SPIDR Analysis Documentation

Darvesh Gorhe

October 17, 2023

Contents

1 Introduction

This document outlines the SPIDR analysis pipeline, which has three main sections: preprocessing, postprocessing, and downstream analysis. Each section covers the following:

Preprocessing:	fastq → bed
Postprocessing:	bed → bedgraph, bigwig
Downstream Analysis:	bedgraph, bigwig → tables, figures, etc.

Preprocessing and Postprocessing use Snakemake to orchestrate several steps in an efficient way. Snakemake allows each pipeline to effectively use high-performance computing resources with minimal human supervision. This implementation allows Snakemake to either submit batch jobs or run interactively.

In many cases, it is *impossible* to run the pipeline on a local machine due to the large amount of data, computational requirements, and unavailable dependencies. With respect to the dependencies, this is particularly an issue for things like `samtools`, `deeptools`, and `pysam`. These tools are often only designed and supported for Linux machines.

2 Setting Up your HPC Cluster Environment

The following steps are not strictly necessary, but they will make your life much easier. The steps below show you how to install `mamba` outside of your home directory. This is necessary because Snakemake uses the mamba dependency resolver by default and your home directory is somewhat limited in space and permissions. For more detailed steps see the steps below, otherwise continue to the next section 3.

To explicitly use `conda` instead of `mamba`, you can add the following line to your `sbatch-run.sh` and `interactive-run.sh`: `--use-conda`.

2.1 Why should I do this?

- Snakemake is very persistent in getting you to use `mamba`
- Sometimes `conda` does not work or takes an *extremely* long time to resolve dependencies, especially for larger environment definition files (e.g. `pipeline/envs/sprite.yaml`)
- `mamba` can also help down the line for testing and debugging `conda`/`mamba` environments.

2.2 Steps

1. Download Mambaforge from this page.
2. Run the installation script from the same place you downloaded it. The command should be something like this, but defer to the instructions on the page you downloaded it from:

```
bash Mambaforge-$(uname)-$(uname -m).sh
```

3. Follow the prompts on screen. Stop when you get to the part where it asks you where to install mamba. For Ginsburg users, I'd recommend `/burg/mjlab/users/<your UNI>/cli-tools`. For other HPC users, determine what a good place is for you based on your permissions and space available.
   - You may need to create the `<your UNI>` directory within `users` if it does not already exist. You can do so by running:

```
mkdir -p /burg/mjlab/projects/cli-tools/<your UNI>
```

   - Don't include the `<>` when you run the command. For example, my UNI is `dsg2157` so I would run:

```
mkdir -p /burg/mjlab/projects/cli-tools/dsg2157
```

4. Within the instructions it will ask you if you want to run `conda init`. Type 'yes' (without the single quotes) and hit enter/return.
5. To see the effects of the changes, you can either log out and log back into your cluster or you can run `source ~/.bashrc` to reload your bash profile (which has the same effect).
   - To check that `mamba` was installed correctly, run:

```
mamba --version
```

   - The output should look something like this (your version numbers may be different):

```
mamba 1.4.1
conda 23.1.0
```

3 Preprocessing

3.1 How to Run `generate-targets.smk`

1. Navigate to the `spidr` repository on your machine (i.e. Ginsburg or your institution's HPC).
2. Create a directory within the `pipeline` directory called `fastq_files`.

```
mkdir -p fastq_files
```

3. Copy or move your read 1 and read 2 fastq files into the `fastq_files` directory you just created. Make sure that the fastq files are compressed using `gzip`.
   - For larger files, consider using `pigz` which allows for parallel compression with multiple threads (`-p -1` tells `pigz` to use all available threads)

```
pigz -p -1 <read_1 file>.fastq
pigz -p -1 <read_2 file>.fastq
```

   - Compressing your `fastq` files is important because many steps in the Snakemake pipeline assume that the fastq files are compressed. As a result, these steps use programs designed to work specifically with compressed files (e.g. `zcat`)
   - Make sure your files are in the following format:

```
<experiment>_R[1|2].fastq.gz
```

   - In this case `[1|2]` means either `1` or `2`. For example, if your experiment is called `my_experiment` then your files should be named:

```
my_experiment_R1.fastq.gz
my_experiment_R2.fastq.gz
```

