Title:

A comprehensive guide to potato transcriptome assembly

Authors:

Maja Zagorščak[1], Marko Petek[1]

Affiliations:

[1] Department of Biotechnology and Systems Biology, National Institute of Biology, Večna pot 111, 1000 Ljubljana, Slovenia

*** Corresponding author:** Maja Zagorščak, e-mail: maja.zagorscak@nib.si

Running head:

Potato transcriptome assembly guide

Abstract

We have witnessed a rapid advancements in high-throughput genome sequencing and the maturation of long-read technologies. However, an accurate assembly of polyploid potato genomes still remains challenging. Sequencing the double monoploid genome of *Solanum tuberosum* Group Phureja (Xu et al., 2011) has enabled functional studies of polyploid potato cultivars using RNA sequencing (RNA-Seq) technologies, although, with the limitation of not covering cultivar specific gene expression. The accumulated RNA-Seq datasets from these cultivars can be leveraged to assemble tetraploid potato transcriptomes that enable analysis of genes that are not limited to reference genome annotations. To increase transcriptomes' quality, short read assemblies are nowadays complemented with full-length transcriptome sequencing using Pacific Biosciences or Oxford Nanopore platforms. In this chapter we give a detailed guide on a pipeline for *de novo* transcriptome assembly of polyploid potato genotypes and their integration into a pan-transcriptome.

Key words: *de novo* transcriptome assembly, high-throughput sequencing (HTS), pan-transcriptome, FASTQ, plant transcriptome, tetraploid potato, mRNA, RNA-Seq

1 Introduction

Several high-throughput sequencing (HTS) short read platforms such as Illumina, SOLiD and Ion Torrent have been developed and used to generate vast amounts of sequence data. The leading long-read sequencing platforms from Pacific Biosciences (PacBio) and Oxford Nanopore also provide affordable high-throughput transcriptome sequencing that is used also in plants [1].

However, an accurate assembly of polyploid potato genomes still remains challenging. Sequencing the double monoploid genome of *Solanum tuberosum* Group Phureja [2] has enabled functional studies of polyploid cultivars using RNA sequencing (RNA-Seq) technologies, although, with the limitation of not covering cultivar specific gene expression. Many tetraploid potato RNA-Seq datasets have been deposited to public repositories and, when properly annotated with cultivar/genotype information, can be leveraged for *de novo* transcriptome assembly.

De novo assembly algorithms can be roughly divided into two types. The first type are overlap-layout-consensus (OLC) greedy algorithms implemented in assemblers such as Celera [3] which was used to assemble the human genome. These assemblers find similar sequences and build contigs by performing multiple sequence alignments (MSA). They work reasonably well on smaller sets of longer reads obtained by Sanger or Roche 454 sequencing, but perform poorly for more extensive short read datasets.

The second type of assembly algorithms utilizes de Bruijn graphs [4] and were designed for short read HTS. These algorithms break reads into k-mers, short fragments of a specified length (k), detect k-mers that

largely overlap and upon that information construct graphs in which k-mers represent nodes [5]. Graphs are further inspected, curated and pruned, based on different criteria and assumptions, to produce individual transcript sequences. Many implementations of this algorithm have been used for *de novo* transcriptome assembly from HTS RNA-Seq reads and have been cross-compared in several studies [6–8]. Outputs from different assemblers vary substantially, but even when using the same assembler with different parameters (e.g. k-mer length, bubble size) the resulting contigs differ. Thus, the real challenge is to retain true biological transcript isoforms.

If transcriptomes of multiple genotypes are assembled, the next step of the analysis is to construct a pan-transcriptome. A pan-transcriptome is a non-redundant collection of all expressed transcripts of a certain species or subspecies, in which the transcripts are classified into core (present in all or most genotypes) and accessory (present in only few genotypes). If transcripts from enough genotypes, tissues and conditions are sequenced in sufficient depth, the resulting pan-transcriptome can be a reasonably good approximation of most genes of the pan-genome. For non-model organisms that lack pan-genome resources and with less well annotated reference genomes, a pan-transcriptome can offer valuable insights into gene presence/absence variations and alternative splicing.

Here we present a potato transcriptome reconstruction protocol that includes quality control (QC), preprocessing, filtering, *de novo* assembly of short reads, processing of long reads, as well as annotation and integration transcriptomes of several genotypes into a pan-transcriptome (Figure 1).

"[Fig 1 near here]"

2 Materials

2.1 Biological Sequences

Sequencing data of short-read sequencing platforms are stored in FASTQ formatted files that contain sequence reads and quality values for each sample replicate. Short RNA-Seq reads can be either single-end (SE) or paired-end (PE). The more recent library preparation methods are strand specific, meaning they preserve the information on transcript directionality or strandedness of reads. For *de novo* transcriptome assemblies, best results are obtained when using PE strand-specific reads with read lengths of 150 nucleotides (nt) or above.

Long transcriptome reads are currently available from PacBio (Iso-Seq) and Oxford Nanopore (Direct RNA or cDNA sequencing). PacBio RS and RS II instruments output raw data in Hierarchical Data Format (HDF) format with filename extensions “bas.h5” and “bax.h5”, whereas the Sequel and Sequel II instruments output data in Binary Alignment Map (BAM) format with filename extension “.bam”. Oxford Nanopore MinION and GridION instruments output reads in HDF format with filename extension “.fast5”. After processing (see Section 3.1.4), the sequences from both long read platforms are stored in FASTQ format.

To capture less abundant genes and rare events deeper sequencing is required. A general recommendation is to use various experimental setups in obtaining RNA-Seq data, such as different platforms and library preparation protocols, i.e. long reads complemented with short PE reads [9].

2.2 Computational Hardware

1. Normalisation and *de novo* assembly: 512GB RAM suggested
2. Analyses steps: 32 threads (1.80GH), 256GB RAM, SSD + 4TB disk space suggested

2.3 Computational Software

Operating system: recent Ubuntu-based distribution is recommended although most commands should work also on other distributions

Programming languages:

1. Python 2.7.12 & Python 3.5.2 or higher
2. R version 3.6.1 or higher

A set of publicly available bioinformatics tools, packages and custom scripts used for the analysis:

1. Conda (docs.conda.io) a package, dependency and environment manager
2. moreutils (rentes.github.io/unix/utilities/2015/07/27/moreutils-package) an UNIX utilities package
3. dos2unix (waterlan.home.xs4all.nl/dos2unix.html) a text file format converter
4. pyfaidx (github.com/mdshw5/pyfaidx) a tool for indexing, retrieval, and modification of FASTA files [10]

5. FastQC (www.bioinformatics.babraham.ac.uk/projects/fastqc/) a QC tool for HTS files e.g. FASTQ files
6. MultiQC (github.com/ewels/MultiQC) a tool that summarizes the reports of QC tools for multiple files [11]
7. FASTX-Toolkit (hannonlab.cshl.edu/fastx_toolkit) a collection of tools for FASTA/FASTQ files preprocessing
8. BBTools (jgi.doe.gov/data-and-tools/bbtools) a package containing various bioinformatic tools for short read alignment (BBMap – Bushnell B. – sourceforge.net/projects/bbmap/)
9. SPAdes (github.com/ablab/spades) an assembly toolkit [12]
10. Oases (github.com/dzerbino/oases) a *de novo* transcriptome assembler based on the Velvet genome assembler core [13, 14]
11. Trinity (github.com/trinityrnaseq/trinityrnaseq) a *de novo* transcriptome assembler [15]
12. CD-HIT (weizhongli-lab.org/cd-hit) a program for clustering and comparing protein or nucleotide sequences [16]
13. Exonerate (www.ebi.ac.uk/about/vertebrate-genomics/software/exonerate) a general tool for sequence comparison
14. NCBI-BLAST+ (www.ncbi.nlm.nih.gov/books/NBK279671/) a set of utilities for sequence databases scanning, similarity of sequences comparisons and homologous candidates identification [17]
15. tr2aacds v2016.07.11 from EvidentialGene (arthropods.eugenes.org/EvidentialGene/evigene/) - a workflow for sequence redundancy reduction and gene loci classification [18]
16. STAR (github.com/alexdobin/STAR) a RNA-Seq aligner [19]
17. samtools (github.com/samtools) tools for manipulating next-generation sequencing data [20]

18. TransRate (github.com/blahah/transrate) a software for *de novo* transcriptome assembly quality analysis [21]
19. InterProScan 5 (github.com/ebi-pf-team/interproscan) a software package for functional characterisation [22]
20. MatchAnnot (github.com/TomSkelly/MatchAnnot) transcripts annotator regarding alignment to a genomic reference and matches to an annotation database in GTF format
21. VecScreen (github.com/aaschaffer/vecscreen_plus_taxonomy) a scanner for vector contamination [23]
22. DIAMOND (github.com/bbuchfink/diamond) an accelerated BLAST compatible local sequence aligner [24]
23. BUSCO (busco.ezlab.org) a software for assessment of genome assembly and annotation completeness [25, 26]
24. Centrifuge (github.com/infphilo/centrifuge) a classifier for metagenomic sequences [27]
25. Pavian (github.com/fbreitwieser/pavian) an interactive browser application for analysis of metagenomics data [28]
26. MAFFT (mafft.cbrc.jp/alignment/software) a multiple sequence alignment tool [29]
27. MUSCLE (www.drive5.com/muscle/) a multiple sequence alignment tool [30]
28. Clustal Omega (www.clustal.org/omega) a multiple sequence alignment tool [31]
29. Simple Phylogeny from ClustalW2 (www.clustal.org/clustal2) a multiple sequence alignment tool [32–34]
30. MView (github.com/desmid/mview) an utility for multiple sequence alignments [35]

In addition, in-house scripts were written to perform various tasks associated with data processing and are freely available at github.com/NIB-SI/_p_stRT.

3 Methods

The workflow can be divided into 4 main steps: raw data processing, *de novo* transcriptome assembly, decreasing redundancy of assemblies and annotation, and pan-transcriptome construction.

Similar to a wet-lab experimental design, we need to prepare a directory tree. For all steps we will use the ISA (Investigation/Study/Assay) directory structure based on the ISA-Tab metadata format [36]. Furthermore, we recommend the usage of input, output, and scripts directories within assays. Collection of associated metadata should be recorded at all steps. When possible, log files generated by the software should be kept. Software and tools not installed globally are recommended to be kept within one designated directory, here generically noted as `pathToSoftwareDirectory`. Interchangeably, `pathTo` term is used when data is included.

