Using SPAdes De Novo Assembler

Andrey Prjibelski,[1,4] Dmitry Antipov,[1,4] Dmitry Meleshko,[1,4]
Alla Lapidus,[1,2,4] and Anton Korobeynikov[1,3,5]

[1]Center for Algorithmic Biotechnologies, Saint Petersburg State University, Saint Petersburg, Russia
[2]Department of Cytology and Histology, Saint Petersburg State University, Saint Petersburg, Russia
[3]Department of Statistical Modelling, Saint Petersburg State University, Saint Petersburg, Russia
[4]Contributed equally
[5]Corresponding author: *a.korobeynikov@spbu.ru*

SPAdes—St. Petersburg genome Assembler—was originally developed for de novo assembly of genome sequencing data produced for cultivated microbial isolates and for single-cell genomic DNA sequencing. With time, the functionality of SPAdes was extended to enable assembly of IonTorrent data, as well as hybrid assembly from short and long reads (PacBio and Oxford Nanopore). In this article we present protocols for five different assembly pipelines that comprise the SPAdes package and that are used for assembly of metagenomes and transcriptomes as well as assembly of putative plasmids and biosynthetic gene clusters from whole-genome sequencing and metagenomic datasets. In addition, we present guidelines for understanding results with use cases for each pipeline, and several additional support protocols that help in using SPAdes properly. © 2020 Wiley Periodicals LLC.

Basic Protocol 1: Assembling isolate bacterial datasets
Basic Protocol 2: Assembling metagenomic datasets
Basic Protocol 3: Assembling sets of putative plasmids
Basic Protocol 4: Assembling transcriptomes
Basic Protocol 5: Assembling putative biosynthetic gene clusters
Support Protocol 1: Installing SPAdes
Support Protocol 2: Providing input via command line
Support Protocol 3: Providing input data via YAML format
Support Protocol 4: Restarting previous run
Support Protocol 5: Determining strand-specificity of RNA-seq data

Keywords: biosynthetic gene clusters • de novo assembly • genome assembly • metagenomes • plasmids • transcriptome

How to cite this article:
Prjibelski, A., Antipov, D., Meleshko, D., Lapidus, A., & Korobeynikov, A. (2020). Using SPAdes de novo assembler. *Current Protocols in Bioinformatics*, *70,* e102. doi: 10.1002/cpbi.102

INTRODUCTION

Similar to any other de novo sequence assembly software, SPAdes (Nurk et al., 2013) aims to build continuous and accurate sequences (often referred to as contigs and scaffolds) from short reads. Initially, SPAdes was designed for assembly of bacterial genomes from short Illumina reads, obtained via single-cell MDA (Lasken, 2007) or conventional isolate sequencing. Further on, SPAdes was adapted for the assembly of bacterial metagenomes (Nurk, Meleshko, Korobeynikov, & Pevzner, 2017), eukaryotic transcriptomes (Bushmanova, Antipov, Lapidus, & Prjibelski, 2019), and small eukaryotic genomes. Also, the SPAdes package contains pipelines for assembly of putative plasmids (Antipov et al., 2016, 2019) and biosynthetic gene clusters (Meleshko et al., 2019) from whole-genome or metagenomic sequencing data.

In addition to single, paired-end, and mate-pair Illumina reads (Prjibelski et al., 2014; Vasilinetc, Prjibelski, Gurevich, Korobeynikov, & Pevzner, 2015), the current version of SPAdes is capable of assembling IonTorrent data (Ershov, Tarasov, Lapidus, & Korobeynikov, 2019), as well as performing hybrid assembly that combines short accurate reads with long error-prone reads (Antipov, Korobeynikov, McLean, & Pevzner, 2015), such as PacBio and Oxford Nanopores.

SPAdes starts its assembly pipeline by constructing a de Bruijn graph from short reads. Further, the constructed graph undergoes a simplification procedure that involves removal of erroneous edges. Such edges are typically caused by sequencing errors or artifacts. Once the assembly graph is simplified, SPAdes maps short paired and long reads (if available) back to the assembly graph using this alignment information to perform repeat resolution and scaffolding via the exSPAder module (Prjibelski et al., 2014), which aims to construct correct and continuous paths for the genome being assembled in the assembly graph.

Here we describe five basic protocols for the main implemented SPAdes pipelines, as well as several support protocols. The main pipeline for assembly of isolate bacterial data is presented in Basic Protocol 1. Basic Protocol 2 describes the metagenome assembly workflow. Basic Protocol 3 contains information on assembly of putative plasmids from whole-genome shotgun (WGS) and metagenomic data. Basic Protocol 4 includes guidelines for de novo transcriptome assembly. Finally, Basic Protocol 5 is dedicated to the discovery of putative biosynthetic gene clusters. Information about the resulting files produced by each pipeline is presented in the corresponding sections in Guidelines for Understanding Results. The SPAdes installation process is described in Support Protocol 1. Other support protocols describe the specification of SPAdes input via command line (Support Protocol 2) and such features as using a YAML file for providing input data (Support Protocol 3), restarting previous runs (Support Protocol 4), and determining strand specificity of RNA-seq data (Support Protocol 5). The set of supported options and examples of output is given for SPAdes version 3.14.

STRATEGIC PLANNING

Resource Planning

The process of genome, metagenome, and transcriptome assembly may require lots of computational, memory and disk resources depending on the input dataset. Many factors contribute to this including genome size, sequencing error rate, repeat content and complexity of a (meta-)genome, and sequencing depth among, others. Unfortunately, many of these properties are not known in advance and it is possible to derive only very rough estimates. Typically, the assembly process of an isolate bacterial dataset requires several gigabytes of RAM and disk space; a small eukaryotic dataset may require from 10 to 50 Gb of RAM, and assembly of large metagenomes and metatranscriptomes could easily utilize hundreds gigabytes of RAM and disk space. Thus, the provided guidelines are very rough and could vary significantly depending on the particular dataset.

Quality Control

The quality of input data is crucial for both the success of the assembly job and the quality of the final result. Prior to assembly one needs to perform the necessary quality control (QC) steps to ensure that the input read files are free from sequencing artifacts and contamination, e.g., adapters. In some cases, quality trimming might be recommended, as it could significantly lower the error rate and therefore reduce the resource consumption of an assembler. However, the trimming process should be performed very carefully, as it is very easy to provoke coverage gaps by applying aggressive quality trimming, resulting in deterioration of the assembly results. In addition, we need to note that any pre-processing of paired-end reads should keep the left and right reads paired in the files.

Understanding SPAdes Input

The particular type of input files could significantly change the outcome of the assembly, and therefore it is recommended to plan both the sequencing run and assembly pipeline together. SPAdes accepts input as paired-end reads, mate-pairs, and single (unpaired) reads in FASTA and FASTQ formats. For IonTorrent data, SPAdes also supports unpaired reads in unmapped BAM format (like the one produced by Torrent Server). However, in order to run the built-in read error correction, reads should be in FASTQ or BAM format. Sanger, Oxford Nanopore, and PacBio CLR reads can be provided in either format since SPAdes does not run error correction for these types of data.

To run SPAdes 3.14.1, one needs to have at least one library of the following types:

- Illumina paired-end/high-quality mate-pairs/unpaired reads
- IonTorrent paired-end/high-quality mate-pairs/unpaired reads
- PacBio CCS reads

SPAdes should not be used if you plan to assemble:

- Illumina and IonTorrent libraries together
- Only PacBio CLR, Oxford Nanopore, Sanger reads or contig sequences

In other words, SPAdes is capable of performing hybrid assemblies of short and long noisy reads, but cannot be used for long-read assemblies alone.

Important notes:

- If providing SPAdes with multiple paired-end and mate-pair libraries, it is strongly recommended to order them according to their insert size (from smallest to longest).
- It is not recommended to run SPAdes on PacBio reads with low coverage (less than 5).
- We suggest not to run SPAdes on PacBio CCS reads for large genomes.
- SPAdes accepts gzip-compressed files as input.

Read-pair Libraries

By using the command-line interface (see Support Protocol 2), you can specify up to nine different paired-end libraries, up to nine mate-pair libraries, and also up to nine high-quality mate-pair libraries (such as Nextera Mate Pair Libraries) simultaneously. To process more libraries, you can use YAML dataset description input file (See Support Protocol 3). For brevity, we will refer to paired-end and mate-pair libraries simply as read-pair libraries.

By default, SPAdes assumes that paired-end and high-quality mate-pair reads have forward-reverse (fr) orientation and that ordinary mate-pairs have reverse-forward (rf) orientation. However, different orientations can be set for any libraries by using options in SPAdes.

To distinguish reads in pairs, we refer to them as left and right reads. For forward-reverse orientation, the forward reads correspond to the left reads and the reverse reads to the right. Similarly, in reverse-forward orientation, left and right reads correspond to reverse and forward reads, respectively, and so on.

Each read-pair library can be stored in several files or several pairs of files. Paired reads can be organized in two different ways:

- *In file pairs.* In this case, left and right reads are placed in different files and go in the same order in respective files.
- *In interleaved files.* In this case, the reads are interlaced, so that each right read goes after the corresponding paired left read.

For example, the Illumina sequencing process produces paired-end reads in two files: `R1.fastq` contains left reads and `R2.fastq` contains right reads from the read-pair. If you choose to store reads in file pairs, make sure that for every read from `R1.fastq`, the corresponding paired read from `R2.fastq` is placed in the respective paired file on the same line. If you choose to use interleaved files, every read from `R1.fastq` should be followed by the corresponding paired read from `R2.fastq`.

If adapter and/or quality trimming software has been used prior to assembly, files with the orphan reads can be provided as "single read files" for the corresponding read-pair library. It is extremely important to use paired-end-aware read trimming and preprocessing software (see Troubleshooting for more information).

If you have merged some of the reads from your paired-end (not mate-pair or high-quality mate-pair) library using tools such as BBMerge (Bushnell, Rood, & Singer, 2017) or STORM, you should provide the file with resulting reads as a "merged read file" for the corresponding library. Note that non-empty files with the remaining unmerged left/right reads (separate or interlaced) must be provided for the same library in order for SPAdes to correctly detect the original read length.

