Article title: De novo genome assembly and annotation of the Uzbek autochthonous grapevine cultivar Parkent (*Vitis vinifera* L.)

Journal: Plant Molecular Biology Reporter

Authors: Ilkhom B. Salakhutdinov, Dmitriy A. Mirzabaev, Venera S. Kamburova, Dilshod E. Usmonov, Farhod S. Radjapov, Shukhrat E. Shermatov, Zabardast T. Buriev, Ibrokhim Y. Abdurakhmonov.

Affiliation: Center of Genomics and Bioinformatics of the Academy of Sciences of the Republic of Uzbekistan

Corresponding author: Dmitriy A. Mirzabaev, email: dmitriy08m@gmail.com.

Online Resource 1. Bioinformatics pipeline commands

1. Read quality assessment, taxonomic classification, and de-contamination

1.1. Quality check of raw reads:

```
fastqc -t <threads> input_R1.fastq.gz input_R2.fastq.gz
```

1.2. Trimming and adapter removal:

```
fastp -i input_R1.fastq.gz -I input_R2.fastq.gz -o clean_R1.fastq.gz -O
clean_R2.fastq.gz -w <threads> --adapter_fasta /path/to/adapters.fasta -f 15 -F 15 -
W 4 -M 15 -3 -l 50 -b 134 --trim_poly_g -h fastp_report.html -j fastp_report.json
```

1.3. First round of decontamination (Kraken2):

```
kraken2 --db /path/to/core_nt_db/ --paired clean_R1.fastq.gz clean_R2.fastq.gz --
threads <threads> --output kraken_classification.txt --report kraken_report.txt --
unclassified-out unclassified_reads#.fq
```

1.4. Extraction of target organism reads (TaxID 3603 for Vitis):

```
python3 extract_kraken_reads.py -k kraken_classification.txt -r kraken_report.txt -t
3603 -s1 clean_R1.fastq.gz -s2 clean_R2.fastq.gz -o filtered_R1.fastq -o2
filtered_R2.fastq --include-children --fastq-output
```

2. K-mer analysis and heterozygosity estimation

2.1. K-mer counting and analysis:

```
for k in 21 33 63; do jellyfish count -m $k -s 1G -t <threads> -C -o
kmer_counts_k${k}.jf filtered_R1.fastq filtered_R2.fastq; jellyfish histo -t
<threads> kmer_counts_k${k}.jf > kmer_counts_k${k}.hist; done
```

3. De novo genome assembly, reference-guided scaffolding, and quality assessment

3.1. De novo assembly:

3.1.1. Primary contig assembly using SPAdes:

```
spades.py -1 filtered_R1.fastq -2 filtered_R2.fastq -o spades_output -k 21,33,63 -t
<threads> -m <RAM_GB>
```

3.1.2. De novo assembly quality assessment (QUAST):

```
quast.py -o quast_denovo_spades_K21-63_S1 --min-contig 200
spades_output/contigs.fasta
```

3.2. A single round of polishing using Pilon:

3.2.1. Indexing the assembly

```
bwa-mem2 index /path/to/contigs_spades_K21_63.fasta
```

3.2.2. Aligning reads to the assembly

```
bwa-mem2 mem -t <threads> /path/to/contigs_spades_K21_63.fasta
/path/to/filtered_R1.fastq.gz /path/to/filtered_R2.fastq.gz 2> bwa.log | samtools
sort -o sorted_reads_contigs.bam - 2> samtools.log
```

3.2.3. Indexing the sorted BAM file

```
samtools index sorted_reads_contigs.bam
```

3.2.4. Polishing the entire assembly

```
java -Xmx<RAM_GB>G -jar /path/to/pilon-1.24.jar --genome
/path/to/contigs_spades_K21_63.fasta --frags sorted_reads_contigs.bam --output
pilon_polished_contigs --diploid
```

3.3. FCS-adaptor screening and cleaning:

```
./run_fcsadaptor.sh --fasta-input blob_cleaned_assembly.fasta --output-dir
./fcs_output –euk
```

```
cat blob_cleaned_assembly.fasta | python3 fcs.py clean genome --action-report
./fcs_output/fcs_adaptor_report.txt --output fcs_cleaned_assembly.fasta
```

3.4. Identification of organellar and rDNA contigs:

3.4.1. Create organelle database:

```
for organelle in rDNA ChloroDNA mitoDNA; do makeblastdb -in
/path/to/sequence_vitis_${organelle}.fna -dbtype nucl -out db_vitis_${organelle};
blastn -query fcs_cleaned_assembly.fasta -db db_vitis_${organelle} -outfmt 6 -out
blast_results_vitis_${organelle}.tsv; done
```

