slide-seq-pipeline

Release 0.1

Hirak

May 30, 2022

CONTENTS

slide-seq-pipeline This pipeline replicates most of the stuff provided in official slide-seq except the dependence to the google sheet and assumed internal file/folder location. We lack some of the generalization as these scripts are meant to run on a local machine. I would highly recommend installing the actual package since some of the functions are directly used in this repo.

Check out the usage section for further information, including how to installation the project.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Pipeline

1.1.1 Pre-requisite softwares

To run the pipeline successfully one needs to install/download several third-party tools.

Tools

- Picard tools
- Dropseq tools
- STAR

Before going into the installation procedure. We should go over the actual input format. The input folder contains the raw sequencing reads.

The operation starts from the `BASECALLS_DIR`, that contains the actual intensities and the `bcl` files. For example the basic structure of such file will be

```
slide-seq-pipeline git:(main) ls /home/hsarkar/code/slide-seq-data/P25255/220329_A00689_
↪0513_BHYK23DRXY/
Config            InterOp                Recipe           RunInfo.xml       ␣
↪SequenceComplete.txt
CopyComplete.txt  Logs                   RTA3.cfg         runParameters.xml  Thumbnail_
↪Images
Data              P25255_SampleSheet.txt  RTAComplete.txt  RunParameters.xml
```

The `bcl` files are present in `Data/Intensities/BaseCalls/`. The `RunInfo.xml` used to be the file that has the information that is to be parsed before writing down the information.

Use the existing package so that we can import existing structures from the `slideseq-tools`.

Install existing `slideseq-tools` package

```
>git clone https://github.com/MacoskoLab/slideseq-tools.git
```

Create a spread-sheet that contains some of the following fields that we need to feel. An example spread-sheet is provided in *spreadsheet* folder of the repo.

```
Index(['library', 'date', 'flowcell', 'run_name', 'bclpath', 'lane',
'sample_barcode', 'bead_structure', 'reference', 'run_barcodematching',
'locus_function_list', 'start_sequence', 'base_quality',
'min_transcripts_per_cell', 'email', 'puckcaller_path', 'bead_type',
```

(continues on next page)

(continued from previous page)

```
'gen_read1_plot', 'gen_downsampling'],
dtype='object')
```

Desctiption

library: Name of the library (Each puck will have a specic name) puckcaller_path: A location containing the files from the image analysis pipeline, `BeasBarcodes.txt` and `BeadLocations.txt` run_name: Depends on how many different runs we are having BCLPath: Actual location to the BCL files sample_barcod: It's a 8bp barcode that can be obtained from `RunInfo.xml` file with the column name `index` (I am not sure about this yet) reference: The actual reference file I used `GRCh38.fasta` bead_structure: Determines the actual length of the cell barcode and the corresponding UMI

Cosntruction

The construction of the dataframe with the help of the `RunInfo.xml` is depicted in the next embedded notebook.

1.2 Pipeline through notebooks

1.2.1 Preprocess output directory

[1]:
```
import sys
print(sys.version)
```

```
3.7.10 (default, Feb 26 2021, 18:47:35)
[GCC 7.3.0]
```

1.2.2 Import the needed packages

[14]:
```
from xml.etree import ElementTree as et
from dataclasses import dataclass
from pathlib import Path
from openpyxl import load_workbook
import pandas as pd
import sys
import os
import csv
from typing import Dict, List, Tuple

sys.path.append('/home/hsarkar/code/slideseq-tools/src/')


from slideseq.metadata import Manifest
from slideseq.metadata import Manifest, split_sample_lanes, validate_run_df
from slideseq.config import get_config
import slideseq.util.constants as constants
from slideseq.util import run_command

import logging
log = logging.getLogger(__name__)
```

[4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Classes

We followed the same structure created for the original slide-seqtools repository. The classes and the containing datastructure are repeatedly used for parsing the excel file containing information about the experiment. The main goal of the associated dataframe is to iterate over all the directories within different libraries to create inermediate files within the output directory.

[4]:
```
def get_flowcell(run_info: et.ElementTree) -> str:
    """
    Get the flowcell name from RunInfo.xml. This should be more reliable than
    trying to parse the run directory, which might have been renamed.

    :param run_info: ElementTree representing RunInfo.xml
    :return: The flowcell name for this run
    """
    flowcell = run_info.find("./Run/Flowcell").text
    return flowcell


def get_read_structure(run_info: et.ElementTree) -> str:
    """
    Get read structure from RunInfo.xml. Assumes one index sequence only,
    will warn if two are present and ignore the second.

    :param run_info: ElementTree representing RunInfo.xml
    :return: Formatting string representing the read structure
    """
    read_elems = run_info.findall("./Run/Reads/Read[@NumCycles][@Number]")
    read_elems.sort(key=lambda el: int(el.get("Number")))

    if len(read_elems) == 4:
        # two index reads. We will just ignore the second index
        log.warning(
            "This sequencing run has two index reads, we are ignoring the second one"
        )
        return "{}T{}B{}S{}T".format(*(el.get("NumCycles") for el in read_elems))
    elif len(read_elems) != 3:
        raise ValueError(f"Expected three reads, got {len(read_elems)}")

    return "{}T{}B{}T".format(*(el.get("NumCycles") for el in read_elems))

def get_lanes(run_info: et.ElementTree) -> range:
    """
    Return the lanes for this run, as a range
    :param run_info: ElementTree representing RunInfo.xml
    :return: range object for the lanes of the run
    """
    lane_count = int(run_info.find("./Run/FlowcellLayout[@LaneCount]").get("LaneCount"))
    return range(1, lane_count + 1)
```