In an unlikely case that some of the reads from your mate-pair (or high-quality mate-pair) library are "merged," you should provide the resulting reads as a *separate* single-read library.

Unpaired (Single-Read) Libraries

By using the command-line interface, you can specify up to nine different single-read libraries. To input more libraries, you can use a YAML dataset description file (see Support Protocol 3). Single libraries are assumed to have high quality and reasonable coverage. For example, you can provide PacBio CCS reads as a single-read library.

Note that you should not specify PacBio CLR, Sanger reads or additional contigs as single-read libraries, as each of them has a separate option with special treatment.

PacBio and Oxford Nanopore Reads

SPAdes can take as an input an unlimited number of PacBio and Oxford Nanopore libraries for hybrid assemblies (e.g., with Illumina or IonTorrent). There is no need to pre-correct this kind of data. SPAdes will use PacBio CLR and Oxford Nanopore reads for gap closure and repeat resolution. For PacBio, you just need to have filtered subreads in FASTQ/FASTA format. PacBio CCS/Reads of Insert reads or pre-corrected (using third-party software) PacBio CLR/Oxford Nanopore reads can be simply provided as single reads to SPAdes.

Additional Contigs

In case you have contigs generated for the same genome that were generated by other assembler(s) and you wish to merge them into a SPAdes assembly, you can specify additional contigs using `--trusted-contigs` or `--untrusted-contigs`. The first option is used when high-quality contigs are available. These contigs will be used for graph construction, gap closure, and repeat resolution. The second option is used for less reliable contigs that may have more errors or contigs of unknown quality. These contigs will be used only for gap closure and repeat resolution. The number of additional contigs is unlimited.

Note that SPAdes does not perform assembly using genomes of closely related species. Only contigs of the same genome should be specified.

ASSEMBLING ISOLATE BACTERIAL DATASETS

BASIC PROTOCOL 1

This protocol describes the basic assembly process for a regular multi-cell isolate dataset by SPAdes. The protocol assumes that the input is a set of reads obtained from a single bacterial genome sequenced with high coverage depth. Also, the coverage is assumed to be uniform across the genome. There are no other restrictions and assumptions, so all kinds of input data (see "Understanding SPAdes input" section) including long noisy reads are supported and could be provided to SPAdes for hybrid assemblies.

Necessary Resources

Hardware

A 64-bit Linux or MacOS system with as much physical memory as possible is recommended

NOTE: The memory and disk consumption heavily depend on the input data, and in many cases cannot be estimated beforehand (see Strategic Planning section for more information). While SPAdes could be run on computational clusters, it cannot utilize the resources of these clusters fully. In particular, it cannot scale over the multiple nodes; only a single node will be used.

Software

Python and SPAdes package (see Support Protocol 1 for installation)

Input files

Input files with reads in FASTA or FASTQ format. Input files can also be compressed with Gzip. For IonTorrent reads, unmapped BAM files can also be provided. At least one library with short reads is required. Multiple different libraries can be provided simultaneously, but IonTorrent and Illumina reads cannot be used together. Although SPAdes works with reads of any length, we recommend not to use reads shorter than 100 bp.

More information can be found in the Understanding SPAdes input section (see Strategic Planning)

1. Categorize your input data.
   Before launching SPAdes, it is important to understand how to provide your input data. Unpaired libraries for both short and long reads can be given in arbitrary order using appropriate options (see Support Protocol 2). For optimal performance, paired-end and mate-pair libraries should be provided in order of their insert size, from smallest to largest. If several libraries have roughly the same insert size, we recommend providing them as a single library that contains multiple files.

2. Specify your input data.
   Follow Support Protocol 2 to correctly specify your input data via command-line options. Alternatively, input data can be specified via a YAML configuration file (see Support Protocol 3).

3. *Optional:* Set additional parameters.

Although SPAdes does not require any specific additional parameters, you may also want to check Advanced Parameters in the Commentary section of this article, which describes parameters to specify, e.g., number of computation threads to be used, the memory limit, etc.

4. Run SPAdes pipeline.

   Once all options are set, run the following command (bold text represents user input):

   ```
   spades.py --isolate input_data_parameters
     additional_parameters -o output_folder
   ```

   For example:

   ```
   spades.py --isolate --pe1-1 path_to/E.coli.10K.R1.
     fastq.gz --pe1-2 path_to/E.coli.10K.R2.fastq.gz -o
     output_folder
   ```

   for the sample data.

   The output directory will be created automatically. If you specify a folder already containing data from a previous run, all old data will be lost.

5. Check that pipeline has completed successfully.

   Once the SPAdes run is complete, check the end of the log for the presence of error messages (the `spades.log` file can be also found in the output folder). If any error occurred, report it to *spades.support@cab.spbu.ru* or *https://github.com/ablab/spades/issues*. Do not forget to attach the `spades.log` and `params.txt` files.

   If the run was successful, the output folder will contain the following files:

   - `contigs.fasta` — resulting contig sequences in FASTA format;
   - `scaffolds.fasta` — resulting scaffold sequences in FASTA format;
   - `assembly_graph.gfa` — assembly graph and scaffolds paths in GFA 1.0 format;
   - `assembly_graph.fastg` — assembly graph in FASTG format;
   - `contigs.paths` — paths in the assembly graph corresponding to contigs.fasta;
   - `scaffolds.paths` — paths in the assembly graph corresponding to scaffolds.fasta;
   - `spades.log` — file with all log messages.

   To interpret results, follow the guidelines mentioned in Understanding SPAdes Output under Guidelines for Understanding Results.

   Sample file. As sample data for Basic Protocol 1, we provide a subset of the publicly available *E. coli* sequencing dataset (accession number ERR008613). This subset includes only reads mapped to the first 10 kbp of *E. coli str. K12 substr. MG1655* genome. The dataset contains 100 bp Illumina paired-end reads with insert size approximately 215 bp.

   - Left reads: `E.coli.10K.R1.fastq.gz`
   - Right reads: `E.coli.10K.R2.fastq.gz`

   Sample data. Running Basic Protocol 1 using files provided as sample files (see Supporting Information), one should obtain a complete assembly of the first 10 kbp of the *E. coli* genome. Formally speaking, the resulting `contigs.fasta` and `scaffolds.fasta` should contain a single sequence with the following ID:

   ```
   NODE_1_length_10000_cov_196.710910
   ```

BASIC PROTOCOL 2

ASSEMBLING METAGENOMIC DATASETS

This protocol describes the assembly process for metagenomic datasets by metaSPAdes. The protocol assumes that the input is a set of reads obtained from a metagenomic DNA.

metaSPAdes is able to effectively assemble sequenced data from a mix of bacteria of different abundances, and from closely related species. metaSPAdes aims to reconstruct the "core metagenome" (Nurk et al., 2017), so if multiple close strains are presented, the most covered is assembled. See "Understanding metaSPAdes Output" in Guidelines for Understanding Results for more information.

Necessary Resources

Hardware

A 64-bit Linux or MacOS system with as much physical memory as possible is recommended

NOTE: The memory and disk consumption heavily depend on the input data, and in many cases cannot be estimated beforehand (see Strategic Planning section for more information). While metaSPAdes could be run on computational clusters, it cannot utilize the resources of these clusters fully. In particular, it cannot scale over multiple nodes; only single node will be used.

Software

Python and SPAdes package (see Support Protocol 1 for installation)

Input files

Input files with reads sequenced from metagenome dataset in FASTA or FASTQ format. Input files can also be compressed with Gzip. Only one library with Illumina short paired reads is required. In addition, multiple different long-read libraries (PacBio, ONT, etc.) can be provided simultaneously. Although metaSPAdes works with reads of any length, we recommend not to use reads shorter than 100 bp.

1. Specify your input data.

   Follow Support Protocol 2 to correctly specify your input data via command-line options. Alternatively, input data can be specified via a YAML configuration file (see Support Protocol 3).

   In addition to short-read data, an unlimited number of files containing long reads can be provided via following options:

   `--nanopore` for Oxford Nanopore reads;
   `--pacbio` for PacBio reads.

   Specifying additional long reads to supplement short reads will enable the hybridSPAdes algorithm for hybrid metagenome assembly.

2. *Optional:* Set additional parameters.

   Although metaSPAdes does not require any specific additional parameters, you may also want to check Advanced Parameters in the Commentary section of this article, which describes parameters to specify, e.g., number of computation threads to be used, the memory limit, etc.

3. Run metaSPAdes pipeline.

   Once all options are set, run the following command (bold text represents user input):

   ```
   metaspades.py input_data_parameters
     additional_parameters -o output_folder
   ```

   The output directory will be created automatically. If you specify a folder already containing data from a previous run, all old data will be lost.

4. Check that pipeline has completed successfully.

   Once the metaSPAdes run is complete, check the end of the log for the presence of error messages (the `spades.log` file can be also found in the output folder). If any error occurred, report it to *spades.support@cab.spbu.ru* or *https://github.com/ablab/spades/issues*. Do not forget to attach the `spades.log` and `params.txt` files.

5. If the run was successful, the output folder will contain the following files:
   - `contigs.fasta` — resulting contig sequences in FASTA format;
   - `scaffolds.fasta` — resulting scaffold sequences in FASTA format;
   - `assembly_graph.gfa` — assembly graph and scaffolds paths in GFA 1.0 format;
   - `assembly_graph.fastg` — assembly graph in FASTG format;
   - `contigs.paths` — paths in the assembly graph corresponding to `contigs.fasta`;
   - `scaffolds.paths` — paths in the assembly graph corresponding to `scaffolds.fasta`;
   - `spades.log` — file with all log messages.

   To interpret results, follow the guidelines mentioned in Understanding metaSPAdes Output under Guidelines for Understanding Results.

   Sample file. As sample data for Basic Protocol 2, we provide a subset of the publicly available *E. coli* sequencing dataset (accession number ERR008613). This subset includes only reads mapped to the first 10 kbp of *E. coli str. K12 substr. MG1655* genome. The dataset contains 100 bp Illumina paired-end reads with insert size approximately 215 bp (see Supporting Information).

