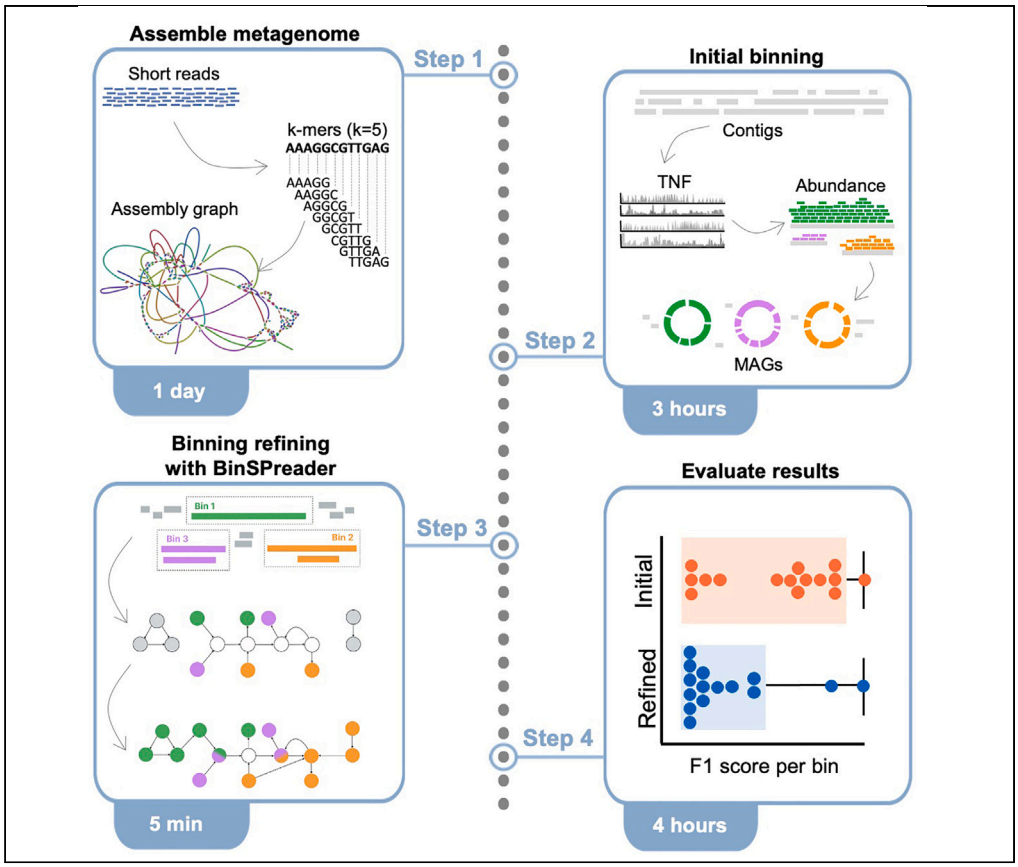
STAR Protocols

CellPress
OPEN ACCESS

Protocol

# Protocol for refining metagenomic binning with BinSPreader




The analysis of metagenomic data obtained via high-throughput DNA sequencing is primarily carried out by a dedicated binning process involving clustering contigs, presumably belonging to the same species. Here, we present a protocol for improving the quality of binning using BinSPreader. We describe steps for typical metagenome assembly and binning workflow. We then detail binning refining, its variants, output, and possible caveats. This protocol optimizes the process of reconstructing more complete genomes of microorganisms that make up the metagenome.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Sofia Ochkalova, Ivan Tolstoganov, Alla Lapidus, Anton Korobeynikov

so.ochkalova@gmail.com (S.O.)
a.korobeynikov@spbu.ru (A.K.)

## Highlights

BinSPreader refines the binning using the assembly graph connectivity

Binning refining improves the completeness of the bins without sacrificing the purity

Binning refining enriches bins with contigs containing important conservative genes

BinSPreader predicts contigs belonging to several MAGs

Ochkalova et al., STAR Protocols 4, 102417 September 15, 2023 
https://doi.org/10.1016/j.xpro.2023.102417

STAR Protocols

CellPress
OPEN ACCESS

Protocol

# Protocol for refining metagenomic binning with BinSPreader

Sofia Ochkalova,[1,4,*] Ivan Tolstoganov,[1] Alla Lapidus,[1,2] and Anton Korobeynikov[1,3,5,*]

[1]Center for Algorithmic Biotechnology, Saint Petersburg State University, 199004 Saint Petersburg, Russia

[2]Department of Cytology and Histology, St. Petersburg State University, 199178 Saint Petersburg, Russia

[3]Department of Statistical Modelling, Saint Petersburg State University, 198504 Saint Petersburg, Russia

[4]Technical contact

[5]Lead contact

*Correspondence: so.ochkalova@gmail.com (S.O.), a.korobeynikov@spbu.ru (A.K.)
https://doi.org/10.1016/j.xpro.2023.102417

## SUMMARY

**The analysis of metagenomic data obtained via high-throughput DNA sequencing is primarily carried out by a dedicated binning process involving clustering contigs, presumably belonging to the same species. Here, we present a protocol for improving the quality of binning using BinSPreader. We describe steps for typical metagenome assembly and binning workflow. We then detail binning refining, its variants, output, and possible caveats. This protocol optimizes the process of reconstructing more complete genomes of microorganisms that make up the metagenome.**

**For complete details on the use and execution of this protocol, please refer to Tolstoganov et al.[1]**

## BEFORE YOU BEGIN

This protocol shows binning refining via BinSPreader, an assembly graph utilizing refiner. Using data from the mock metagenomic dataset MBARC26[2] we describe the full workflow from raw sequencing data to refined binning. Reported timings of the protocol steps were recorded for the MBARC26 dataset and machine Intel(R) Xeon(R) CPU E7-4880 v2 @ 2.50 GHz with 10 cores. The analysis time may vary depending on the computer environment used.

All files and scripts needed to reproduce the protocol, as well as the files that are generated during its operation, are uploaded to Figshare (link in "key resources table"). Below we provide a list of required equipment, resources and preparatory activities.

### Hardware

Although BinSPreader itself can be run on a standard desktop computer, other steps of the protocol (read mapping and genome assembly) require a high-performance computer with a multi-core CPU.

The need for a high-performance computing system depends on the data set being analyzed.

If such resources are not accessible, the ready-made files can be downloaded from Figshare (see key resources table). Besides that, a 64-bit Linux or macOS operating system is needed to execute all necessary tools.

Summarizing the above, the following is required to run the complete protocol end-to-end.