1. Create an investigation directory (_I_STRT) designated for potato transcriptome assembly, and within this directory create four study directories:

```
$ mkdir _I_STRT
$ mkdir _I_STRT/_S_01_sequences _I_STRT/_S_02_denovo \
_I_STRT/_S_03_stCuSTr _I_STRT/_S_04_stPanTr
```

2. Within the first study (_S_01_sequences), data will be gathered and preprocessed (including trimming and quality control). This study also includes *in silico* normalisation steps and the PacBio Iso-Seq processing pipeline. The second study (_S_02_denovo) is designated for de Bruijn graph based *de novo*

assembly of short reads and over-assembly (i.e. assortment of *de novo* assemblies) containing unique sequence identifiers per genotype generation. The third study (_S_03_stCuSTr), containing the most steps, is designated for decreasing redundancy of assemblies and annotation. Finally, within the fourth study (_S_04_stPanTr), a pipeline for pan-transcriptome construction will be demonstrated.

3.1 Study 01 - raw data preprocessing

This section describes how to obtain publicly available data and how to preprocess it before running *de novo* transcriptome assembly.

3.1.1 Gathering raw data

Publicly available raw sequence files can be obtained from international sequence repositories such as SRA (www.ncbi.nlm.nih.gov/sra) or ENA (www.ebi.ac.uk/ena). At SRA, experiments can be queried and FASTQ files directly downloaded from the Traces/Download/FASTA/FASTQ webpage (trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=search_seq_name). Similarly, ENA provides the ENA Browser www.ebi.ac.uk/ena/browser/home) that enables FASTQ file download.

3.1.2 Raw data quality control

As noted previously, assembly quality depends on the input datasets and used computational methods. First step in obtaining high quality input data is to establish robust and quality controlled wet-lab procedures

which includes sampling, RNA isolation and RNA-Seq library preparation. Additional wet-lab sequencing library normalisation procedures can be used to reduce differences in gene expression abundance.

After sequencing, raw read quality control (QC) is an essential step in all RNA-Seq dry-lab experimental procedures and should be done before any assembly. A popular QC software used for sequencing data is FastQC (www.bioinformatics.babraham.ac.uk/projects/fastqc/).

1. Follow the instructions on the website to install FastQC. To run FastQC on all samples, create a new assay directory and subdirectories within `_S_01_sequences`:

```
$ mkdir _A_01_FastQC _A_01_FastQC/input/ _A_01_FastQC/output/
```

2. Copy all downloaded FASTQ files into `_A_01_FastQC/input/ directory`. To execute FastQC on multiple threads (`-t`) for all files execute:

```
$ fastqc -t 24 _A_01_FASTQ/input/*.f* -o _A_01_FASTQ/output
```

3. FastQC reports can be aggregated into one HTML report file using MultiQC [11]. After installation run:

```
$ multiqc _A_01_FASTQ/output -o _A_01_FASTQ/output
```

3.1.3 Short reads data preprocessing

Trimming of adapter sequences, polyA tails and low quality terminal bases should be performed and reads below a certain mean quality score should be discarded [37] (*see* **Note 1**). We recommend using well-

documented tools such as cutadapt [38], trimmomatic [39] or TrimGalore (github.com/FelixKrueger/TrimGalore). Before *de novo* assembly, we generally recommend to trim bases that are below Phred quality score 20 and remove reads containing any ambiguous nucleotides (N).

1. To preprocess Illumina PE reads using TrimGalore, run the following commands:

```
$ conda install trim-galore
$ conda activate
$ trim_galore --quality 20 --paired --retain_unpaired --keep --max_n 0 \
  --fastqc --fastqc_args "--outdir ../output/fastqc" \
  -output_dir ../output R1_001.fastq.gz R2_001.fastq.gz
$ conda deactivate
```

In the above TrimGalore command `R1_001.fastq.gz` represents a file with first reads in pair and `R2_001.fastq.gz` the second reads in pair. FastQC quality control reports for preprocessed reads are automatically generated and stored under the ../output/fastqc directory. For more details on TrimGalore *see* **Note 2**.

For more details on quality assessment of raw short read RNA-Seq data and read trimming see **Chapter 3** and **Chapter 11**.

2. To simplify the execution of assembly programs, trimmed reads from datasets of the same genotype produced by the same sequencing platform with similar read lengths can be concatenated into one file using `cat` (the command below shows how to combine three files into one). This has to be done

separately for PE and SE reads; take care to concatenate forward and reverse PE reads into separate files (*see* **Note 3**).

```
$ cat file_SE1.fastq file_SE2.fastq file_SE3.fastq > Illumina_SE.fastq
```

For large short read datasets, assemblers may require more memory as available to the system, therefore assembly might take unreasonable time to finish or even fail. To avoid this, read error correction and digital normalisation can be performed prior to *de novo* assembly with BBNorm from BBTools software package (sourceforge.net/projects/bbmap/) using the following commands (for parameter details see documentation at jgi.doe.gov; for alternative *see* **Note 4**):

```
$ pathTo/bbmap/tadpole.sh -Xmx400g in=Illumina_SE.fastq \
  out=Illumina_SE.tadpole.fastq mode=correct k=50
$ pathTo/bbmap/bbnorm.sh -Xmx400g in=Illumina_SE.tadpole.fastq \
  out=SE.tadpole.BBnorm.fastq target=100 min=5 \
$ pathTo/bbmap/tadpole.sh -Xmx400g in=Illumina_PE.fastq \
  out=Illumina_PE.tadpole.fastq mode=correct k=50 ignorebadquality=t
$ pathTo/bbmap/bbnorm.sh -Xmx400g in=Illumina_PE.tadpole.fastq \
  out=Illumina_PE.tadpole.BBnorm.fastq target=100 min=5
```

3.1.4 Long read data preprocessing

PacBio Iso-Seq sequencing generates reads up to several kb in length and uses circular consensus sequencing method to generate high fidelity consensus sequences. The Iso-Seq data preprocessing pipeline differs slightly based on the version of the sequencing instrument. PacBio offers a free software SMRT

Link server for processing raw data, both, in a web-browser GUI format (www.pacb.com/support/software-downloads), and as a command line version (github.com/PacificBiosciences/IsoSeq). The Iso-Seq pipeline includes the conversion of raw reads into 1) circular consensus sequences (CCS), 2) primer removal, 3) demultiplexing and identification of full-length CCS, 4) polyA-tail and concatemer removal, 5) isoform clustering and 6) polishing. Redundant high-quality full-length PacBio isoforms can be further 7) collapsed based on their alignment to the reference genome and 8) filtered based on their counts using PacBio Cupcake ToFU Python scripts (github.com/Magdoll/cDNA_Cupcake). The pipeline is still actively developed and thus might change in the future.

Oxford Nanopore Technologies provides a data preprocessing toolkit called Guppy for their raw sequencing data, that includes base-calling from `.fast5` files, demultiplexing and adapter trimming (nanoporetech.com/nanopore-sequencing-data-analysis). Recently, an open source Nextflow-based workflow has also been published for streamline direct RNA nanopore sequencing analysis [40].

3.2 Study 02 - de novo transcriptome assembly

In this study we perform de Bruijn graph based *de novo* over-assembly of short reads. For each genotype we generate unique sequence identifiers. We give commands for frequently used *de novo* transcriptome assemblers (*see* **Note 5**): rnaSPAdes [12], Velvet/Oases [13, 14] and Trinity [15].

Even with flawlessly preprocessed reads, *de novo* assembly is computationally and memory-wise exhaustive. In addition to sequencing and library preparation errors, palindromic sequences create

unresolvable paths in de Bruijn graphs. To prevent this, k-mer lengths for assembly should be odd. Also, when possible, k-mers should be longer than the length of a repeat.

To start with this section, create a new assay directory for each assembler within the directory of the second study (`_S_02_denovo`):

```
$ mkdir _A_01_rnaspades _A_02_velvet _A_03_trinity
```

3.2.1 rnaSPAdes *de novo* assembly

A *de novo* transcriptome assembler, rnaSPAdes [12], with the emphasis on assembling alternatively spliced isoforms, automatically calculates k-mer sizes based on the read length (two k-mer sizes are used by default), and in parallel offers the option of setting the k-mer sizes manually. The minimal contig length cut-off is 200 bp. It supports Illumina and Ion Torrent short reads; recent versions also enable hybrid transcriptome assembly of short reads and long PacBio/Oxford Nanopore reads. It outputs three FASTA files that differ in filtration stringency based on length and estimated expression level.

Installation instructions for rnaSPAdes are found at the SPAdes GitHub page (github.com/ablab/spades) that also includes a more detailed explanation of program functionality. To generate multiple *de novo* assemblies under various k-mer lengths (e.g. 23, 33, 43, 53, 63, 73 and 83) using Illumina non-strand specific, paired-end (PE) and single-end (SE) reads in forward-reverse fragment orientation execute the following code:

```
$ python3 pathTo/rnaspades.py \
```

```
--pe1-12 PE_input_file_name.fastq \
--pe1-s SE_input_file_name.fastq \
--pe1-fr -t 48 -m 500 \
--tmp-dir ./tempdir -k 23,33,43,53,63,73,83 -o output_directory
```

`PE_input_file_name.fastq` represents a generic filename for combined PE reads in FASTQ format. Likewise `SE_input_file_name.fastq` represents a generic filename for combined SE reads in FASTQ format. The parameter `-t` defines the number of threads, `-m` defines memory limit in Gb, `--tmp-dir` the directory for temporary files and `-k` the comma-separated list of k-mer sizes listed in ascending order. For more details *see* **Note 6**.

3.2.2 Velvet/Oases *de novo* assembly

The Velvet assembler [13] can handle long reads, SOLiD colour space data, Illumina and 454 platforms data. The assembler consists of two subprograms: velveth (constructs k-mers from the data) and velvetg (builds the de Bruijn graph and finds and extracts contigs from it). Oases pipeline [14] then segments the graph and extracts transcript isoforms. You should run several assemblies with different k-mer lengths and merge them.

To install Velvet/Oases follow the instructions at github.com/dzerbino/oases where a more detailed procedure explanation and parameter descriptions are available, including computational resources

recommendations. Define the number of threads (e.g. 48) that will be available for the Velvet assembler by typing:

```
$ export OMP_NUM_THREADS=48
```

You need to adapt your commands depending on the sequencing platform (*see* **Note 7**). The following commands are appropriate for a multiple k-mer assembly of Illumina PE and SE reads (including orphan PE reads):

```
$ pathTo/velveth baseDirectoryName 23,84,10 -fastq \
  -shortPaired Illumina_PE.tadpole.BBnorm.fastq \
  -short Illumina_SE.tadpole.BBnorm.fastq -strand_specific

$ for((n=23; n<=83; n=n+10)); do
  pathTo/velvetg baseDirectoryName_"$n" -ins_length 175 \
  -read_trkg yes -min_contig_lgth 200 -cov_cutoff 1 -exp_cov auto;
  done

$ for((n=23; n<=83; n=n+10)); do
  pathTo/oases baseDirectoryName_"$n";
  done
```

This will create output directory names `baseDirectoryName` concatenated with the used k-mer. The parameter `-strand_specific` informs the assembler that the supplied reads originate from strand-specific library preparation. The `for` loops define to use k-mer lengths starting at 23, increment by 10, until reaching 83. When using PE reads, library insert size, i.e. the average size of the fragments including the sequenced reads, must be defined using `-ins_length`. The expected short-read k-mer coverage can be defined using

`-exp_cov` (values between 10 and 20 are recommended); or is automatically estimated to the length weighted median contig coverage (auto).

