PETRI-seq Detailed Computational Pipeline

Related to "Prokaryotic single-cell RNA sequencing by in situ combinatorial indexing" and "Identification and genetic dissection of convergent persister cell states"
Last Updated November 11, 2024 by Sydney B. Blattman

Implementation

- To use our pipeline for generating a BC x Gene/Operon count matrix from fastq files, download the PETRI-seq scripts v2 folder.
- Open README.txt, and follow the instructions to run the code.

Details of our Pipeline

- After sequencing and Illumina demultiplexing (bcl2fastq), the result may be 2 merged fastq.gz files (reads 1 and 2) or 8 unmerged fastq.gz files for lanes 1-4 and reads 1-2. Our pipeline is flexible for use with any number of input files. If lanes are not merged, the pipeline can run more quickly as steps will be done in parallel. As described in the README, our scripts can be used on merged lanes by setting n_lanes = 1.
- Our pipeline begins with the script sc_pipeline_15_generic.py. As described in the README, create a folder with a sample name (e.g. "random20000") and transfer fastq.gz files to this folder. File names should be in the form {sample name}_{S#}_L00*_R1_001.fastq.gz. For example, create the folder "random20000" and place in it the files "random20000_S1_L001_R1_001.fastq.gz" and "random20000_S1_L001_R2_001.fastq.gz" (etc. for additional lanes, as we have already done in the demo folder).
- sc_pipeline_15_generic.py implements a number of steps:
  1. Run fastqc on all fastq files. Results will be in the folder with sample name (e.g. directory named "random20000").
  2. Run cutadapt to remove poor quality reads.
  3. Run PEAR (Paired-End reAd mergeR) to assemble overlapping reads 1 and 2. Then, assembled reads are split back into non-overlapping reads 1 and 2. Assembled contigs that are too short (<75 bases) are removed.
  4. Extract UMI sequences using umi_tools extract. This generates a fastq file with the first seven bases of R1 (the UMI) moved to the sequence name.
  5. Demultiplex reads by barcode 3 sequence. This generates 96 * 2 fastq files per lane.
  6. Merge lanes. This consolidates the files for all lanes, leaving just 96 * 2 fastq files for all lanes.
  7. Demultiplex reads by barcode 2 sequence.
  8. Demultiplex reads by barcode 1 sequence.
  9. Final outputs are:
     - Folder called {sample name}_bc1 which contains individual fastq files for each 3 barcode combination (BC).
     - {sample name}_bc1_ReadsPerBC.eps which is a histogram of reads per BC.
     - {sample name}_bc1_kneePlot.eps which is the knee plot of reads per BC.
     - {sample name}_bc1_cumulative_frequency_table.txt is a tab-delimited table showing the BCs in descending order of number of reads and the number of reads corresponding ("count" column).

  - The knee plot and histogram can be used to determine the number of BCs (n_BCs) to use in the next step.

- Next, pipeline_v2_generic.sh implements the following steps:
  1. Remove BCs with few reads (below the threshold set by n_BCs) and rename the folder {sample name}_bc1 as {sample name}_selected_cells (python script: remove_cells_v1.py)
     - Output folder {sample name}_selected_cells contains the fastq files of selected BCs
     - Output file {sample name}_selected_cumulative_frequency_table.txt is a trimmed version of {sample name}_bc1_cumulative_frequency_table.txt containing only the selected BCs.
  2. Trim read 2 sequences that include parts of barcode 1 or linker with cutadapt. These are from fragments where tagmentation occurred very close or within the barcode sequence. These should be handled during the paired-end merge, so this step may be redundant. If the remaining read 2 sequence is less than 17 bp, then the read is omitted because it cannot be aligned reliably. (python script: trim_R2_v4.py)
  3. Using cutadapt, trim read 2 sequences that include the reverse complement of the barcode or linker sequences. (script: hairpin_2nd_trim.py)
  4. Align read 2 to the fasta file using bwa. (python script: align_HiSeq_after_hairpin_trim_v2.py)
  5. Annotate aligned bam files with feature names (operon, gene) using feature counts and identify groups of reads with likely the same UMI (python script: featureCounts_directional_5.py)
     - This script first takes the sam output from bwa and removes the XT tag because this interferes with feature counts. Then, it converts sam to bam. Then, it runs feature counts, which generates annotated bam files. Finally, umi_tools group identifies reads corresponding to the same feature with the same or similar UMI sequence. If similar UMI sequences are likely actually the same, then those errors are edited and re-annotated as the same UMI.
  6. Collapse reads by UMI sequence and feature annotation. Generate a single file with the UMI sequence, feature annotation, and number of reads (for that feature:UMI combination) for all BCs. Final output is a file with the suffix '_filtered_mapped_UMIs.txt'. (python script: sc_sam_processor_11_generic.py)
  7. Finally, make a matrix of UMIs per feature by BC. This uses the '_filtered_mapped_UMIs.txt' and reformats it into a count matrix. Final output is a file with the suffix 'mixed_species_gene_matrix.txt'. (python script: make_matrix_mixed_species.py)