SPIDR Analysis Documentation

Darvesh Gorhe, Ahmed Abdou, Erica Wolin

October 3, 2023

Contents

1 Introduction

This document will outline the analysis pipeline for SPIDR. The pipeline is divided into three main sections: preprocessing, postprocessing, and downstream analysis. The preprocessing section will outline the steps taken to go from fastq files → bed files. The post-pipeline section will outline the steps taken to prepare the data from bed files to bedgraph files. The downstream analysis will outline the steps taken to analyze the data from bedgraph files to the final results.

The preprocessing and the postprocessing pipeline use Snakemake to orchestrate several steps in an efficient way using `sbatch` jobs. These `sbatch` jobs are ways to run individual sections of the pipeline on high-performance computing clusters.

2 Preprocessing

2.1 Overview

Above we see an overview of the primary files involved in configuration of the preprocessing pipeline. `run_pipeline.sh` is the shell file that actually gets called to run the pipeline. `cluster.yaml` is the 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` is the file that configures the pipeline itself. This has parameters like the filepaths for indicies for the bowtie2 and STAR aligners. `Snakefile` is the file that actually runs the pipeline and contains all the rules. The rules are the individual steps that are run in the pipeline.

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 so Snakemake can determine how to optimally schedule rules on a system. 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.

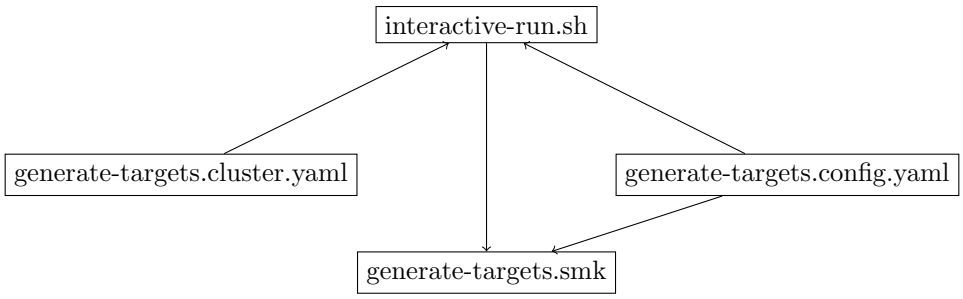

Figure 1: Overview of preprocessing configuration with the interactive pipeline.

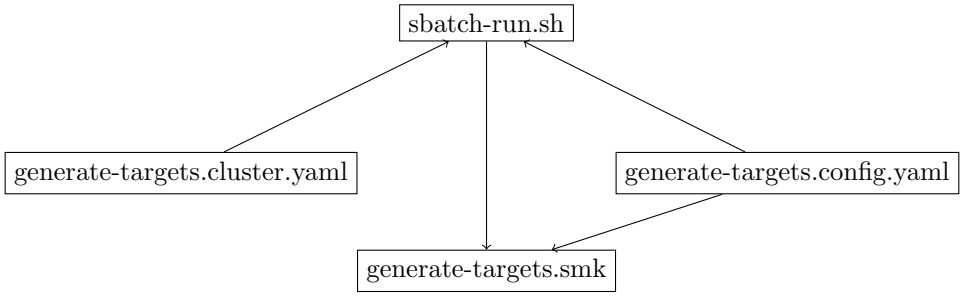

Figure 2: Overview of preprocessing configuration with the sbatch pipeline.

2.1.1 Relationship Between Files

As you can see, both are the same in terms of their relationship to one another. The only difference is their entry point. `interactive-run.sh` allows you to see the progress of the pipeline as it runs. `sbatch-run.sh` submits the pipeline to the cluster and you can check the status of the pipeline with `squeue -u <your_username>`.

2.2 `generate-targets.smk`

Each color corresponds to a different rule. To see the contents of each rule you can view the `Snakefile`. 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.

