algorithm which decides which path should be kept and removes the other. In the simplified example above, nodes 1, 3 and 4 result in a contig.

Figure 8: String Graph example. Genomic sequences replaced by alphabetical sequence for better visualization. The graph on the left represents an overlap graph between 3 sequences, which are used as nodes. Since the path from nodes 1 to 3 skip node 2, it is a transient edge and can be safely removed, building a string graph.

IDBA
IDBA is a DBG based assembler designed for single cell data, which tends to have uneven genome coverage and sequencing depth and calls for special assembly optimization.

Due to uneven sequencing depth, handling incorrect k-mers becomes trickier, because incorrect k-mers from high sequencing depth regions may be more abundant than correct k-mers from low sequencing depth regions. Therefore, using a global threshold to detect incorrect k-mers can be ineffective. IDBA tries to solve this issue by using variable thresholds for detecting incorrect contigs, which are determined by the depth of neighboring contigs.

Building the DBG by IDBA is an iterative process (Figure 9), which starts by using a minimal value for $k$, and increases by one in each iteration. The contigs created by an iteration of $k$ will be used
as input for the next iterations, to attempt to preserve information that larger $k$ values may miss. Once the iteration process is over, scaffolding is performed by aligning reads to the assembled contigs.

Single-cell assemblers have shown good results at assembling RNA-seq in previous research (Bushmanova et al., 2016; Hölzer & Marz, 2019), where the single-cell version of the assembler SPAdes (Bankevich et al., 2012) has even outperformed the RNA-seq version. For this reason, both IDBA-UD (single-cell) and IDBA-Tran, which is the RNA-seq version of IDBA (Peng et al., 2013), assemblies were performed and compared. Assemblies were performed using default parameters, and with the default max k-mer and step size parameters of each version inverted.

**SPAdes**

SPAdes is an assembler designed for bacterial single-cell and multi-cell data. SPAdes uses a different DBG version, referred to as *Paired De Bruijn Graph* (Medvedev et al., 2011), that incorporates mate pair data into the graph structure, by constructing the graph using pairs of k-mers alongside their estimated distance instead of single k-mers. These pairs are referred to as *k-bimers*. Similarly to IDBA, SPAdes uses multiple k-mer sizes as an iterative process, in order to create a DBG that can benefit from both small and larger k values (Figure 9).

SPAdes step-by-step assembly is outlined below:

1. An iterative DBG is constructed, using multiple k-values (Figure 9).
2. Using paired end distance analysis and paths in the assembly graph, accurate distance estimates for k-mers in the graph are derived and k-bimers are adjusted accordingly.
3. A paired assembly graph is constructed using the adjusted k-bimers.
4. The resulting graph is simplified and backtracked to derive contigs and read mappings.

Like IDBA, SPAdes also has an RNA-seq specific version (rnaSPAdes). Assemblies with both versions will be performed and compared. Since the developers do not recommend changing k-mer size for this assembler, assemblies for each version will be performed with default parameters.

![Figure 9: Iterative DBG example. The first iteration $K=K_{(min)}$ uses the minimal k-value and construct a DBG from the reads. In the next iteration $K=K_{(min)}+1$, the value for k has been increased by 1 and the contigs from the first iteration are used as input for re-building the graph, producing an “accumulated DBG”. This process continues until the maximal value for k is reached.]
Evaluation software

TransRate

TransRate (TR) (Smith-Unna et al., 2016) is a transcriptome evaluation tool which aligns the paired end reads used for assembly to the assembled contigs to detect errors and benchmark the overall assembly. There are many possible errors in transcriptome assemblies due to various reasons, such as sequencing process errors, incomplete transcript coverages, biological variabilities, and algorithmic simplifications. Common assembly errors include transcript collapsing, chimerism, unsupported insertions, incompleteness, fragmentation, misassembly, and redundancy. Table 3 shows more detailed information about these artifacts and how TR handles them.

Table 3: Description of artifacts and detections by TR.

