Jan 16, 2023

SAHDA: Sequence Alignment from Haplotype De Novo Assembly

DOI
dx.doi.org/10.17504/protocols.io.q26g78w58lwz/v1

Joseph C Blackwell[1], Megan Mcdonald[2]
[1]University of Warwick; [2]University of Birmingham

Joseph C Blackwell

DOI: https://dx.doi.org/10.17504/protocols.io.q26g78w58lwz/v1

External link: https://github.com/JoeBlackSci/SAHDA

Protocol Citation: Joseph C Blackwell, Megan Mcdonald 2023. SAHDA: Sequence Alignment from Haplotype De Novo Assembly. **protocols.io https://dx.doi.org/10.17504/protocols.io.q26g78w58lwz/v1**

Manuscript citation:
Scripts and basic workflow for this protocol were adapted from: Rapid Parallel Evolution of Azole Fungicide Resistance in Australian Populations of the Wheat Pathogen Zymoseptoria tritici Megan C. McDonald, Melanie Renkin, Merrin Spackman, Beverley Orchard, Daniel Croll, Peter S. Solomon, Andrew Milgate Applied and Environmental Microbiology Feb 2019, 85 (4) e01908-18; DOI: 10.1128/AEM.01908-18

Protocol status: Working
We use this protocol and it's working

Created: April 16, 2021

Last Modified: January 16, 2023

Protocol Integer ID: 49179

Keywords: Genome, Assembly, de novo assembly, haploid, genomics, snakemake, workflow, bioinformatics, sequence alignment from haplotype de novo assembly sahda, de novo genome assembly fasta, haplotype de novo assembly sahda, haploid genomes de novo, genome assembly, sequencing library, small haplotype genome, assembly blast search result, multiple alignment fasta file, assembled genome, local blast database, analyse gene, retrieved fasta sequence, sequence alignment, genome, multiple alignment fasta file for each query, sequencing, fasta sequences for each query, gene, genes of interest, query on each assembly, sahda, alignment, workflow tool

Funders Acknowledgements:

Abstract

SAHDA is a Snakemake based workflow tool designed identify and analyse genes/features of interest within small haplotype genomes. The workflow will assemble haploid genomes *de novo* from short paired-end reads, create a local BLAST database and query genes of interest. Where genes of interest are detected, a multiple alignment is performed for each gene. The tool also allows the input of intermediary files meaning that a combination of both raw sequencing libraries and assembled genomes can be investigated simultaneously.

Primary input:
- Raw paired end sequencing libraries fastq.gz

Optional intermediary inputs/outputs:
- Trimmed paired end sequencing libraries fastq.gz
- *De novo* genome assembly fasta
- Local BLAST database for each assembly
- BLAST search results for each query on each assembly
- Retrieved fasta sequences for each query on each assembly

Primary output:
- Multiple alignment fasta file for each query.

GitHub repository: https://github.com/JoeBlackSci/SAHDA

Image Attribution

Photo of *Zymoseptoria tritici* infecting wheat by Megan C. McDonald. Location: Wagga Wagga Agricultural Institue, NSW Department of Primary Industries, Wagga Wagga, NSW, Australia.

Guidelines

Naming conventions:
When importing intermediary files, they must be named appropriately in order for the workflow to identify them as files to incorporate into the workflow.

For example, these are the file and directory names required for a config.yml file containing the following specifications.