4. Run `python fastq2json.py -f fastq_files` from the `pipeline` directory. This will create a `samples.json` file which will be used by the pipeline to determine which fastq files to run.

```
python fastq2json.py -f fastq_files
```

5. Modify the `cluster.generate-targets.yaml` file to reflect your account ID. You can change other parameters as well, but the `account` parameter is the only one you need to change. The account parameter should be the one that you use to submit jobs to the cluster (`mjlab` for anyone in the lab using Ginsburg). To show this more concretely, here's a snippet of the `cluster.generate-targets.yaml` file:

```
__default__:
    account: "mjlab" # <-- Change this to your account ID
    time: "01:00:00"
    mem: 20g
    cpus: 1
    nodes: 1
    output: "workup/logs/cluster/{rule}.{wildcards.sample}.out"
    error: "workup/logs/cluster/{rule}.{wildcards.sample}.err"
splitfq:
    account: "mjlab" # <-- Change this to your account ID
    time: "02:00:00"
    mem: 64g
    cpus: 1
    nodes: 1
    output: "workup/logs/cluster/{rule}.{wildcards.sample}.out"
    error: "workup/logs/cluster/{rule}.{wildcards.sample}.err"
compress_fastq:
    account: "mjlab" # <-- Change this to your account ID
    time: "02:00:00"
    mem: 64g
```

```
        cpus: 8
        nodes: 1
        output: "workup/logs/cluster/{rule}.{wildcards.sample}.out"
        error: "workup/logs/cluster/{rule}.{wildcards.sample}.err"
    ...
```

6. Modify `config.generate-targets.yaml` to your liking. This config file looks like this

```
# email to which errors will be sent
email: "dsg2157@columbia.edu"

# Location of the config file for barcodeIdentification
bID: "config_6_rounds_mTOR.txt"

# Location of the samples json file produced with the fastq2json.py
   script
samples: "./samples.json"

# Output directory
output_dir: ""

# Temporary directory
temp_dir: "/burg/mjlab/projects/spidr_preprocessing_dg/scratch"

# Unique tags in the ROUND1 tags within config
conditions: ['CNTRL', 'TORIN']

# Currently "mm10" and "hg38" available
assembly: "hg38"

proportion: 0.7

min_oligos: 3

# Number of barcodes used
num_tags: "7"

# Number of chunks to split fastq
num_chunks: 10

# Format for splitting clusters
rounds_format: 'format_6_rounds_mTOR.txt'

# File for cutadapt
cutadapt_oligos: "oligo2_reverse.fasta"

# Bowtie2 Indexes
# Note: mm10 index currently not available at specified filepath
bowtie2_index:
    hg38:
       "/burg/mjlab/projects/spidr_preprocessing_dg/ncRNA_bt2/ncRNA"

# Star Indexes
# Note: mm10 index currently not available at specified filepath
star_index:
    hg38: "/burg/mjlab/projects/spidr_preprocessing_dg/hg38"
```

- The most likely things that you need to change are: `conditions`, `assembly`, `bowtie2_index`, `star_index`, `bID`, `rounds_format`, and `num_tags`.
- You may need to change `cutadapt_oligos` if you've used modified adapater sequences.

- You may want to change `num_chunks` if you have large fastq files. Generally, higher is better if your cluster has the capacity to process many jobs in parallel. Otherwise, lower is fine. 10 is a good number to start with.
- Everything else you can experiment with.
- It should be noted that `output_dir` is not well supported so it's recommended to leave it as an empty string (i.e. what's shown in the example above).

7. Check that Snakemake is installed by running the following (you should get an output like `7.25.0`)

```
snakemake --version
```

   - If it's not installed then you can install it with `mamba` (or `conda`) in your current environment with:

```
mamba install -c conda-forge snakemake
```

   - It's recommended that you install Snakemake in your base environment for convenience. This specifically helps with `sbatch` jobs since loading anaconda on SLURM systems usually defaults to loading the base environment.

8. For interactive runs, run the following command (otherwise skip to the next step):

```
bash interactive-run.sh generate
```

9. For `sbatch` runs, run the following command:

```
bash sbatch-run.sh generate
```

   - You can check the status of the jobs that Snakemake dispatches by running:

```
squeue -u <your UNI>
```

   - when you submit the `sbatch` job you'll get a job id and hopefully a corresponding output file in the format: `slurm-<job id>.out`. You enter that file and you will see the outputs of the snakemake pipeline (similar to what you would see in your terminal if you ran the pipeline interactively).