   - Left reads: `E.coli.10K.R1.fastq.gz`
   - Right reads: `E.coli.10K.R2.fastq.gz`

   Sample data. Running Basic Protocol 2 using the files provided as sample files (see Supporting Information), one should obtain a complete assembly of the first 10 kbp of the *E. coli* genome. Formally speaking, the resulting `contigs.fasta` and `scaffolds.fasta` should contain a single sequence with the following ID:
   `NODE_1_length_10000_cov_208.790347`

BASIC PROTOCOL 3

ASSEMBLING SETS OF PUTATIVE PLASMIDS

Plasmids are stably maintained extrachromosomal genetic elements that replicate independently from the host cell's chromosomes.

This protocol describes algorithms and software for assembling plasmids from genomic and metagenomic datasets with plasmidSPAdes and metaplasmidSPAdes

In general, plasmidSPAdes and metaplasmidSPAdes pipelines are mostly based on the main SPAdes and metaSPAdes, with an additional stage that makes it possible to remove contigs that correspond to bacterial chromosomes using the sequence coverage information. plasmidSPAdes and metaplasmidSPAdes perform best on high-copy-number plasmids. For low-copy-number plasmids, Basic Protocol 1 or 2 (depending on dataset type) may provide even better results.

Necessary Resources

Hardware

A machine with a significant amount of physical memory. Recommended RAM minimum is 8 GB, 16 GB should be enough for most bacterial isolates, but more can be required for metagenomic datasets. A 64-bit Linux system or MacOS is also required.

Software

Python and SPAdes package (see Support Protocol 1 for installation)

Input files

Input files with reads sequenced from a genomic or metagenomic dataset in FASTA or FASTQ format. Input files can also be compressed with Gzip. Only one library with Illumina short paired reads is required.

1. Specify your input data.
   Follow Support Protocol 2 to correctly specify your input data via command-line options. Alternatively, input data can be specified via a YAML configuration file (see Support Protocol 3).
   Long reads can be provided via following options:

   `--nanopore` for Oxford Nanopore reads;
   `--pacbio` for PacBio reads;

   We do not recommend using long reads for plasmid assembly.
   Specifying additional long reads to supplement short reads will enable the hybridSPAdes algorithm for hybrid assembly.

2. *Optional:* Set additional parameters.
   Although plasmidSPAdes does not require any specific additional parameters, you may also want to check Advanced Parameters in the Commentary section of this article, which describes parameters to specify, e.g., number of computation threads to be used, the memory limit, etc.

3. Run plasmidSPAdes pipeline.
   Once all options are set, run the following command (bold text represents user input):

   ```
   plasmidspades.py input_data_parameters
     additional_parameters -o output_folder
   ```

   for an isolated bacterial dataset or

   ```
   metaplasmidspades.py input_data_parameters
     additional_parameters -o output_folder
   ```

   for a metagenomic dataset
   The output directory will be created automatically. If you specify a folder already containing data from a previous run, all old data will be lost.

4. Check that pipeline has completed successfully.
   Once the plasmidSPAdes or metaplasmidSPAdes run is complete, check the end of the log for the presence of error messages (the `spades.log` file can be also found in the output folder). If any error occurred, report it to *spades.support@cab.spbu.ru* or *https://github.com/ablab/spades/issues*. Do not forget to attach the `spades.log` and `params.txt` files.

5. If the run was successful, the output folder will contain the following files:
   - `contigs.fasta` — resulting contig sequences in FASTA format;
   - `scaffolds.fasta` — resulting scaffold sequences in FASTA format;
   - `assembly_graph.gfa` — assembly graph and scaffolds paths in GFA 1.0 format;
   - `assembly_graph.fastg` — assembly graph in FASTG format;
   - `contigs.paths` — paths in the assembly graph corresponding to `contigs.fasta`;
   - `scaffolds.paths` — paths in the assembly graph corresponding to `scaffolds.fasta`;
   - `spades.log` — file with all log messages.

   To interpret results, follow the guidelines mentioned in Understanding plasmidSPAdes and metaplasmidSPAdes Output under Guidelines for Understanding Results.

Sample file. As sample data for Basic Protocol 3 we provide a set of reads simulated from *Yersinia pestis strain Cadman plasmid pPCP1* and artificially (*in silico*) mixed with *E. coli* chromosomal reads. The dataset contains 100 bp Illumina paired-end reads with insert size approximately 215 bp (see Supporting Information).

- Left reads: `Y.pestis.pPCP1.R1.fastq.gz`
- Right reads: `Y.pestis.pPCP1.R2.fastq.gz`

Sample data. Running Basic Protocol 3 using the files provided as sample files (as isolated bacterial dataset; see Supporting Information), one should obtain a complete assembly of *Y. pestis plasmid pPCP1*. Formally speaking, the resulting `contigs.fasta` and `scaffolds.fasta` should contain a single sequence with the following ID:

```
NODE_1_length_9689_cov_4.668331_component_0
```

BASIC PROTOCOL 4

ASSEMBLING TRANSCRIPTOMES

RNA sequencing data is typically used for gene expression analysis via mapping reads to a reference genome. However, for organisms without a high-quality reference genome or gene annotation, de novo transcriptome assembly is a viable alternative. This protocol describes the transcriptome assembly process with rnaSPAdes—a pipeline implemented within the SPAdes package. RnaSPAdes performs assembly from short Illumina or IonTorrent reads, but additionally can take PacBio Iso-seq or Oxford Nanopore reads as an input for hybrid assembly.

In general, the rnaSPAdes pipeline is very similar to the main SPAdes workflow. However, rnaSPAdes implements different simplification algorithms, which were designed specifically for transcriptomic data (Bushmanova et al., 2019). Later, instead of repeat resolution and scaffolding, rnaSPAdes performs reconstruction of the sequences of the transcripts (including alternatively spliced isoforms) via the same exPAnder module.

Necessary Resources

Hardware

A machine with a significant amount of physical memory. Recommended RAM minimum is 8 GB, but more can be required for typical RNA-seq datasets. A 64-bit Linux system or MacOS is also required.

Software

Python and SPAdes package (see Support Protocol 1 for installation)

Input files

Input files with RNA reads in FASTA or FASTQ format. Input files can also be compressed with Gzip. For IonTorrent reads, unmapped BAM files can also be provided. At least one library with short reads is required. Multiple different libraries can be provided simultaneously, but IonTorrent and Illumina reads cannot be used together. Although rnaSPAdes works with reads of any length, we recommend not to use reads shorter than 100 bp.

Strand-specific data can also be provided as an input. However, if multiple short-read libraries are specified, strand specificity must be the same for all of them. Combining strand-specific and non-stranded data, or mixing different kinds of strand-specific data, is not possible.

Besides assembly of conventional short-read RNA-seq data, rnaSPAdes also supports hybrid transcriptome assembly (Prjibelski, Korobeynikov, & Lapidus, 2019). In addition to accurate short reads, it takes as an input long noisy RNA reads, such as Pacific Biosciences Iso-seq or Oxford Nanopore, which do not need to be corrected prior to assembly in any case and can be provided in any amount and combination.

More information can be found in Understanding SPAdes Input (see Strategic Planning).

1. Categorize your input data.
   Before launching rnaSPAdes, it is important to understand how to provide your input data. Unpaired libraries for both short and long reads can be given in arbitrary order using appropriate options (see Support Protocol 2). For optimal performance, paired-end libraries should be provided according to their insert size, from smallest to largest. If several libraries have roughly the same insert size, we recommend providing them as a single library that contains multiple files. This recommendation also applies when you aim to assemble a total transcriptome from different samples (e.g., collected from different tissues or under different conditions).
   As mentioned above, IonTorrent and Illumina data cannot be used together. Similarly, libraries with different strand specificity characteristics should not be provided simultaneously.

2. *Optional:* Determine strand specificity of short-read libraries.
   Typically, a library preparation kit specifies whether your data are strand specific or not. However, if you use a dataset of an unknown origin, or simply want to check your library, follow Support Protocol 5.

3. Specify your input data.
   Follow Support Protocol 2 to correctly specify your input data via command-line options. Alternatively, input data can be specified via a YAML configuration file (see Support Protocol 3).
   When using strand-specific data, use the following options:
   - `--ss fr` for forward-reverse strand-specific data;
   - `--ss rf` for reverse-forward strand-specific data.

   In addition to short-read data, an unlimited number of files containing long reads or full transcript sequences can be provided via following options:
   - `--nanopore` for Oxford Nanopore RNA reads;
   - `--pacbio` for PacBio Iso-seq reads;
   - `--fl-rna` for full-length reads/transcript sequences.

4. *Optional:* Set additional parameters.
   Although rnaSPAdes does not require any additional parameters, you may also want to check Advanced Parameters in the Commentary section of this article.

5. Run rnaSPAdes pipeline.
   Once all options are set, run the following command (bold text represents user input):

   ```
   rnaspades.py input_data_parameters
     additional_parameters -o output_folder
   ```

   The output directory will be created automatically. If you specify a folder already containing data from a previous run, all old data will be lost.

6. Check that pipeline has completed successfully.
   Once the rnaSPAdes run is complete, check the end of the log for presence of error messages (the `spades.log` file can be also found in the output folder). If any error occurred, report it to *spades.support@cab.spbu.ru* or *https://github.com/ablab/spades/issues*. Do not forget to attach the `spades.log` and `params.txt` files.
   If the run was successful, the output folder will contain the following files:
   - `transcripts.fasta` — resulting transcript sequences in FASTA format;

- `assembly_graph.gfa` — assembly graph and scaffolds paths in GFA 1.0 format;
- `assembly_graph.fastg` — assembly graph in FASTG format;
- `transcripts.paths` — paths in the assembly graph corresponding to transcripts.fasta;
- `spades.log` — file with all log messages.

To interpret the results, follow the guidelines for understanding rnaSPAdes output.

Sample file. As sample data for Basic Protocol 4 we provide a set of 126 bp paired-end Illumina reads simulated from a single isoform of *H. sapiens* GYPC gene (see Supporting Information).