3.2.3 Trinity *de novo* assembly

The Trinity assembler consists of three separate programs, used in three consecutive stages: Inchworm, Chrysalis and Butterfly [15]. In Trinity, k-mer length is fixed at length 25; the shortest reported contig default length is 200 bpand the output FASTA file contains all assembled isoforms.

See installation instructions and all related Trinity information at github.com/trinityrnaseq/trinityrnaseq/wiki**.** Prepare a space separated list of all your Illumina PE reads as `--left` or `--right` in the same order and run:

```
$ Trinity --seqType fq --max_memory 128G --CPU 24 \
  --left PE_set_1_1.fq.gz PE_set_2_1.fq.gz PE_set_n_1.fq.gz \
  --right PE_set_1_2.fq.gz PE_set_2_2.fq.gz PE_set_n_2.fq.gz
```

Option `--max_memory` defines the allocated memory for the tool and `--CPU` the number of threads. Assembled transcripts under default Trinity k-mer size 25 are stored in the `trinity_out_dir/Trinity.fasta` file.

3.2.4 Post-assembly processing

First, make assembly FASTA file names unique using `mv` or `cp` commands. We recommended the naming convention `assemblerNameGenotypeCount.fasta`. Instead of full assembler name and full genotype name an abbreviation can be used. Count represents an integer number of assembler-genotype combinations, e.g. for 5 various k-mers used under the same assembler it will be 1, 2, .. , 5 per genotype. Now, using renamed assembler names, assembler defined contig IDs can be renamed to an overall unique IDs per genotype, e.g. `assemblerNameGenotypeCount_sequenceCount`. To do this, download the script `S02_3.2.4_uniqueIDs.py` from github.com/NIB-SI/_p_stRT to the scripts directory (*see* **Note 8**) and run the command:

```
$ chmod 755 S02_3.2.4_uniqueIDs.py
```

Adapt paths of input/output files within the script as wanted and run:

```
$ python2 S02_3.2.4_uniqueIDs.py
```

Concatenate all .log files containing pairs of old and new IDs using `cat` command. Merge all contigs into one FASTA file per genotype using `cat` command, e.g.

```
$ cat ../output/*Genotype* >> Genotype.all.tr
```

3.3 Study 03 - decreasing redundancy of assemblies and annotation

Evaluations of assembly tools have shown that no assembly strategy is optimal for all RNA-Seq datasets which might encompass different species, genome ploidy, amount and input data quality [6]. Therefore, currently the optimal solution is, instead of choosing the best assembler, to choose the best assembled

sequence from the over-assembly (of various assemblers with various parameter settings). Such a selection eliminates assembler-specific biases. The best assembled sequence is defined as the most likely candidate representation of the real isoform occurring in the transcriptome. By choosing the best assembled sequence, misassembled contigs are removed, thus improving results of follow-up analyses (e.g. differential expression). Some pipelines that exploit common bioinformatic tools have been proposed, one of them being EvidentialGene [18].

The tr2aacds workflow, a part of EvidentialGene pipeline, feeds on the information from so-called over-assembly, where all produced *de novo* assemblies merged into one and true transcripts are expected to be assembled in at least one of the multiple assembly runs. First, tr2aacds predicts coding regions in assembled contigs and defines coding sequences with their complementary polypeptide sequences. In the next step it decreases redundancy. Keeping only the transcripts with the longest and most complete coding region helps to eliminate chimeric and misassembled transcripts. Lastly, it segregates transcripts that are likely isoforms, alleles, or other variations seen at a single locus.

After reference transcriptome construction, it needs to be validated. In general, assemblies can be evaluated either by reference-free metrics, i.e. from statistics such as length and accuracy of contigs, or reference-based metrics, i.e. using biological evidence. Just as for *de novo* assembly, there is no single best evaluation metric for all datasets. Furthermore, the metrics shown as most relevant for *de novo* genome assembly evaluation are not the most appropriate for transcriptome assembly evaluation, due to spatial and temporal dynamics of transcript abundance.

One straight-forward evaluation is to map input reads to the constructed transcriptome. In case of a well-constructed transcriptome it is expected that the majority of reads (e.g. 90%) will map back to the transcriptome. Moreover, the number of all obtained transcripts, as well as their mean, median, N10 and N90 lengths should be within the interval of closely related species transcriptomes. The TransRate software package [21] can be used to access these metrics. TransRate can be also used to analyse constructed polypeptides in respect to proteins or transcripts from a related species. BUSCO [25, 26], on the other hand, measures the percentage of fully reconstructed protein-coding transcripts according to the evolutionarily-informed expectations. In both analyses, the lower the number of contigs for a comprehensive transcriptome description, the better. Other popular transcriptome evaluation tools are DETONATE [41] (https://github.com/deweylab/detonate) and rnaQUAST [42](cab.spbu.ru/software/rnaquast/).

For biological assessment, you can inspect protein homology using BLAST [17] or DIAMOND [24], or protein domain identification using HMMER [43] and PFAM [44].

3.3.1 Decreasing redundancy of assemblies

Navigate to your dedicated software installation directory (`pathToSoftwareDirectory`) and obtain EvidentialGene scripts set containing `tr2aacds2.pl` script (*see* **Note 9**) version 2016 from arthropods.eugenes.org/EvidentialGene/.

Install dependencies: exonerate, cd-hit and BLAST+ separately or fetch all together from arthropods.eugenes.org/EvidentialGene/other/sra2genes_testdrive/evigene_apps_linux_x86_64_19may14

.tar.gz archive using `wget` command and extract archived files into the dedicated software directory. Documentation on EvidentialGene `tr2aacds v2016` procedure can be found at the National Center for Genome Analysis Support Blog (https://blogs.iu.edu/ncgas/2018/12/17/how-evigene-works/).

Within Study 03, make a new assay directory using the `mkdir` command:

```
$ mkdir _A_01_evigene
```

Move `Genotype.all.tr` to the working directory. In case of multiple genotypes, create genotype specific subdirectories, move `Genotype.all.tr` accordingly and run listed commands within them. The `tr2aacds2` script demands one directory for input/output data from which also the script should be run.

In the command line define the number of threads available for this step, as the amount of memory in Mb. Take care that the multiple of defined threads and memory does not exceed 50% of all available memory. Type `free -m` to see memory and swap space usage, then define number of threads and the amount of memory (*see* **Note 10**). For example, if you have 8 treads and 32 Gb memory available, type:

```
$ ncpu=8
$ maxmem=4000
```

Also define where you put EvidentialGene scripts, what will be your working directory, including input/output data, and name of the input FASTA file:

```
$ bapps=pathToSoftwareDirectory
$ evigene=$bapps/evigene/scripts # where tr2aacds2 script is located
$ datad=pathToWorkingDirectory # directory containing input/output data
```

```
$ env trset=Genotype.all.tr # name of the input file
```

Export paths of software tools so that the tr2aacds2 script can find them. Replace 2.x.y with the actual installed NCBI-BLAST+ version:

```
$ export PATH=$bapps/cdhit-master/:$PATH
$ export fastanrdb=$bapps/exonerate/usr/bin/fastanrdb
$ export PATH=$bapps/ncbi-blast-2.x.y+/bin:$PATH
```

The following commands check if all predispositions are set up and run `tr2aacds2` script typing:

```
# check if evigene can see input data
$ if [ "X" = "X$trset" ]; then
  echo "Missing env trset=xxxx.tr"; usage;
  fi
$ if [ "X" = "X$datad" ]; then
  echo "Missing env datad=/path/to/data"; usage;
  fi
# run tr2aacds2 script from evigene using defined parameters
$ $evigene/prot/tr2aacds2.pl -tidy -NCPU $ncpu -MAXMEM $maxmem \
  -log -cdna $trset
```

Within the working directory, results for further steps will be stored in the `okayset` subdirectory as follows:

- okay transcript, coding sequences and polypeptide FASTA files; further named as `all.okay.tr`, `all.okay.cds` and `all.okay.aa`
- okalt transcript, coding sequences and polypeptide FASTA files; further named as `all.okalt.tr`, `all.okalt.cds` and `all.okalt.aa`

3.3.2 Mapping reads back to the EvidentialGene output

To get insight into the percentage of preprocessed reads (see Study 01) that map back to the EvidentialGene generated output, install STAR in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/alexdobin/STAR. Also install samtools and dependencies as described at github.com/samtools.

Within Study 03, create a new assay directory for the STAR step:

```
$ mkdir A_02.1_STAR
```

Concatenate transcripts files (okay and okalt per genotype) using `cat` command from the previous step output, i.e. EvidentialGene `okayset` subdirectory, into `genotype.transcripts.fasta`.

Increase the limit of open files allowed:

```
$ ulimit -n 10000
```

Prepare directory for reference genome file indexing:

```
$ mkdir ./genotype/star_index
```

Generate mapping reference index for merged okay and alt EvidentialGene transcripts (`transcripts.fasta`) using the command:

```
$ pathToSoftwareDirectory/STAR/bin/Linux_x86_64/STAR \
```

```
  --runThreadN 32 --runMode genomeGenerate \
  --genomeDir ./genotype/star_index \
  --genomeFastaFiles pathTo/genotype.transcripts.fasta \
  --limitGenomeGenerateRAM=242371378560
```

where the number of threads is set to 32 (`--runThreadN`) and can be changed arbitrarily.