(continues on next page)

(continued from previous page)

```
@dataclass
class RunInfo:
    """A dataclass to represent RunInfo.xml for a sequencing run (a.k.a. flowcell)"""

    run_dir: Path
    flowcell: str
    lanes: range
    read_structure: str

    @property
    def demux_log(self):
        return f"demultiplex.{self.flowcell}.L00$TASK_ID.log"

    @property
    def basecall_dir(self):
        return self.run_dir / "Data" / "Intensities" / "BaseCalls"

    def alignment_log(self, lane: int):
        return f"alignment.{self.flowcell}.L{lane:03d}.$TASK_ID.log"


def get_run_info(run_dir: Path) -> RunInfo:
    # open the RunInfo.xml file and parse it with element tree
    with (run_dir / "RunInfo.xml").open() as f:
        run_info = et.parse(f)

    return RunInfo(
        run_dir,
        get_flowcell(run_info),
        get_lanes(run_info),
        get_read_structure(run_info),
    )
```

[5]:
```
run_info = get_run_info(
    Path('/home/hsarkar/code/slide-seq-data/P25255/220329_A00689_0513_BHYK23DRXY')
)
```

Run infomation shows that there is one flow-cell and the files are present in the `run_dir`. A library contains two lanes.

[6]:
```
run_info
```

[6]:
```
RunInfo(run_dir=PosixPath('/home/hsarkar/code/slide-seq-data/P25255/220329_A00689_0513_
↪BHYK23DRXY'), flowcell='HYK23DRXY', lanes=range(1, 3), read_structure='42T8B41T')
```

We now construct the required sheet for running further steps in slide-seq pipeline

[7]:
```
ws = load_workbook('/home/hsarkar/notebooks/slide_seq_analysis/2023/example_metadata.xlsx
↪').active
from itertools import islice
data = ws.values
cols = next(data)
```

(continues on next page)

(continued from previous page)

```
data = list(data)
#idx = [r[0] for r in data]
#data = (islice(r, 1, None) for r in data)
df = pd.DataFrame(data, columns=cols)
run_df = df[constants.METADATA_COLS]
run_df.columns = [c.lower() for c in constants.METADATA_COLS]
import numpy as np
run_df = run_df.fillna(value=0)
run_df = run_df.astype(constants.METADATA_TYPES)
```

[8]:
```
run_df
```

[8]:

```
       library        date    flowcell  run_name  \
0  P25255_1001  2022-28-03  Kharchenko         1
1  P25255_1002  2022-28-03  Kharchenko         1
2  P25255_1003  2022-28-03  Kharchenko         1
3  P25255_1004  2022-28-03  Kharchenko         1
4  P25255_1001  2022-28-03  Kharchenko         1
5  P25255_1002  2022-28-03  Kharchenko         1
6  P25255_1003  2022-28-03  Kharchenko         1
7  P25255_1004  2022-28-03  Kharchenko         1

                                          bclpath  lane sample_barcode  \
0  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       TAAGGCGA
1  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       CGTACTAG
2  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       AGGCAGAA
3  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       TCCTGAGC
4  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       TAAGGCGA
5  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       CGTACTAG
6  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       AGGCAGAA
7  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       TCCTGAGC

  bead_structure                                      reference  \
0  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
1  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
2  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
3  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
4  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
5  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
6  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
7  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...

   run_barcodematching  locus_function_list  start_sequence  base_quality  \
0                False                    0               0             0
1                False                    0               0             0
2                False                    0               0             0
3                False                    0               0             0
4                False                    0               0             0
5                False                    0               0             0
6                False                    0               0             0
7                False                    0               0             0
```

(continues on next page)

(continued from previous page)

```
   min_transcripts_per_cell                    email  \
0                         0  hiraksarkar.cs@gmail.com
1                         0  hiraksarkar.cs@gmail.com
2                         0  hiraksarkar.cs@gmail.com
3                         0  hiraksarkar.cs@gmail.com
4                         0  hiraksarkar.cs@gmail.com
5                         0  hiraksarkar.cs@gmail.com
6                         0  hiraksarkar.cs@gmail.com
7                         0  hiraksarkar.cs@gmail.com

                                  puckcaller_path  bead_type  \
0  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
1  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
2  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
3  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
4  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
5  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
6  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
7  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0

   gen_read1_plot  gen_downsampling
0           False             False
1           False             False
2           False             False
3           False             False
4           False             False
5           False             False
6           False             False
7           False             False
```

The above data-frame are manually curated. One needs to check with the actual file-structure before creating such a worksheet. Traditionally, the researchers at Broad work with google sheets and fetch data with the sheet id.