3.2 Relationship Between Files

Below is a schematic of how various files in the pipeline are related to one another. The file at the head of the arrow uses data provided by the file at the tail of the arrow. For example, `interactive-run.sh` is using information from `cluster.generate-targets.yaml`. As you can see, both are the same in terms of their relationship to one another. The only difference is how it's run.

- `interactive-run.sh` allows you to see the progress of the pipeline as it runs.
- `sbatch-run.sh` submits the pipeline to the cluster as a batch job.
- You can check the status of the pipeline on any SLURM system with `squeue -u <your UNI>`

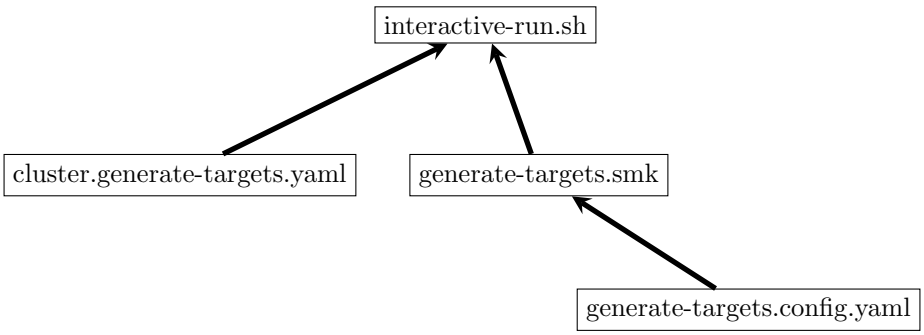


Figure 1: Overview of preprocessing configuration with the interactive pipeline.

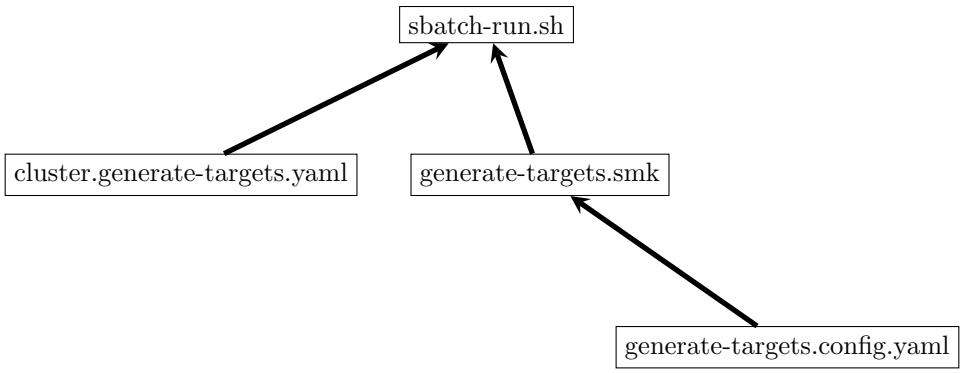
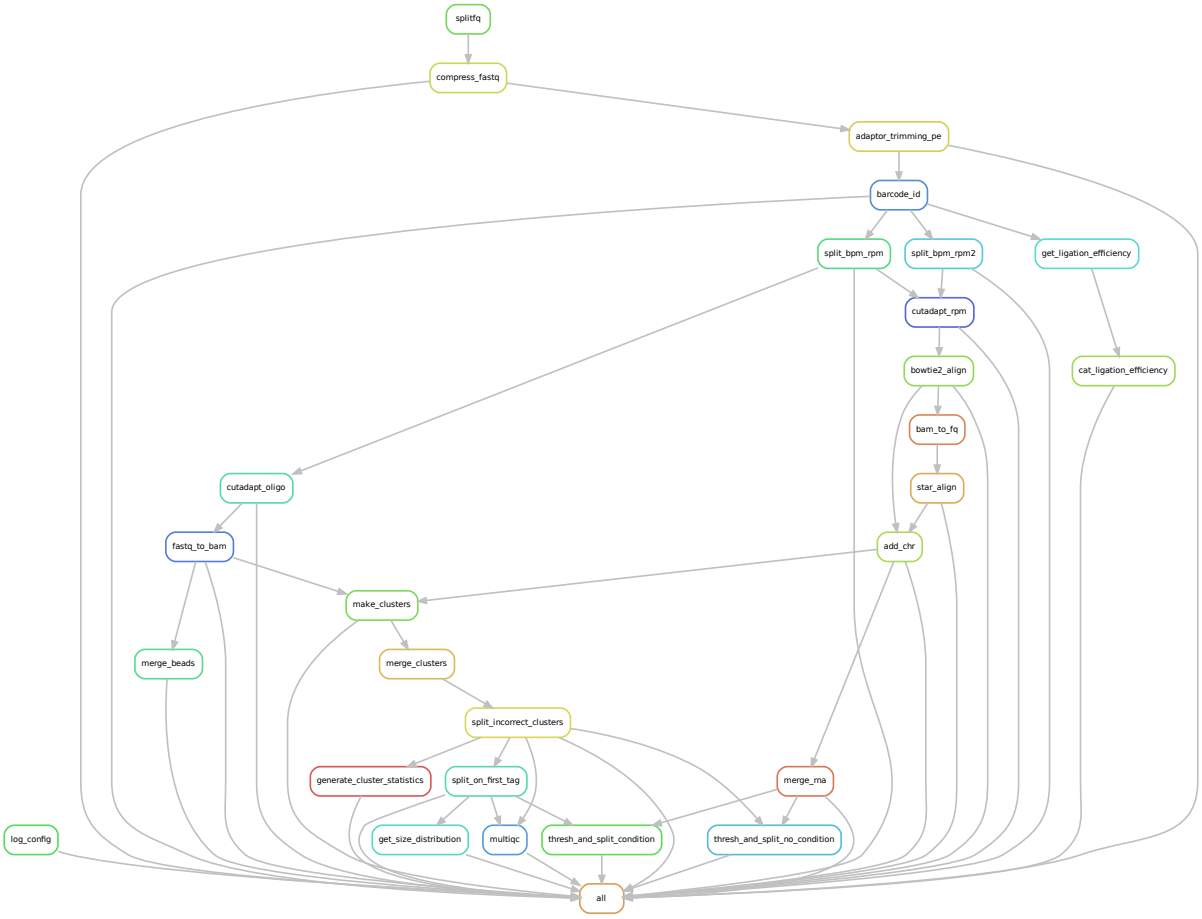