Run STAR commands per genotype to map Illumina SE preprocessed reads as follows:

```
$ pathToSoftwareDirectory/STAR/bin/Linux_x86_64/STAR \
  --runMode alignReads --runThreadN 32 \
  --genomeDir ./genotype/star_index \
  --readFilesIn pathTo/Illumina_SE.fastq \
  --outFileNamePrefix ./genotype/genotype_mapToEvigene_STAR_SE_ \
  --outSAMtype BAM SortedByCoordinate
```

To map Illumina PE reads to the generated *de novo* transcriptome, split any interleaved PE FASTQ files into two PE files using the following command:

```
$ pathToSoftwareDirectory/bbmap/reformat.sh -Xmx200g \
  in=pathTo/Illumina_PE.fastq out1=./genotype/Illumina_PE_R1.fastq \
  out2=./genotype/Illumina_PE_R2.fastq fastawrap=700 overwrite=true \
  ignorebadquality=t int=t
```

Now run STAR commands per genotype to map Illumina PE preprocessed reads as follows:

```
$ pathToSoftwareDirectory/STAR/bin/Linux_x86_64/STAR \
--runMode alignReads --runThreadN 32 --genomeDir ./genotype/star_index \
--readFilesIn ./genotype/input/Illumina_PE_R1.fastq
./genotype/input/Illumina_PE_R2.fastq \
```

```
--outFileNamePrefix ./genotype/genotype_mapToEvigene_STAR_PE_ \
--outSAMtype BAM SortedByCoordinate
```

Lastly, generate index files for the SE and PE set separately; this will create a .bai file for each `.bam` file. These .bai files are required for for visualisation in graphical viewers for HTS alignments:

```
$ pathToSoftwareDirectory/samtools-1.6/samtools index
./genotype/genotype_mapToEvigene_STAR_SE_Aligned.sortedByCoord.out.bam

$ pathToSoftwareDirectory/samtools-1.6/samtools index
./genotype/genotype_mapToEvigene_STAR_PE_Aligned.sortedByCoord.out.bam
```

Summary of results in tabular format showing the percentage of reads mapped back to the EvidentialGene output transcriptome can be found in files named `Log.final.out`.

3.3.3 Sequence based and reference based validation of EvidentialGene output

To assess the quality of transcriptomes via size-based and reference-based metrics, install TransRate in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/blahah/transrate.

Within Study 03, create a new assay directory for the TransRate step:

```
$ mkdir A_02.2_transrate
```

Download reference genome files (Merged-PhurejaDM-geneModels, *see* **Note 8**) from the GitHub (github.com/NIB-SI/_p_stRT) and uncompress them (e.g. using the `unzip` command).

Concatenate coding sequence files from the evigene assay, `okayset` directory output into `genotype.cds` file using `cat` command.

Run TransRate commands per genotype as follows:

```
$ transrate --assembly=pathTo/genotype.cds \
--reference=pathTo/StPGSC4.04n_seq_3_PGSC-cds-rep_ITAG-cds.fasta \
--threads=24 --loglevel \
--output=genotype_initial_transrate_usingReference >
genotype_initial_transrate_usingReference.log
```

Again, the number of threads (`--threads`) can be adapted. Results summary is stored in `genotype_initial_transrate_usingReference.log` tabular format files.

3.3.4 Biological evidence based annotations

To validate the reliability of transcripts and coding sequences, characterise them by detecting homologous sequences and functional domains and screen them for nucleic acid sequences that may be of vector origin.

3.3.4.1 InterProScan functional analysis

Install InterProScan in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/ebi-pf-team/interproscan/wiki.

Within Study 03, create a new assay directory for the InterProScan annotation step.

```
$ mkdir A_02.3_InterProScan
```

Then, input files can be read using a relative path from the subdirectory `../_A_01_evigene/genotype/okayset/`. Just as for the `tr2aacds2` step, InterProScan (IPS) commands will be run per genotype. IPS can be run separately for okay and okalt set, as shown here, or merged prior to the analysis using the `cat` command. In the case of multiple genotypes, a subdirectory for each should be created.

```
$ pathToSoftwareDirectory/interproscan-5.32-71.0/interproscan.sh \
  --seqtype p -i ../_A_01_evigene/genotype/okayset/all.okay.aa \
  --output-dir genotype_IPS --disable-precalc --formats TSV

$ pathToSoftwareDirectory/interproscan-5.32-71.0/interproscan.sh \
  --seqtype p -i ../_A_01_evigene/genotype/okayset/all.okalt.aa \
  --output-dir genotype_IPS --disable-precalc --formats TSV
```

If using another version of InterProScan, replace 5.32-71.0 with the version number that corresponds to your installed version. Example of outputs, used in further steps, can be found in input_dummy_files/genotype_IPS at github.com/NIB-SI/_p_stRT repository.

To aggregate IPS results per sequence the R `data.table` library can be used. To do this, download `S03_3.3.4.1_aggregateIPS.R` from github.com/NIB-SI/_p_stRT (*see* **Note 8** and **Note 11**), place it in the scripts directory, and run the following commands:

```
$ chmod 755 S03_3.3.4.1_aggregateIPS.R
```

Adapt paths of input/output files within the script as wanted and run it by typing:

```
$ Rscript S03_3.3.4.1_aggregateIPS.R
```

Repeat for each genotype.

3.3.4.2 Transcripts annotation using STARlong followed by MatchAnnot

This step contains STARlong mapping commands followed by MatchAnnot. First, clone (*see* **Note 8**) MatchAnnot in your dedicated software installation directory (`pathToSoftwareDirectory`) from GitHub as follows:

```
$ git clone https://github.com/TomSkelly/MatchAnnot.git
```

Within Study 03, create a new assay directory for the MatchAnnot annotation step:

```
$ mkdir A_02.4_STARlong_matchAnnot
```

As reference genome, the FASTA file accompanied with the corresponding GTF file, already downloaded from the GitHub (*see* section 3.3.3) will be used.

Increase the limit of open files allowed:

```
$ ulimit -n 10000
```

Create reference genome index using 28 threads:

```
$ pathTo/STARlong --runThreadN 28 --runMode genomeGenerate
--genomeDir ./star_index \
--genomeFastaFiles StPGSC4.04n_seq_3_PGSC-cds-rep_ITAG-cds.fasta \
--sjdbGTFfile StPGSC4.04n_PGSC-ITAG-merged_representative-transcript_genes.gtf \
--limitGenomeGenerateRAM 210000000000
```

Setup one-time reference genome loading to be used by all the samples:

```
$ STARlong --genomeLoad LoadAndExit --genomeDir ./star_index
```

Map transcripts to the reference, per genotype. You can do this separately for okay and okalt set or concatenate files prior to STARlong run. Example code for run on the `all.okay.tr` set:

```
$ STARlong --runMode alignReads \
  --outSAMattributes NH HI NM MD --outFilterType BySJout \
  --outFilterMultimapNmax 20 \
  --alignSJoverhangMin 8 --alignSJDBoverhangMin 1 \
  --outFilterMismatchNmax 999 --outFilterMismatchNoverReadLmax 0.08 \
  --alignIntronMin 20 --alignIntronMax 1000000 \
  --alignMatesGapMax 1000000 --outSAMtype SAM \
  --limitBAMsortRAM 100000000000 --runThreadN 28 \
  --genomeDir ./star_index \
  --readFilesIn ../_A_01_evigene/genotype/okayset/all.okay.tr \
  --outFileNamePrefix ./genotype.tr.okay_ref_STARlong. \
```

```
  --seedPerReadNmax 100000 --seedPerWindowNmax 1000 \
  --seedSearchLmax 30 --seedSearchStartLmax 30 \
  --alignTranscriptsPerReadNmax 100000 \
  --alignTranscriptsPerWindowNmax 10000 \
  --scoreGapNoncan -20 --scoreDelOpen -1 --scoreDelBase -1 \
  --scoreInsOpen -1 --scoreInsBase -1 \
  --chimMultimapNmax 20 --chimSegmentMin 100
```

Repeat for all genotypes, by adapting `--readFilesIn` and `--outFileNamePrefix` values.

Unload reference

```
$ STARlong --genomeLoad Remove --genomeDir ./star_index
```

Sort all generated SAM files in all subdirectories

```
$ for i in $(ls ./*.sam | sed s/.sam// )
  do sort -k 3,3 -k 4,4n ${i}.sam > ${i}.sorted.sam
  done
```

Run MatchAnnot on all sorted SAM files

```
$ for i in $(ls ./*.sorted.sam ); do
  pathToSoftwareDirectory/MatchAnnot/matchAnnot.py > --gtf=./StPGSC4.04n_PGSC-
ITAG-merged_representative-transcript_genes.gtf --format=alt \
  ${i} > ${i}.matchAnnot.txt; done
```

Parse MatchAnnot results, i.e keep only columns containing transcriptID, reference geneID, reference transcriptID, exon match (i.e. which exon) and match score:

```
$ for i in $(ls ./*.sorted.sam.matchAnnot.txt | sed s/.txt// ); do
  grep "result:" ${i}.txt | tr -s ' ' | awk -F'[ ]' '{print $2, $3, $4, $6, $8}'
> ${i}.parsed.txt; done
```

3.3.4.3 Vector segment contamination scanning

Install Vecscreen in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/aaschaffer/vecscreen_plus_taxonomy. Within Study 03, create a new assay directory for the Vecscreen annotation step:

```
$ mkdir _A_02.5_Vecscreen
```

Export path to Vecscreen installation directory

```
$ VECPLUSDIR=pathToSoftwareDirectory/Vecscreen
```

Download UniVec from NCBI and create database (DB) for Vecscreen in your DB dedicated directory:

```
$ wget ftp://ftp.ncbi.nih.gov/pub/UniVec/UniVec_Core
$ makeblastdb -in UniVec_Core -input_type fasta -dbtype nucl \
  -parse_seqids -out bioDB_dedicated_directory/UniVecDB
```

Run Vecscreen per genotype transcripts FASTA file:

```
$ $VECPLUSDIR/scripts/vecscreen -query pathTo/transcripts.fasta \
```

```
-db pathTo/UniVecDB -out vecscreen.out -text_output
```

Get the first column containing transcript IDs from Vecscreen output and save it as `IDs_column.txt` file. Make a subsetted FASTA file including only sequences with those IDs and run blastn against nt DB.

```
$ xargs faidx -d ' ' pathTo/transcripts.fasta < IDs_column.txt > subset.fasta
$ blastn -query subset.fasta -db pathTo/nt -out nt.out -num_threads 24
```

Tweak number of threads used by changing the value of `-num_threads`.

Obtain the best blastn result per query using `sumablastplus.pl` script from github.com/NIB-SI/_p_stRT. Download (*see* **Note 8**) and run the script:

```
$ pathTo/sumablastplus.pl -maxhits 1 -empties nt.out nt.out_top1.tsv
```

where `-maxhits` defines the maximum number of hits per query to include in output, `-empties` defines whether to include empty queries (with no hits found), `nt.out` is the input file (should be output from BLAST program) and `nt.out_top1.tsv` is the output file (tab separated).

Merge Vecscreen and blastn top hit results using a spreadsheet software e.g. Excel. Add a coverage column calculated as ratio of query sequence length and blastn alignment length.

3.3.4.4 Annotations using DIAMOND

Install DIAMOND in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/bbuchfink/diamond. Within Study 03, create a new assay directory for the DIAMOND annotation step:

```
$ mkdir _A_02.6_DIAMOND
```

Obtain FASTA files from UniProt using query keywords "*Solanum tuberosum*'', "Solanaceae'' and "Viridiplantae". Also download FASTA files of the complete SwissProt and TrEMBL databases from www.uniprot.org/downloads and concatenate them in one UniProt FASTA file.