Now we would describe the generation process step by step.

Step 1 (Preparing for demultiplexing)

Create folders and files for carrying out demultilexing

[9]:
```
from slideseq.metadata import Manifest
from typing import Dict, List, Tuple
import logging
log = logging.getLogger(__name__)
import csv

def gen_barcode_file(manifest: Manifest, flowcell: str, lane: int, output_file: Path):
    with output_file.open("w") as out:
        wtr = csv.writer(out, delimiter="\t")
        wtr.writerow(("barcode_sequence_1", "library_name", "barcode_name"))

        for library in manifest.libraries:
            if (flowcell, lane) in library.samples:
```

(continues on next page)

(continued from previous page)

```
                for barcode in library.samples[flowcell, lane]:
                    # we don't write out barcode_name but the column is required
                    wtr.writerow((barcode, library.name, ""))


def gen_library_params(manifest: Manifest, flowcell: str, lane: int, output_file: Path):
    with output_file.open("w") as out:
        wtr = csv.writer(out, delimiter="\t")
        wtr.writerow(("OUTPUT", "SAMPLE_ALIAS", "LIBRARY_NAME", "BARCODE_1"))

        for sample in manifest.samples:
            if sample.flowcell == flowcell and sample.lane == lane:
                # output the uBAM directly to library directory
                sample.lane_dir.mkdir(exist_ok=True, parents=True)

                for barcode, output_ubam in zip(sample.barcodes, sample.barcode_ubams):
                    wtr.writerow((output_ubam, sample.name, sample.name, barcode))


def prepare_demux(run_info_list: List[RunInfo], manifest: Manifest):
    """create a bunch of directories, and write some input files for picard"""
    # Create directories
    log.info(
        f"Creating directories in {manifest.workflow_dir} and {manifest.library_dir}"
    )

    for run_info in run_info_list:
        for lane in run_info.lanes:
            output_lane_dir = manifest.workflow_dir / run_info.flowcell / f"L{lane:03d}"

            output_lane_dir.mkdir(exist_ok=True, parents=True)
            (output_lane_dir / "barcodes").mkdir(exist_ok=True)

            # Generate barcode_params.txt that is needed by ExtractIlluminaBarcodes
            gen_barcode_file(
                manifest,
                run_info.flowcell,
                lane,
                output_lane_dir / "barcode_params.txt",
            )

            # Generate library_params that is needed by IlluminaBasecallsToSam
            gen_library_params(
                manifest,
                run_info.flowcell,
                lane,
                output_lane_dir / "library_params.txt",
            )


def validate_demux(manifest: Manifest):
    """verify that `prepare_demux` was run previously"""
```

(continues on next page)

(continued from previous page)

```
    if not manifest.workflow_dir.exists():
        log.error(f"{manifest.workflow_dir} does not exist")
        return False

    for flowcell_dir in manifest.flowcell_dirs:
        run_info = get_run_info(flowcell_dir)

        # Create directories
        log.info(f"Checking directories in {manifest.workflow_dir / run_info.flowcell}")
        for lane in run_info.lanes:
            output_lane_dir = manifest.workflow_dir / run_info.flowcell / f"L{lane:03d}"

            for p in (
                output_lane_dir,
                output_lane_dir / "barcodes",
                output_lane_dir / "barcode_params.txt",
                output_lane_dir / "library_params.txt",
            ):
                if not p.exists():
                    log.error(f"{p} does not exist, demux looks incomplete")
                    return False

    return True


def validate_alignment(manifest: Manifest, n_libraries: int):
    """verify that alignment was run and output is present"""

    for i in range(n_libraries):
        library = manifest.get_library(i)

        for p_list in (
            library.polya_filtering_summaries,
            library.star_logs,
            library.alignment_pickles,
            library.processed_bams,
        ):
            for p in p_list:
                if not p.exists():
                    log.error(f"{p} does not exist, alignment looks incomplete")
                    return False
    else:
        return True
```

[10]:
```
run_name = '1'
output_dir = Path('/home/hsarkar/code/slide-seq-data/P25255/processed_data')/\
    Path(run_name)
flowcell_dirs = sorted(Path(fd) for fd in set(run_df.bclpath))
manifest_file = output_dir / "manifest.yaml"
metadata_file = output_dir / "metadata.csv"
```