1. Computer.
   a. Typically 128 GB of memory or more (for the metagenome assembly step of the protocol; memory requirements of BinSPreader itself are very low compared to the assembly step).
   b. A multi-core CPU with at least 8 computational cores (for metagenome assembly and read mapping).
   c. At least 320 Gb of the disk space to store FASTQ reads.
2. Operating system.
   a. Linux or macOS.

### Software

Install the tools listed in the "key resources table" using conda and the YAML environment file provided in this protocol (described below).

### Raw data

As an example, this protocol uses shotgun sequencing data from the mock microbial community MBARC26[2] accessible from NCBI SRA: SRX1836716. This artificial metagenome is composed of 23 bacterial and 3 archaeal strains isolated from heterogeneous soil and aquatic environments as well as derived from human, bovine and frog. For 25 species reference genomes are known spanning a wide range of genome sizes (1.8–6.5 Mbp), GC-contents (28.4%–72.7%), repeat contents (0%–18.3%) and encompass a diverse abundance profiles. Moreover, the data set in the analysis contains three closely-related enterobacteria, namely *Escherichia coli*, *Salmonella bongori* and *Salmonella enterica*. Therefore, despite being a synthetic dataset, MBARC26 is complex enough to demonstrate the refining procedure.

## Prepare a conda environment and install required packages

Timing: 20 min

The easiest way to install all required tools is by using Conda, an open-source environment management system.[3] It should be installed if you do not already have it. Detailed installation instructions can be found in the documentation, below we briefly describe the installation of Miniconda which is a minimal distribution of Conda.

3. Download the latest Miniconda installer suitable for your machine from https://repo.anaconda.com/miniconda/. For example:

```
> wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh.
```

4. Run installation:

```
> bash Miniconda3-latest-Linux-x86_64.sh
```

*Note:* During the installation process directory for Conda have to be specified. The installer also asks for the approval to add Conda to the PATH and start it automatically when the system is activated. We assume you will agree to this.

5. Refresh the shell settings for the changes to take effect:

```
> source ~/.bashrc
```

*Note:* If conda is installed successfully and active, the command line prompt will show ''(base)'' e.g., (base) username@hostname:$$. If ''(base)'' does not appear even after executing source ~/.bashrc, then restart your command line prompt.

6. Modify conda config adding channels from which required packages will be installed:

```
> conda config --add channels defaults
> conda config --add channels conda-forge
> conda config --add channels bioconda
> conda config --add channels agbiome
```

7. Create a working directory and enter it:

```
> mkdir refining && cd refining
```

8. Download folder scripts, files adapters.fa and environment.yml from Figshare (see key resources table) and place them in the working directory. To unarchive the downloaded folder run:

```
> tar -xzf scripts.tar.gz
```

*Alternatives:* If the folder has been automatically ungzipped (for example, if it was downloaded using Safari on MacOS), run the following:

```
> tar -xf scripts.tar
```

9. Create a conda environment specified in the yml file. This environment will contain all required tools and packages to execute this protocol.

```
> conda env create -f environment.yml
```

10. Activate environment:

```
> conda activate bsp_protocol
```

*Note:* ''(bsp_protocol)'' will appear at the beginning of your command line prompt. When the terminal is restarted, the environment has to be reactivated. If it is needed to deactivate the environment, use the command:

```
> conda deactivate
```

11. One of the tools installed, namely CheckM,[4] relies on a number of precalculated data files which needs to be downloaded to your computer.

a. Run this command to create folder called checkm_data in the current directory and download the archive into it (you can modify the command to download it wherever you want):

```
> wget -P checkm_data https://zenodo.org/record/7401545/files/checkm_data_2015_01_16.tar.gz.
```

b. To unpack the archived folders run:

```
> tar -xzf checkm_data/checkm_data_2015_01_16.tar.gz -C checkm_data
```

c. Specify to CheckM path where the reference data is stored:

```
> checkm data setRoot checkm_data
```

### Download the demonstration public dataset

Timing: 3–4 h

12. Download raw reads from NCBI SRA database using prefetch from SRA-toolkit. Raw reads will take 32 Gb of your disk space. Option "–max-size" has to be specified because by default files larger than 20 Gb are skipped.

```
> prefetch --max-size 50GB --output-directory . SRR3656745
```

*Note:* Download time may vary depending on your internet connection. After downloading is finished, a new folder called SRR3656745 will appear in the current directory.

13. Unpack NCBI-formatted file to regular paired-end reads in FASTQ format.

```
> fastq-dump --split-files SRR3656745
```

*Note:* When the extraction process is completed, files SRR3656745_1.fastq and SRR3656745_2.fastq, 63 Gb each, will appear in the current directory. The source SRA SRR3656745 will no longer be used and can be deleted to save the disk space:

```
> rm -r SRR3656745/
```

## KEY RESOURCES TABLE

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---|---|---|
| Deposited data | | |
| MBARC26 dataset | Singer et al.[2] | NCBI SRA, accession number SRX1836716 |
| Custom scripts, assembly graph, scaffolds, abundance profiles, binning results generated in this protocol | This study | https://figshare.com/articles/dataset/BinSPreader_binning_refining_protocol/21970808 |
| Software and algorithms | | |
| Anaconda | Anaconda Software Distribution[3] | https://docs.anaconda.com/ |
| BinSPreader v.3.16-dev | Tolstoganov et al.[1] | https://cab.spbu.ru/software/binspreader/ |
| CheckM v.1.2.2 | Parks et al.[4] | https://github.com/Ecogenomics/CheckM |

(*Continued on next page*)

*Continued*

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---|---|---|
| MetaBAT2 v.2.15 | Kang et al.[5] | https://bitbucket.org/berkeleylab/metabat/src/master/ |
| metaQUAST v.5.2.0 | Mikheenko et al.[6] | https://cab.spbu.ru/software/metaquast/ |
| SPAdes v.3.15.4 | Nurk et al.[7] | https://cab.spbu.ru/software/spades/ |
| SRA Toolkit v.3.0.3 | SRA Toolkit Development Team[8] | https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software |
| bbtools v.37.62 | Bushnell[9] | https://sourceforge.net/projects/bbmap/ |
| minimap2 v.2.24 | Li[10] | https://github.com/lh3/minimap2 |
| SAMtools v.1.6 | Danecek et al.[11] | https://github.com/samtools/samtools/ releases/download/1.15/samtools-1.15.tar.bz2 |
| Python v.3.9.16 | Rossum & Drake[12] | https://www.python.org |
| Biopython package v.1.81 | Cock et al.[13] | https://biopython.org |
| pandas package v.1.5.3 | Reback et al.[14] | https://pandas.pydata.org |
| matplotlib package v.3.7.1 | Hunter[15] | https://matplotlib.org |
| seaborn package v.0.12.2 | Waskom et al.[16] | https://seaborn.pydata.org |
| scipy package v.1.10.1 | Virtanen et al.[17] | https://scipy.org/install/ |
| Other | | |
| Hardware: Intel(R) Xeon(R) CPU E7-4880 v2 @ 2.50 GHz, 1.5T RAM, Debian 4.16.12 | N/A | N/A |

## STEP-BY-STEP METHOD DETAILS

### Assemble short reads into metagenomic contigs

Timing: 22 h

This protocol assumes that the metagenome sequencing data is composed of short Illumina paired-end reads generated from whole metagenome shotgun sequencing and stored in FASTQ files (however, BinSPreader itself could be used on the assembly graphs generated from e.g., 3rd generation sequencing data). Below we briefly outline the process of assembling metagenomes, which has been extensively described earlier (for example, refer to Thomas et al.,[18] Prjibelski et al.[19]).