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Description</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family collapse</td>
<td>When there are multiple transcripts belonging to the same family, they may have very similar sequences which could cause the assembler to assemble one hybrid version of the transcripts instead of assembling multiple contigs.</td>
<td>When a contig has inconsistent read support, such as the support rate dropping suddenly at certain bases of the contig only, that is evidence of family collapsing.</td>
</tr>
<tr>
<td>Chimerism</td>
<td>When multiple transcripts become concatenated into a single contig.</td>
<td>If two transcripts with different expression level are concatenated together, the read coverage in one transcript will be different from the other, causing a sudden change in read coverage in the middle of a contig.</td>
</tr>
<tr>
<td>Unsupported insertion</td>
<td>When bases are wrongly “added” in the middle of a transcript.</td>
<td>A gap of read coverage will be detectable in the middle of the contig, where the added bases are located.</td>
</tr>
<tr>
<td>Incompleteness</td>
<td>Transcripts missing a significant number of bases at either end.</td>
<td>If a read pair has only one of its reads aligned to a contig, or if reads only align partially to one of the contig ends.</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>Transcript gets divided into multiple contigs.</td>
<td>If read pairs form a “bridge” between two contigs, i.e., read pair aligns to two different contigs.</td>
</tr>
<tr>
<td>Local misassembly</td>
<td>If a transcript is assembled with sequences on wrong position of the transcript.</td>
<td>If a read pair aligns to a contig in the wrong order, meaning the left read aligns to the second half of the contig while the right read aligns to the first half.</td>
</tr>
<tr>
<td>Redundancy</td>
<td>When a single transcript gets assembled into multiple overlapping contigs.</td>
<td>Same reads will align to multiple contigs.</td>
</tr>
</tbody>
</table>

Based on the alignment analyses described on Table 3, each contig on the assembly will get assigned score s(C) ranging from 0 to 1, where 0 indicates very bad assembly quality and 1 indicates very good assembly quality. The score is based on these four main criteria: (1) The contig has accurate base identity at each position, (2) The contig has the same length as a single...
transcript, (3) The bases of the contig are in the right order, and (4) the contig represents a single transcript.

To approximate whenever or not these four criteria are met, four metrics are calculated: $s(C_{\text{nuc}})$, the proportion of bases in the mapped reads that are equal to the ones in the contig; $s(C_{\text{cov}})$, the proportion of bases in the contig that do not have a single supporting read; $s(C_{\text{ord}})$, the extent to which the order of the bases in the contig are correct; and $s(C_{\text{seg}})$, the probability that the contig represents a single transcript. The contig score $s(C)$ is the product of these four metrics.

The overall TR score $T$ is then calculated as the product of the geometric mean of the contig scores, as shown on Equation 2:

$$T = \sqrt[n]{\frac{1}{n} \prod_{c=1}^{n} s(C)}$$

(Equation 2)

TR will also separate contigs qualified as “good” from contigs qualified as “bad” and output a new fasta file for each. This means that the “good” contigs fasta file will basically be a filtered version of the assembly, and therefore a higher quality version. An “optimal score” is also reported, which is the score of the “good” contigs subset.

**BUSCO**

BUSCO is a software tool that is designed to evaluate the completeness of genomic assemblies. It evaluates the contigs in an assembly and searches for the presence of ortholog genes. Due to evolution, every organism is expected to contain a certain amount of known genetic content that is shared with its ancestors. BUSCO uses a database of these known ortholog genes, OrthoDB (Waterhouse et al., 2013), and aligns it to the contigs of the assembly. Theoretically, most of the genes in the transcriptome will have a common ancestor present in the database, therefore a complete assembly would get a high proportion of matches. BUSCO classifies the contigs in the assembly into Complete and single copy, Complete but duplicated, Fragmented, or Missing. If a gene has a match with an ortholog, it is classified as Complete and single copy if its length is within two standard deviations of the BUSCO group mean length and found only once, and as Complete but duplicated if it is found with more than one copy. Fragmented genes are the genes that contain only partial matches, and Missing genes have no matches. BUSCO outputs a percentage score for each of these four classifications.

**Results**

**Pre-processing**

Prior to assembly, the data has been inspected with FastQC and pre-processed with Trimmomatic, Rcorrector and Kraken2.