- Left reads: `H.sapiens.GYPC.R1.fastq.gz`
- Right reads: `H.sapiens.GYPC.R2.fastq.gz`

Sample data. Running Basic Protocol 4 using the files provided as sample files (see Supporting Information), one should obtain a single sequence of the ENST00000356887.12 isoform of the human gene *GYPC*. Formally speaking, the resulting `transcripts.fasta` should contain a single sequence with the following ID:

```
NODE_1_length_1802_cov_33.685238_g0_i0
```

BASIC PROTOCOL 5

ASSEMBLING PUTATIVE BIOSYNTHETIC GENE CLUSTERS

Biosynthetic gene clusters (BGC) are operons encoding secondary metabolites, and are often present in bacterial and fungal genomes. Assembly of some BGC classes is challenging, since they may contain multiple homologous genes inside a single cluster, making the assembly graph highly repetitive. This basic protocol describes the biosyntheticSPAdes assembly process, which takes a single paired-read library with optional long-read libraries and produces an assembly of putative biosynthetic gene-cluster nucleotide sequences. BiosyntheticSPAdes is based on the metaSPAdes pipeline, but introduces additional modules that allow the annotation of BGC domains on the assembly graph, decrease the number of mismatches and indels, and enumerate all putative BGCs.

Necessary Resources

Hardware

A machine with a significant amount of physical memory. Recommended RAM minimum is 8 GB, but typical metagenomics datasets require significantly more memory. 64-bit Linux system or MacOS are also required.

Software

Python and SPAdes package (see Support Protocol 1 for installation).

Input files

Input files with reads sequenced from metagenomics or an isolated bacterial/fungal dataset in FASTA or FASTQ format. Input files can also be compressed with Gzip. Exactly one library with Illumina short paired reads is required. Multiple different long-read libraries (PacBio, ONT, etc.) can be provided simultaneously. Although biosyntheticSPAdes works with reads of any length, we recommend not to use reads shorter than 100 bp.

Sample file: As sample data for Basic Protocol 5 we provide a set of reads simulated from the abyssomicin C biosynthetic gene cluster from *Verrucosispora maris str. AB-18-032*. The dataset contains 150-bp simulated Illumina paired-end reads with insert size of approximately 500 bp.

1. Specify your input data.
   Follow Support Protocol 2 to correctly specify your input data via command-line options. Alternatively, input data can be specified via a YAML configuration file (see Support Protocol 3).
   In addition to short-read data, an unlimited number of files containing long reads can be provided via the following options:

   `--nanopore` for Oxford Nanopore reads;
   `--pacbio` for PacBio reads;

2. *Optional:* Set additional parameters.
   Although biosyntheticSPAdes does not require any additional parameters, you may also want to check Advanced Parameters in the Commentary section of this article, which describes parameters.

3. Run biosyntheticSPAdes pipeline.
   Once all options are set, run the following command (bold text represents user input):

   ```
   spades.py --bio input_data_parameters
     additional_parameters -o output_folder
   ```

   The output directory will be created automatically. If you specify a folder already containing data from a previous run, all old data will be lost.

4. Check that pipeline has completed successfully.
   Once the biosyntheticSPAdes run is complete, check the end of log for presence of error messages (the `spades.log` file can be also found in the output folder). If any error occurred, report it to *spades.support@cab.spbu.ru* or *https://github.com/ablab/spades/issues*. Do not forget to attach the `spades.log` and `params.txt` files.

5. If the run was successful, the output folder will contain the following files:
   - `gene_clusters.fasta` — resulting gene clusters sequences in FASTA format;
   - `bgc_statistics.txt` — file with basic annotation of putative gene clusters;
   - `domain_graph.dot` —domain graph in dot-format;
   - `spades.log` — file with all log messages.

   To interpret the results, follow guidelines mentioned in Understanding biosynthetic-SPAdes Output under Guidelines for Understanding Results.

Sample data

After successful completion of biosyntheticSPAdes on the sample data, output should contain the following:

- `gene_clusters.fasta` should contain two fasta records named `NODE_1_length_59311_cluster_1_candidate_1` and `NODE_1_length_59311_cluster_1_candidate_2`
- `bgc_statistics.txt` should contain information about two putative biosynthetic gene clusters found. The domain sequence for both candidates should be `TE-AT-KR-AT-KR-AT-KR-AT-AT-KR-AT-AT-KR-AT-AT`.
- `domain_graph.dot` should contain domain graph description in dot format. Visual representation should image two isoform subgraphs.
- `spades.log` should contain no error messages.

To interpret the results, follow the guidelines mentioned in Understanding biosynthetic-SPAdes Output under Guidelines for Understanding Results.

SUPPORT PROTOCOL 1

INSTALLING SPAdes

SPAdes is freely available to download in the form of the source code package. Pre-built binaries are available for download as well. While we do our best to ensure that the pre-built binaries are universal enough to run on the wide variety of Linux and Mac platforms, we cannot guarantee that they will be run on your particular system. If it does not, building from the source code is desired. In addition to this, SPAdes is available from several package managers including Conda and Homebrew, among the others.

Necessary Resources

Hardware

SPAdes requires a 64-bit Linux system or Mac OS with as much physical memory as possible (16 Gb or more for larger assembly tasks)

Software

- Python (supported versions are Python2: 2.7, and Python3: 3.2 and higher)
- C++14 compliant C++ compiler. In particular, GCC version 5.8.3 or newer, or clang version 7 or newer, are required
- cmake (version 2.8.12 or higher)
- zlib
- libbz2

1. Download the latest version of SPAdes from *http://cab.spbu.ru/software/spades*.
2. Unpack the tar file (we are using SPAdes 3.14 as an example) and move into SPAdes directory:

```
tar -xzf SPAdes-3.14.1.tar.gz
cd SPAdes-3.14.1
```

3. Build SPAdes with the following script:

```
./spades_compile.sh
```

4. SPAdes will be built in the directory `./bin`. If you wish to install SPAdes into another directory, you can specify the full path of the destination folder by running the following command in bash or sh:

```
PREFIX=<destination_dir>./spades_compile.sh
```

for example:

```
PREFIX=/usr/local./spades_compile.sh
```

which will install SPAdes into `/usr/local/bin`.

5. After installation, you will get the same files (listed below) in the `./bin` directory (or `<destination_dir>/bin` if you specified `PREFIX`). We also suggest adding SPAdes installation directory to the PATH variable.
6. In the case of successful installation, the following files will be placed in the `./bin` directory, among others:
   - `spades.py` (main executable script)
   - `metaspades.py` (main executable script for metaSPAdes)
   - `plasmidspades.py` (main executable script for plasmidSPAdes)
   - `metaplasmidspades.py` (main executable script for metaplasmidSPAdes)
   - `rnaspades.py` (main executable script for rnaSPAdes)
   - `spades-core` (assembly module)
   - `spades-gbuilder` (standalone graph builder application)
   - `spades-gmapper` (standalone long read to graph aligner)
   - `spades-kmercount` (standalone *k*-mer counting application)

- `spades-hammer` (read error correcting module for Illumina reads)
- `spades-ionhammer` (read error correcting module for IonTorrent reads)
- `spades-bwa` (BWA alignment module which is required for mismatch correction)
- `spades-corrector-core` (mismatch correction module).

7. Verify your installation.

   For testing purposes, SPAdes comes with a sample data set (reads that align to the first 1000 bp of *E. coli* genome). To try SPAdes on this data set, run:

   ```
   <spades installation dir>/spades.py --test
   ```

   If you added SPAdes installation directory to the PATH variable, you can run:

   ```
   spades.py --test
   ```

   For simplicity we assume that SPAdes installation directory is added to the PATH variable.

   If the installation is successful, you will find the following information at the end of the log:

```
===== Assembling finished. Used k-mer sizes: 21, 33, 55
* Corrected reads are in spades_test/corrected/
* Assembled contigs are in spades_test/contigs.fasta
* Assembled scaffolds are in
  spades_test/scaffolds.fasta
* Assembly graph is in spades_test/assembly_graph.fastg
* Assembly graph in GFA format is in
  spades_test/assembly_graph.gfa
* Paths in the assembly graph corresponding to the
  contigs are in spades_test/contigs.paths
* Paths in the assembly graph corresponding to the
  scaffolds are in spades_test/scaffolds.paths
======= SPAdes pipeline finished.
========= TEST PASSED CORRECTLY.
SPAdes log can be found here: spades_test/spades.log
Thank you for using SPAdes!
```

PROVIDING INPUT VIA COMMAND LINE

SUPPORT PROTOCOL 2

SPAdes supports multiple input libraries and can utilize them all in order to obtain best possible results. The command-line interface allows basic specification of a single input library as well as an advanced way to specify multiple libraries of different kinds.

Necessary Resources

Hardware

See Basic Protocol 1

Software

See Basic Protocol 1

Specifying single library (paired-end or single-read)

`--12 <file_name>` File with interlaced forward and reverse paired-end reads
`-1 <file_name>` File with forward reads
`-2 <file_name>` File with reverse reads
`--merged <file_name>` File with merged paired reads. If the properties of the library permit, overlapping paired-end reads can be merged using special software. Non-empty files with (remaining) unmerged left/right reads (separate

or interlaced) must be provided for the same library for SPAdes to correctly detect the original read length.

`-s <file_name>` File with unpaired reads

Specifying multiple libraries

Single-read libraries

`--s<#> <file_name>` File for single-read library number `<#> (<#> = 1,2,..,9)`. For example, for the first paired-end library the option is `--s1 <file_name>`. Do not use `-s` options for single-read libraries, since they specifies unpaired reads for the first paired-end library.

Paired-end libraries

`--pe<#>-12 <file_name>` File with interlaced reads for paired-end library number `<#> (<#> = 1,2,..,9)`. For example, for the first single-read library the option is: `--pe1-12 <file_name>`.

`--pe<#>-1 <file_name>` File with left reads for paired-end library number `<#> (<#> = 1,2,..,9)`

`--pe<#>-2 <file_name>` File with right reads for paired-end library number `<#> (<#> = 1,2,..,9)`

`--pe<#>-m <file_name>` File with merged reads from paired-end library number `<#> (<#> = 1,2,..,9)`. If the properties of the library permit, paired reads can be merged using special software. Non-empty files with (remaining) unmerged left/right reads (separate or interlaced) must be provided for the same library for SPAdes to correctly detect the original read length.