(continues on next page)

(continued from previous page)

```
run_info_list = [ run_info ]
config = get_config()

manifest = Manifest(
            run_name=run_name,
            flowcell_dirs=flowcell_dirs,
            workflow_dir=output_dir,
            library_dir=Path('/home/hsarkar/code/slide-seq-data/P25255/processed_data/
↪libraries'),
            metadata_file=metadata_file,
            metadata=split_sample_lanes(run_df, run_info_list),
            email_addresses=sorted(
                set(e.strip() for v in run_df.email for e in v.split(","))
            ),
        )
```

At this point we have read the config file stored within the `slideseq-tools` directory, curtailed to our need

[11]:
```
config
```

[11]:
```
Config(picard=PosixPath('/home/hsarkar/code/slideseq-tools/soft/picard/build/libs/picard.
↪jar'), dropseq_dir=PosixPath('/home/hsarkar/code/slideseq-tools/soft/Drop-seq_tools-2.
↪5.1'), reference_dir=PosixPath('/broad/macosko/reference'), workflow_dir=PosixPath('/
↪broad/macosko/data/workflows/flowcell'), library_dir=PosixPath('/broad/macosko/data/
↪libraries'), gsecret_name='projects/velina-208320/secrets/sequencing-credentials/
↪versions/latest', gsheet_id='1kwnKrkbl80LyE9lND0UZZJXipL4yfBbGjkTe6hcwJic', worksheet=
↪'Experiment Log', gs_path=PosixPath('macosko_data/libraries'))
```

[12]:
```
manifest
```

[12]:
```
Manifest(run_name='1', flowcell_dirs=[PosixPath('/home/hsarkar/code/slide-seq-data/
↪P25255/220329_A00689_0513_BHYK23DRXY')], workflow_dir=PosixPath('/home/hsarkar/code/
↪slide-seq-data/P25255/processed_data/1'), library_dir=PosixPath('/home/hsarkar/code/
↪slide-seq-data/P25255/processed_data/libraries'), metadata_file=PosixPath('/home/
↪hsarkar/code/slide-seq-data/P25255/processed_data/1/metadata.csv'), metadata=      ␣
↪library        date   flowcell  run_name  \
0  P25255_1001  2022-28-03  HYK23DRXY         1
1  P25255_1002  2022-28-03  HYK23DRXY         1
2  P25255_1003  2022-28-03  HYK23DRXY         1
3  P25255_1004  2022-28-03  HYK23DRXY         1
4  P25255_1001  2022-28-03  HYK23DRXY         1
5  P25255_1002  2022-28-03  HYK23DRXY         1
6  P25255_1003  2022-28-03  HYK23DRXY         1
7  P25255_1004  2022-28-03  HYK23DRXY         1

                                          bclpath  lane sample_barcode  \
0  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       TAAGGCGA
1  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       CGTACTAG
2  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       AGGCAGAA
3  /home/hsarkar/code/slide-seq-data/P25255/22032...     1       TCCTGAGC
4  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       TAAGGCGA
5  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       CGTACTAG
6  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       AGGCAGAA
```

(continues on next page)

(continued from previous page)

```
7  /home/hsarkar/code/slide-seq-data/P25255/22032...     2       TCCTGAGC

  bead_structure                                          reference  \
0  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
1  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
2  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
3  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
4  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
5  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
6  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...
7  8C18X6C1X8M1X  /home/hsarkar/code/slide-seq-data/slide-seq-re...

   run_barcodematching  locus_function_list  start_sequence  base_quality  \
0                False                     0               0             0
1                False                     0               0             0
2                False                     0               0             0
3                False                     0               0             0
4                False                     0               0             0
5                False                     0               0             0
6                False                     0               0             0
7                False                     0               0             0

   min_transcripts_per_cell                    email  \
0                         0  hiraksarkar.cs@gmail.com
1                         0  hiraksarkar.cs@gmail.com
2                         0  hiraksarkar.cs@gmail.com
3                         0  hiraksarkar.cs@gmail.com
4                         0  hiraksarkar.cs@gmail.com
5                         0  hiraksarkar.cs@gmail.com
6                         0  hiraksarkar.cs@gmail.com
7                         0  hiraksarkar.cs@gmail.com

                                  puckcaller_path  bead_type  \
0  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
1  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
2  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
3  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
4  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
5  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
6  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0
7  /home/hsarkar/code/slide-seq-data/P25255/Barco...          0

   gen_read1_plot  gen_downsampling
0           False             False
1           False             False
2           False             False
3           False             False
4           False             False
5           False             False
6           False             False
7           False             False  , email_addresses=['hiraksarkar.cs@gmail.com'])
```

Create folders using the manifest files

[ ]:
```
prepare_demux(run_info_list, manifest)
```

Step 2 Demultiplex (Has to be run in command line)