Figure 2: Overview of preprocessing configuration with the sbatch pipeline.

3.3 Details of `generate-targets.smk`

Rules are chained together automatically based on the name of input and output file specified within each rule. Snakemake will create a directed acyclic graph (DAG) upon launching the pipeline. A DAG is a data structure which tells Snakemake the dependencies between rules. This allows Snakemake to determine in what order to execute the rules. Snakemake will then run the rules in the correct order and in parallel when possible. Below is an example of a DAG for the preprocessing pipeline with `num_chunks`= 2. As the number of chunks becomes larger, the middle of the graph becomes correspondingly wider.



Each node corresponds to a different rule. To see the contents of each rule you can view the `generate-targets.smk` file. The arrows represent the flow of data and the dependencies between rules. A rule will not run until all of its dependencies have successfully completed.

4 Postprocessing Pipeline

I'm not sure how the post-processing pipeline works to be completely honest, so I'm not comfortable writing documentation for it.

5 Downstream Analysis (Ahmed)

Not sure where Ahmed's code and figures are located.

6 Downstream Analysis (Darvesh)

Below is a comparison of the ENCODE data to the outputs of SPIDR. Specifically, we compare the transcript proportion as given by the supplementary data of ENCODE along with the ENCODE raw data processed through the same pipeline as the SPIDR data.