1. Trim adapter sequences and poor-quality bases using bbduk:

```
> bbduk.sh in1=SRR3656745_1.fastq in2=SRR3656745_2.fastq out1=SRR3656745_1.trimm.fastq out2=SRR3656745_2.trimm.fastq ref=adapters.fa ktrim=r k=23 mink=11 hdist=1 tpe tbo qtrim=rl trimq=17
```

*Note:* Here, "ref" option is used to provide a reference fasta file with adapter sequences to remove. "ktrim = r" specifies that sequence matching to ref will be trimmed together with all the bases to the right side of a read, leaving only the bases to the left. "k", "mink" and "hdist" set k-mer size, minimal size of a kmer to look for, and hamming distance (allowed number of mismatches), respectively. "tbo" means running in a mode when adapters are trimmed not only based on a kmer match, but also on pair overlap detection using BBMerge. "tpe" specifies trimming of both reads to the same length. "trimq = 17" means quality-trimming to Q17 using the Phred algorithm, "qtrim = rl" makes bbduk to trim the both sides of a read.

⚠ CRITICAL: Depending on the data source additional preprocessing steps might be needed (for example, exclusion of host-derived reads). Adapt the protocol according to the context of your research.

2. Run metaSPAdes assembler with read error correction:

```
> metaspades.py -1 SRR3656745_1.trimm.fastq -2 SRR3656745_2.trimm.fastq -o assembly -t 10
```

*Note:* The assembly may take quite a long time. After it is finished, the assembly graph, scaffolds, contigs and intermediate files will be saved to the assembly directory. You may need to adapt the command line (e.g., number of used CPU threads specified via "-t" option) depending on the available computation resources. Refer to Prjibelski et al.[19] and SPAdes manual (https://cab.spbu.ru/files/release3.15.5/manual.html#sec3.1) for more information.

*Alternatives:* Preassembled metagenome is available from the assembly folder of Figshare dataset.

3. Run QUAST[6] to assess contiguity of generated assembly. Option "–fast" prompts QUAST to output only main statistics of the assembly in a plain text form.

```
> quast.py --fast -o quast assembly/scaffolds.fasta
```

*Note:* The results of downstream metagenome analysis depend directly on the assembly quality, and thus their evaluation allows one to make assumptions about further binning outcomes. By using this command one can obtain standard assembly statistics such as total length, N50 and contigs counts by length.

QUAST report can be viewed by using the following command.

```
> less quast/report.txt
```

Table 1 shows the example report generated by this step.

### Perform initial binning by using MetaBAT2

Timing: 3 h

This protocol uses MetaBAT2 to obtain a preliminary binning, although any other binner or binning pipeline (such as MetaWRAP[20]) of your choice can be used. Apart from characteristics computed from contig sequences, MetaBAT2 can use contig coverages to distinguish original organisms by their variable abundance in the source microbial community. Contig coverage improves binning accuracy, so despite the fact that MetaBAT2 can bin without input abundances, we strongly recommend to map reads and create abundance profiles in advance.

*Alternatives:* Considering that the read mapping step is rather time-consuming, pre-made abundance profiles are available from FigShare (see key resources table).

4. Map trimmed reads to the assembly:

```
> minimap2 -t 10 -a -x sr assembly/scaffolds.fasta SRR3656745_1.trimm.fastq SRR3656745_2.
trimm.fastq | samtools view -F 3584 -b -o - | samtools sort -@ 10 -o mbarc.bam
```

**Table 1. Example of QUAST report for MBARC26 metagenome assembled with metaSPAdes**

| Assembly | Scaffolds |
|---|---|
| # contigs (>= 0 bp) | 20631 |
| # contigs (>= 1000 bp) | 2868 |
| # contigs (>= 5000 bp) | 1155 |
| # contigs (>= 10000 bp) | 957 |
| # contigs (>= 25000 bp) | 644 |
| # contigs (>= 50000 bp) | 415 |
| Total length (>= 0 bp) | 103568544 |
| Total length (>= 1000 bp) | 98021022 |
| Total length (>= 5000 bp) | 94826206 |
| Total length (>= 10000 bp) | 93385583 |
| Total length (>= 25000 bp) | 88271863 |
| Total length (>= 50000 bp) | 79915017 |
| # contigs | 4988 |
| Largest contig | 2331453 |
| Total length | 99485665 |
| N50 | 199092 |
| N90 | 21347 |
| L50 | 102 |
| L90 | 699 |
| #N's per 100 kbp | 12.76 |

*Note:* This one-liner performs mapping, filtering and sorting of the alignment in the BAM format without producing intermediate files. The "-x" argument tells minimap2 which preset to use, here it is one for mapping of short reads to genome assembly ("sr"); option "-a" sets output file format to SAM. "-F 3584" filters out alignments that are unmapped, are secondary or supplemental alignments, or have a low mapping quality. The number 3584 is the bitwise flag for these alignment states. The resulting bam file is sorted and saved to mbarc.bam. "-@" specifies the number of threads to be used during the sorting step.

5. Compute abundance profiles of contigs using the script that comes with MetaBAT2:

```
> jgi_summarize_bam_contig_depths --outputDepth abundances.tsv mbarc.bam
```

6. Bin contigs using MetaBAT2:
   a. Create output directory for MetaBAT2:

```
> mkdir -p metabat2/bins
```

   b. Run the binner:

```
> metabat2 -i assembly/scaffolds.fasta -o metabat2/bins/bin -a abundances.tsv --numThreads 10
--seed 42
```

*Note:* Calculated abundance profiles are specified using "-a" option. Option "–seed" is used to fix random value seed to guarantee the reproducibility of the binning result. MetaBAT2 produced 35 bins in this example case; they will appear in metabat2/bins/ directory as individual FASTA files.