`demultiplex.sh` should contain the locations that can be determined from the worksheet and the manifest file

```
TMP_DIR="/home/hsarkar/code/slide-seq-data/P25255/processed_data/tmp"
PICARD_JAR="/home/hsarkar/code/slideseq-tools/soft/picard/build/libs/picard.jar"
SGE_TASK_ID="2"
BASECALLS_DIR="/home/hsarkar/code/slide-seq-data/P25255/220329_A00689_0513_BHYK23DRXY/
↪Data/Intensities/BaseCalls"
READ_STRUCTURE="42T8B41T"
OUTPUT_DIR="/home/hsarkar/code/slide-seq-data/P25255/processed_data/1"
FLOWCELL="HYK23DRXY"
```

Run the following script

```
./scripts/demultiplex.sh
```

We can see the created files here

[13]:
```
n_libraries = len(list(manifest.libraries))
for library_index in range(n_libraries):
    for run_info in run_info_list:
        for lane in run_info.lanes:

            print('[{},{},{}]'.format(
                library_index,run_info.flowcell, lane))
            sample = manifest.get_sample(
                library_index, run_info.flowcell, lane)


            for barcode_ubam in sample.barcode_ubams:
                print(barcode_ubam)
```

```
[0,HYK23DRXY,1]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1001/
↪HYK23DRXY/L001/P25255_1001.TAAGGCGA.unmapped.bam
[0,HYK23DRXY,2]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1001/
↪HYK23DRXY/L002/P25255_1001.TAAGGCGA.unmapped.bam
[1,HYK23DRXY,1]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1002/
↪HYK23DRXY/L001/P25255_1002.CGTACTAG.unmapped.bam
[1,HYK23DRXY,2]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1002/
↪HYK23DRXY/L002/P25255_1002.CGTACTAG.unmapped.bam
[2,HYK23DRXY,1]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1003/
↪HYK23DRXY/L001/P25255_1003.AGGCAGAA.unmapped.bam
[2,HYK23DRXY,2]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1003/
↪HYK23DRXY/L002/P25255_1003.AGGCAGAA.unmapped.bam
```

(continues on next page)

(continued from previous page)

```
[3,HYK23DRXY,1]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1004/
↪HYK23DRXY/L001/P25255_1004.TCCTGAGC.unmapped.bam
[3,HYK23DRXY,2]
/home/hsarkar/code/slide-seq-data/P25255/processed_data/libraries/2022-28-03_P25255_1004/
↪HYK23DRXY/L002/P25255_1004.TCCTGAGC.unmapped.bam
```

After demultiplexing it multiple unmapped bamfiles will be created for each barcode within lane. If there is one `sample_barcode` for each lane then it will only create one file.

Step 3 Merge the barcode.unmapped ubams and

```
[ ]: n_libraries = len(list(manifest.libraries))
for library_index in range(n_libraries):
    for run_info in run_info_list:
        for lane in run_info.lanes:

            print('[{},{},{}]'.format(
                library_index,run_info.flowcell, lane))
            sample = manifest.get_sample(
                library_index, run_info.flowcell, lane)


            for barcode_ubam in sample.barcode_ubams:
                print(barcode_ubam)
            bead_structure = sample.get_bead_structure()
            xc_range = ":".join(f"{i}-{j}" for c, i, j in bead_structure if c == "C")
            xm_range = ":".join(f"{i}-{j}" for c, i, j in bead_structure if c == "M")

            if not sample.raw_ubam.exists():
                if len(sample.barcode_ubams) > 1:
                    cmd = config.picard_cmd("MergeSamFiles", manifest.tmp_dir)
                    cmd.extend(
                        [
                            "--OUTPUT",
                            sample.raw_ubam,
                            "--SORT_ORDER",
                            "unsorted",
                            "--ASSUME_SORTED",
                            "true",
                        ]
                    )

                    for ubam_file in sample.barcode_ubams:
                        cmd.extend(["--INPUT", ubam_file])
                    run_command(cmd, "MergeBamFiles", sample)
                else:
                    if sample.raw_ubam.exists():
                        print(sample.raw_ubam)
                    else:
                        os.rename(sample.barcode_ubams[0], sample.raw_ubam)
```

Step 4 recover whitelisted barcodes (Takes long run from command prompt)

1.2.3 Step 4a: Extract the barcodes (42bp) from the bam files

[17]:
```
fp = open('../../../scripts/write_shell.txt','w')
for library_index in range(n_libraries):
    for run_info in run_info_list:
        for lane in run_info.lanes:
            print('[{},{},{}]'.format(
                library_index,run_info.flowcell, lane))
            sample = manifest.get_sample(
                library_index, run_info.flowcell, lane)
            if sample is None:
                print('sample not present')

            awk_command = "awk 'NR%2 {{print > \"{}\"}}'".format(str(sample.raw_barcode))
            cmd = [
                "samtools",
                "view",
                str(sample.raw_ubam),
                "|",
                "cut -f10 |",
                awk_command
            ]
            fp.write("{}\n".format(' '.join(cmd)))
fp.close()
```

```
[0,HYK23DRXY,1]
[0,HYK23DRXY,2]
[1,HYK23DRXY,1]
[1,HYK23DRXY,2]
[2,HYK23DRXY,1]
[2,HYK23DRXY,2]
[3,HYK23DRXY,1]
[3,HYK23DRXY,2]
```