The data has been inspected with FastQC prior to pre-processing and succeeding pre-processing. The FastQC plots of interest are “quality score across all bases”, which check the quality distributions across all reads for every position in the reads, and "% Adapter" which shows the cumulative percentage of adapters found at each position of the reads. Prior to pre-processing, “quality scores across all bases” (Figure 10.A) shows high mean quality (blue line) from positions 1-151, however there are some deviations from position 120 represented by the whiskers reaching lower quality scores in the Y axis. After trimming the data with Trimmomatic, these deviations are no longer present (Figure 10.C), meaning that the lower data reads have been
removed. “% Adapter” shows 24% of illumina universal adapter detected at position 140 prior to pre-processing (Figure 10.B), which have also been removed after trimming (Figure 10.D).

Rcorrector has detected a small percentage of likely base errors in the reads, and successfully corrected them, while kraken2 has identified a percentage of contaminant containing reads and successfully removed them. The exact proportions of data pre-processing are documented on Table 4 and 5.

(A)

(B)
Figure 10: FastQC Quality scores across all bases (1, 3) and % Adapter (2, 4) reports for the AA R1 dataset before (1,2) and after (3,4) pre-processing. As shown in the graphs, the quality scores of the processed data are higher, denoted by the lack of whiskers in the boxplot of the processed data. The adapters detected in the raw data have also been completely removed, denoted by the disappearance of the red line in the adapter plot of the processed data.

Table 4: pre-processing results for the AA dataset.

<table>
<thead>
<tr>
<th>Software</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
</table>
| Trimmomatic | 118 270 517 read pairs. | -Removed 1 989 396 (1.68%) read pairs.  
- Unpaired 39 609 719 (33.49%) reads. |
| Rcorrector | 76 671 402 read pairs. | -Corrected 52 481 115 (0.2%) bases. |
| Kraken2 | 76 671 402 read pairs. | -Removed 12 103 971 (15%) read pairs. |

Table 5: pre-processing results for the MM dataset.

<table>
<thead>
<tr>
<th>Software</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
</table>
| Trimmomatic | 118 270 517 read pairs. | -Removed 1 629 068 (2.49%) read pairs.  
- Unpaired 10 407 588 (15.88%) reads. |
| Rcorrector | 53 488 997 read pairs. | -Corrected 29 563 770 (0.2%) bases. |
| Kraken2 | 53 488 997 read pairs. | -Removed 3 794 904 (7%) read pairs. |
**Best assemblies**

For each assembler in this study with exception of Trinity and SGA, multiple assemblies with different parameter have been performed and benchmarked with TR. The top assembly of each assembler has been selected by comparing optimal scores (see Methods: TransRate), since theoretically the best assembly will achieve the highest score.

For the AA dataset, Velvet has shown very low efficiency at assembling the transcriptome. The MM dataset has gotten better scores, with the k=59 assembly reaching an optimal TR score of 0.31 (Figure 11.A). In both cases, the score positively correlated with the k-mer size, suggesting Velvet is most efficient at assembling transcriptomes when using higher k-mer sizes. For both datasets the best assembly was the one achieved using a k-mer size of 59.

Similarly to Velvet, the MM dataset had overall higher assembly scores than the AA dataset from the IDBA assemblies. For the AA dataset, IDBA-UD with default parameters had the best score, while for the MM dataset IDBA-Tran with IDBA-UD default parameters had the best score, followed by IDBA-UD with a slightly lower score (Figure 11.B). This result suggests that the single cell version of IDBA can be even more effective than the Tran version to assemble transcriptome data, specially if the software is used at default parameters. While a higher score has been achieved using the RNA version for one of the datasets, that only happened once the max k-mer size and step parameters from the RNA version were set to match the default of the UD version. However, if the parameters are not changed, the UD version outperforms the Tran version in both cases.