## Refine bins with BinSPreader

Timing: 5 min

This section of the protocol provides a detailed description of input files requirements of BinSPreader, and introduces different run modes and options available. Finally, there are example commands and thorough instructions on how to run BinSPreader depending on your data conditions.

7. Prepare the input data for BinSPreader:
   a. Assembly graph.

   *Note:* Unlike contig sequences alone, an assembly graph accumulates complete information about the assembly. It is constructed of edges that are the maximal nonbranching DNA sequences (unitigs), resulting from merger of k-mers from unambiguous paths, and links between edges, inferred from read overlaps. After the assembler resolves repeat paths across the assembly graph, it outputs non-branching paths as contigs. This results in the loss of a wealth of information about the connectivity of DNA sequences that can be beneficial for downstream analysis. Therefore the use of assembly graph for downstream analysis (instead of contig / scaffold sequences) might provide better results as important link information is preserved and can be utilized by graph-aware tools.

   Most often assembly graphs are represented in GFA file format (http://gfa-spec.github.io/GFA-spec/GFA1.html). It is a tab-separated format with each line corresponding to a single record. Most usually there are three types of them: segment, link or path, and record type is set in the first column with capital "S", "L" or "P", respectively. Segment record describes nucleotide sequence and its unique name. Link record characterizes overlap between two sequences represented by their names, orientation of these sequences, and overlap length. Path record describes a scaffold as an ordered list of oriented segments, supported by link records.

   b. Assembly.

   *Note:* metaSPAdes outputs assembled contigs as contigs.fasta. File scaffolds.fasta contains assembled scaffolds and metaSPAdes-produced assembly graph encodes scaffold paths in path records. Therefore we recommend using scaffolds for metagenome binning and further downstream analysis. If there is a specific demand to use contigs.fasta, their paths though the assembly graph should be additionally specified. metaSPAdes provides contigs paths in the separate contigs.paths file that could be provided to BinSPreader with the "-paths" option. Should the "-paths" option be specified, BinSPreader will process contig paths described in the path file instead of scaffold paths embedded in the assembly graph. In any case, the binned sequences must correspond to the paths in the assembly graph.

   c. Initial binning.
      BinSPreader takes input binning in a tab-separated format with the first column containing sequence (scaffold) name and second being a unique bin identifier. To generate the appropriate file based on a directory with FASTA files of individual bins one can use convert_fasta_bins_to_tsv_format.py script from Figshare. Before running it, ensure that bin files are named akin to "bin_*.fasta" format.
      i. Rename bins named "bin.*.fa" created by MetaBAT2 using the command below:

```
> for file in metabat2/bins/bin.*.fa; do mv "$file" "${file/bin./bin_}"sta; done
```

      ii. After file name format is normalized, run the script:

```
> python scripts/convert_fasta_bins_to_tsv_format.py --o metabat2/binning.tsv metabat2/bins/*
```

   d. Additional linkage information **(optional)**.

*Note:* BinSPreader can map paired-end reads to obtain additional linkage evidence from original sequencing reads or HiC data. It can be useful to compensate for the fragmentation of the graph and obtain connections between detached edges. However, it can produce spurious connections between similar sequences originating from distinct genomes and cause their co-clustering. Also in this case refining will take more time because read mapping might be time consuming. To use a paired-end library one should provide its description in YAML format via the "–dataset" option. For example, corresponding file for MBARC26 will be:

```
[
  {
  orientation: "fr",
  type: "paired-end",
  right reads: ["SRR3656745_1.trimm.fastq"],
  left reads: ["SRR3656745_2.trimm.fastq"]
  }
]
```

For detailed information on specifying input data with YAML dataset file, please, refer to SPAdes manual (https://cab.spbu.ru/files/release3.15.5/manual.html#sec3.1).

8. Select run mode:
   a. Refining types: Correction vs. Propagation

*Note:* BinSPreader can perform two types of refining: propagate input binning to unbinned edges of the assembly graph without any revision of initially binned edges, or try to correct initial binning using the graph contiguity information. Propagation or correction modes can be chosen via "-Rprop" or "-Rcorr" options, respectively. Propagation mode is much more conservative and essentially will not influence any scaffold assignments made by a binner, only unbinned sequences will be assigned to their (in terms of assembly graph) neighboring bins. Correction mode on one hand could correct some incorrect sequence assignment (especially if such contigs are relatively short), but on the other hand might possibly increase possible contamination especially if the initial binning was not quite pure. We recommend using propagation mode on relatively pure binning when the goal is to complete the binning via assigning of short unbinned scaffolds to relevant bins, and correction mode otherwise.

   b. Running mode types: Default vs. Sparse

*Note:* In addition to the default launch modes, BinSPreader can work in a special mode designed to refine sparse binning, which assumes that the total length of the binned contigs is much lower than the total length of the assembly. In this mode binning propagation from initially binned edges is limited by constant graph distance threshold. To enable sparse mode use "–sparse-propagation" option. Usually sparse mode is required when the initial binning undergoes additional filtering / processing (e.g., by tools such as MetaWRAP[20] or DAS_Tool[21]) or there are lots of unbinned sequences.

   c. Binning types: Single vs. Multiple

*Note:* BinSPreader supports multiple assignment of bins, which means that a contig may be included into several independent bins after refining. This might be helpful to append highly conservative sequences encoding house-keeping genes, such as rRNA, to recovered MAGs. Use "-m" option to toggle this mode. Unlike other modes, which output resulting binning in a

tab-separated file binning.tsv containing two columns (“tall” format), in multiple assignment mode by default the output file has several columns (“wide” format). A dedicated “–tall-multi” flag could be used to control this behavior and switch output from wide to tall format.