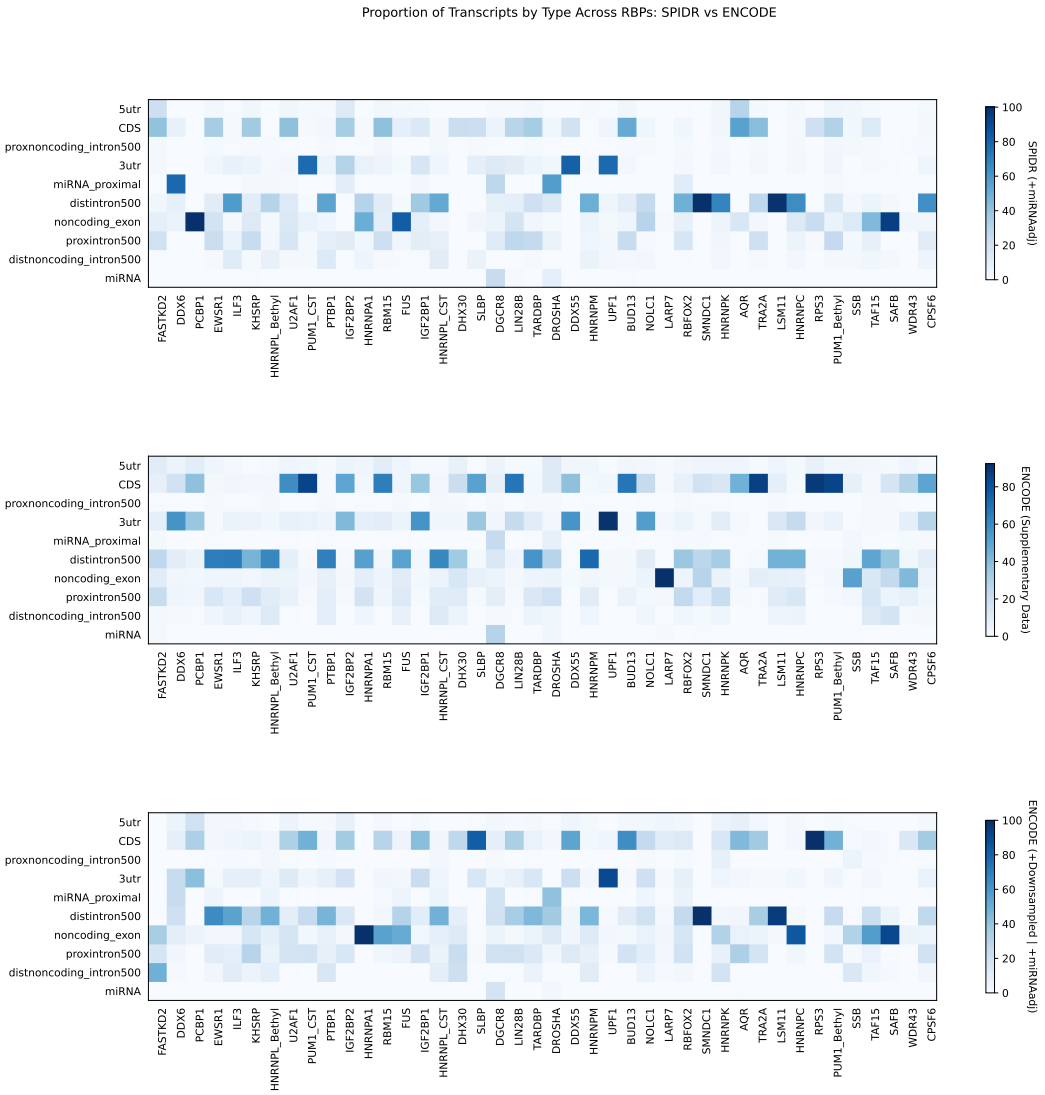


Figure 3: Comparing ENCODE to SPIDR

We also compared the distribution of imputed transcript proportions versus the true transcript proportions. Essentially, we wanted to ensure that our observed transcript proportions were significantly different from those that we would expect by chance. Indeed, this was the case.

7 Glossary

- **fastq**: A file format for storing a biological sequence and its corresponding quality scores.
- **bed**: A file format for storing genomic intervals.
- **bedgraph**: A file format for storing genomic intervals and their corresponding values.
- **bowtie2**: A short read aligner.
- **STAR**: A short read aligner.
- **Snakemake**: A workflow management system.
- **DAG**: A data structure which tells Snakemake the dependencies between rules so Snakemake can determine how to optimally schedule rules on a system.
- **rule**: A step in the pipeline.
- **cluster.yaml**: A file that configures the `sbatch` jobs. It sets things like the account ID so jobs actually get queued, the memory per job, the number of cpus per job, the maximum time a job should run, etc.
- **config.yaml**: A file that configures the pipeline itself. This has parameters like the filepaths for indicies for the bowtie2 and STAR aligners.
- **Snakefile**: The file that actually runs the pipeline and contains all the rules. The rules are the individual steps that are run in the pipeline.
- **rulegraph**: A visual representation of the DAG.
- **mamba**: A package manager for conda packages.
- **conda**: A package manager for python packages.
- **conda-forge**: A repository of conda packages.
- **base environment**: The default environment that conda uses.
- **environment**: A collection of packages that can be used together.
- **environment.yml**: A file that specifies the packages that should be installed in an environment.
- **mambaforge**: A version of conda that uses mamba as its dependency resolver.
- **dependency resolver**: A tool that determines which packages need to be installed in order to satisfy the dependencies of a package.