Run the commands

```
parallel -j 8 < /home/hsarkar/code/slideseq-tools/write_shell.txt
```

1.2.4 Step 4b: Create whitelist (Ideally one should borrow information from image analysis)

All the follwoing commands should be run from command line and not notebook

The barcodes will be written using the following script

```
python scripts/createwhitelist.py
```

Step 5: Create the refined bam files

```
python scripts/recoverbam.py
```

Step 6: Run the alignment step

```
python scripts/alignment.py
```

Step 7: Run processing pipeline

```
python scripts/processing.py
```

[18]:
```
!pwd
```

```
/home/hsarkar/code/slide-seq-pipeline/docs/source/notebooks
```

1.3 Recovery algorithm details

1.3.1 Bead Matrix

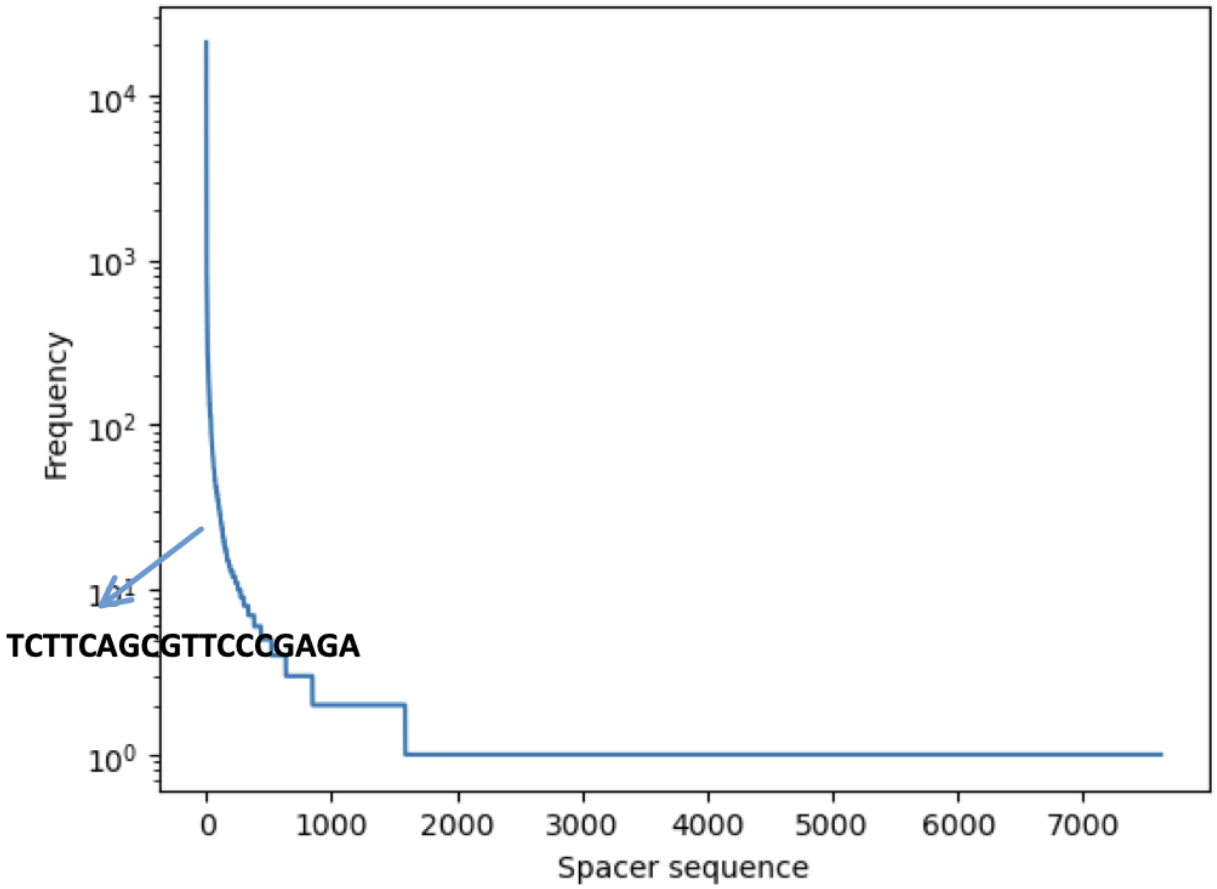
Within slideseq dataset the left end of the read contains 42 base-pairs. Each of this 42 bp sequence is structured in a pre-defined arrangement known as bead-matrix. The arrangement dictates which base-pairs to consider while forming the bead barcode and the UMI.