`--pe<#>-s <file_name>` File with unpaired reads from paired-end library number `<#> (<#> = 1,2,..,9)`. For example, paired reads can become unpaired during the error correction procedure.

`--pe<#>-<or>` Orientation of reads for paired-end library number `<#> (<#> = 1,2,..,9; <or> = "fr","rf","ff")`. The default orientation for paired-end libraries is forward-reverse (`--> <--`). For example, to specify reverse-forward orientation for the second paired-end library, you should use the flag: `--pe2-rf` (should not be confused with FR and RF strand-specificity for RNA-seq data; see Basic Protocol 3).

Mate-pair libraries

`--mp<#>-12 <file_name>` File with interlaced reads for mate-pair library number `<#> (<#> = 1,2,..,9)`

`--mp<#>-1 <file_name>` File with left reads for mate-pair library number `<#> (<#> = 1,2,..,9)`

`--mp<#>-2 <file_name>` File with right reads for mate-pair library number `<#> (<#> = 1,2,..,9)`

`--mp<#>-<or>` Orientation of reads for mate-pair library number `<#> (<#> = 1,2,..,9; <or> = "fr","rf","ff")`. The default orientation for mate-pair libraries is reverse-forward (`<-- -->`). For example, to specify forward-forward orientation for the first mate-pair library, you should use the flag `--mp1-ff`.

High-quality mate-pair libraries (can be used for mate-pair only assembly)

`--hqmp<#>-12 <file_name>` File with interlaced reads for high-quality mate-pair library number `<#> (<#> = 1,2,..,9)`

`--hqmp<#>-1 <file_name>` File with left reads for high-quality mate-pair library number `<#> (<#> = 1,2,..,9)`

`--hqmp<#>-2 <file_name>` File with right reads for high-quality mate-pair library number `<#> (<#> = 1,2,..,9)`

`--hqmp<#>-s <file_name>` File with unpaired reads from high-quality mate-pair library number `<#> (<#> = 1,2,..,9)`
`--hqmp<#>-<or>` Orientation of reads for high-quality mate-pair library number `<#> (<#> = 1,2,..,9; <or> = "fr","rf","ff").` The default orientation for high-quality mate-pair libraries is forward-reverse (`--> <--`). For example, to specify reverse-forward orientation for the first high-quality mate-pair library, you should use the flag: `--hqmp1-rf`.

Specifying data for hybrid assembly

`--pacbio <file_name>` File with PacBio CLR reads. For PacBio CCS reads use `-s` option
`--nanopore <file_name>` File with Oxford Nanopore reads
`--sanger <file_name>` File with Sanger reads
`--trusted-contigs <file_name>` Reliable contigs of the same genome, which are likely to have no mis-assemblies and small rate of other errors (e.g., mismatches and indels). This option is not intended for contigs of the related species.
`--untrusted-contigs <file_name>` Contigs of the same genome, quality of which is average or unknown. Contigs of poor quality can be used, but may introduce errors in the assembly. This option is also not intended for contigs of the related species.

PROVIDING INPUT VIA YAML FILE

SUPPORT PROTOCOL 3

An alternative way to specify an input dataset for SPAdes is to create a YAML configuration file. By using a YAML file, you can provide an unlimited number of paired-end, mate-pair, and unpaired libraries. Basically, a YAML data set file is a text file in which input libraries are provided as a comma-separated list in square brackets. Each library is provided in brackets as a comma-separated list of attributes.

Necessary Resources

Hardware

See Basic Protocol 1

Software

See Basic Protocol 1

The following YAML attributes are available:

1. Orientation (“fr”, “rf”, “ff”).
2. Type (“paired-end”, “mate-pairs”, “hq-mate-pairs”, “single”, “pacbio”, “nanopore”, “sanger”, “trusted-contigs”, “untrusted-contigs”).
3. Interlaced reads (comma-separated list of files with interlaced reads).
4. Left reads (comma-separated list of files with left reads).
5. Right reads (comma-separated list of files with right reads).
6. Single reads (comma-separated list of files with single reads or unpaired reads from paired library).
7. Merged reads (comma-separated list of files with merged reads).
   To properly specify a library, you should provide its type and at least one file with reads. Orientation is an optional attribute. Its default value is “fr” (forward-reverse) for paired-end and high-quality mate-pair libraries and “rf” (reverse-forward) for standard mate-pair libraries. The value for each attribute is given after a colon.

Comma-separated lists of files should be given in square brackets. For each file, you should provide its full path in double quotes. Make sure that files with right reads are given in the same order as corresponding files with left reads.
For example, if you have one paired-end library, split it into two pairs of files:

```
lib_pe1_left_1.fastq lib_pe1_right_1.fastq
lib_pe1_left_2.fastq lib_pe1_right_2.fastq
```

one mate-pair library:

```
lib_mp1_left.fastq lib_mp1_right.fastq
```

and PacBio CCS and CLR reads:

```
pacbio_ccs.fastq pacbio_clr.fastq
```

Then, the YAML dataset description file should look like this:

```
[
   -- orientation: “fr”,
      type: “paired-end”,
      right reads: [

  “/FULL_PATH_TO_DATASET/lib_pe1_right_1.fastq”,

  “/FULL_PATH_TO_DATASET/lib_pe1_right_2.fastq”
      ],
      left reads: [

  “/FULL_PATH_TO_DATASET/lib_pe1_left_1.fastq”,

  “/FULL_PATH_TO_DATASET/lib_pe1_left_2.fastq”
      ]
   },
   -- orientation: “rf”,
      type: “mate-pairs”,
      right reads: [

  “/FULL_PATH_TO_DATASET/lib_mp1_right.fastq”
      ],
      left reads: [
         “/FULL_PATH_TO_DATASET/lib_mp1_left.fastq”
      ]
   },
   -- type: “single”,
      single reads: [
         “/FULL_PATH_TO_DATASET/pacbio_ccs.fastq”
      ]
   },
   -- type: “pacbio”,
      single reads: [
         “/FULL_PATH_TO_DATASET/pacbio_clr.fastq”
      ]
   }
]
```

Once you have created a YAML file, save it with a `.yaml` extension (e.g., as `my_data_set.yaml`) and run SPAdes using the `--dataset` option: `--dataset <your YAML file>`.

Notes

1. The `--dataset` option cannot be used with any other options for specifying input data. We recommend to nest all files with long reads of the same data type in a single library block.
2. `spades.py` creates the `input_dataset.yaml` file in the output folder from the input command-line options. One could use this file as a template for the further modifications.

RESTARTING A PREVIOUS RUN

SUPPORT PROTOCOL 4

Often, an assembly job will require several days if not weeks to complete, depending on the complexity of the dataset in question. Many computational clusters and shared servers have disk, execution time, and memory quotas, making long-running assemblies quite challenging. SPAdes includes a checkpoint engine that allows fine- and coarse-grained saving of the assembler internal state for future continuation of assembly with optionally changed parameters such as hard memory limit, etc.

One needs to ensure that enough disk space is available should checkpoints be used, as in this case SPAdes saves its entire internal space to disk and therefore the disk space consumption is proportional to the amount of memory used. For large metagenomes, this could easily take up several hundred gigabytes of disk space.

Necessary Resources

Hardware

See Basic Protocol 1

Software

See Basic Protocol 1

1. Decide if coarse-grained checkpoints will be enough for the assembly job. If coarse-grained checkpoints are used, SPAdes will be able to continue only from the start of each of its stages, although neither additional disk space nor computational time required to save checkpoints will be utilized. This is the default SPAdes execution mode. The coarse checkpoints are made after:
   - error correction module is finished
   - iteration for each specified *k* value of assembly module is finished
   - mismatch correction is finished for contigs or scaffolds
2. If fine-grained checkpoints are necessary, decide whether to keep the whole history of checkpoints or only the last successful one. In the case of the latter, SPAdes will be able to continue from the last checkpoint; however, restarting will be possible only from separate stages. The execution time for writing fine-grained checkpoints is the same, and the disk usage is significantly less should only the last checkpoint be kept. Fine-grained checkpoints are enabled via the command-line option `--checkpoints`. The argument should be `all` or `last`, depending on whether all checkpoints or only the last are kept. For example:

   ```
   spades.py -o <output_directory> <other options>
     --checkpoints last
   ```

   saves only the last checkpoint.
3. Run SPAdes using any of the protocols provided in this article.

4. Should SPAdes crash for some reason (e.g., insufficient memory, disk quota exceeded, etc.), continue the SPAdes job using the `--continue` option. In addition, the output directory should be specified, as this is where the checkpoints are created:

   ```
   spades.py --continue -o <output_directory>
   ```

   No other options can be specified along with `--continue`.

5. Alternatively, it is possible to restart SPAdes after changing some execution parameters. For this use the `--restart-from <checkpoint>` command-line option. Checkpoints are:

   - `ec` – start from error correction
   - `as` – restart assembly module from the first iteration
   - `k<int>` – restart from the iteration with specified *k* values, e.g., k55 (not available in RNA-seq mode)
   - `mc` – restart mismatch correction
   - `last` – restart from the last available check-point (similar to –continue)

   In contrast to the `--continue` option, you can change some of the options when using `--restart-from`. You can change any option except: all basic options, all options for specifying input data (including `--dataset`), the `--only-error-correction` option, and the `--only-assembler option`. For example, if you ran the assembler with *k* values 21,33,55 without mismatch correction, you can add one more iteration with *k*=77 and run the mismatch correction step by running SPAdes with the following options:

   ```
   --restart-from k55 -k 21,33,55,77
     --mismatch-correction -o
     <previous_output_dir>
   ```

   Since all files will be overwritten, do not forget to copy your assembly from the previous run if you need it.

SUPPORT PROTOCOL 5

DETERMINING STRAND-SPECIFICITY OF RNA-seq DATA

Strand-specific RNA-seq data can be beneficial for both reference-based analysis and de novo assembly. Typically, a library preparation kit specifies whether your data is strand-specific or not. However, if you use a dataset of unknown origin, or simply want to check your library, follow this protocol.