9. Run BinSPreader:
Below there are example commands to run BinSPreader with different options. It is also recommended to use tee command to save log information.
a. Correction mode (other options as default):

```
> bin-refine assembly/assembly_graph_with_scaffolds.gfa metabat2/binning.tsv binspreader-Rcorr -t 10 -Rcorr | tee binspreader-Rcorr.log
```

b. Propagation mode (other options as default):

```
> bin-refine assembly/assembly_graph_with_scaffolds.gfa metabat2/binning.tsv binspreader-Rprop -t 10 -Rprop | tee binspreader-Rprop.log
```

c. Correction mode with multiple assignment:

```
> bin-refine assembly/assembly_graph_with_scaffolds.gfa metabat2/binning.tsv binspreader-Rcorr-Pmax -t 10 -Rcorr --tall-multi -m | tee binspreader-Rcorr-Pmax.log.
```

d. Correction mode with sparse propagation:

```
> bin-refine assembly/assembly_graph_with_scaffolds.gfa metabat2/binning.tsv binspreader-Rcorr-sparse -t 10 -Rcorr --sparse-propagation | tee binspreader-Rcorr-sparse.log.
```

**Caution:** Pay attention to the edge degree value that is calculated during BinSPreader execution. For the given MBARC26 assembly the edge degree is equal to 1.26 (it can be slightly different in your case), which roughly can be interpreted as on average each edge of the graph has more than one neighbor. BinSPreader relies on connectivity information provided by the assembly graph, so edge degree less than 1 usually indicates fragmented assembly graph where refining may have no effect. You can pull the required line from the BinSPreader log by executing the following.

```
> less binspreader-Rprop.log | grep "edge degree"
```

A possible solution for a disconnected assembly graph with an edge degree below 1 is to provide BinSPreader with an additional paired-end linkage information (“–dataset” option, described above) that increases graph connectivity and facilitates refining. Note that fragmented assembly graph often is an indicator of upstream problems, e.g., issues with sequencing library preparation, insufficient sequencing depth, elevated error rates, etc. and might suggest QC of the assembly input data.

*Optional:* Sometimes binners split a single genome into two or more bins, which overall tend to be very pure but otherwise not complete. Such bins usually overlap in an assembly graph, and great extent of the overlap could potentially be a signal to merge such bins. Apart from that, bins overlap could measure shared genome content for bins that represent closely-related species.

To expose this information, BinSPreader can optionally compute refined bins pairwise similarity matrix based on their overlap in an assembly graph (controlled by “–bin-dist” flag). Refer to Tolstoganov et al.[1] for details on the graph overlap similarity function definition.

You can visualize hierarchical clustering of bin distance values calculated as (1 – bin similarity) using script `visualize_bin_dist.py`. Replace `<folder_name>` with a real path.

```
> python scripts/visualize_bin_dist.py -i <folder_name>/bin_dist.tsv -o figures/dendrogram.png
```

Figure 1 shows a clustering tree calculated from MBARC26 bins obtained with MetaBAT2 and refined with BinSPreader in correction mode. Branch color indicates taxonomic assignment of bins determined using reference genomes that are available for MBARC26 mock dataset. As expected, bins representing the same species, as well as closely related species, typically cluster together.

### Evaluate results

**Timing: 4 h**

Most often the quality of bacterial MAGs is assessed by CheckM tool (Parks et al., 2015), which scans bins for a determined set of single-copy genes to estimate their completeness and contamination. This section describes how to visualize results of binning before and after refining for comparison.

10. Prepare input bins as FASTA files. To create bins in fasta format required by CheckM use extract_fasta_bins.py script:

```
> for FOLDER in binspreader-Rcorr binspreader-Rprop binspreader-Rcorr-Pmax binspreader-Rcorr-sparse ; do python scripts/extract_fasta_bins.py -b $FOLDER/binning.tsv -i assembly/scaffolds.fasta -o $FOLDER/bins/ ; done
```

*Note:* this script assumes that the input binning file is tab-separated and contains two columns ("long" format). If it has many columns ("wide" format), conversion should be done using wide2long.py script:

```
> python scripts/wide2long.py <binning_wide.tsv> > <binning_long.tsv>
```

For example.

```
> python scripts/wide2long.py binspreader-Rcorr-Pmax/binning.tsv > binspreader-Rcorr-Pmax/binning_long.tsv
```

11. Run CheckM on original and refined bins. This command will sequentially run CheckM on five sets of bins:

```
> for FOLDER in metabat2 binspreader-Rcorr binspreader-Rprop binspreader-Rcorr-Pmax binspreader-Rcorr-sparse ; do checkm lineage_wf -t 10 -x fasta --tab_table -f $FOLDER/checkm_result.tsv $FOLDER/bins $FOLDER/checkm ; done
```

*Note:* "lineage_wf" states for CheckM workflow in which quality is estimated with lineage-specific markers identified individually for each bin. Argument "-x" is used to specify non-default file extension of binning files. Option "–tab_table" is used to write a report in tab-separated format to a file specified via the "-f" flag.

12. Inspect text reports and visualize results with script draw_binning_stats.py:

```
> python scripts/draw_binning_stats.py --stats metabat2/checkm/storage/bin_stats_ext.tsv,binspreader-Rcorr/checkm/storage/bin_stats_ext.tsv,binspreader-Rprop/checkm/storage/bin_stats_ext.tsv,binspreader-Rcorr-Pmax/checkm/storage/bin_stats_ext.tsv,binspreader-Rcorr-sparse/checkm/storage/bin_stats_ext.tsv --checkm --output figures --name MBARC26 --labels MetaBAT2,BinSPreader-Rcorr,BinSPreader-Rprop,BinSPreader-Rcorr-Pmax,BinSPreader-Rcorr-sparse
```

*Note:* The script will output plots that demonstrate completeness, contamination and F1 score for each binning assessed with CheckM.

Here we estimate F1 as.

(Completeness + Contamination) / (Completeness * Contamination)

to reflect overall balance between these two metrics. After execution of this command six figures called <metrics_name>_rank_index.png and <metrics_name>_total_length.png will appear in directory figures/.

## EXPECTED OUTCOMES

It is recommended to begin the analysis of binning results with a review of the metagenomic sample and the resulting assembly. Since the MBARC26 sample contains several closely related microorganisms with similar genome sequences, a significant number of short contigs is formed during the assembly process due to the impossibility of resolving interspecific repeats. QUAST results (Table 1) demonstrate that