Export paths to your DIAMOND installation directory and dedicated biological databases directory:

```
$ myDIAMOND="pathTo/diamond/diamond"
$ myDB=pathTo/myDB
```

Prepare DIAMOND databases:

```
$ myDIAMOND makedb --in pathTo/solanumTuberosum.fasta -d solanumTuberosum
$ myDIAMOND makedb --in pathTo/Solanaceae.fasta -d Solanaceae
$ myDIAMOND makedb --in pathTo/Swissprot_TrEMBL_plants.fasta -d plants
$ myDIAMOND makedb --in pathTo/UniProt.fasta -d uniprot
```

Prepare query files: concatenate cds and transcript okay and okalt files from the evigene okayset output directory using `cat` command.

Run DIAMOND for each genotype per transcript and cds set per database. Example for `solanumTuberosum` database and transcripts query as follows:

```
$ $myDIAMOND blastx --db $myDB/solanumTuberosum.dmnd \
  -q pathTo/genotype.tr.fasta -F 15 --range-culling --top 10 --sensitive \
  -f 0 --threads 30 --evalue 0.00001 -o tr_solanumTuberosum.out
```

Change the number of threads (`--threads`) and e-value cut-off (`--evalue`) as preferred.

Obtain best blast result per query using `sumablastplus.pl` script from github.com/NIB-SI/_p_stRT (*see* **Note 8**). Example for transcripts query hits within `solanumTuberosum` DB:

```
$ pathTo/sumablastplus.pl -maxhits 1 -empties \
  tr_solanumTuberosum.out genotype.tr_solanumTuberosum_top1.tsv
```

where `-maxhits` defined the maximum number of hits per query to include in output, `-empties` defines whether to include empty queries (with no hits found), `tr_solanumTuberosum.out` is the input file (should be output from DIAMOND blast program) and `genotype.tr_solanumTuberosum_top1.tsv` is the output file (tab separated).

3.3.5 Transcriptome filtering and annotation

As tr2aacds from EvidentialGene uses only the internal evidence to reduce the initial over-assembly, obtaining external evidence, such as one from PFAM DB, BLAST DB of closely related organisms, Gene Ontology [45], KEGG [46], MapMan [47], etc., is a necessary follow-up step. The results from MatchAnnot, InterProScan and DIAMOND can be used as biological evidence for further filtering. Transcripts that do not map to the reference nor have any significant hits in either InterPro or UniProt should be eliminated from further analysis to obtain more reliable transcriptomes.

3.3.5.1 Transcriptome filtering

Within Study 03, create a new assay directory for the biological filtering step:

```
$ mkdir _A_03.1_filtering
```

Within the assay directory, create subdirectories input, output, scripts and intermediate:

```
$ mkdir input output scripts intermediate
```

Make necessary subdirectories while positioned in the scripts subdirectory per each used genotype using `mkdir` command as follows:

```
$ mkdir ../intermediate/genotype/ ../output/genotype/
```

Calculate sequence lengths. Prior to length calculation, transcript and coding sequences from okay and okalt EvidentialGene output directory should be concatenated:

```
$ cat ../../_A_01_evigene/genotype/okayset/all.okay.tr \
  ../../_A_01_evigene/genotype/okayset/all.okalt.tr | seqkit fx2tab \
  --length --name --header-line >
../intermediate/genotype/genotype.tr_lengths.tsv

$ cat ../../_A_01_evigene/genotype/okayset/all.okay.cds \
  ../../_A_01_evigene/genotype/okayset/all.okalt.cds | seqkit fx2tab \
  --length --name --header-line >
../intermediate/genotype/genotype.cds_lengths.tsv
```

Move or copy all relevant output files from the biological annotation steps to the `../intermediate/genotype/` subdirectory. Those include outputs from: Vecscreen, InterProScan, DIAMOND and MatchAnnot. Rename DIAMOND top hit outputs by adding integers with leading zero at the beginning of the file, in alternating order for *Solanum tuberosum*, Solanaceae, Viridiplantae and UniProt .cds and .tr results (see example in input_dummy_files/intermediate/genotype at github.com/NIB-SI/_p_stRT). Script `S03_3.3.5.1_filtering.R` from github.com/NIB-SI/_p_stRT uses this information to filter out sequences without biological evidence. For DIAMOND output, E-value cut-off of 10^{-5} and query transcript/cds and target sequence alignment coverage higher or equal to 50% is used. Only *Solanum tuberosum*, Solanaceae and Viridiplantae custom databases are considered.

Make sure that you adapt all relative paths and genotype names to your system prior using the script. Here, a generic name 'genotype' will be used for an example. Download the script (*see* **Note 8** and **Note 11**) to the scripts subdirectory and run it by typing (for an alternative *see* **Note 12**):

```
$ Rscript S03_3.3.5.1_filtering.R
```

The script generates three files: i) `../output/genotype/genotype_tr.cds.tsv` – merged information on which filtering criteria were used, ii) `../output/genotype/genotype_keep_IDs.tsv` – list of IDs which passed the filtering cut-off, and iii) `../output/genotype/genotype_drop_IDs.tsv` – list of IDs which didn't pass the filtering cut-off criteria.

Transcripts that do not map to the genome nor have any significant hits in either InterPro or custom UniProt databases (here also including hits for their accompanying coding sequence) are eliminated from further analysis.

3.3.5.2 Paralogue clusters and transcriptome annotation

Within Study 03, create a new assay directory:

```
$ mkdir _A_03.2_components
```

This step uses `igraph` library collection for network analysis [48] (available for Python and R) to re-assign post-filtering representative and alternative classes and to obtain finalised genotype-specific transcriptomes (*see* **Note 13**).

First obtain sequence relations from polypeptide EvidentialGene FASTA output (i.e. file extensions *.aa in the okayset folder of EvidentialGene output):

```
$ for f in okayset/*; do ll $f; grep '>' $f | wc -l; done
$ touch IDs.txt
$ for f in okayset/*.aa; do grep '>' $f >> IDs.txt; done
$ sed -e 's/>//g' -e 's/ /\t/g' IDs.txt | sponge IDs.txt
$ cut -f1,8 IDs.txt | sponge IDs.txt
$ sed -e 's/evgclass=/\t/g' -e 's/,okay,/\t/g' -e 's/match:/\t/g' -e
's/,pct:/\t/g' -e 's/\/-sense\//\t/g' -e 's/\/\.\//\t/g' -e 's/;//g' -e 's/-
sense;//g' IDs.txt | sponge IDs.txt
$ cut -f1,5,7 IDs.txt | sponge IDs.txt
```

Then calculate sequence length for polypeptide EvidentialGene FASTA output:

```
$ touch seq_len.txt
```

```
$ for f in okayset/*.aa; do
  seqkit fx2tab --length --name --header-line $f >> seq_len.txt;
  done
$ sed -e 's/ /\t/g' seq_len.txt | sponge seq_len.txt
$ cut -f1,11 seq_len.txt | sponge seq_len.txt
$ sed 1d seq_len.txt | sponge seq_len.txt
```

`Seq_len.txt`, `IDs.txt` and `genotype_keep_IDs.tsv` (from the previous step) are input files for `S03_3.3.5.2_clusters.R`. Move all required files in the input subdirectory (see input_dummy_files/input at github.com/NIB-SI/_p_stRT) and run script (*see* **Note 8** and **Note 11**) from scripts subdirectory typing:

```
$ Rscript S03_3.3.5.2_clusters.R
```

Script `S03_3.3.5.2_clusters.R` creates representative-alternatives groups from the EvidentialGene classification (i.e. evigene IDs and accompanying descriptions in .fasta headers), filters-out ones that didn't pass the filtering threshold, and re-defines the transcript with longest coding sequence within the group as a representative one.

According to post-filtering re-classification, create annotated FASTA files (here an example for transcripts FASTA file) in a few steps as follows:

```
# Concatenate evigene output and remove evigene descriptions for header
$ cat ../../_A_01_evigene/genotype/okayset/all.okay.tr
../../_A_01_evigene/genotype/okayset/all.okalt.tr > ../output/genotype_all.tr
$ cut -f1 -d " " ../output/genotype_all.tr | sponge ../output/genotype_all.tr
```

```
$ grep ">" ../output/genotype_all.tr | wc -l

# create subsetted FASTAs
$ xargs faidx -d ' ' ../output/genotype_all.tr <
../output/representativesIDs.tsv > ../output/representatives.tr 2>
../output/representatives.err.txt
$ xargs faidx -d ' ' ../output/genotype_all.tr < ../output/alternativesIDs.tsv
> ../output/alternatives.tr 2> ../output/alternatives.err.txt

# linearise FASTAs
$ fasta_formatter -i ../output/representatives.tr -o
../output/genotype_representatives.tr -w 0;
$ fasta_formatter -i ../output/alternatives.tr -o
../output/genotype_alternatives.tr -w 0;

$ rm ../output/representatives.tr
$ rm ../output/alternatives.tr
$ rm ../output/genotype_all.tr*

$ awk 'NR==FNR{a[$1]=$2;next} NF==2{$2=a[$2]; print ">" $2;next} 1' FS='\t'
../output/alias.rep FS='>' ../output/genotype_representatives.tr | sponge
../output/genotype_representatives.tr
$ awk 'NR==FNR{a[$1]=$2;next} NF==2{$2=a[$2]; print ">" $2;next} 1' FS='\t'
../output/alias.alt FS='>' ../output/genotype_alternatives.tr | sponge
../output/genotype_alternatives.tr
```

Step with command `xargs faidx` creates a FASTA subset according to the provided list of IDs stored in `.tsv` file. If `*err.txt` is not empty, something has gone wrong. Check if empty and remove them:

```
$ rm ../output/*.err.txt
```

Command `fasta_formatter` linearises the FASTA file. The set of `awk` commands adds user defined description to headers in the FASTA file according to `alias` files, which are two column files containing existing FASTA headers and desired ones, obtained through the R `S03_3.3.5.2_clusters.R` script (Windows R users *see* **Note 14**). Users can easily modify or completely change alias files using a spreadsheet application. Instead of `awk`, commands described in **Note 15** can be used.

Repeat FASTA generation steps for coding sequences and polypeptides FASTA files. Repeat all steps per genotype.