3.4.2 Calculate contig lengths for filtering:

```
awk '/^>/ {if (seqlen){print seqlen}; printf "%s\t",substr($0,2);seqlen=0;next; } {
seqlen = seqlen +length($0)}END{print seqlen}' fcs_cleaned_assembly.fasta >
contig_lengths.txt
```

3.4.3. Applying custom filtering script based on specific coverage thresholds: 60% (rDNA), 50% (mitoDNA), and 40% (ChloroDNA):

```
python custom_filter_script.py blast_results_vitis_rDNA.tsv contig_lengths.txt 60
contigs_to_remove_rDNA.txt
python custom_filter_script.py blast_results_vitis_mitoDNA.tsv contig_lengths.txt 50
contigs_to_remove_mitoDNA.txt
python custom_filter_script.py blast_results_vitis_ChloroDNA.tsv contig_lengths.txt
40 contigs_to_remove_ChloroDNA.txt
```

3.4.4. Merge lists of contigs to remove:

```
cat contigs_to_remove_*.txt | sort -u > final_contigs_to_remove.txt
```

3.4.5. Generate the organelle-filtered assembly:

```
grep "^>" fcs_cleaned_assembly.fasta | sed 's/>//' > all_contigs.txt
grep -v -F -f final_contigs_to_remove.txt all_contigs.txt > contigs_to_keep.txt
seqtk subseq fcs_cleaned_assembly.fasta contigs_to_keep.txt >
organelle_filtered_assembly.fasta
```

3.5. Redundans (haplotig removal):

```
redundans.py -f organelle_filtered_assembly.fasta -o redundans_output -t <threads> -
-noscaffolding --nogapclosing --minimap2reduce
```

3.6. Reference-guided scaffolding:

```
ragtag.py scaffold -o ragtag_output -u -t <threads> /path/to/reference_genome.fasta
redundans_output/contigs.reduced.fa
```

3.7. Three iterative rounds of polishing with Pilon:

```
ASSEMBLY="ragtag_output/ragtag.scaffold.fasta"; for i in {1..3}; do bwa-mem2 index
${ASSEMBLY}; bwa-mem2 mem -t <threads> ${ASSEMBLY} filtered_R1.fastq
filtered_R2.fastq | samtools sort -o aligned_round${i}.bam -; samtools index
aligned_round${i}.bam; java -Xmx<RAM_GB>G -jar pilon.jar --genome ${ASSEMBLY} --
frags aligned_round${i}.bam --output polished_round${i} --diploid;
ASSEMBLY="polished_round${i}.fasta"; done
```

3.8. Removing sequences shorter than 200 bp from the final polished assembly:

```
seqtk seq -L 200 polished_round3.fasta > final_genome_assembly.fasta
```

3.9. Quality and completeness assessment of assemblies:

3.9.1. Quality assessment using reference genome (QUAST):

```
quast.py -o quast_results -r reference.fasta -g annotation.gff -s -e --circos -f --
min-contig 200 -t <threads> final_genome_assembly.fasta
```

3.9.2. Quality assessment using reference genome (BUSCO):

```
busco -i final_genome_assembly.fasta -m genome -l
/path/to/busco_db/lineages/eudicots_odb10 -f
```

4. Identification of repetitive elements and genome annotation

4.1. De novo repeats library construction and masking:

4.1.1. Genome database construction for repeat modeling:

```
BuildDatabase -name genome_db final_genome_assembly.fasta
```

4.1.2. De novo identification of repetitive elements and LTR structural search:

```
RepeatModeler -database genome_db -threads <threads> -LTRStruct
```

4.1.3. Repeat annotation and soft-masking of the genome:

```
RepeatMasker -lib genome_db-families.fa -pa <threads> -gff -xsmall
final_genome_assembly.fasta
```