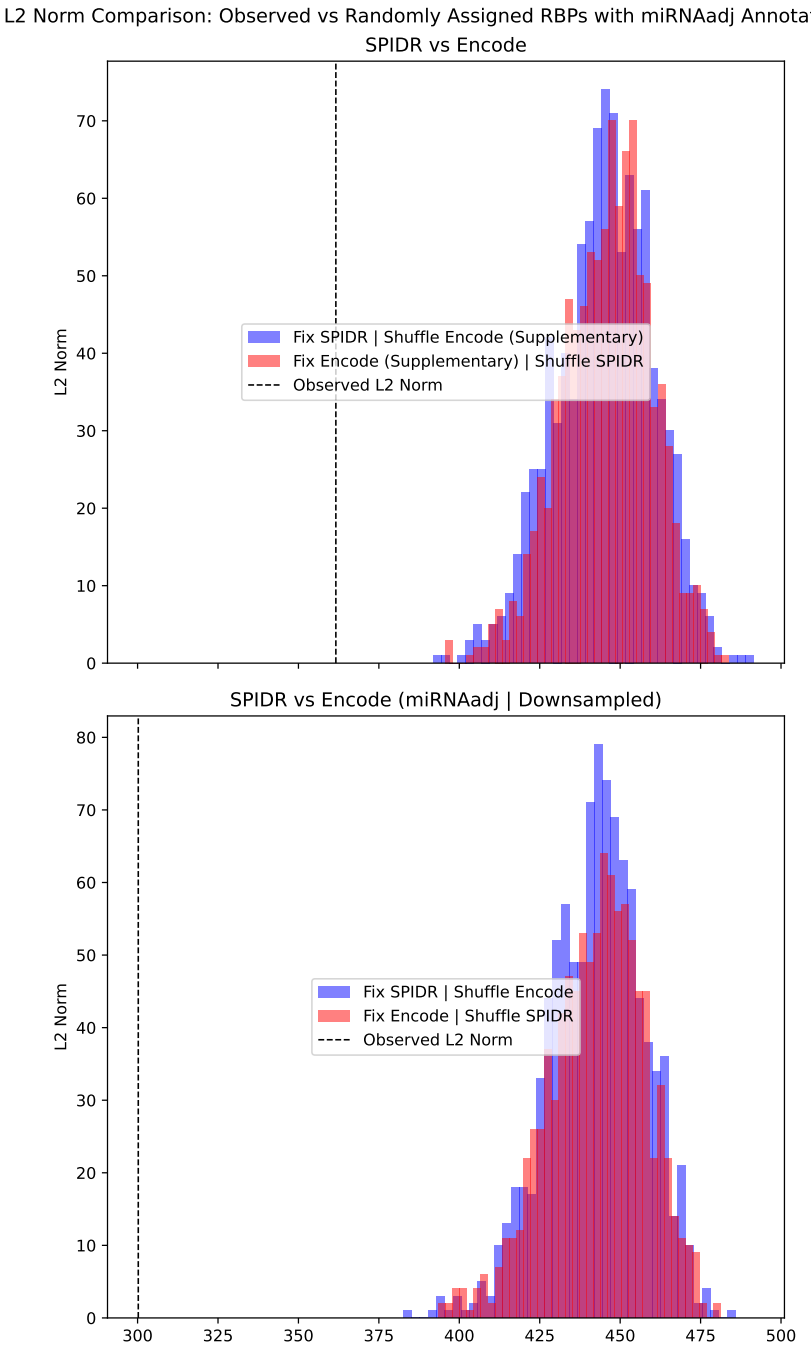


Figure 4: Comparing ENCODE to SPIDR