3.3.6 Reference transcriptome sequence based and reference based validation

Re-run TransRate statistics to obtain improvement in quality of final transcriptomes due to filtering step via size-based and reference-based metrics. Within Study 03, create a new assay directory for this TransRate step:

```
$ mkdir _A_04_transrate
```

Concatenate coding sequences from the previous step into one file, e.g. `genotype_cds_all.fasta`. The command is analogous to the previous TransRate command:

```
$ transrate --assembly=output_from_previous_step/genotype_cds_all.fasta \
--reference=pathTo/StPGSC4.04n_seq_3_PGSC-cds-rep_ITAG-cds.fasta \
--threads=24 --loglevel --output=genotype_cds_transrate_usingReference >
genotype_cds_transrate_usingReference.log
```

3.3.7 Reference transcriptome BUSCO completeness estimation

Install BUSCO in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at busco.ezlab.org/. Within Study 03, create a new assay directory for the BUSCO completeness estimation step:

```
$ mkdir _A_05_BUSCO
```

Calculate completeness of each *de novo* transcriptome using embryophyta lineage data and/or solanaceae. Total complete BUSCOs as sum of complete and single-copy BUSCOs and complete and duplicated BUSCOs.

Download lineage data from busco.ezlab.org/frames/plants.htm to your dedicated biological databases directory, e.g. `buscoDB`. Run BUSCO command on the polypeptide dataset to speed up the analysis as follows, considering that all polypeptide FASTA files are within the same directory when running BUSCO and that they all have 'aa' and '.fasta' as a part of their name.

Export path to BUSCO python script and to BUSCO databases directory:

```
$ myBUSCO=pathToSoftwareDirectory/busco/scripts/run_BUSCO.py
```

```
$ myDB=pathTo/buscoDB
```

Run BUSCO on all polypeptide fasta files using for e.g. Solanaceae v10 and Embryophyta v9 databases:

```
$ for f in pathTo/*aa*.fasta
  do
    python $myBUSCO
    -i $f -o BUSCO_"${f##*/}"_solanaceae_odb10 \
    -l $myDB/solanaceae_odb10 --mode prot -c 30 -t ./tmp;
    python $myBUSCO \
    -i $f -o BUSCO_"${f##*/}"_embryophyta_odb9 \
    -l $myDB/embryophyta_odb9 --mode prot -c 30 -t ./tmp;
  done
```

Increase or decrease number of threads changing value of `-c` parameter. See results within `./*/short_summary_BUSCO*.`

3.3.8 Reference transcriptome taxonomic classification

Install Centrifuge in your dedicated software installation directory (`pathToSoftwareDirectory`) as described at github.com/infphilo/centrifuge. Within Study 03, create a new assay directory for the Centrifuge taxonomic classification of transcriptomes step:

```
$ mkdir _A_07_centrifuge
```

You can download the prebuild Centrifuge NCBI nt database index from the webpage (cb.jhu.edu/software/centrifuge) or build a custom database within your biological databases directory as documented in the centrifuge manual. The potato taxonomy ID 4113 is supplied (`--host-taxid`).

Run centrifuge read taxonomic classification using the command:

```
$ pathToSoftwareDirectory/centrifuge -x pathTo/centrifugeDB/nt -f \
  -U transcriptome.fasta -S output_centrifuge.txt \
  --report-file output_report.txt --host-taxid 4113 --seed 42 -p 30
$ centrifuge-kreport -x pathTo/centrifugeDB/nt \
  output_centrifuge.txt > output_kreport.txt
```

Repeat centrifuge commands per each genotype. To visualise the results, run the online application Pavian (fbreitwieser.shinyapps.io/pavian/) and upload your `output_kreport` files. After the data is processed, click on the link "Generate HTML report ... " in the menu on the left to save the generated report.

3.3.9 Multiple Sequence Alignment of paralogue clusters

Inspect your paralogue cluster sequence assignment using multiple sequence alignment (MSA) algorithms. Web service GUI tools are available at www.ebi.ac.uk/Tools/msa/. For nucleotide multiple alignment we recommend Clustal Omega [31] (use at least 5 iterations), while for polypeptides MAFFT [29]. Visualise alignments with MView [35].

To automatically generate MSA of all paralogue clusters you can use the `S03_3.3.9_MSA.py` script from github.com/NIB-SI/_p_stRT/. Prior to running the script, a reference FASTA file and a directory

containing sequence identifiers of interest should be prepared. One of the options is to list sequence identifiers per cluster (i.e. create multiple files containing IDs, see input_dummy_files/folderWithIds at github.com/NIB-SI/_p_stRT) from `S03_3.3.5.2_clusters.R` paralogue clusters table output (see `clusters.tsv` in minimal_example_out/output at github.com/NIB-SI/_p_stRT). Also, install the following prerequisites: pyfaidx [10], Clustal Omega, ClustalW2 (with Simple Phylogeny), MAFFT, Muscle [30] and MView and adapt paths to the software in the `print.sh` script (*see* **Note 16**).

R code example for paralogue clusters (from `clusters.tsv` file) containing equal or more than 30 (`n`) sequences:

```
> pathToFile = getwd()
> folderWithIds = "folderWithIds"
> n = 30

> clusters = read.delim(file = paste0(pathToFile , "/clusters.tsv"),
                        header = TRUE, sep = "\t", quote = NULL, dec = ".",
                        stringsAsFactors = FALSE, na.strings = "NA",
                        comment.char = "#", fill = TRUE)

> interestingClusters = clusters[clusters$seqCount >=n,]

> dir.create(file.path(folderWithIds), showWarnings = FALSE)
> dir.create(file.path(folderWithIds, paste0("n_", n)), showWarnings = FALSE)

> myClu = sort(unique(interestingClusters$clu))
> for (i in myClu) {
```

```
    subset = interestingClusters[which( i == interestingClusters$clu),]
    subset = dplyr::arrange(subset, desc(status), ID)
    fname = gsub(" ", "_", i)
    write.table(x = subset$ID,
                file = paste0("folderWithIds/n_", n, "/", fname, "IDs.txt"),
                append = FALSE, quote = FALSE, sep = "\t", eol = "\n",
                na = "NA", dec = ".", row.names = FALSE, col.names = FALSE)
  }
```

Let's say that your FASTA file is `pathTo/aa.fasta` (see input_dummy_files/my.test.aa.fasta at github.com/NIB-SI/_p_stRT) and a set of selected IDs per cluster are within `folderWithIds` directory (see input_dummy_files/folderWithIds at github.com/NIB-SI/_p_stRT). Download `S03_3.3.9_MSA.py` script and its dependent `print.sh` script (*see* **Note 8**) and run it by typing:

```
$ python2 S03_3.3.9_MSA.py
```

You will be asked to enter some choices, such as:

```
Path to your reference .fasta file: pathTo/aa.fasta
Path to your folder containing files with written gene IDs of interest, line by
line: pathTo/folderWithIds
Where would you like to store your results (directory name, e.g.
here/and/there)?: your top_level_output_directory
Preferred width of the alignment is (full or integer): 180
Consensus on or off: off
```

e.g. for ClustalOmega MSA you may use the following choices:

```
Algorithm choice [1-3]: 1
Number of iterations: 5
```

```
Number of CPU (up to 30): 24
```

MSA outputs (.html visualisation, phylogenetic tree, etc.) will be available at output subdirectory (within your top-level output directory) named according to the selected parameters.

3.4 Study 04 - pan-transcriptome construction

Combine genotype-specific representative coding sequences with coding sequences from the merged Phureja DM gene models (StPGSC4.04n_seq_3_PGSC-cds-rep_ITAG-cds.fasta downloaded in the previous steps from github.com/NIB-SI/_p_stRT, containing 17,880 and 31,442 non-redundant PGSC and ITAG genes, respectively) and cluster them using `cdhit-est` algorithm [16]; we recommend using global sequence identity threshold 90%, alignment coverage for the shorter sequence 75%, bandwidth of alignment 51 nt and word length of 9 using the command:

```
$ pathToSoftwareDirectory/cd-hit-est -i pathTo/combined.cds.fasta \
  -o stCuSTr_cds_repAlt_cdhitEST-conda_aS.75_c.90 \
  -G 1 -n 9 -g 1 -c 0.90 -aS 0.75 -p 1 -bak 1 -d 200 -b 51 -uS 0.25 \
  -M 0 -T 28
```

See an example of an output in minimal_example_out/cdhit-est_out at github.com/NIB-SI/_p_stRT.

Separate sequences that did not cluster using `cdhit-est` into tetraploid (`singletons.tetra.seqID.txt`) and DM (`singletons.DM.seqID.txt`) datasets, and generate .txt files containing their IDs (one ID per line) using a spreadsheet software or R script `S03_3.4_parse_cdhit-est-output.R` available from github.com/NIB-SI/_p_stRT (*see* **Note 8** and **Note 11**). If files containing sequence IDs were not generated on a Linux based system *see* **Note 17**.

Combine genotype-specific representative polypeptide sequences with polypeptide sequences from the merged Phureja DM gene models into `combined.aa.fasta` and create polypeptide FASTA subsets using the following commands:

```
$ xargs faidx -d ' ' pathTo/combined.aa.fasta <
pathTo/singletons.tetra.seqID.txt > stPanTr_aa_singletons.tetraploid.fasta 2>
stPanTr_aa_singletons.tetraploid.error;

$ xargs faidx -d ' ' pathTo/combined.aa.fasta < pathTo/singletons.DM.seqID.txt
> stPanTr_aa_singletons.DM.fasta 2> stPanTr_aa_singletons.DM.error
```

Replace * in polypeptide ITAG sequences (identifiers beginning with Sotub) with X:

```
$ sed 's/[*]/X/g' stPanTr_aa_singletons.DM.fasta | sponge
stPanTr_aa_singletons.DM.fasta
```

Subject sequences to the cdhit-2d algorithm using local sequence identity threshold 90%, alignment coverage for the shorter sequence 45%, bandwidth of alignment 45 nt and word length of 5 as follows:

```
$ pathToSoftwareDirectory/cdhit-2d -i stPanTr_aa_singletons.DM.fasta \
-i2 stPanTr_aa_singletons.tetraploid.fasta \
-o stPanTr_aa_singletons_tetra_on_DM_aS.45_c.90 \
-G 0 -n 5 -g 1 -c 0.90 -aS 0.45 -p 1 -bak 1 -d 200 -b 45 -uS 0.55 \
-M 0 -T 28;

$ pathToSoftwareDirectory/cdhit-2d -i stPanTr_aa_singletons.tetraploid.fasta \
-i2 stPanTr_aa_singletons.DM.fasta \
```

```
-o stPanTr_aa_singletons_DM_on_tetra_aS.45_c.90 \
-G 0 -n 5 -g 1 -c 0.90 -aS 0.45 -p 1 -bak 1 -d 200 -b 45 -uS 0.55 \
-M 0 -T 28
```

See an example of an output see /minimal_example_out directory at github.com/NIB-SI/_p_stRT.