Note: **importing an assembled genome** requires the genome to be in fasta format, named 'scaffolds.fasta' and placed inside a directory named '<sample_name>_assembled', that is in turn placed inside the '02_assemble' Directory (see below).

```
# Config File

Samples:
  - EXAMPLE1
  - EXAMPLE2

Queries:
  - EXAMPLE-QUERY
```

```
# Resulting file structure

results
├── 00_append
│   ├── EXAMPLE2_R1.fastq.gz
│   └── EXAMPLE2_R2.fastq.gz
├── 01_trim
│   ├── EXAMPLE1_trimmed_1P.fastq
│   ├── EXAMPLE1_trimmed_1U.fastq
│   ├── EXAMPLE1_trimmed_2P.fastq
│   ├── EXAMPLE1_trimmed_2U.fastq
│   ├── EXAMPLE2_trimmed_1P.fastq
│   ├── EXAMPLE2_trimmed_1U.fastq
│   ├── EXAMPLE2_trimmed_2P.fastq
│   └── EXAMPLE2_trimmed_2U.fastq
├── 02_assemble
│   ├── EXAMPLE1_assembled
│   │   ├── scaffolds.fasta
│   └── EXAMPLE2_assembled
│       ├── scaffolds.fasta
├── 03_blastdb
│   ├── EXAMPLE1_blastdb
│   │   ├── EXAMPLE1_assembly.fasta
│   │   ├── EXAMPLE1_assembly.nhr
│   │   ├── EXAMPLE1_assembly.nin
│   │   └── EXAMPLE1_assembly.nsq
│   └── EXAMPLE2_blastdb
│       ├── EXAMPLE2_assembly.fasta
│       ├── EXAMPLE2_assembly.nhr
│       ├── EXAMPLE2_assembly.nin
│       └── EXAMPLE2_assembly.nsq
├── 04_search
│   └── EXAMPLE-QUERY
│       ├── EXAMPLE-QUERY_EXAMPLE1.txt
│       └── EXAMPLE-QUERY_EXAMPLE2.txt
├── 05_retrieve
│   └── EXAMPLE-QUERY
│       ├── EXAMPLE-QUERY_EXAMPLE1_retrieved.fasta
│       ├── EXAMPLE-QUERY_EXAMPLE1_retrieved.gff
│       ├── EXAMPLE-QUERY_EXAMPLE2_retrieved.fasta
│       └── EXAMPLE-QUERY_EXAMPLE2_retrieved.gff
├── 07_combine
│   └── EXAMPLE-QUERY_all.fasta
└── 08_align
```

```
    └── EXAMPLE-QUERY_aligned.fasta
```

Troubleshooting

Safety warnings

This protocol is not compatible with windows machines as SPAdes does not have a windows version. The workflow has been tested on mac OS but should also run on linux operating systems, this includes linux side loaded on windows 10 devices as explained by James P B Lloyd **here**.

Before start

Assembling genomes *de novo* takes a long time, so assembled genomes can be input as an intermediary file, allowing the workflow to investigate a combination of raw sequencing reads and assembled genomes without having to repeat any computational analysis.

If users have access to a HPC where they cannot run SAHDA directly, they may wish to assemble genomes independently and them import them into the workflow.

Overview

1 The following is an overview of the workflow. When run, the workflow will automatically take inputs and perform subsequent steps. Each step can act as an input for files and generates an intermediary output that can be extracted from the workflow. The software used in each step is linked and the inputs and outputs of each step is listed below.

1. Trimming of raw fastq files [**trimmomatic**]
   Input: Raw paired end sequencing files in fastq.gz format
   Output: Trimmed and QC controlled fastq.gz files

2. *De Novo* assembly [**SPAdes**]
   Input: Trimmed paired end and unpaired sequencing fastq.gz files
   Output: Assembled genome scaffold.fasta file

3. Create local BLAST databases for each de novo assembly [**BLAST+**]
   Input: Assembled genome scaffold.fasta file
   Output: Indexed local BLAST database for each

4. Query each assembly BLAST database with gene of interest [**BLAST+**]
   Inputs: Local BLAST database and sequences to query in .fasta format
   Output: BLAST search results

5. Extract sequence from the database [**BLASTtoGFF**]
   Input: BLAST search results and assembled genome scaffold.fasta
   Output: Separate retrieved fasta files for the best BLAST hit

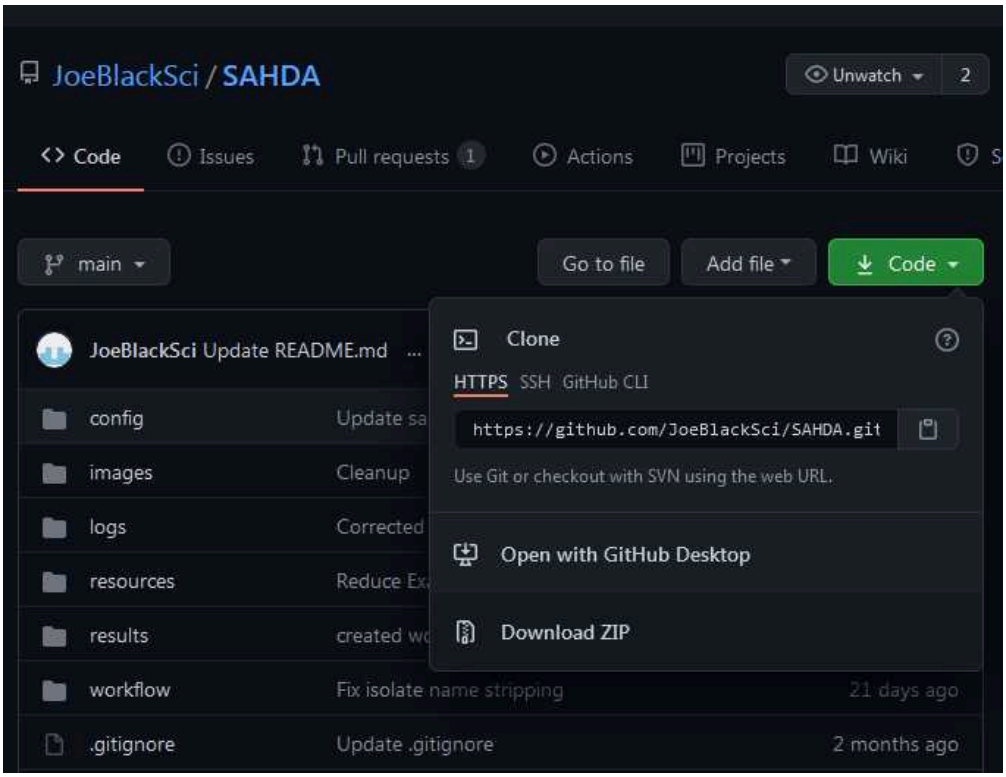
6. Combine fasta files based on query sequence
   Input: Separate fasta files
   Output: Combined fasta files