For the SPAdes assemblies, once again, overall higher scores have been observed for the MM dataset. Since this is true for all the assembler tested, it can be concluded that the MM read data has higher quality and it's therefore easier to assemble. SPAdes assemblies scores show the same pattern for both datasets (Figure 11.C). The RNA version out-performs the SC assemblies, reaching the highest scores observed in the entire study for both datasets (AA=0.48, MM=0.58). However, the SC scores (AA=0.40, MM=0.47) are still quite high and surpasses most of the assemblers in the study. That along with the results from IDBA (Figure 11.B) suggests that single-cell assemblers can in fact be suitable for RNA-seq transcriptome data.
(B)

**AA**

![Optimal Transfer score for AA parameters](image)

**MM**

![Optimal Transfer score for MM parameters](image)
Figure 11: Optimal TR score benchmarking for different assemblers and parameters. (A) Velvet assemblies benchmarks for different k-mer sizes. (B) IDBA assemblies benchmarks. UD signifies single-cell versions while Tran signifies RNA version. mk60 signifies max k-mer parameter set to 60 and step value set to 10 while mk100 signifies max k-mer parameter set to 100 and step value set to 20. (C) SPAdes assemblies benchmarks. SC signifies single-cell version while RNA signifies RNA version.
Assembler comparison
The top assembly for each assembler (selected by optimal TR score, as shown on Figure 11) alongside Trinity and SGA assemblies have been evaluated and compared using TR and BUSCO, Figure 12 shows TR scores plus optimal scores (see Methods: TranRate) for each assembly. Additional measures can be found on Table 6:

Figure 12: Top assembly of each assembler TR scores and optimal scores benchmarks.
Table 6: Top assembly of each assembler transcript amount and lengths metrics.

<table>
<thead>
<tr>
<th></th>
<th>AA Assembly</th>
<th>Velvet</th>
<th>SGA</th>
<th>IDBA</th>
<th>Trinity</th>
<th>SPAdes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of transcripts</td>
<td>239 630</td>
<td>1 521 971</td>
<td>116 803</td>
<td>217 821</td>
<td>160 421</td>
<td></td>
</tr>
<tr>
<td>Mean transcript length</td>
<td>324</td>
<td>66</td>
<td>808</td>
<td>824</td>
<td>802</td>
<td></td>
</tr>
<tr>
<td>Transcripts under 200 bp of length</td>
<td>98 061</td>
<td>1 240 557</td>
<td>0</td>
<td>34</td>
<td>557</td>
<td></td>
</tr>
<tr>
<td>Transcripts over 1 000 bp of length</td>
<td>16 159</td>
<td>11 147</td>
<td>27 291</td>
<td>51 860</td>
<td>35 496</td>
<td></td>
</tr>
<tr>
<td>Transcripts over 10 000 bp of length</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td>60</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MM Assembly</th>
<th>Velvet</th>
<th>SGA</th>
<th>IDBA</th>
<th>Trinity</th>
<th>SPAdes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of transcripts</td>
<td>1 023 969</td>
<td>3 894 739</td>
<td>1 202 366</td>
<td>1 522 828</td>
<td>880 857</td>
<td></td>
</tr>
<tr>
<td>Mean transcript length</td>
<td>485</td>
<td>137</td>
<td>588</td>
<td>572</td>
<td>945</td>
<td></td>
</tr>
<tr>
<td>Transcripts under 200 bp of length</td>
<td>237 951</td>
<td>2 758 656</td>
<td>0</td>
<td>123</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>Transcripts over 1 000 bp of length</td>
<td>119 968</td>
<td>86 277</td>
<td>151 375</td>
<td>168 457</td>
<td>201 134</td>
<td></td>
</tr>
<tr>
<td>Transcripts over 10 000 bp of length</td>
<td>19</td>
<td>1</td>
<td>189</td>
<td>589</td>
<td>3446</td>
<td></td>
</tr>
</tbody>
</table>

SGA is the assembler that assembled the highest number of transcripts (AA = 1 521 971, MM = 3 894 739), however, the vast majority of these transcripts are under 200 bp (AA = 1 240 557, MM= 2 758 656) which probably indicates that SGA produces a high amount of erroneous small transcripts. These get filtered out by TR (see Methods: TransRate), explaining the bigger gap between the score and the optimal score compared to the other assemblers. For both datasets, velvet and SGA have failed to assemble long transcripts (over 10 000 bp of length) which seems to be a significant issue with these assemblies.

For the MM dataset, SPAdes has assembled a great amount of long transcripts (3446 transcripts over 10 000 bp) and a higher mean transcript length, suggesting SPAdes was more efficient at retrieving transcripts of bigger size, while Trinity has assembled the highest amount of transcripts overall (excluding SGA). For the AA dataset, however, both assemblers “agreed” on the amount of very long transcripts (over 10 000 bp) with 60 transcripts from Trinity and 50 from SPAdes.