R script `S03_3.4_parse_cdhit-2d-output.R`, available from github.com/NIB-SI/_p_stRT, can be used to parse cdhit-2d outputs (*see* **Note 8** and **Note 11**).

Sequences that are shared by the DM merged gene model and *de novo* assembled cultivar-specific transcriptomes can be designated as "core" transcripts, and sequences that were assembled in only one transcriptome can be designated "genotype-specific".

4 Notes

1. In de Brujin graph-based *de novo* assembly erroneous tails of the reads can cause dead ends in the graph and faulty isoforms definition thus increasing the number of nodes, computation time and memory usage. Take into consideration that a few *de novo* assemblers already include some read preprocessing steps.
2. TrimGalore auto-detects adapter sequences. If only one read from the pair passes the filtering threshold, it can be retained using `--retain_unpaired` parameter and the trimmed intermediate file is kept using the `--keep` parameter. PE files need to be listed in a pairwise fashion, e.g. `R1_001.fastq.gz R2_001.fastq.gz R1_002.fastq.gz R2_002.fastq.gz`. Reads shorter than 35 bp will be discarded by default. To find out more options run:

   ```
   $ trim_galore –help
   ```

3. When one of the two reads in a pair (PE reads) is discarded, the remaining orphaned reads are treated as SE reads. SOLiD reads are in colour-space and thus have to be preprocessed separately and converted into double encoding using Perl script "`denovo_preprocessor_solid_v2.2.1.pl`" for the Velvet assembler [13].
4. Alternatively, digital normalisation can be performed also with the khmer software package [49] using the script "`normalize-by-median.py`". More on khmer documentation can be found here: https://khmer.readthedocs.io/en/latest/user/scripts.html#digital-normalization.
5. Apart from rnaSPAdes, Velvet/Oases and Trinity, you can also use other genome-independent *de novo* assemblers e.g. SOAPdenovo-Trans [50], Trans-ABySS [51], T-IDBA [52], TransLiG [53] and SAT-Assembler [54]. For more information about short read *de novo* assemblers we kindly see review and benchmarking articles [6, 55].
6. For strand-specific reverse-forward oriented PE reads, `--ss-rf` should be added to the command. Input files can also be in compressed format, i.e. `fastq.gz` files.

7. In the case you are using SOLiD reads, remove the parameter `-ins_length` and add parameter `-amos_file yes`. The Illumina PE reads used in the `velveth` command (`Illumina_PE.tadpole.BBnorm.fastq`) are in interleaved FASTQ file format. If your PE reads are in two separate FASTQ files (one file with first and the other with second in pair reads) add the parameter `-separate` before the file . Also for PE reads, use `-scaffolding no` to turn off the default scaffolding option because this can introduce unwanted Ns in the output sequences.

8. Obtaining materials from the GitHub:

   i) use wget to pull down the raw file

   ```
   $ wget https://raw.githubusercontent.com/username/reponame/path/to/file
   ```

   ii) use git clone to pull the complete repository (prerequisites: user with sudo privileges)

   ```
   $ sudo apt update
   $ sudo apt install git
   $ git clone https://github.com/username/reponame.git
   ```

9. Installing Perl:

   i) Install Perl on Ubuntu-like Linux OS (prerequisites: user with sudo privileges)

   ```
   $ sudo apt update
   $ sudo apt-get install perl
   # check version
   $ perl -v
   ```

   More details for installing Perl can be found at www.perl.org/get.html#unix_like

   ii) Use CPAN ('Comprehensive Perl Archive Network', www.cpan.org) to install Perl modules.

   Also, install `cpanm` to make installing other modules easier (metacpan.org/pod/App::cpanminus)

   ```
   $ cpan App::cpanminus
   ```

   iii) Install different version of Perl

   ```
   $ sudo cpan App::perlbrew
   $ perlbrew init
   # see which versions are available:
   ```

```
$ perlbrew available
# install version 5.X.Y
$ perlbrew install perl-5.X.Y
# list all installed versions
$ perlbrew list
# change Perl for the current shell # (or per your sessions)
$ perlbrew use perl-5.X.Y # (or perlbrew switch perl-5.X.Y)
$ which perl
# revert version to default for the current shell # (or per your sessions)
$ perlbrew off # (or perlbrew switch-off)
```

More details can be found at perlbrew.pl.

10. Total memory usage while running EvidentialGene scripts will be a multiplication of the defined `ncpu` and `maxmem` parameters.

11. Install R on Ubuntu-like Linux OS (prerequisites: user with sudo privileges)

with `add-apt-repository` (adapt values in the square brackets and delete the brackets)

i) Import the repository public key

```
$ apt-key adv --keyserver [from this location or server] --recv-keys [retrieve key(s)]
```

ii) Add the CRAN repository to your system sources' list

```
$ sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu [type
appropriate selection from https://cloud.r-project.org/bin/linux/ubuntu/]'
```

iii) Install the complete R system

```
$ sudo apt-get update
$ sudo apt install r-base
$ sudo apt-get install r-base-dev
$ sudo apt install build-essential
```

More details available at https://cran.r-project.org/bin/linux/ubuntu/README.html

12. Users can merge and filter the results in their own way using a spreadsheet software, if that generates the necessary output files containing sequence IDs that passed the filtering threshold. If you are running R

scripts on Windows, take care to call the `dos2unix` command on the output that will be used in the later steps performed in Linux, e.g. for output of `S03_3.3.5.1_filtering.R` script containing sequence IDs to keep or drop run:

```
$ sudo apt-get install dos2unix
```

`$ dos2unix genotype_keep_IDs.tsv` and `dos2unix genotype_drop_IDs.tsv`.

13. One can use cd-hit-2d to ensure optimal alternatives assignment post filtering. Exemplary code as follows:

```
$ cdhit-2d -i representatives.fasta -i2 alternatives.fasta -o aa_local_aS.51_c.70 -G 0 -n
4 -g 1 -c 0.70 -aS 0.51 -p 1 -bak 1 -d 200 -M 0 -T 24;
```

14. If you run R script `S03_3.3.5.2_clusters.R` on Windows OS, type for the outputs:

```
$ dos2unix ../output/representativesIDs.tsv
$ dos2unix ../output/alternativesIDs.tsv
```

15. Alternative to the `awk` command, to add the user defined description to headers in the FASTA files according to the `alias.tsv` files, you can use:

```
$ git clone https://github.com/nylander/translate_fasta_headers.git
$ for f in ../pathTo/*.fasta
  do
    tmp=${f##*/}
    echo "${f##*/}"
    grep ">" ../pathTo/$tmp | wc -l
    translate_fasta_headers/translate_fasta_headers.pl \
        -tabfile=../pathTo/alias.tsv --in ../pathTo/$tmp \
        --out ../output/"annot_"$tmp
    grep ">" ../output/"annot_"$tmp | wc -l
  done
```

16. Installing MSA prerequisites:

i) Installations details for MSA tools can be found at:

www.ebi.ac.uk/seqdb/confluence/display/JDSAT/Multiple+Sequence+Alignment

ii) Installing pip for Python 3 and Python2 on Ubuntu-like OS (prerequisites: user with sudo privileges)

```
# Installing pip for Python 3
$ sudo apt update
$ sudo apt install python3-pip
# Installing pip for Python 2
$ sudo apt update
$ sudo apt install python2
$ curl https://bootstrap.pypa.io/get-pip.py --output get-pip.py
$ sudo python2 get-pip.py
```

iii) Installing pyfaidx

```
$ pip install pyfaidx
```

See more details at pypi.org/project/pyfaidx. Use pip install for any other missing modules and packages.

iv) Adapt paths according to your installation location within `print.sh` script under the `# write proper paths` comment line for: Muscle, PERL5LIB, Mview, ClustalW, simple_phylogeny.pl, ClustalOmega and MAFFT.

17. To make sequence IDs files compatible with Linux, run `dos2unix` command on `singletons.tetra.seqID.txt` and `singletons.DM.seqID.txt` files.

Acknowledgments

The authors would like to thank the coworkers from the Department of Biotechnology and Systems Biology of the National Institute of Biology for providing raw RNA-Seq datasets, Henrik Krnec for BLAST output parser, and the Omics bioinformatics team for critical discussions. This project was supported by the Slovenian Research Agency (grants P4-0165, J4-1777, J4-4165, J4-7636, J4-8228, J4-9302 and Z7-1888), COST actions CA15110 (CHARME) and CA15109 (COSTNET).

References

1. Zhao L, Zhang H, Kohnen M V., et al (2019) Analysis of transcriptome and epitranscriptome in plants using pacbio iso-seq and nanopore-based direct RNA sequencing. Front Genet 10:1–14. https://doi.org/10.3389/fgene.2019.00253
2. Xu X, Pan S, Cheng S, et al (2011) Genome sequence and analysis of the tuber crop potato. Nature 475:189–195. https://doi.org/10.1038/nature10158
3. Denisov G, Walenz B, Halpern AL, et al (2008) Consensus generation and variant detection by Celera Assembler. Bioinformatics 24:1035–1040. https://doi.org/10.1093/bioinformatics/btn074
4. de Bruijn NG (1946) A combinatorial problem. Proc Sect Sci K Ned Akad van Wet te Amsterdam 49:758–764
5. Moreton J, Izquierdo A, Emes RD (2016) Assembly, Assessment, and Availability of De novo Generated Eukaryotic Transcriptomes. Front Genet 6:361. https://doi.org/10.3389/fgene.2015.00361
6. Hölzer M, Marz M (2019) De novo transcriptome assembly: A comprehensive cross-species comparison of short-read RNA-Seq assemblers. Gigascience 8:1–16. https://doi.org/10.1093/gigascience/giz039
7. Wang S, Gribskov M (2017) Comprehensive evaluation of de novo transcriptome assembly programs and their effects on differential gene expression analysis. Bioinformatics 33:327–333. https://doi.org/10.1093/bioinformatics/btw625
8. Zhao QY, Wang Y, Kong YM, et al (2011) Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study. BMC Bioinformatics 12 Suppl 1: https://doi.org/10.1186/1471-2105-12-S14-S2
9. Zhang G, Sun M, Wang J, et al (2019) PacBio full-length cDNA sequencing integrated with RNA-seq reads drastically improves the discovery of splicing transcripts in rice. Plant J 97:296–305. https://doi.org/10.1111/tpj.14120
10. Shirley M, Ma Z, Pedersen B, Wheelan S (2015) Efficient “pythonic” access to FASTA files using pyfaidx. https://doi.org/10.7287/peerj.preprints.970v1
11. Ewels P, Magnusson M, Lundin S, Käller M (2016) MultiQC: Summarize analysis results for multiple tools and samples in a single report. Bioinformatics 32:3047–3048. https://doi.org/10.1093/bioinformatics/btw354
12. Bushmanova E, Antipov D, Lapidus A, Prjibelski AD (2019) RnaSPAdes: A de novo transcriptome assembler and its application to RNA-Seq data. Gigascience 8:1–13. https://doi.org/10.1093/gigascience/giz100
13. Zerbino DR (2010) Using the Velvet de novo assembler for short-read sequencing technologies. Curr. Protoc. Bioinforma. 31:11.5.1-11.5.12
14. Schulz MH, Zerbino DR, Vingron M, Birney E (2012) Oases: Robust de novo RNA-seq assembly across the dynamic range of expression levels. Bioinformatics 28:1086–1092. https://doi.org/10.1093/bioinformatics/bts094
15. Grabherr MG, Haas BJ, Yassour M, et al (2011) Full-length transcriptome assembly from RNA-