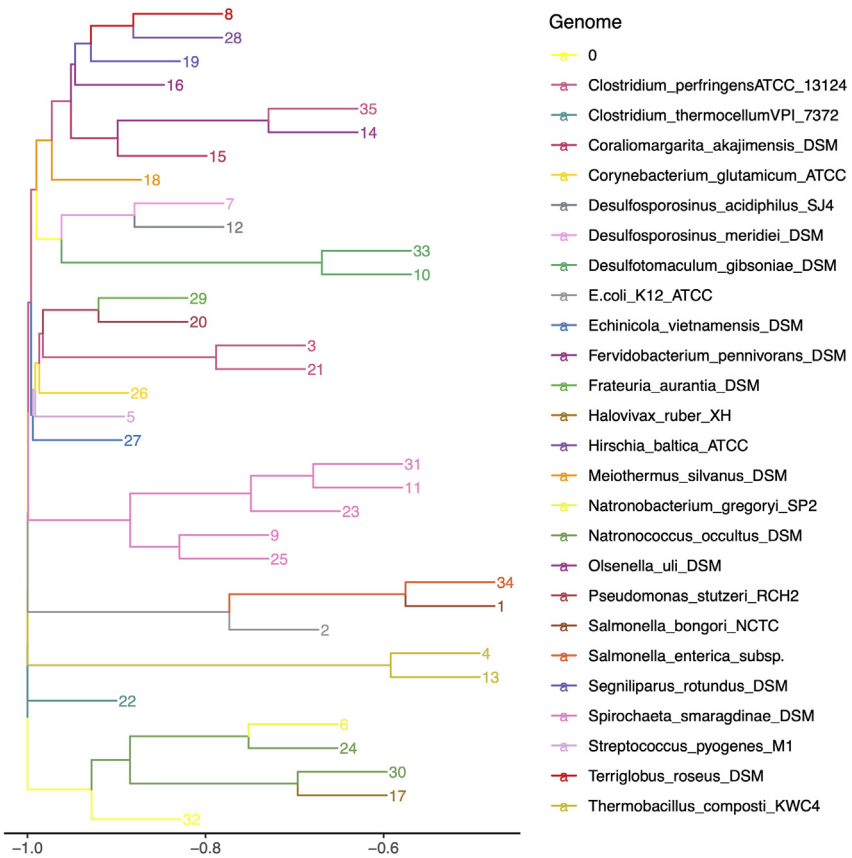



**Figure 1. Hierarchical clustering of bin distances calculated for MetaBAT2 bins refined with BinSPreader in correction mode (MBARC26 dataset)**

**Table 2. (1) Percentages of binned contigs for all used binning methods; (2) The number of 16S rRNA genes depending on their assembled sequence length in relation to the full 16S rRNA length on MBARC26 dataset**

| | (1) Percentage of binned | | (2) Number of detected 16S rRNA genes | |
|---|---|---|---|---|
| Method | Sequences | bp | Length > 50% | Length > 90% |
| MetaBAT2 | 43.7 | 93.5 | 2 | 0 |
| BinSPreader-Rcorr | 97.6 | 99.9 | 15 | 7 |
| BinSPreader-Rprop | 97.6 | 99.9 | 15 | 7 |
| BinSPreader-Rcorr-Pmax | 97.6 | 99.9 | 132 | 66 |
| BinSPreader-Rcorr-sparse | 96.1 | 99.3 | 15 | 7 |
| Total in assembly | – | – | 15 | 7 |

Label description:
BinSPreader-Rcorr – refining in correction+propagation mode.
BinSPreader-Rprop – refining in propagation only mode.
BinSPreader-Rcorr-Pmax – refining in correction+propagation mode with multiple assignment of contigs.
BinSPreader-Rcorr-sparse – refining in correction+propagation mode using sparse propagation algorithm.

17,763 out of 20,631 contigs are shorter than 1000 bp, which is about 5.4% of the total assembly length. Table 2 represents percentages of binned contigs for all used binning methods.

Due to the fact that MetaBAT2 omits short contigs, they are not included in bins, making the MAGs fragmentation problem not so crucial for this particular binning, but still relevant. In addition, MetaBAT2 generally tends to generate high purity bins at the sacrifice of completeness. These observations are confirmed by the CheckM binning assessment (Figure 2). As can be seen from the plots, many of the produced bins have low completeness. Moreover, the number of bins markedly exceeds the expected number of organisms in the sample, which means that several MAGs are split into two or more bins.

After refining with BinSPreader, regardless of the options chosen, the percentage of binned sequences increases significantly, from 43.7% to 96.1–97.6% (Table 2, (1)).

By allocating unbinned contigs to the corresponding bins, BinSPreader increases the value of their completeness, and correspondingly F1 score, without change or with minimal drop in the purity of the bins (Figure 2). Effects of refining may seem little on CheckM estimation because CheckM generally has low sensitivity to short contigs. Its evaluation is based on a set of marker single-copy genes, which are likely to be missing on short contigs. At the same time, these contigs may contain other sequences of interest, such as CRISPRs or genes of rRNA and antimicrobial resistance.

Analysis of bins with Barrnap[22] before and after refining clearly shows a remarkable rise in the number of detected rRNA genes as a consequence of refining (Table 2, (2)). Original MetaBAT2 binning contained only 2 sequences whereas BinSPreader in any mode was able to recover all assembled rRNA genes. Moreover, adoption of a graph-based approach helps to selectively add rRNAs to contig sets of the corresponding species.

However, we need to outline one particular artifact that is related to the multiple contig assignment mode (BinSPreader-Rcorr-Pmax). One can see from Table 2 (2), that the number of contigs with rRNA genes is inadequately high in this mode. The reason is that 16S rRNA genes are highly conservative and therefore their sequences form complex interspecies repeat structures in the assembly graphs on which bins belonging to the different species overlap. In multiple assignment mode such conservative repeated contigs get assigned to several MAGs at once inflating the total number of sequences with rRNA genes. For more information on evidence of the enrichment of refined bins in other functional genetic elements see Tolstoganov et al.[1]

The effects of the different BinSPreader options on binning refining performance for MBARC26 dataset are the following.