2.3 Running the Pipeline

1. Clone the repository from GitHub onto the cluster. (`git clone https://github.com/mjlab-Columbia/spidr.git`)
2. Change directories into the pipeline directory (`cd pipeline`).
3. Modify the `cluster.yaml` file to reflect your account ID, email, and other parameters. The email written in the `cluster.yaml` will be the one which error emails are sent to. The account ID should be the one that you use to submit jobs to the cluster (`mjlab` for anyone in the lab).
4. Modify the `config.yaml` to specify the locations of the indices you want to use for alignment (bowtie2 and STAR). Set any other parameters as you would like.
5. (Optional) Install mamba outside of your home directory and change your `$HOME/.bashrc` file to include the path to mamba. This will allow you to use mamba to install the necessary conda packages. This is necessary because Snakemake uses the mamba dependency resolver by default and your home directory is somewhat limited in space and permissions. For more detailed steps see the steps below, otherwise continue.
    - Download mambaforge:
      curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-($uname$)−(uname -m).sh"
    - Run the installation script from the same place you downloaded it:
      bash Mambaforge-($uname$)−(uname -m).sh
    - Follow the prompts on screen and when asked where to install mamba, specify a location outside of your home directory. I chose `/burg/mjlab/users/dsg2157/cli-tools`

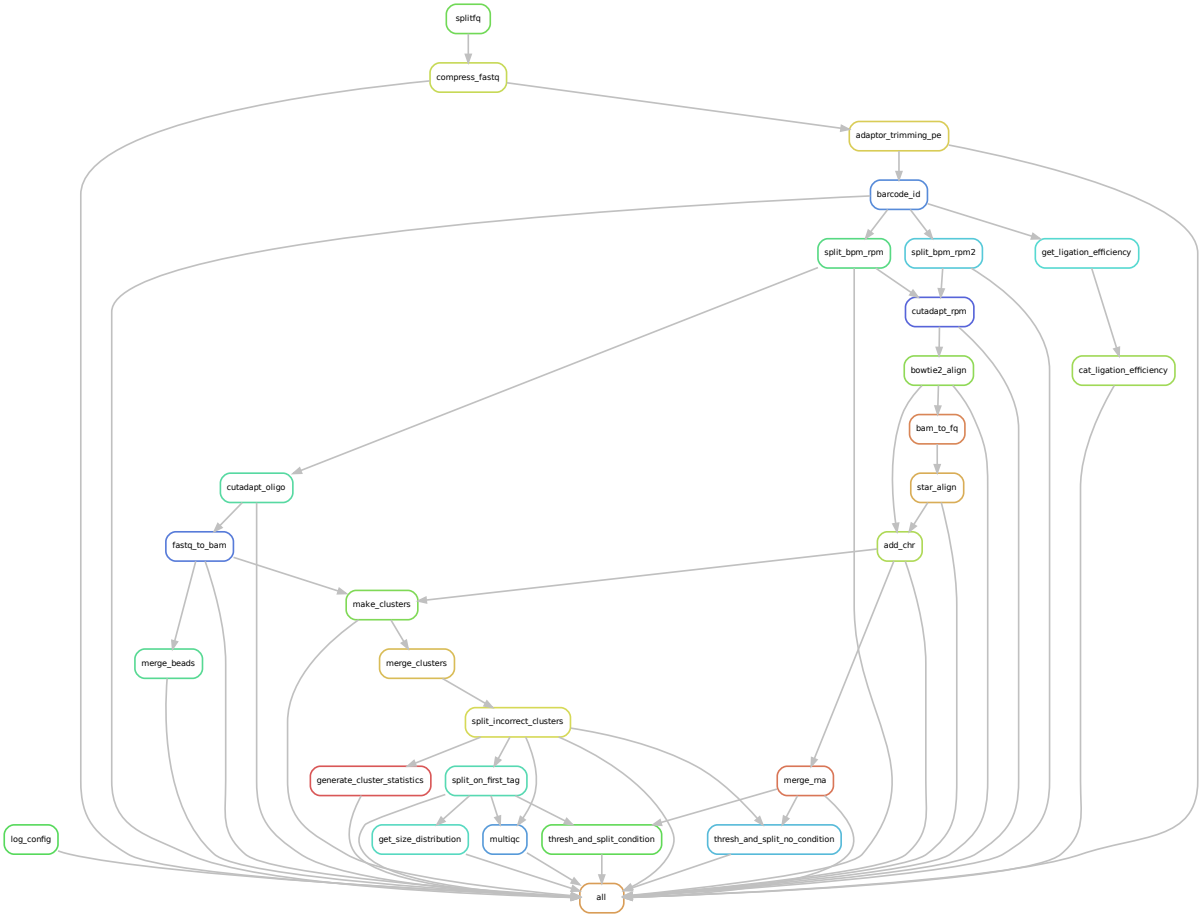

Figure 3: Example of directed acyclic graph (DAG) for preprocessing pipeline with `num_chunks`= 2