Seq data without a reference genome. Nat Biotechnol 29:644–652. https://doi.org/10.1038/nbt.1883
16. Fu L, Niu B, Zhu Z, et al (2012) CD-HIT: Accelerated for clustering the next-generation sequencing data. Bioinformatics 28:3150–3152. https://doi.org/10.1093/bioinformatics/bts565
17. Camacho C, Coulouris G, Avagyan V, et al (2009) BLAST+: Architecture and applications. BMC Bioinformatics 10:421. https://doi.org/10.1186/1471-2105-10-421
18. Gilbert DG (2019) Genes of the pig, Sus scrofa, reconstructed with EvidentialGene. PeerJ 2019:. https://doi.org/10.7717/peerj.6374
19. Dobin A, Davis CA, Schlesinger F, et al (2013) STAR: Ultrafast universal RNA-seq aligner. Bioinformatics 29:15–21. https://doi.org/10.1093/bioinformatics/bts635
20. Li H, Handsaker B, Wysoker A, et al (2009) The Sequence Alignment/Map format and SAMtools. Bioinformatics 25:2078–2079. https://doi.org/10.1093/bioinformatics/btp352
21. Smith-Unna R, Boursnell C, Patro R, et al (2016) TransRate: Reference-free quality assessment of de novo transcriptome assemblies. Genome Res 26:1134–1144. https://doi.org/10.1101/gr.196469.115
22. Jones P, Binns D, Chang HY, et al (2014) InterProScan 5: Genome-scale protein function classification. Bioinformatics 30:1236–1240. https://doi.org/10.1093/bioinformatics/btu031
23. Schäffer AA, Nawrocki EP, Choi Y, et al (2018) VecScreen-plus-taxonomy: Imposing a tax(onomy) increase on vector contamination screening. Bioinformatics 34:755–759. https://doi.org/10.1093/bioinformatics/btx669
24. Buchfink B, Xie C, Huson DH (2014) Fast and sensitive protein alignment using DIAMOND. Nat. Methods 12:59–60
25. Simão FA, Waterhouse RM, Ioannidis P, et al (2015) BUSCO: Assessing genome assembly and annotation completeness with single-copy orthologs. Bioinformatics 31:3210–3212. https://doi.org/10.1093/bioinformatics/btv351
26. Waterhouse RM, Seppey M, Simao FA, et al (2018) BUSCO applications from quality assessments to gene prediction and phylogenomics. Mol Biol Evol 35:543–548. https://doi.org/10.1093/molbev/msx319
27. Kim D, Song L, Breitwieser FP, Salzberg SL (2016) Centrifuge: Rapid and sensitive classification of metagenomic sequences. Genome Res 26:1721–1729. https://doi.org/10.1101/gr.210641.116
28. Breitwieser FP, Salzberg SL (2020) Pavian: Interactive analysis of metagenomics data for microbiome studies and pathogen identification. Bioinformatics 36:1303–1304. https://doi.org/10.1093/bioinformatics/btz715
29. Nakamura T, Yamada KD, Tomii K, Katoh K (2018) Parallelization of MAFFT for large-scale multiple sequence alignments. Bioinformatics 34:2490–2492. https://doi.org/10.1093/bioinformatics/bty121
30. Edgar RC (2004) MUSCLE: Multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32:1792–1797. https://doi.org/10.1093/nar/gkh340
31. Sievers F, Higgins DG (2018) Clustal Omega for making accurate alignments of many protein sequences. Protein Sci 27:135–145. https://doi.org/10.1002/pro.3290
32. Kimura M (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J Mol Evol 16:111–120. https://doi.org/10.1007/BF01731581
33. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing

phylogenetic trees. Mol Biol Evol 4:406–425. https://doi.org/10.1093/oxfordjournals.molbev.a040454
34. Larkin MA, Blackshields G, Brown NP, et al (2007) Clustal W and Clustal X version 2.0. Bioinformatics 23:2947–2948. https://doi.org/10.1093/bioinformatics/btm404
35. Brown NP, Leroy C, Sander C (1998) MView: A web-compatible database search or multiple alignment viewer. Bioinformatics 14:380–381. https://doi.org/10.1093/bioinformatics/14.4.380
36. Sansone SA, Rocca-Serra P, Field D, et al (2012) Toward interoperable bioscience data. Nat. Genet. 44:121–126
37. Del Fabbro C, Scalabrin S, Morgante M, Giorgi FM (2013) An Extensive Evaluation of Read Trimming Effects on Illumina NGS Data Analysis. PLoS One 8:e85024. https://doi.org/10.1371/journal.pone.0085024
38. Martin M (2011) Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet.journal 17:10. https://doi.org/10.14806/ej.17.1.200
39. Bolger AM, Lohse M, Usadel B (2014) Trimmomatic: A flexible trimmer for Illumina sequence data. Bioinformatics 30:2114–2120. https://doi.org/10.1093/bioinformatics/btu170
40. Cozzuto L, Liu H, Pryszcz LP, et al (2020) MasterOfPores: A Workflow for the Analysis of Oxford Nanopore Direct RNA Sequencing Datasets. Front Genet 11:211. https://doi.org/10.3389/fgene.2020.00211
41. Li B, Fillmore N, Bai Y, et al (2014) Evaluation of de novo transcriptome assemblies from RNA-Seq data. Genome Biol 15:553. https://doi.org/10.1186/s13059-014-0553-5
42. Bushmanova E, Antipov D, Lapidus A, et al (2016) RnaQUAST: A quality assessment tool for de novo transcriptome assemblies. Bioinformatics 32:2210–2212. https://doi.org/10.1093/bioinformatics/btw218
43. Eddy SR (2011) Accelerated profile HMM searches. PLoS Comput Biol 7:1002195. https://doi.org/10.1371/journal.pcbi.1002195
44. Finn RD, Bateman A, Clements J, et al (2014) Pfam: The protein families database. Nucleic Acids Res 42:222–230. https://doi.org/10.1093/nar/gkt1223
45. Ashburner, M., Ball, C., Blake J et al. (2000) Gene ontology: Tool for the unification of biology. Nat. Genet. 25:25–29
46. Kanehisa M, Goto S, Sato Y, et al (2012) KEGG for integration and interpretation of large-scale molecular data sets. Nucleic Acids Res 40:109–114. https://doi.org/10.1093/nar/gkr988
47. Thimm O, Bläsing O, Gibon Y, et al (2004) MAPMAN: A user-driven tool to display genomics data sets onto diagrams of metabolic pathways and other biological processes. Plant J 37:914–939. https://doi.org/10.1111/j.1365-313X.2004.02016.x
48. Csárdi G, Nepusz T (2006) The igraph software package for complex network research. InterJournal Complex Syst 1695:1695. https://doi.org/10.3724/SP.J.1087.2009.02191
49. Crusoe MR, Alameldin HF, Awad S, et al (2015) The khmer software package: enabling efficient nucleotide sequence analysis. F1000Research 4:900. https://doi.org/10.12688/f1000research.6924.1
50. Xie Y, Wu G, Tang J, et al (2014) SOAPdenovo-Trans: De novo transcriptome assembly with short RNA-Seq reads. Bioinformatics 30:1660–1666. https://doi.org/10.1093/bioinformatics/btu077
51. Robertson G, Schein J, Chiu R, et al (2010) De novo assembly and analysis of RNA-seq data. Nat Methods 7:909–912. https://doi.org/10.1038/nmeth.1517

52. Peng Y, Leung HCM, Yiu SM, Chin FYL (2011) T-IDBA: A de novo iterative de Bruijn graph assembler for transcriptome (Extended abstract). In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Berlin, Heidelberg, pp 337–338
53. Liu J, Yu T, Mu Z, Li G (2019) TransLiG: A de novo transcriptome assembler that uses line graph iteration. Genome Biol 20:81. https://doi.org/10.1186/s13059-019-1690-7
54. Zhang Y, Sun Y, Cole JR (2014) A Scalable and Accurate Targeted Gene Assembly Tool (SAT-Assembler) for Next-Generation Sequencing Data. PLoS Comput Biol 10:1003737. https://doi.org/10.1371/journal.pcbi.1003737
55. Geniza M, Jaiswal P (2017) Tools for building de novo transcriptome assembly. Curr Plant Biol 11–12:41–45. https://doi.org/10.1016/j.cpb.2017.12.004

Figure captions

Figure 1: **Bioinformatics pipeline for generation of potato transcriptomes.** *Tools used in specific steps are emphasised. Input datasets (sequence reads) and output data (assemblies and transcriptomes) are depicted as cylinders. Abbreviations: tr2aacds – "transcript to amino acid coding sequence" Perl script from EvidentialGene pipeline.*

Pre-processing, per genotype

short reads

short reads

short reads

Quality control
FastQC + MultiQC

Adapter removal, trimming and filtering
TrimGalore

Digital normalisation
BBNorm

SE reads

PE reads

long reads

Long reads pre-processing pipeline

long reads

long reads

***de novo* transcriptome assembly, per genotype (1, ..., n)**

***de novo* assembly of short reads**

assembler 1
multiple parameter settings

multiple transcriptome assemblies

assembler 2
multiple parameter settings

multiple transcriptome assemblies

…

assembler m
multiple parameter settings

multiple transcriptome assemblies

over-assembly

Redundancy reduction
tr2aacds

Biological evidence based annotations

Functional analysis
InterProScan

Vector contamination scanning
VecScreen

Genome based annotations
STARlong + MatchAnnot

Homology based annotation
DIAMOND

Transcriptome filtering and annotation

Transcriptome filtering
custom scripts (R)

Paralogue clusters classification
custom scripts (R)

Representative and alternative sequences annotation
custom scripts (R)

Transcriptome evaluation

Mapping reads back to transcriptome
STAR

Sequence based and reference based validation
TransRate

Completeness estimation
BUSCO

Taxonomic calssification
Centrifuge + Pavian

Multiple Sequence Alignment of paralogue clusters
CustalOmega/MAFFT

genotype 1 reference transcriptome

genotype n reference transcriptome

pan-transcriptome