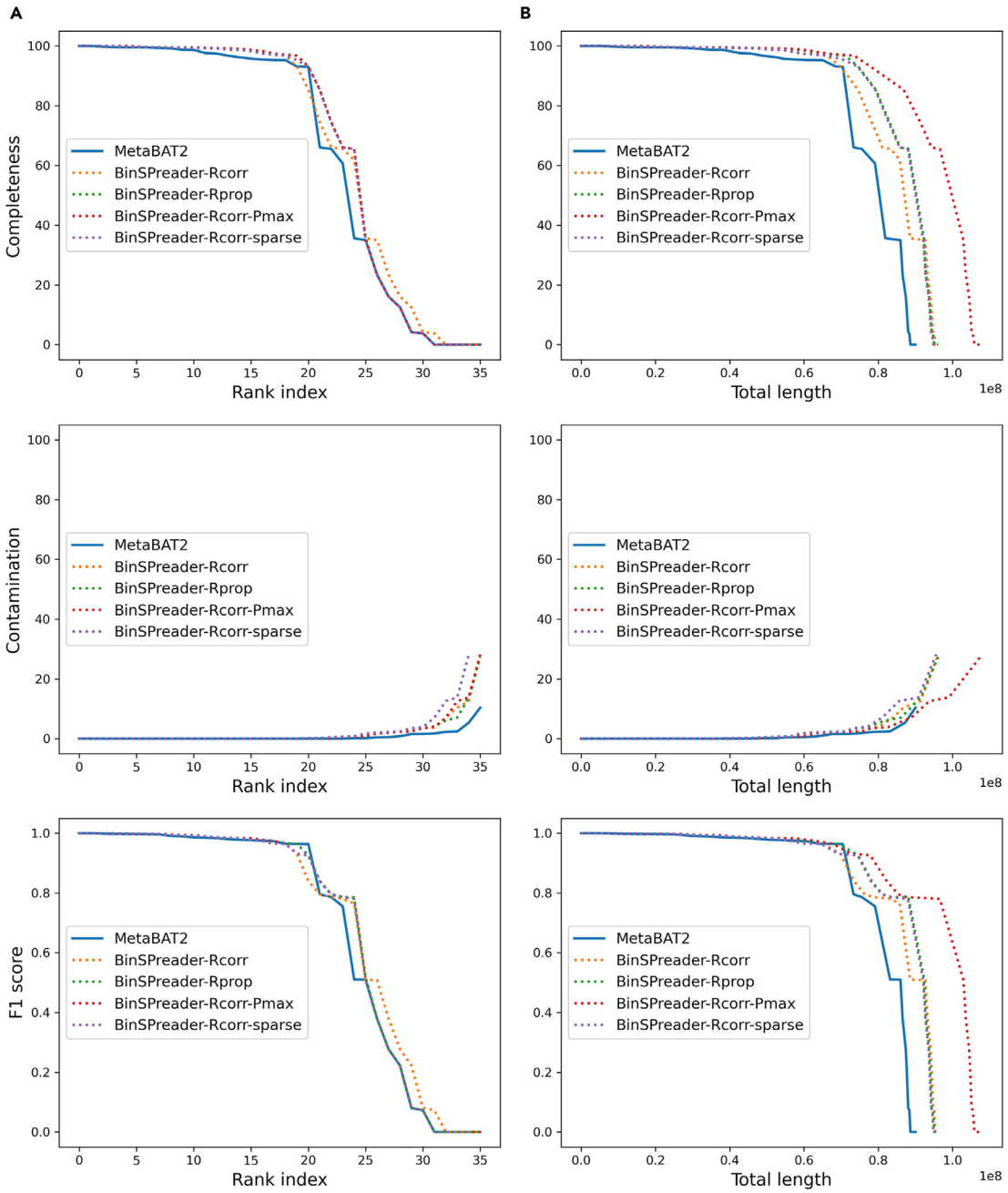



**Figure 2. Comparison of CheckM completeness, contamination and F1 score of MBARC26 bins produced by different binning methods**

(A and B) (A) – ranked by bins' score, (B) – ranked by total length. Label description: BinSPreader-Rcorr – refining in correction+propagation mode; BinSPreader-Rprop – refining in propagation only mode; BinSPreader-Rcorr-Pmax – refining in correction+propagation mode with multiple assignment of contigs; BinSPreader-Rcorr-sparse – refining in correction+propagation mode using sparse propagation algorithm.

- refining in correction+propagation mode (denoted as Rcorr) and propagation only mode (denoted as Rprop) produced very similar binnings according to CheckM. However, Rprop performed better than Rcorr, recovering more complete bins (Figure 2, 1a) and, as a result, demonstrating higher F1 score per total length (Figure 2, 1b).

- sparse propagation mode (referred to as Rcorr-sparse) compared to refining in correction mode with default propagation (referred to as Rcorr) worked in the same way as Rprop and also produced bins with greater increase in completeness (Figure 2, 1b) than Rcorr.
- use of multiple binning (denoted Rcorr-Pmax) considerably increased the length of most bins (Figure 2, 1b-3b), but at the same time does not change per bin values of contamination and completeness significantly (Figure 2, 1a-3a) comparing to other BinSPreader modes, probably due to the mentioned CheckM limitations.

In summary, in the case of MBARC26 metagenome the best options are Rprop and Rcorr-Pmax, both of which produced bins with optimal tradeoffs between completeness and contamination. Overall, the binning refining with BinSPreader successfully compensates for the frequent inability of binners to handle short contigs, potentially enriching the MAG with functional elements and providing additional insights for subsequent studies.

## LIMITATIONS

The BinSPreader algorithm relies on the connectivity obtained from the input assembly graph to infer whether or not the scaffolds belong to certain bins. If the graph is highly fragmented, that is, contains a large fraction of edges disconnected from the main graph, BinSPreader will lack the necessary information for binning propagation. In that case, refining will be unsuccessful without additional connectivity evidence (e.g., from paired-end reads or HiC data).

Besides the assembly graph itself, BinSPreader also relies on the initial binning, while it can correct minor bin assignment mistakes (in its correction mode), it cannot correct major faults of initial binning such as contaminated bins results from several MAGs joined together.

BinSPreader only refines initial binning, therefore it cannot create new bins or recover MAGs that are missed by the initial binner.

BinSPreader implicitly assumes that the whole assembly graph is covered by the bins. This is often not the case if additional filtering or postprocessing was done on the bins. For example, the popular MetaWRAP tool produces filtered binnings using a consensus from 3 underlying binning algorithms. In such cases the initial bins are quite pure, often incomplete and all ambiguous MAGs (e.g., contaminated due to closely-related species) are missed from the initial binning. In such cases BinSPreader might spuriously inflate the bins propagating the binning from binned to completely unbinned parts of the assembly graph. This is a typical use case for sparse binning propagation mode of BinSPreader.

Lastly, while the graph-based bin distance could be a very useful metric for bin merging, its calculation could be very time consuming in case of many bins and complex assembly graphs.

## TROUBLESHOOTING

Overall BinSPreader tries to correctly report possible problems, if any. Comprehending error / warning messages is advised.

### Problem 1

BinSPreader produces error message "Failed to determine k-mer length"

### Potential solution

BinSPreader does not support an arbitrary assembly graph in GFA format. It assumes that the segments all have the same overlaps. Such graphs are produced by de Bruijn graph based assemblers (such as SPAdes or MEGAHIT) or long read assemblers such as Flye. In particular, variable overlaps

are not supported and BinSPreader cannot use such input graphs. The only potential solution is to re-assemble input data using a compatible assembler.

### Problem 2

BinSPreader produces warning message "Unknown scaffold" mentioning scaffold name.

### Potential solution

The paths for the aforementioned scaffold were not found (in the assembly graph file or provided .paths file). Check that the binned sequences correspond to the paths on the assembly graph. Oftentimes this problem appears when contigs and not scaffolds are used for binning. Likely the binning process will need to be redone using the correct set of sequences.

### Problem 3

Excessive running time of BinSPreader during graph propagation step.

### Potential solution

Reduce total number of iterations ("-n" option, 5000 by default) and / or the convergence threshold ("-e" option, 1e-5 by default).