In our experiment the bead-matrix is '8C18X6C1X8M1X'

```
'C' - bead barcode
'X' - discard
'M' - UMI barcode
```

According to the previous structure we have the following scenario, where the bead-barcode consists of 8+6 = 14 base-pairs. There is 18 base-pair spacer between the two different barcodes. This spacer should be fixed and placed in the predefined location (in this case 9th position) but due to noise and quality of the beads the spacer can have noisy bases and is shifting from it's original position.

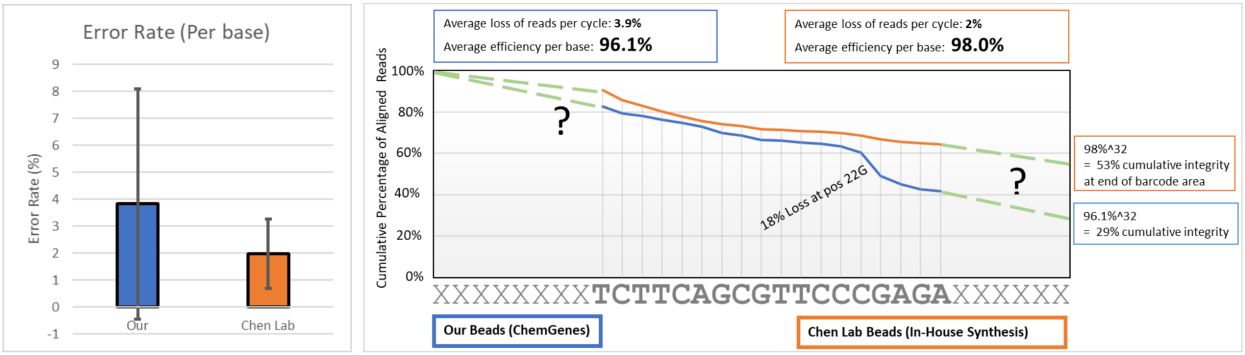
For example in the beads we see




```
ATGCGAATTCTTCAGCGTTCCCGAGACGATCAGCCATTGCTT
AGCGCCTCTTCAGCGTTCCCGAGACGGTTACTTCCACTAGAT -- If there are more than one shift we can␣
↪not much (not recoverable)
ATTCGGTTCTTCAGCGTTCCCGAGATACCGCTGTGGGTTTTT -- There is possibility of recovery
GACCTAATCTTCAGCGTTCAGTCATAATCCCACAGATGGTAC -- no
GGGGTCCTTCTTCAGCGTTCCCGAGACTTAGTTCAGCGTCTT --
CGCAGACTTCTTCAGCGTTCTTTTTAAAATATCTGTAGAGGC
TCCGTCCCTCTTCAAAGCAGGCCCGAGCCGCCTGGATACCGC
AGTCAATTTCTTCAGCGTTCCGAGAGCTAAACGGCTGCTTTT
CTTCCAATTTTTCAGCGTTCCCGGACCAAGCAAGTTAGTCTT
```

70K barcode of 14bp long — Exact match to create a 60K whitelist

Johan has prepared a very comprehensive comparison of the base qualities.



Error Rate calculated by checking the integrity of each position in the Universal Primer Region, when the earlier sequence was in frame.

Our beads have almost twice as high error rate as the Chen Lab beads, but also show a more inconsistent error rate. For example, 18% of molecules correct at the CCC region, lose alignment at the following G for our beads.