Necessary Resources

Hardware

Similar to basic requirements (see Basic Protocol 1)

Software

- STAR (Dobin et al., 2013) or Hisat2 (Kim, Langmead, & Salzberg, 2015) aligners
- samtools (Li et al., 2009)
- bedops (Neph et al., 2012)
- RSeQC (Wang, Wang, & Li, 2012)

Files

- Input short-reads RNA-seq data in FASTA or FASTQ format
- Reference genome in FASTA format
- Gene annotation in GTF/GFF/BED format

Note: Bold text represents user input.

1. Construct index of your reference genome.

One of the following spliced aligners can be used:

a. STAR:

```
STAR --runMode genomeGenerate --runThreadN threads
  --genomeDir index --genomeFastaFiles reference.fasta
```

a. Hisat2:

```
hisat2-build reference.fasta index -p threads
```

2. Map reads.
   Use the same tools as in step 1.

a. STAR:

```
STAR --runThreadN threads --genomeDir index
  --readFilesIn left_reads.fastq right_reads.fastq
  --outFileNamePrefix output --outSAMtype BAM
  SortedByCoordinate
```

a. Hisat2:

```
hisat2 -x index -1 left_reads.fastq -2
  right_reads.fastq -p threads -S output.sam
```

3. Sort alignments.
   The resulting alignments are required to be sorted (automatically done when using STAR with `--outSAMtype BAM SortedByCoordinate` option), which can be done by using samtools package:

```
samtools sort input.sam -o sorted.bam
```

4. Convert gene annotation to BED format.
   Genome annotation needs to be converted to BED format. If your annotation is stored in typical GTF/GFF format, conversion can be done using bedops package:

```
convert2bed -i format < annotation > annotation.bed
```

5. Determine strand specificity.
   To determine strand specificity using mapped reads, you will need to run `infer_experiment.py` scripts from the RSeQC package:

```
python infer_experiment.py -i sorted.bam -r annotation.bed
```

If the majority of reads are explained by "1++,1--,2+-,2-+", strand-specificity is likely to be reverse-forward. If the majority of reads are explained by "1+-,1-+,2++,2--", the data seems to have reverse-forward strand-specificity. If both fractions are similar, your data is not strand-specific. For more details see the RSeQC manual at http://rseqc.sourceforge.net/#infer-experiment-py.

GUIDELINES FOR UNDERSTANDING RESULTS

Generic SPAdes output

SPAdes stores all output files in `<output_dir>`, which is set by the user via the -o command line option.

- `<output_dir>/corrected/` contains reads corrected by BayesHammer in `*.fastq.gz` files; if compression is disabled, reads are stored in uncompressed `*.fastq` files
- `<output_dir>/scaffolds.fasta` contains resulting scaffolds (recommended for use as resulting sequences)
- `<output_dir>/contigs.fasta` contains resulting contigs
- `<output_dir>/assembly_graph.gfa` contains SPAdes assembly graph and scaffold paths in GFA 1.0 format

- `<output_dir>/assembly_graph.fastg` contains SPAdes assembly graph in FASTG format
- `<output_dir>/contigs.paths` contains paths in the assembly graph corresponding to `contigs.fasta` (see details below)
- `<output_dir>/scaffolds.paths` contains paths in the assembly graph corresponding to scaffolds.fasta (see details below).

Contig and scaffold names in SPAdes output FASTA files have the following format:

```
NODE_3_length_237403_cov_243.207
```

Here, 3 is the number of the contig/scaffold (contigs are ordered by their length), 237403 is the sequence length in nucleotides, and 243.207 is the *k*-mer coverage for the last (largest) *k*-mer length used. Note that the *k*-mer coverage is always lower than the read (per-base) coverage.

In general, SPAdes uses two techniques for joining contigs into scaffolds. The first one relies on read-pairs and tries to estimate the size of the gap separating contigs. The second one relies on the assembly graph: e.g., if two contigs are separated by a complex tandem repeat that cannot be resolved exactly, contigs are joined into a scaffold with a fixed gap size of 100 *N* symbols inserted to outline this. Contigs produced by SPAdes do not contain ambiguous IUPAC symbols (including *N*).

Understanding Scaffold Paths

To view FASTG and GFA files, we recommend using a Bandage visualization tool (Wick, Schultz, Zobel, & Holt, 2015). Note that sequences stored in `assembly_graph.fastg` correspond to contigs before repeat resolution (edges of the assembly graph). Paths corresponding to contigs after repeat resolution (scaffolding) are stored in `contigs.paths (scaffolds.paths)` in the format accepted by Bandage (see Bandage Wiki for details). The example is given below.

Let the contig with the name `NODE_5_length_100000_cov_215.651` consist of the following edges of the assembly graph:

```
>EDGE_2_length_33280_cov_199.702
>EDGE_5_length_84_cov_321.414'
>EDGE_3_length_111_cov_175.304
>EDGE_5_length_84_cov_321.414'
>EDGE_4_length_66661_cov_223.548
```

Then, `contigs.paths` will contain the following entry:

```
NODE_5_length_100000_cov_215.651
2+,5-,3+,5-,4+
```

Since the current version of Bandage does not accept paths with gaps, paths corresponding contigs/scaffolds jumping over a gap in the assembly graph are split by a semicolon at the gap position. For example, the following record:

```
NODE_3_length_237403_cov_243.207
21-,17-,15+,17-,16+;
31+,23-,22+,23-,4-
```

states that `NODE_3_length_237403_cov_243.207` corresponds to the path with 10 edges, but jumps over a gap between edges `EDGE_16_length_21503_cov_482.709` and `EDGE_31_length_140767_cov_220.239`.

Understanding metaSPAdes Output

There is no universal outcome for metagenome assembly. The ideal scenario is to obtain the complete genome sequence of each and every species presented in the dataset. However, usually this is not possible, especially when strains are involved, as strain variations need to be distinguished from sequencing errors (see Nurk et al., 2017, and Prjibelski et al., 2019, for extensive discussion).

In short, to deal with this problem, metaSPAdes aims to achieve a so-called consensus assembly by detecting and masking variations between seemingly related strains. Consensus assemblies as opposed to strain assemblies have both advantages and disadvantages for downstream analysis.

Strain assemblies:

- Pros: ability to accurately reconstruct information about individual strains
- Cons: assembly quality is often inferior due to long unresolved repeats

Consensus assemblies:

- Pros: ability to reconstruct the backbone structure of the species genome
- Cons: can result in annotation artifacts and lose information about individual strains

The set of output files of metaSPAdes is exactly the same as that of SPAdes. However the files `contigs.fasta` and `scaffolds.fasta` contain the consensus assembly representing the backbone structure of a metagenome.

Understanding rnaSPAdes Output

Most files in rnaSPAdes output are the same as the ones produced by genomic SPAdes. However, instead of contigs and scaffolds, rnaSPAdes produces transcript sequences, which are written to `transcripts.fasta`. Corresponding paths in the assembly graph are stored in the `transcripts.paths` file.

Beside length and coverage, sequence IDs in the FASTA file also have additional information. Each FASTA header contains a gene id (follows after `_g`) and an isoform number within this gene (follows after `_i`). For example, sequence header `NODE_41_length_1352_cov_13.370137_g53_i2` means that this particular transcript is the isoform #2 in the gene with ID=53 (numbering starts with 0 for both). Isoforms of the same gene must have at least 300 common consecutive nucleotides, which means that transcripts of paralogous genes are often classified as alternative isoforms of a single gene. Moreover, isoforms of short genes may not be grouped together.

In addition to `transcripts.fasta`, rnaSPAdes produces `hard_filtered_transcripts.fasta` and `soft_filtered_transcripts.fasta`. The first file is generated by applying stronger coverage and length filters to the original sequences from `transcripts.fasta`. The second one includes low-covered and short sequences in addition to all transcripts from `transcripts.fasta`. Exact parameters used for filtering assembled sequences are given in the supplementary material of Bushmanova et al. (2019).

In case of properly used strand-specific data, all transcripts are expected to have the same orientation (strand) as the original RNA molecule that it was sequenced from.

Understanding plasmidSPAdes and metaplasmidSPAdes Output

plasmidSPAdes and metaplasmidSPAdes output only DNA sequences of putative plasmids. Output file names and formats remain the same as in SPAdes (see previous sections), with the following differences.

For all contig names in `contigs.fasta`, `scaffolds.fasta`, and `assembly_graph.fastg`, plasmidSPAdes appends a suffix of the form `_component_X`, where `X` is the identifier of the putative plasmid that the contig belongs to. Note that plasmidSPAdes may not be able to separate similar plasmids, and thus their contigs may appear with the same identifier.

For metaplasmidSPAdes, only complete putative plasmids (i.e., circular contigs) are output, so the`_component_X` suffix is not attached.

Understanding biosyntheticSPAdes Output

In addition to the files outlined above biosyntheticSPAdes outputs three files of interest:

- `gene_clusters.fasta` – contains DNA sequences from putative biosynthetic gene clusters (BGC). Since each sample may contain multiple BGCs, and biosyntheticSPAdes can output several putative DNA sequences for each cluster, for each contig name we append the suffix `_cluster_X_candidate_Y`, where `X` is the identifier of the BGC and `Y` is the identifier of the candidate from the BGC.
- `bgc_statistics.txt` – contains statistics about BGC composition in the sample. First, it outputs number of domain hits in the sample. Then, for each BGC candidate we output domain order, with positions on the corresponding DNA sequence from `gene_clusters.fasta`.
- `domain_graph.dot` – contains domain graph structure, which can be used to assess complexity of the sample and structure of BGCs. For more information about domain graph construction, please refer to the original article (Meleshko et al., 2019).

COMMENTARY

Background Information

SPAdes development began back in 2011 with a group of young Russian researchers under the supervision of Prof. Pavel Pevzner from UCSD. The assembler takes its name from the city where the lab is located—St. Petersburg Assembler. The original goal was to develop an assembly method for a novel type of data—single-cell bacterial sequencing performed via the Multiple Displacement Amplification protocol (Lasken, 2007). During development, it appeared that most of the designed algorithms and techniques were well suited for assembling conventional isolate bacterial data as well.