While the MM dataset had higher TR scores than the AA dataset, BUSCO analyses (Figure 13) show a different pattern. Velvet, SGA and IDBA gotten more completed genes for the AA dataset, while Trinity and SPAdes have gotten a better result for the MM dataset, but with a smaller margin. Interestingly, IDBA has achieved a great BUSCO score for the AA dataset, almost matching Trinity and SPAdes with 60% completed and single copy genes, while getting a very low score for the MM dataset (over 60% missing). This could be an indication that IDBA performs better at assembling smaller transcriptomes.

Compared to the other assemblers, Trinity has a large portion of duplicated genes. While accordingly to BUSCO this could indicate erroneous assembly of haplotypes (Simão et al., 2015), it is also caused by multiple gene isoforms, leading to genes being detected as duplicated (Manni, Berkeley, Seppey, & Zdobnov, 2021).

Figure 14 shows a BUSCO comparison between Single-cell and RNA versions of IDBA and SPAdes. For both datasets, IDBA shows slightly lower scores for the RNA version. SPAdes shows similar results on both versions for the AA dataset, but much higher score on the RNA version for the MM dataset. The RNA version of SPAdes is the only one that has gotten duplicated genes, meaning it...
was the only to detect splice variants. IDBA-Tran has not detected different isoforms, therefore the advantages of using IDBA-Tran over IDBA-UD are unclear.

Figure 13: Top assembly of each assembler BUSCO benchmarks. The bars represent the percentages of contigs classified as Missing, Fragmented, and Complete (see Methods for details).
Discussion

Transcriptome assembly have important applications in the field of biology, since it establishes databases for genes and proteins and makes differential gene expression analyses possible (Pita et al., 2018; Raghavan et al., 2022). Therefore, the ability to assemble transcriptomes of good quality is highly desired. Although reference-based assembly performs better than de novo methods (Behera et al., 2021), a reference isn't always available to use, making the ability to
assemble transcriptomes via de novo very valuable. There are many different de novo tools that can assemble a transcriptome, however, it is not always clear which assembler should be used to get the best assembly. In order to get a good idea about which assemblers to use, studies that assemble the same data with different tools and compare the results (Amin et al., 2014; Chopra et al., 2015; Honaa\textsuperscript{s} et al., 2016; Hölzer & Marz, 2019; Madritsch et al., 2021; Rana et al., 2016) are very useful. Unfortunately, there is not a great abundance of such studies available, and the existing studies tend to majorly focus on RNA-seq specific assemblers. That is logical, since theoretically assemblers specially designed for RNA-seq will always perform better than other assembler types at transcriptome assembly. But that leaves a big knowledge gap on how other assembler types perform, and thus how important exactly is it to use RNA-seq specific assemblers. In this study, transcriptome assembly was performed with different types of assemblers in order to benchmark the performance of different types of assemblers, rather than just assemblers of the same type. This was partially done in the study by Hölzer & Marz (Hölzer & Marz, 2019), which has also included one single-cell assembler in their comparison. The raw data used in this study was chosen due to the fact that mussels have a great lack of sequencing data available in databases, making it a good candidate for de novo assembly.

It is unusual to utilize whole-genome assemblers to assemble RNA-seq data, because of important differences in the nature of transcriptome assembly. One example of that is different expression levels between transcripts; it is not unlikely that some transcripts will have a much higher expression than others, and therefore much higher coverage in the read data. Since this issue only affects RNA-seq data, whole-genome assemblers are not properly equipped to handle it and may by example consider lowly expressed transcripts to be sequencing errors and remove it from the assembly. There is also the issue of alternative splicing: since a locus can produce multiple different but similar transcripts (isoforms), that will form multiple paths in the graph which can be difficult to properly resolve. For these reasons, transcriptome assembly studies avoid utilizing whole-genome assemblers to assemble RNA-seq data. In this study, however, two whole genome assemblers (Velvet and SGA) were included in order to compare their efficiency at assembling RNA-seq data against RNA-seq specific assemblers. It is quite clear from the BUSCO analyses that both Velvet and SGA have failed to retrieve a good amount of transcripts compared to the RNA-seq specific assemblers, specially Trinity and SPAdes. At this aspect, SGA has had the worst performance, matching only 10\% of BUSCO orthologs for the MM dataset, and 20\% for the AA dataset. For comparison, Trinity and SPAdes have each matched over 70\% for the MM dataset and over 60\% for the AA dataset. Velvet performed better than SGA at 22\% for the MM dataset and 37\% for the AA dataset, but still nowhere close to Trinity and SPAdes.