4.2. Training species-specific AUGUSTUS model using BUSCO:

4.2.1. BUSCO configuration (config.ini):

```
[busco_run]
# Input file
;in = final_genome_assembly.fasta.masked
# Output directory name
```

```
;out = VitisVinifera_AUG_BUSCO
# Where to store the output directory
out_path = /path/to/busco_working_dir
# Path to the BUSCO dataset
lineage_dataset = /path/to/eudicots_odb10
# Which mode to run (genome / proteins / transcriptome)
mode = genome
# Run lineage auto selector
auto-lineage = False
# Run auto selector only for eukaryote datasets
auto-lineage-euk = False
# How many threads to use for multithreaded steps
;cpu = <threads>
# Force rewrite if files already exist (True/False)
force = False
# Restart a previous BUSCO run (True/False)
restart = False
# Blast e-value
evalue = 1e-3
# How many candidate regions (contigs, scaffolds) to consider for each BUSCO
limit = 3
# Metaeuk parameters for initial run
;metaeuk_parameters=''
# Metaeuk parameters for rerun
;metaeuk_rerun_parameters=""
# Augustus parameters
;augustus_parameters=''
# Quiet mode (True/False)
quiet = False
# Local destination path for downloaded lineage datasets
download_path = /path/to/busco_db/
# Run offline
offline=True
# Ortho DB Datasets version
datasets_version = odb10
# URL to BUSCO datasets
download_base_url = https://busco-data.ezlab.org/v4/data/
# Download most recent BUSCO data and files
update-data = True
# Use Augustus gene predictor instead of metaeuk
use_augustus = True
```

4.2.2. BUSCO training execution script (run_busco_train.sh):

```
#!/bin/bash

# Activate environment

source activate busco_env
```

```
CPU_COUNT=<threads>
INPUT_FASTA="final_genome_assembly.fasta.masked"
OUTPUT_NAME="VitisVinifera_AUG_BUSCO"

echo "Starting BUSCO" $(date)

# run BUSCO
busco \
--config config.ini \
--in ${INPUT_FASTA} \
--out ${OUTPUT_NAME} \
-l eudicots_odb10 \
-m genome \
-f \
--long \
--cpu ${CPU_COUNT} \
--augustus_parameters='--progress=true' \
--offline

echo "BUSCO Mission complete!" $(date)
```

Execution command:

```
nohup ./run_busco_train.sh > busco_training.log 2>&1 &
```

4.2.3. Migration of trained parameters to AUGUSTUS config directory:

```
#!/bin/bash

# 1. Set the path to the newly generated parameters
SOURCE_PARAMS="/path/to/busco_working_dir/VitisVinifera_AUG_BUSCO/run_eudicots_odb10
/augustus_output/retraining_parameters/BUSCO_VitisVinifera_AUG_BUSCO"

# 2. Determine where Augustus stores species configurations
if [ ! -z "$AUGUSTUS_CONFIG_PATH" ]; then
    DEST_DIR="$AUGUSTUS_CONFIG_PATH/species"
elif [ ! -z "$CONDA_PREFIX" ]; then
    DEST_DIR="$CONDA_PREFIX/config/species"
else
    echo "Failed to automatically find the Augustus config path."
fi

# 3. Copy and verify the result
if [ -d "$DEST_DIR" ]; then
    echo "Copying parameters to: $DEST_DIR"
    cp -r "$SOURCE_PARAMS" "$DEST_DIR/"

    # Verification
    if [ -d "$DEST_DIR/BUSCO_VitisVinifera_AUG_BUSCO" ]; then
        echo "SUCCESS! Parameters installed."
    else
```

```
        echo "ERROR during copying."
    fi
else
    echo "ERROR: Target folder $DEST_DIR does not exist. Check Augustus
installation."
fi
```

4.3. Structural annotation (Funannotate):

4.3.1. Assembly cleaning and size filtering:

```
funannotate clean -i final_genome_assembly.fasta.masked -o
final_genome_masked_cleaned.fasta -m 10000
```

4.3.2. Evidence-driven structural gene prediction:

```
funannotate predict -i final_genome_masked_cleaned.fasta -o fun_out --species "Vitis
vinifera" --isolate <CultivarName> --name <LocusTag> --cpus <threads> --
transcript_evidence /path/to/vitis_vinifera_ESTs.fasta --augustus_species
BUSCO_VitisVinifera_AUG_BUSCO --busco_db embryophyta --busco_seed_species cacao --
organism other --max_intronlen 10000
```

4.4. Functional annotation tools:

4.4.1. Protein domain, motif, and GO term annotation (InterProScan):

```
interproscan.sh -i fun_out/predict_results/Vitis_vinifera_Parkent.proteins.fa -f xml
-o iprscan.xml -goterms -pa -dp --cpu <threads>
```

4.4.2. Orthology assignment and functional classification (eggNOG-mapper):

```
emapper.py -i fun_out/predict_results/Vitis_vinifera_Parkent.proteins.fa --output
eggnog_out --output_dir ./ --data_dir /path/to/eggnog_db/ -m diamond --cpu <threads>
--override
```

4.5. Final functional annotation merge:

```
funannotate annotate -i fun_out --eggnog eggnog_out.emapper.annotations --iprscan
iprscan.xml --species "Vitis vinifera" --isolate "Parkent" --busco_db embryophyta -d
/path/to/funannotate_db --cpus <threads>
```