As more and more algorithms were developed, SPAdes turned into a large universal assembly framework containing multiple different computational methods. Thus, construction of novel pipelines for different types of data became a logical continuation of the lab's research. Later on, SPAdes accumulated various algorithms for repeat resolving and scaffolding, hybrid assembly, assembly of metagenomes and transcriptomes, detection of putative plasmids in WGS and metagenomic data, etc.

As multiple assembly tools were developed, the problem of choosing the optimal computational method for each particular type of assembly became more challenging and important. Thus, special tools were created for quality assessment of genome (Gurevich, Saveliev, Vyahhi, & Tesler, 2014), metagenome (Mikheenko, Saveliev, & Gurevich, 2016), and transcriptome de novo assemblies (Bushmanova, Antipov, Lapidus, Suvorov, & Prjibelski, 2016; Smith-Unna, Boursnell, Patro, Hibberd, & Kelly, 2016). In addition, several independent studies performed comparisons between different assembly tools. For example, comparison between assembly tools on bacterial datasets was performed in a GAGE-B study (Magoc et al., 2013). Another recent paper presents a comprehensive comparison between various transcriptome assemblers on multiple RNA-seq datasets obtained from different species (Hölzer & Marz, 2019).

Critical Parameters

K-mer size selection

K-mer size is an extremely important parameter for all de Bruijn graph−based assemblers. Improperly selected *k* values may lead to deterioration of assembly quality. Selecting the optimal *k*-mer size is a non-trivial task, which depends on multiple different characteristics of the data being assembled: dataset type (isolate, single-cell, metagenome etc.), dataset coverage, and read length. Moreover, different assemblers may produce optimal results on the same dataset with completely different *k* values.

The SPAdes assembler implements iterative graph construction with multiple *k*-mer sizes. First, it constructs an assembly graph with a rather small *k* value, and passes the resulting edge sequences as an input (along with all reads) to the next iteration with larger *k*. Once the assembly graph with the final (largest) *k*-mer size is constructed, SPAdes performs repeat resolution and scaffolding. In other words, SPAdes uses all the information from all *k*-mer sizes; it does not merely "select best *k*-mer size from the set" like, e.g., Velvet Optimizer.

Every pipeline implemented in the SPAdes package automatically detects a set of nearly optimal *k*-mer sizes depending on data type and read length. For example, when assembling conventional Illumina sequencing data, SPAdes uses k =21,33,55 when read length is less than 150 bp. For read length varying between 150 bp and 249 bp, k = 21,33,55,77 are used. For 250-bp reads or longer, SPAdes sets k = 21,33,55,77,99,127. Alternatively, rnaSPAdes pipeline always uses two *k*-mer sizes, the lower of which is set to one-third of read length and the larger one equal to one-half of read length.

Although these sets of *k*-mer sizes may not give the best results in absolutely all cases, they seem to provide nearly optimal assemblies on most of the tested datasets. Thus, we highly recommend not to change this parameter, unless you are completely sure. If you finally decide to set *k*-mer sizes manually, use the `-k` option. All *k* values must be odd, must be provided in ascending order, and be larger than 10 and less than 128.

In general, it is not recommended to change the default set of *k*-mer values unless it is explicitly necessary for some reason. Also, the tools that do automatic selection of a single *k*-mer length, like KmerGenie (Chikhi & Medvedev, 2013) are not designed to be used with multi *k*-mer assemblers like SPAdes, and therefore their use is not advised. We also refrain from using the famous N50 statistics for selecting the best *k*-mer size, as larger *k*-mer sizes might provide better N50 at a price of much elevated mis-assembly, rate due to coverage gaps in low-covered regions of a genome.

Correcting sequencing errors in input reads

Correcting sequencing errors in reads prior to the assembly can be done in order to increase assembler's performance and accuracy. The SPAdes package includes two modules for read-error correction: BayesHammer (Nikolenko, Korobeynikov, & Alekseyev, 2013) for correcting Illumina reads and IonHammer (Ershov et al., 2019) for correcting IonTorrent data.

Each pipeline implemented in SPAdes automatically launches the error-correction procedure if needed. However, if your reads were already trimmed or pre-corrected by another tool, you may turn off the error correction stage by using `--only-assembler` flag.

In addition, if you do not need to perform the assembly, you may run only the error-correction module by setting `--only-error-correction` flag when running SPAdes. BayesHammer (Illumina error correction) is launched by default. To launch the IonTorrent error correction module IonHammer, set `--iontorrent` flag.

Reducing the number of mismatch and short indel errors in the assembly

The SPAdes package also includes a MismatchCorrector module for polishing the resulting assemblies, which maps raw reads onto the assembled sequences and corrects mismatches and short indels. To enable assembly polishing, set the `--careful` flag. Note, that this option can be used only when running the standard genome-assembly SPAdes pipeline.

Troubleshooting

The assembly process is a complex task, with success depending on both the quality of input data and available resources. Many things could go wrong, and it is extremely important to carefully read the `spades.log` file for the possible errors, as SPAdes tries to provide meaningful error messages in the majority of common cases. We outline some of these below. Still, if the error message appears to be cryptic, we suggest contacting the SPAdes support team at *spades.support@cab.spbu.ru* or opening an issue at *https://github.com/ablab/spades/issues*, attaching the `spades.log` and `params.txt` files from the output directory for further explanation.

Corrupted input files

One needs to ensure that the input to SPAdes is correct both structurally and logically. In particular, the input files should be in the correct FASTA/FASTQ formats; otherwise, the following errors might be observed:

• the length of the sequence line for some read in the FASTQ file does not correspond to the length of the quality line;

• inability of SPAdes to detect the Phred offset of FASTQ file

Also, the left reads of paired-end datasets should correspond to the right ones. This correspondence is typically broken when using non-paired-end trimmers or other filtering or pre-processing tools. As a result, the number of left and right reads in the files will be different and SPAdes will end with an error.

Typically SPAdes will provide meaningful error messages in these cases. These errors are fatal, and essentially the only solution in this case is making the input correct. This involves stepping back in the computational pipeline and fixing whatever caused the broken input files.

Out-of-memory errors

These kinds of errors are probably the ones occurring most often. As outlined above, it is usually impossible to estimate the amount of memory needed solely from the size of the input files. Still, SPAdes tries to do its best, and where possible estimate the necessary amount of memory for the next steps. If it is determined that the selected hard memory limit (see Advanced Options) will not allow proceeding with the next step, then SPAdes terminates prematurely, allowing the user to either increase the memory limit and restart (see Support Protocol 4) or move to a larger machine.

Unfortunately, it is not always possible to provide a meaningful error message in the case of out-of-memory errors, as producing such a message would require additional memory allocation. Even more, some resource management systems as seen on computational clusters would kill the SPAdes job should it try to overshoot the memory, precluding any proper error reporting. In addition, sometimes there is some free RAM available on the system available, but due to memory fragmentation, the operating system might be unable to fulfil SPAdes' request to allocate a single large chunk of memory. There is no way to overcome this error, and therefore the SPAdes process will terminate.

The only viable solution to these problems is to allocate more RAM, either on the same machine or by moving to a larger one. Another possibility (although with different results) is to try to perform some filtering of the input data to reduce its complexity. Heavy quality trimming and/or removing low-covered reads (for example using the `spades-read-filter` tool) might help; however, the assembly results might appear suboptimal.

I/O problems

SPAdes uses disk storage for scratch intermediate files and accesses input reads several times at different steps of the pipeline. It is extremely important to ensure that all these files are being written and read without problems. Many internal consistency checks are introduced, and therefore I/O errors usually cause them to fail. While these errors are fatal, usually they are transient and therefore could simply be "fixed" by a restart. Otherwise, we suggest contacting your local system administrator for possible solutions to these problems.

Problems with the orientation of paired-end reads

Sometimes the orientation of paired-end reads is non-standard (e.g., not default forward-reverse) and needs to be specified explicitly. The happens more often for mate pairs, as their orientation could be specific to a particular library preparation and sequencing protocol. While SPAdes provides a sensible default, it also tries to detect whether the orientation of paired-end reads or mate-pairs is specified properly. If not, then typically the warning about negative insert size is displayed, and the user is advised to check if the orientation is specified properly.

Problems with fragment length distribution

SPAdes relies on the fragment length distribution of paired-end reads for its repeat resolution and scaffolding process. Since the fragment length is not known in advance, it estimates it via aligning paired-end reads to the long edges of the assembly graph. Sometimes, no paired reads can be aligned to the edges of the assembly graph. There are several possibilities for why this could happen:

• Very fragmented assembly: all long edges of an assembly graph are shorter than the typical fragment length.

• The input reads are corrupted, as the left reads do not correspond to the right ones.

The user is then advised to check parameters used and input data to find and solve possible problems (e.g., using a non-paired-end aware filtering tool).

Uneven coverage

Some modes of the SPAdes pipeline (e.g., isolate and normal multicell mode as well as plasmidSPAdes) rely on the uniform coverage of a genome in question. This assumption is used everywhere in the pipeline, and the assembly results might be bad if it is violated. There are many reasons for uneven coverage:

• Contamination

• Sequencing bias and artifacts

• Assembling just a subset of some reads (e.g., aligned to a particular database) and not the whole genome or a large part of it.

SPAdes tries to detect whether the assumption of uniform coverage is violated in any places in the pipeline both prior to and after assembly. Sometimes the violation could be considered fatal; sometimes only a warning is issued. While a user could overcome the errors by switching to aSPAdes modes that does not use the assumption of even coverage (e.g., single cell mode or metaSPAdes), it is strongly recommended to try to understand the reason for uneven coverage, as it could easily lead to suboptimal assembly results.

Advanced Parameters

Beside input data parameters (see Support Protocols 2 and 3) and several important options (described in Critical Parameters), SPAdes has additional options, which are not required to be set during the assembly, but may be used to optimize the process.