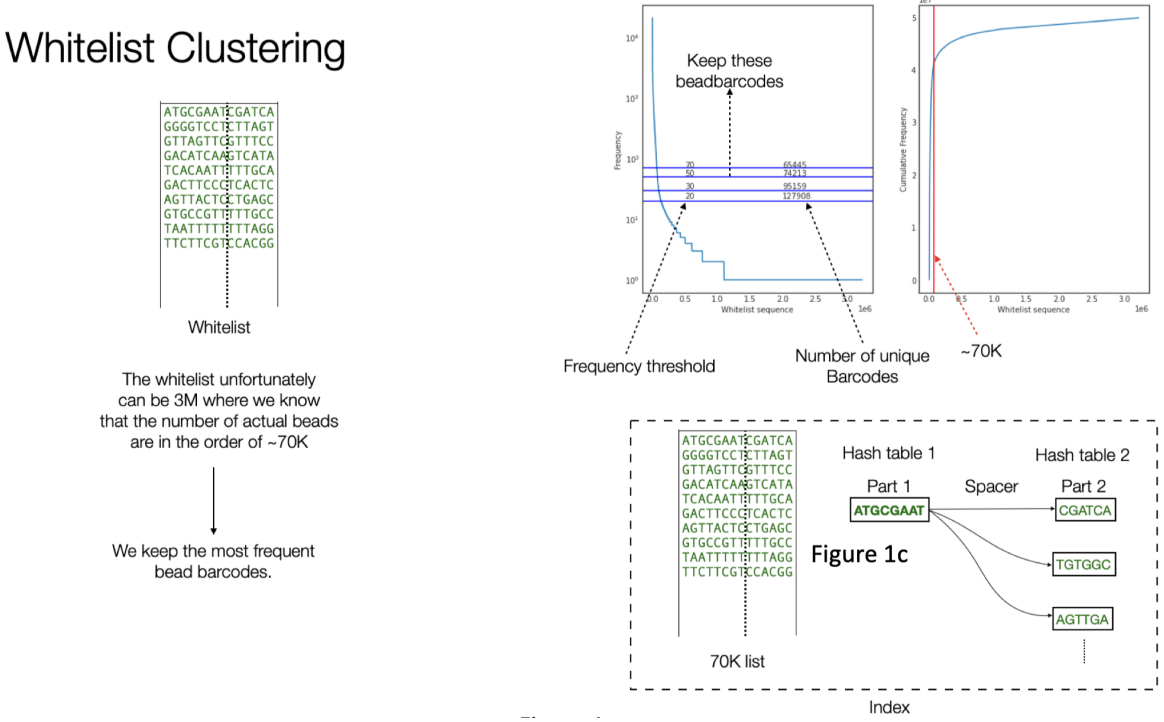
If the errors were 1bp-deletions or substitutions, this could all be salvaged, but unfortunately most of them are much larger deletions, especially near the end.

Johan Boström – Adameyko Lab – May 2022

One thing to notice is that the quality drops a lot therefore we need to correct these errors by algorithmic means.

1.3.2 Current Algorithm

Index Building



The whitelist contains around 70K barcodes. The data from Johan contains around 118M reads. The index is organized with two hash tables. First one containing the first 8bp part of the barcodes and the second hash table that contains the 6bp. We build a connection between them via their co-occurrence in the white list.

Building the whitelist consists of two parts. In the first part we filter out the reads that has a perfect match with the spacer in the expected position (start position 8). We know that the bead barcode consists of the first 8bp and then the 6bp after the spacer. The initial whitelist is built by clipping this 14bp from the reads filtered in the previous step. The expected number of bead barcodes in a puck is ~70K. However In most of the cases because of noise, we end up with more bead barcodes. In order to extract the expected number of bead barcodes, we built a frequency table for these barcodes as shown in Figure 1b, the y-axis shows the frequency of the bead barcodes. The horizontal line shows the frequency cutoff we can use and the resulting number of unique bead barcodes when using such a threshold.

The final whitelist consists of around ~70K bead barcodes.

Out of all reads in our data 38% matches exactly with one of the barcodes in the whitelist (out of these around 3% has a wrong spacer sequence). The aim of the alignment algorithm is to recover the rest 62% reads.

The alignment algorithm uses different heuristics and can be parameterized to tune the number final reads that can be recovered. Alignment algorithm made use of the structure shown in Figure 1c.

The recovery algorithm starts with the alignment of the spacer to the actual read. A typical spacer alignment is shown in Figure 2.

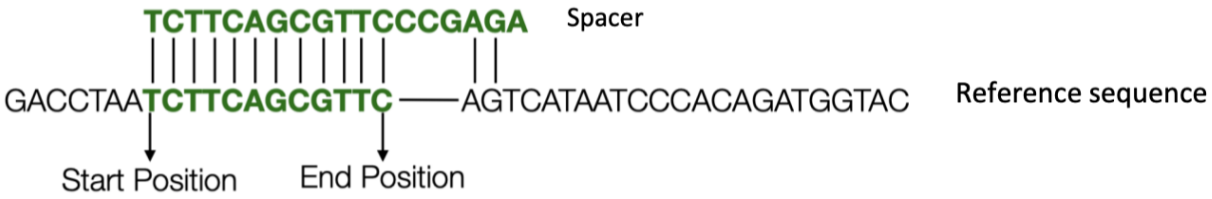


Figure 2

Using this method we recover around 7% of the noisy reads. This leads us to 45% 56% of the beads to recover.

Tuning the allowable rate

There are a couple of different ways to make this algorithm flexible to allow more number of beads at the expense of accepting reads that are less sensitive to the substitution noise.

```
1. We can increase the size of the whitelist to add more bead barcodes in the index.␣
↪Increasing the index size accommodate wide range of mapping.
2. Increasing the edit distance threshold.
3. Increasing the length of the reference used to search for the second part.
```

Instead of stopping at the expected position of the UMI the search can further progress to the end of sequence.

The above mentioned algorithm is implements in `scripts/recoverbam.py`