7. Perform multiple alignment for each identified gene [**MAFFT**]
   Input: Combined fasta files
   Output: Multiple DNA alignment fasta file

Installation

2 **Download SAHDA** via either of:

a. zip from the git repository: https://github.com/JoeBlackSci/SAHDA



b. git clone command on the commandline

```
cd path/to/desired/directory

git clone https://github.com/JoeBlackSci/SAHDA
```

3

After unpacking or cloning the repository, you should see the following file structure.

```
SAHDA
├── README.md                 #  Workflow users guide
├── config                    #  Config file for customising the
workflow
│   ├── config.yml            #  Workflow configuration options
│   ├── env_snakemake_mgw.yml #  Workflow conda environment
├── images                    #  Images for the README.md document
├── logs                      #  Lists of errors and standard
output from the workflow
├── resources
│   ├── adapters              #  Adapter files
│   ├── query                 #  Query sequences
│   └── reads                 #  Sequencing libraries
├── results
└── workflow
    ├── envs                  #  Specifications for software
downloads
    ├── scripts               #  Python Scripts
    └── Snakefile             #  Workflow instructions
```

4 **Install Prerequisite Programs**

1. Install Miniconda 3 using the **miniconda3 installation instructions**.

This will install the latest version of the **Conda** environment management software and it's minimal dependencies.

2. Install Snakemake using Conda.

Snakemake is a python based workflow management language for reproducible and scalable data analysis. It is recommended to recreate the snakemake environment used to develop this workflow.

To do so, navigate using the commandline, to the top level of the workflow directory (containing the README.md file, see above):

```
cd path/to/SAHDA/
```

And recreate the snakemake environment from the environment.yml "env_snakemake_mgw.yml" file in the config folder using:

```
conda env create -f ./config/env_snakemake_mgw.yml
```

Test Installation

10m

5 Before running your own data, it is recommended to run the included example workflow. This will test that the workflow is working correctly, generate the intermediary file structures, provide example files at each step and download software required for the workflow via conda.

Navigate using the commandline, to the top level of the workflow directory (containing the README.md file, see above)

```
cd path/to/SAHDA/
```

Activate the snakemake conda environment

```
conda activate env_snakemake_mgw
```

Run the example workflow

```
snakemake --cores 1 --use-conda
```

6 Running the example workflow successfully will result in the following file structure and results files:

```
SAHDA
├── config
├── images
├── logs
├── resources
│   ├── adapters
│   ├── query
│   └── reads
├── results
│   ├── 00_append
│   │   ├── EXAMPLE2_R1.fastq.gz
│   │   └── EXAMPLE2_R2.fastq.gz
│   ├── 01_trim
│   │   ├── EXAMPLE1_trimmed_1P.fastq
│   │   ├── EXAMPLE1_trimmed_1U.fastq
│   │   ├── EXAMPLE1_trimmed_2P.fastq
│   │   ├── EXAMPLE1_trimmed_2U.fastq
│   │   ├── EXAMPLE2_trimmed_1P.fastq
│   │   ├── EXAMPLE2_trimmed_1U.fastq
│   │   ├── EXAMPLE2_trimmed_2P.fastq
│   │   └── EXAMPLE2_trimmed_2U.fastq
│   ├── 02_assemble
│   │   ├── EXAMPLE1_assembled
│   │   └── EXAMPLE2_assembled
│   ├── 03_blastdb
│   │   ├── EXAMPLE1_blastdb
│   │   └── EXAMPLE2_blastdb
│   ├── 04_search
│   │   └── EXAMPLE-QUERY
│   ├── 05_retrieve
│   │   └── EXAMPLE-QUERY
│   ├── 07_combine
│   │   └── EXAMPLE-QUERY_all.fasta
│   └── 08_align
│       └── EXAMPLE-QUERY_aligned.fasta
└── workflow
```

Input Data

7 **Paired end reads**

Paired end files should be added to the `resources/reads` directory, and the sample name added to the config file described later.