- The basic pattern is: `/burg/mjlab/users/<your_username>/cli-tools`. You may need to create the `your_username` directory within `users` if it does not already exist.
- Edit your `$HOME/.bashrc` file to include the following line:
  `export PATH=/burg/mjlab/users/cli-tools/mamba/bin:$PATH`

6. Install `snakemake` in your base environment via `mamba install -c conda-forge snakemake`. This will allow you to run the pipeline.
7. Create a folder called `fastq_files` and place gzipped fastq files in there.
8. 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.
9. For testing purposes, change `num_chunks` to 2 in order to test the pipeline more quickly.
10. Finally run `bash run_pipeline.sh` from the `pipeline` directory. This will submit the pipeline to the cluster.
11. You can check the status of your jobs with `squeue -u <your_username>`. You can also generate your own DAG of the pipeline by running `bash dag.sh`.

3 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.

4 Downstream Analysis (Ahmed)

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

5 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.

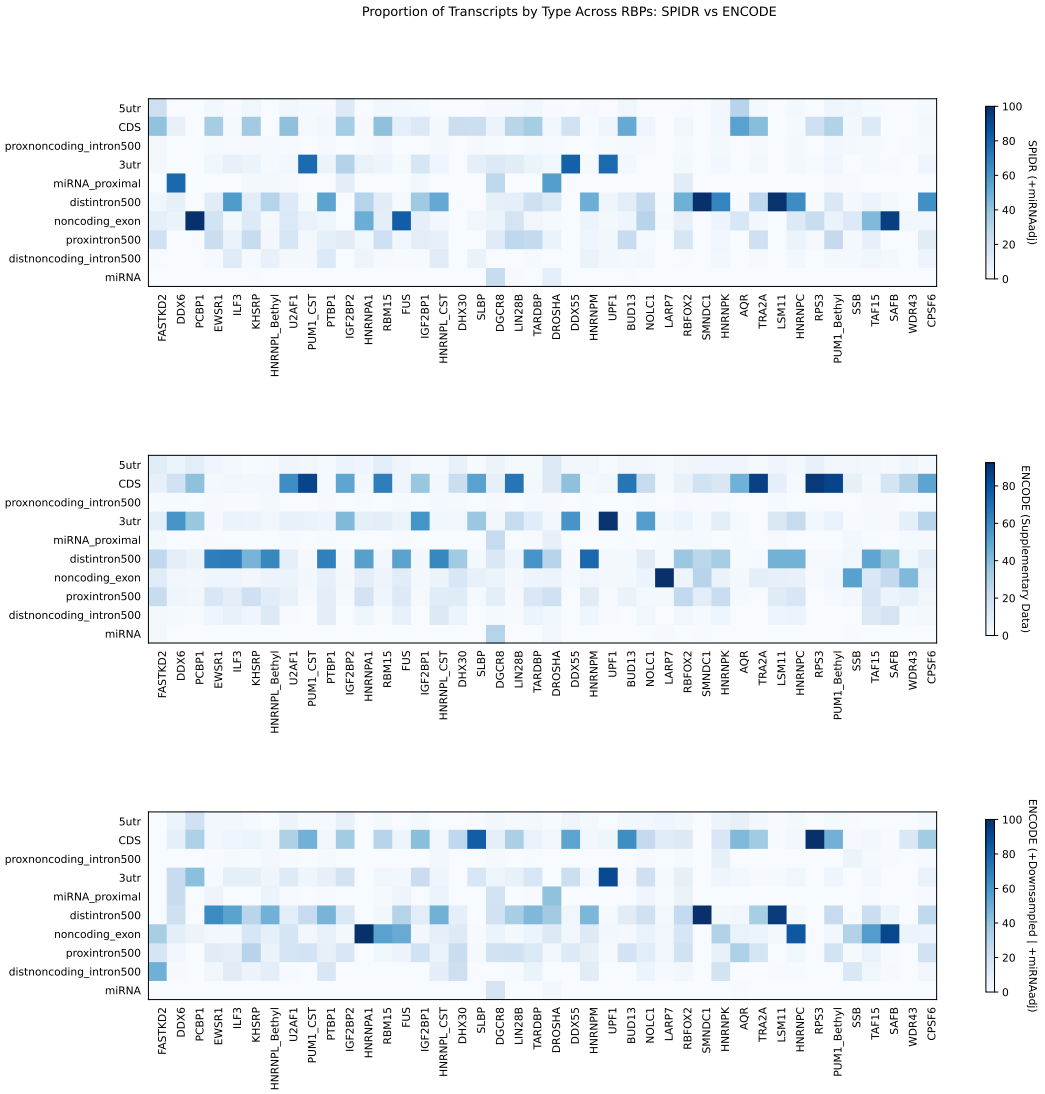

Figure 4: Comparing ENCODE to SPIDR

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.

6 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.

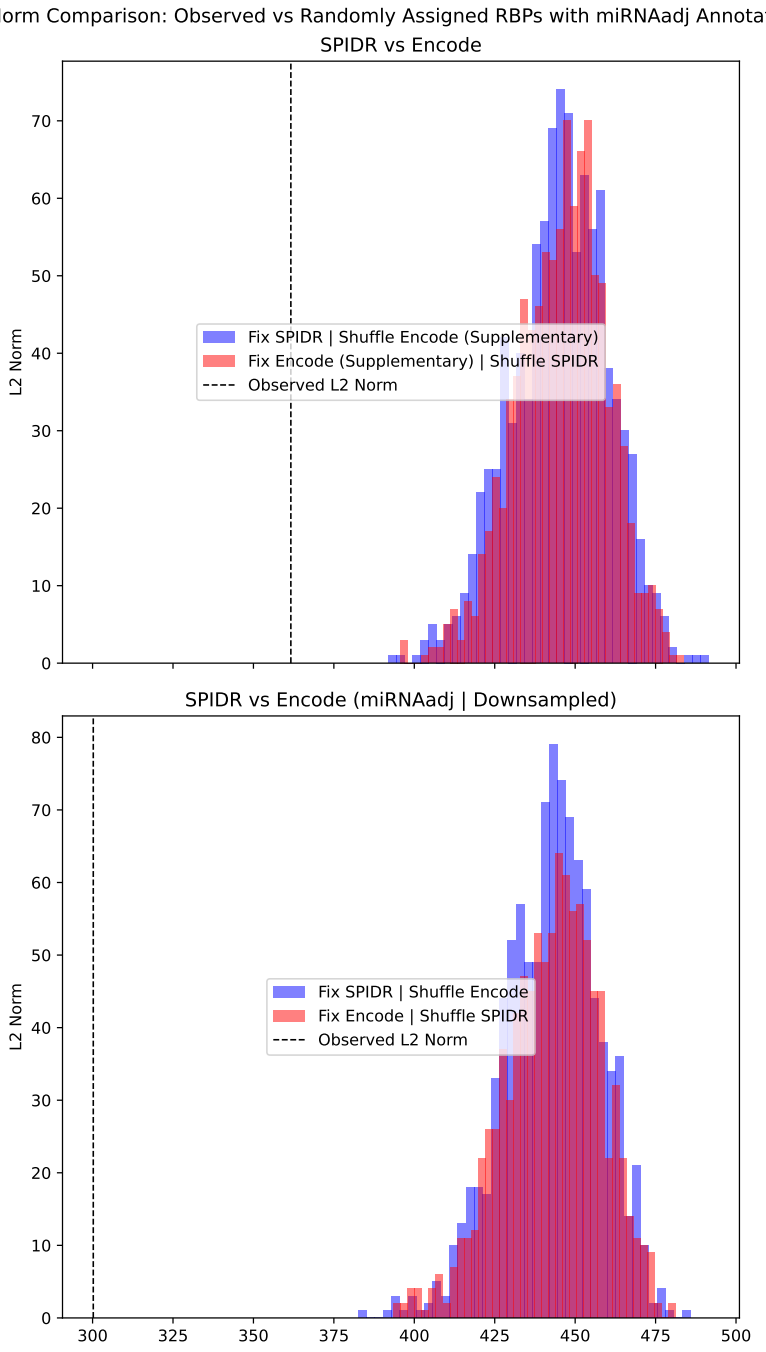

Figure 5: Comparing ENCODE to SPIDR