Assembling multiple transcript isoforms has proven to be a challenge for every assembler included in the study, with the exception of Trinity. Velvet, SGA and IDBA have not assembled any different isoforms, denoted by 0\% duplicated BUSCO scores. SPAdes has assembled a small portion of alternative isoforms, reaching a duplicated BUSCO score of 10\% for the AA dataset and 17\% for the MM dataset. Trinity has shown great efficiency at retrieving multiple isoforms, reaching 25\% duplicated (compared to 40\% single-copy) for the AA dataset and 37\% (compared to 38\% single-copy) for the MM dataset. IDBA-Tran did not assemble different isoforms. This results are comparable to the study by by Hölzer & Marz (2019) where Trinity often achieved a much higher proportion of duplicated matches than IDBA-Tran.

Single-cell assemblers in this study have shown quite decent performance at transcriptome assembly, SPAdes single-cell version has reached very similar TR and BUSCO scores as Trinity for the AA dataset and has surpassed Trinity’s TR score for the MM dataset. Trinity did however reach a much better BUSCO score for the MM dataset than SPAdes single-cell. Both RNA and single-cell versions of IDBA have gotten very similar TR and BUSCO scores for both datasets. It has been
shown before that single-cell assemblers perform well with RNA-seq data by the developers of RNAQuast, a reference based transcriptome assembly benchmarking tool (Bushmanova et al., 2016). This is most likely because of the similarities in nature of single-cell and RNA-seq data, single-cell data tends to have uneven genome coverage while RNA-seq data tends to have uneven transcript expression, so both assembler types have to be optimized at assembling data with uneven nature.

A good strategy used for achieving a very high-quality transcriptome assembly is to perform multiple assemblies using multiple tools and generating a consensus assembly from multiple assemblies. This is called the ensemble approach (Behera et al., 2021), and is already utilized by a few tools, such as TransPi (Rivera-Vicéns et al., 2022), EvidentialGene (Gilbert, 2019), TransBorrow (Yu et al., 2020), the method proposed in (Cerveau & Jackson, 2016) and ConSemble (Voshall et al., 2021). All of the aforementioned tools perform multiple assemblies using three to five RNA-seq specific assemblers and from these assemblies, a higher quality assembly is derived. According to the results of this study, it may be viable to also include single-cell assemblers in ensemble methods. Three of these tools, by example, included the RNA-seq version of IDBA, while according to the results of this study, the single-cell version of IDBA can perform even better than the RNA-seq version. As for today, there is no de novo transcriptome assembly ensemble tool that includes a great multitude of assemblers, however, if a robust ensemble tool that relies on a higher number of included assemblers were to be developed in the future, both single-cell and RNA-seq versions of assemblers could be included. Although as for today, such tool would be challenging to use due to the immense computing power and time required to run a large number of assemblies.

**Conclusion**

To conclude, the aim of this study was to achieve the best possible assemblies for the included datasets and to access the performance of different types of assemblers at assembling transcriptome data. As the results show, whole-genome assemblers do not perform well at assembling transcriptomes and should be avoided for that purpose. While the best assembly was achieved by an RNA-specific assembler (rnaSPAdes), single-cell assemblers have also demonstrated good efficiency at assembling transcriptomes in this study. This result has been noted before when single-cell SPAdes outperformed rnaSPAdes in the study by Hölzer & Marz (2019), and while in this particular study rnaSPAdes has performed better than single-cell SPAdes, single-cell IDBA has arguably performed better than RNA IDBA, due to slightly better BUSCO scores signifying more complete assemblies. This finding could be potentially useful for the development of future assembly ensemble tools.