### Problem 4

Excessive running time of BinSPreader during bin distance estimation.

### Potential solution

Disable calculation of bin distance. Reduce the number of input bins. For example, to remove all fasta bins shorter than 200 kb in the current directory, use the following bash code.

```
> for file in *.fasta ; do length=$(grep -v ">" $file | wc | awk '{print $3-$1}') ; if [[ $length -lt 200000 ]] ; then rm $file ; fi ; done
```

### Problem 5

BinSPreader produces large contaminated bins.

### Potential solution

Likely caused by binning not covering the whole assembly graph (see 'Limitation' section). Use sparse propagation mode ("–sparse-propagation" option).

## RESOURCE AVAILABILITY

### Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Anton Korobeynikov (a.korobeynikov@spbu.ru).

### Materials availability

This study did not generate new unique reagents.

### Data and code availability

- The paper analyzes existing, currently available data. The accession ID for the dataset is included into the key resources table.
- BinSPreader is publicly available online from cab.spbu.ru/software/binspreader and BioConda.
- All files and scripts required to reproduce the protocol and files that are generated during it are uploaded to https://figshare.com/articles/dataset/BinSPreader_binning_refining_protocol/21970808.

- Any additional information required to reproduce the protocol reported in this paper is available from the lead contact upon request.

## ACKNOWLEDGMENTS

The research was carried out in part by computational resources provided by the Resource Center "Computer Center of SPbU." The authors are grateful to Saint Petersburg State University for the overall support of this work. I.T., A.L., S.O., and A.K. were supported by the Russian Science Foundation (grant 19-14-00172).

## AUTHOR CONTRIBUTIONS

Conceptualization, S.O., A.K.; methodology, S.O., A.K.; writing – original draft, S.O., A.L., A.K.; resources, S.O., I.T.; software, I.T., A.K.; funding acquisition, A.K.; supervision, A.L., A.K.

## DECLARATION OF INTERESTS

The authors declare no competing interests.

## REFERENCES

1. Tolstoganov, I., Kamenev, Y., Kruglikov, R., Ochkalova, S., and Korobeynikov, A. (2022). BinSPreader: refine binning results for fuller MAG reconstruction. iScience *25*, 104770. https://doi.org/10.1016/j.isci.2022.104770.

2. Singer, E., Andreopoulos, B., Bowers, R.M., Lee, J., Deshpande, S., Chiniquy, J., Ciobanu, D., Klenk, H.-P., Zane, M., Daum, C., et al. (2016). Next generation sequencing data of a defined microbial mock community. Sci. Data *3*, 160081. https://doi.org/10.1038/sdata.2016.81.

3. Anaconda Software Distribution. Anaconda. https://anaconda.com.

4. Parks, D.H., Imelfort, M., Skennerton, C.T., Hugenholtz, P., and Tyson, G.W. (2015). CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. Genome Res. *25*, 1043–1055. https://doi.org/10.1101/gr.186072.114.

5. Kang, D.D., Li, F., Kirton, E., Thomas, A., Egan, R., An, H., and Wang, Z. (2019). MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. PeerJ *7*, e7359. https://doi.org/10.7717/peerj.7359.

6. Mikheenko, A., Saveliev, V., and Gurevich, A. (2016). MetaQUAST: evaluation of metagenome assemblies. Bioinformatics *32*, 1088–1090. https://doi.org/10.1093/bioinformatics/btv697.

7. Nurk, S., Meleshko, D., Korobeynikov, A., and Pevzner, P.A. (2017). metaspades: a new versatile metagenomic assembler. Genome Res. *27*, 824–834. https://doi.org/10.1101/gr.213959.116.

8. SRA Toolkit Development Team. SRA Toolkit. https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software.

9. Bushnell, B. BBMap. sourceforge.net/projects/bbmap/ (Accession on 1 September, 2022)

10. Li, H. (2018). Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics *34*, 3094–3100. https://doi.org/10.1093/bioinformatics/bty191.

11. Danecek, P., Bonfield, J.K., Liddle, J., Marshall, J., Ohan, V., Pollard, M.O., Whitwham, A., Keane, T., McCarthy, S.A., Davies, R.M., and Li, H. (2021). Twelve years of SAMtools and BCFtools. GigaScience *10*, giab008. https://doi.org/10.1093/gigascience/giab008.

12. Van Rossum, G., and Drake, F.L. (2009). Python 3 Reference Manual (CreateSpace).

13. Cock, P.J.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M.J.L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics *25*, 1422–1423. https://doi.org/10.1093/bioinformatics/btp163.

14. Reback, J., McKinney, W., Jbrockmendel, Bossche, J.V.D., Augspurger, T., Cloud, P., Gfyoung, Hawkins, S., Sinhrks, Roeschke, M., et al. (2021). Pandas-Dev/Pandas: Pandas 1.2.2. https://doi.org/10.5281/ZENODO.4524629.

15. Hunter, J.D. (2007). Matplotlib: A 2D Graphics Environment. Comput. Sci. Eng. *9*, 90–95. https://doi.org/10.1109/MCSE.2007.55.

16. Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D.C., Augspurger, T., Halchenko, Y., Cole, J.B., Warmenhoven, J., et al. (2017). Mwaskom/Seaborn: V0.8.1. https://doi.org/10.5281/ZENODO.883859.

17. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2.

18. Thomas, T., Gilbert, J., and Meyer, F. (2012). Metagenomics - a guide from sampling to data analysis. Microb. Inform. Exp. *2*, 3. https://doi.org/10.1186/2042-5783-2-3.

19. Prjibelski, A., Antipov, D., Meleshko, D., Lapidus, A., and Korobeynikov, A. (2020). Using SPAdes De Novo Assembler. Curr. Protoc. Bioinformatics *70*, e102. https://doi.org/10.1002/cpbi.102.

20. Uritskiy, G.V., DiRuggiero, J., and Taylor, J. (2018). MetaWRAP—a flexible pipeline for genome- resolved metagenomic data analysis. Microbiome *6*, 158. https://doi.org/10.1186/s40168-018-0541-1.

21. Sieber, C.M.K., Probst, A.J., Sharrar, A., Thomas, B.C., Hess, M., Tringe, S.G., and Banfield, J.F. (2018). Recovery of genomes from metagenomes via a dereplication, aggregation and scoring strategy. Nat. Microbiol. *3*, 836–843. https://doi.org/10.1038/s41564-018-0171-1.

22. Seeman Torsten. (2013). Barrnap 0.9: Rapid Ribosomal Rna Prediction. https://github.com/tseemann/barrnap.