This workflow will accept paired-end short read sequences with filenames of the following formats.

```
<sample_name>_<R1/R2>.fastq.gz
<sample_name>_<lane_no.>_<R1/R2>.fastq.gz

for example:
SRRXXXXXXX_R1.fastq.gz
AAAXXXX_L001_R1.fastq.gz
```

Note: The workflow currently only supports samples spread over two lanes. i.e. only L001 and L002 are currently supported.

```

8 **Adapters**

The adapter file, in `.fasta` format, should be added to the `resources/adapters` file and the full filename specified in the `config/config.yml` file described later.

9 **Target gene database**

Query genes should be added as individual `.fasta` files to the `resources/query` directory.

## Configuring the Workflow

10 **Configuration**

The workflow configuration file `config/config.yml` can be edited as a text file to specify parameters for the workflow. Including samples, query genes and trimming options. See the config.yml file for all options.

10.1 **Sample Names**

Sample names should be included without lane, direction or file extension. For example:

- The paired end library EXAMPLE1_R1.fastq.gz and EXAMPLE1_R2.fastq.gz should be included as EXAMPLE1.
- The paired end library EXAMPLE2_L001_R1.fastq.gz, EXAMPLE2_L001_R2.fastq.gz, EXAMPLE2_L002_R1.fastq.gz and EXAMPLE2_L002_R2.fastq.gz should be included as EXAMPLE2.
```

```
Samples:
  - EXAMPLE1
  - EXAMPLE2
```

note: if your filename includes a lane number but is not split over multiple lanes, then the lane number should be edited out or included in the sample name in the config file.

If you are using intermediary files as inputs, ensure that they are placed in the correct directory and follow the example naming conventions.

10.2 **Query Database**

The names of individual query sequence files should be included without file extension. For example:

- The EXAMPLE-QUERY.fasta query should be included as EXAMPLE-QUERY.

```
Queries:
  - EXAMPLE-QUERY
```

10.3 **Adapter file**

The name of the adapter file **including** file extension.

```
Adapter_file:
  - EXAMPLE-adapters.fa
```

Running the Workflow

11 **Activate the Environment**

To run the workflow, navigate using the commandline to the top level directory containing the README.md file.

```
cd /path/to/SAHDA/
```

Activate the workflow's conda environment.

```
conda activate env_snakemake_mgw
```

12 **Executing the Default Workflow**

To execute the default workflow run the following command

```
snakemake --cores all --use-conda
```

Note this will use the maximum cores available on your local machine. To specify the number of cores use --cores N where N is the maximum number of cores to be assigned.

Snakemake will automatically determine which jobs need to be run to generate all target files. Meaning that output files that are already present will not be regenerated.

If you have input intermediary files such as an already assembled genome, then SAHDA will also run on these files so long as the sample is named in the config.

After running this command, your multiple alignment files will be found in 'results/08_align/'

Optional Parameters

13 Parameters For a full list of snakemake parameters see the **snakemake documentation**.

[--use-conda] **Required** This workflow should always be run with with --use-conda flag to enable each rule to download and utilise a specific conda environment. This ensures a reproducible workflow, but also ensures that required software will not conflict.

[-j --jobs --cores] **Required** Specify the maximum number of CPU cores to use in parallel. Specify all to use all cores available.

[-n --dryrun] Dry run the Snakefile with -n prior to performing an analysis to check for potential errors.

[-p --printshellcmds] Print the shell commands to be executed to the terminal.

[-r --reason] Print snakemake's reason for executing each job.

[-R --forcerun] List the rules that should be re-run. Snakemake does not re-perform analysis if new files are added prior to a step that combines outputs from previous steps. For example if new data has been introduced prior to a multiple alignment. In these instances -R mafft-align would need to be specified. A full list of snakemake rules can be seen in the Rules List.

--list-input-changes gives a list of output files that are affected by the inclusion of new data. This can be used to automatically trigger a --forceun with the following code.

```
snakemake -cores all --forcerun $(snakemake --list-input-changes)
--use-conda
```