**Ethical Aspects**

Since this study has worked with publicly available non-human data, no samples were cultured or extracted from any living organism, therefore no particular permits or ethical approves were required. The gender of the mussel samples used in this study is unknown, so gender-specific expressed genes are not reflected in the assemblies. No potential conflicts of interests are declared, and no copyright laws were violated in this study.
References


Ekim, B., Berger, B., & Chikhi, R. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. (2405-4720 (Electronic)).


Kraken 2, KrakenUniq and Bracken indexes. https://benlangmead.github.io/aws-indexes/k2


### Supplementary data

#### Supplementary table 1: Documentation of the parameters used for each software included in the project.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastQC</td>
<td>Default</td>
</tr>
<tr>
<td>Trimmomatic</td>
<td>illuminaclip:TruSeq3-PE.fa: 2:30:10 leading:5</td>
</tr>
<tr>
<td></td>
<td>trailing:5</td>
</tr>
<tr>
<td></td>
<td>slidingwindow:4:20 minlen:36</td>
</tr>
<tr>
<td>Rcorrector</td>
<td>Default</td>
</tr>
<tr>
<td>Kraken2</td>
<td>db: k2_pluspfp_20221209</td>
</tr>
<tr>
<td>Trinity</td>
<td>Default</td>
</tr>
<tr>
<td>SGA</td>
<td>Default</td>
</tr>
<tr>
<td>IDBA-UD</td>
<td>Default (max k = 100, step = 20) and max k =60, step = 10</td>
</tr>
<tr>
<td>IDBA-Tran</td>
<td>Default (max k = 60, step = 10) and max k =100, step = 20</td>
</tr>
<tr>
<td>rnaSPADes</td>
<td>Default</td>
</tr>
<tr>
<td>SPAdes</td>
<td>SC with default parameters</td>
</tr>
<tr>
<td>TransRate</td>
<td>Default</td>
</tr>
<tr>
<td>BUSCO</td>
<td>Lineage set: mollusca, mode: transcriptome</td>
</tr>
</tbody>
</table>

#### Supplementary table 2: Public standard contaminant databases for Kraken2. Size refers to the storage size, while Index size refers to the memory requirement needed to build the index for the database.

<table>
<thead>
<tr>
<th>Name</th>
<th>Contains</th>
<th>Date</th>
<th>Size (GB)</th>
<th>Index size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viral</td>
<td>Viral</td>
<td>12/9/2022</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>MinusB</td>
<td>archaea, viral, plasmid, human, UniVec_Core</td>
<td>12/9/2022</td>
<td>6.1</td>
<td>8.7</td>
</tr>
<tr>
<td>Standard</td>
<td>archaea, bacteria, viral, plasmid, human, UniVec_Core</td>
<td>12/9/2022</td>
<td>48</td>
<td>62</td>
</tr>
<tr>
<td>Standard-8</td>
<td>Standard with DB capped at 8 GB</td>
<td>12/9/2022</td>
<td>5.5</td>
<td>7.5</td>
</tr>
<tr>
<td>Standard-16</td>
<td>Standard with DB capped at 16 GB</td>
<td>12/9/2022</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>PlusPF</td>
<td>Standard plus protozoa &amp; fungi</td>
<td>12/9/2022</td>
<td>51</td>
<td>66</td>
</tr>
<tr>
<td>PlusPF-8</td>
<td>PlusPF with DB capped at 8 GB</td>
<td>12/9/2022</td>
<td>5.5</td>
<td>7.5</td>
</tr>
<tr>
<td>PlusPF-16</td>
<td>PlusPF with DB capped at 16 GB</td>
<td>12/9/2022</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>PlusPFP</td>
<td>Standard plus protozoa, fungi &amp; plant</td>
<td>12/9/2022</td>
<td>104</td>
<td>142</td>
</tr>
<tr>
<td>PlusPFP-8</td>
<td>PlusPFP with DB capped at 8 GB</td>
<td>12/9/2022</td>
<td>5.1</td>
<td>7.5</td>
</tr>
<tr>
<td>PlusPFP-16</td>
<td>PlusPFP with DB capped at 16 GB</td>
<td>12/9/2022</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>EuPathDB46²</td>
<td>Eukaryotic pathogen genomes with contaminants removed</td>
<td>12/9/2022</td>
<td>26.4</td>
<td>34.1</td>
</tr>
</tbody>
</table>