• `-t` sets the number of threads (default is 16 or the number of CPU cores on the system, whichever is less)

• `-m` sets RAM upper limit in gigabytes (default is 250 or the total amount of RAM available, whichever is less); SPAdes will halt if the memory limit is exceeded

• `--disable-gzip-output` disables compression of error-corrected reads

• `--disable-rr` turns off repeat resolution and scaffolding stage

• `--sc` enables pipeline for assembling single-cell data obtained via the MDA protocol

• `--tmp-dir` sets destination for temporary files (default is `output_dir/tmp`)

• `--cov-cutoff` sets read-coverage cutoff: can be positive float number, or ‘`auto`’, or ‘`off`’ (default is `off`).

Acknowledgments

This work was supported by the Russian Fund for Basic Research (grant 18-54-74004). Research was carried out in part by computational resources provided by Resource Center “Computer Center of SPbU.” The authors are grateful to Saint Petersburg State University for the overall support of this work (project id: 51555639).

Literature Cited

Antipov, D., Hartwick, N., Shen, M., Raiko, M., Lapidus, A., & Pevzner, P. (2016). PlasmidSPAdes: Assembling plasmids from whole genome sequencing data. *Bioinformatics*, *32*(22), 3380–3387.

Antipov, D., Korobeynikov, A., McLean, J. S., & Pevzner, P. A. (2015). hybridSPAdes: An algorithm for hybrid assembly of short and long reads. *Bioinformatics*, *32*(7), 1009–1015. doi: 10.1093/bioinformatics/btv688.

Antipov, D., Raiko, M., Lapidus, A., & Pevzner, P. A. (2019). Plasmid detection and assembly in genomic and metagenomic data sets. *Genome Research*, *29*(6), 961–968. doi: 10.1101/gr.241299.118.

Bushmanova, E., Antipov, D., Lapidus, A., & Prjibelski, A. D. (2019). rnaSPAdes: A de novo transcriptome assembler and its application to rna-seq data. *GigaScience*, *8*(9), giz100. doi: 10.1093/gigascience/giz100.

Bushmanova, E., Antipov, D., Lapidus, A., Suvorov, V., & Prjibelski, A. D. (2016). rnaQUAST: A quality assessment tool for de novo transcriptome assemblies. *Bioinformatics*, *32*(14), 2210–2212. doi: 10.1093/bioinformatics/btw218.

Bushnell, B., Rood, J., & Singer, E. (2017). BBMerge: Accurate paired shotgun read merging via overlap. *PloS One*, *12*(10), e0185056. doi: 10.1371/journal.pone.0185056.

Chikhi, R., & Medvedev, P. (2013). Informed and automated *k*-mer size selection for genome assemble. *Bioinformatics*, *30*(1), 31–37. doi: 10.1093/bioinformatics/btt310.

Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., … Gingeras, T. R. (2013). Star: Ultrafast universal rna-seq aligner. *Bioinformatics*, *29*(1), 15–21.

Ershov, V., Tarasov, A., Lapidus, A., & Korobeynikov, A. (2019). IonHammer: Homopolymer-space hamming clustering for iontorrent read error correction. *Journal of Computational Biology*, *26*(2), 124–127. doi: 10.1089/cmb.2018.0152.

Gurevich, A., Saveliev, V., Vyahhi, N., & Tesler, G. (2013). QUAST: Quality assessment tool for genome assemblies. *Bioinformatics*, *29*, 1072–1075. doi: 10.1093/bioinformatics/btt086.

Hölzer, M., & Marz, M. (2019). De novo transcriptome assembly: A comprehensive cross-species comparison of short-read rna-seq assemblers. *GigaScience*, *8*(5), giz039. doi: 10.1093/gigascience/giz039.

Kim, D., Langmead, B., & Salzberg, S. L. (2015). Hisat: A fast spliced aligner with low memory requirements. *Nature Methods*, *12*(4), 357.

Lasken, R. S. (2007). Single-cell genomic sequencing using Multiple Displacement Amplification.

Current Opinion in Microbiology, *10*, 510–516. doi: 10.1016/j.mib.2007.08.005.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., … Durbin, R. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. doi: 10.1093/bioinformatics/btp352.

Magoc, T., Pabinger, S., Canzar, S., Liu, X., Su, Q., Puiu, D., … Salzberg, S. L. (2013). GAGE-B: An evaluation of genome assemblers for bacterial organisms. *Bioinformatics*, *29*(14), 1718–1725. doi: 10.1093/bioinformatics/btt273.

Meleshko, D., Mohimani, H., Tracanna, V., Hajirasouliha, I., Medema, M. H., Korobeynikov, A., & Pevzner, P. A. (2019). BiosyntheticSPAdes: Reconstructing biosynthetic gene clusters from assembly graphs. *Genome Research*, *29*(8), 1352–1362. doi: 10.1101/gr.243477.118.

Mikheenko, A., Saveliev, V., & Gurevich, A. (2016). Metaquast: Evaluation of metagenome assemblies. *Bioinformatics*, *32*(7), 1088–1090. doi: 10.1093/bioinformatics/btv697.

Neph, S., Kuehn, M. S., Reynolds, A. P., Haugen, E., Thurman, R. E., & Johnson, A. K. … others.(2012) BEDOPS: High-performance genomic feature operations. *Bioinformatics*, *28*(14), 1919–1920. doi: 10.1093/bioinformatics/bts277.

Nikolenko, S., Korobeynikov, A., & Alekseyev, M. (2013). BayesHammer: Bayesian clustering for error correction in single-cell sequencing. *BMC Genomics*, *14*, S7. doi: 10.1186/1471-2164-14-S1-S7.

Nurk, S., Bankevich, A., Antipov, D., Gurevich, A. A., Korobeynikov, A., Lapidus, A., … Pevzner, P. A. (2013). Assembling single-cell genomes and mini-metagenomes from chimeric MDA products. *Journal of Computational Biology*, *20*(10), 714–737. doi: 10.1089/cmb.2013.0084.

Nurk, S., Meleshko, D., Korobeynikov, A., & Pevzner, P. A. (2017). metaSPAdes: A new versatile metagenomic assembler. *Genome Research*, *27*(5), 824–834. doi: 10.1101/gr.213959.116.

Prjibelski, A. D., Puglia, G. D., Antipov, D., Bushmanova, E., Giordano, D., Mikheenko, A., … Lapidus, A. (2019). Extending rnaSPAdes functionality for hybrid transcriptome assembly. *BMC Bioinformatics*, *20*(S17), 2.

Prjibelski, A. D., Vasilinetc, I., Bankevich, A., Gurevich, A., Krivosheeva, T., Nurk, S., … Pevzner, P. A. (2014). ExSPAnder: A universal repeat resolver for DNA fragment assembly. *Bioinformatics*, *30*(12), i293–i301. doi: 10.1093/bioinformatics/btu266.

Prjibelski, A. D., Korobeynikov, A. I., & Lapidus, A. L. (2019). Sequence analysis. *Encyclopedia of Bioinformatics and Computational Biology*, *3*, 292–322. doi: 10.1016/B978-0-12-809633-8.20106-4.

Smith-Unna, R., Boursnell, C., Patro, R., Hibberd, J. M., & Kelly, S. (2016). TransRate: Reference-free quality assessment of de novo transcriptome assemblies. *Genome Research*, *2*(8), 1134–1144. doi: 10.1101/gr.196469.115.

Vasilinetc, I., Prjibelski, A. D., Gurevich, A., Korobeynikov, A., & Pevzner, P. A. (2015). Assembling short reads from jumping libraries with large insert sizes. *Bioinformatics*, *31*(20), 3262–3268. doi: 10.1093/bioinformatics/btv337.

Wang, L., Wang, S., & Li, W. (2012). RSeQC: Quality control of RNA-seq experiments. *Bioinformatics*, *28*(16), 2184–2185. doi: 10.1093/bioinformatics/bts356.

Wick, R. R., Schultz, M. B., Zobel, J., & Holt, K. E. (2015). Bandage: Interactive visualization of de novo genome assemblies. *Bioinformatics*, *31*(20), 3350–3352. doi: 10.1093/bioinformatics/btv383.

Key References

Antipov et al. (2015). See above.

This paper describes algorithm for hybrid (NGS data accomplished with third generation sequencing data) assembly used in SPAdes.

Antipov et al. (2016). See above.

This paper describes SPAdes modification for extraction and better assembly of plasmid data from bacterial datasets.

Antipov et al. (2019). See above.

This follow-up paper describes updates for plasmidSPAdes (Antipov et al., 2016) that allow to extract plasmids from metagenomic datasets.

Bushmanova et al. (2019). See above.

This paper describes modifications of SPAdes algorithm for RNA data assembly.

Bushmanova et al. (2020). See above.

This paper updated follow-up paper for hybrid (NGS data with third generation sequencing data) RNA assembly.

Ershov et al. (2019). See above.

This follow-up paper describes further updates to BayesHammer algorithm (Nikolenko et al., 2013) for correction of IonTorrent data.

Meleshko et al. (2019). See above.

This paper describes BiosyntheticSPAdes algorithms for assembly of complex biosynthetic gene clusters.

Nikolenko et al. (2013). See above.

This paper describes in detail the read error correction stage of SPAdes.

Nurk et al. (2013). See above.

This follow-up paper provides more details on SPAdes algorithms that are used for single-cell datasets and mini-metagenomes.

Nurk et al. (2017). See above.

This paper is devoted to SPAdes algorithm modifications for metagenomic dataset assembly.

Prjibelski et al. (2014). See above.

This paper describes an algorithm for utilization of paired-end information in SPAdes.

Vasilinetc et al. (2015). See above.

This follow-up paper updates PathExtend (Prjibelski et al., 2014) with an algorithm for mate-pair (large insert-size paired end) reads.

Internet Resources

http://cab.spbu.ru/software/spades/

Center for Algorithmic Biotechnology website. On this webpage, you can find links, manuals, benchmarks, reviews, download binaries, etc.

https://github.com/ablab/spades

A GitHub repository for SPAdes assembler. Contains manuals and allows you to download previous release versions, as well as SPAdes versions used in journal articles.