Matthew Wright DeDuper HW1

Define problem

Performing PCR on your library, while providing enough cDNA to be read during sequencing, also introduces bias. For example, if you have multiple reads from a gene after sequencing, how do you tell if these are real expression, or bias introduced by sequencing? Perhaps this gene just amplified extraordinarily well and the expression you see isn't real. Removing PCR duplicates will help reduce this problem.

Examples are provided in the before and after example SAM files.

Strategy

Before starting:

Turn SAM file into BAM file

Sort BAM file

Convert BAM back to SAM

Actual DeDuper program:

Read in file with all UMIs and store in a dictionary

Read in .sam file

Read all @ lines and immediately output to file

Read first line and extract: UMI from the first column, position from the fourth column, and CIGAR string from the 5th column. (The chromosome info might also be informative, but if the file is sorted, I don't think it will be problem. This functionality can be easily introduced later though)

Check if UMI exists in dictionary. If not, move to next line (toss read).

If UMI exists: Check if CIGAR string contains soft masking at the 5' end. If so, adjust the position by subtracting the number of soft clipped bases from the position. Store adjusted position and UMI for first read as reference variables. Also, output this read to the file.

Read next line. Check UMI exists. Check for CIGAR string and adjust for soft clipping if necessary. Check current UMI and adjusted position against the the reference line. If they match, throw out the line, keep the reference, and move on to the next line. If they don't match, write the line to output and overwrite the reference UMI and adjusted position with the current UMI and position.

Pseudocode

```
#!/usr/bin/env python

import argparse

def get_arguments():
  Give description of deduper, take UMI input, SAM input, and output files
  return UMI input, SAM input and output arguments

def positionchange(pos, cigar):
  if cigar starts with [num1]S[num2]M:
```

```
    return pos - num1
  else:
    return pos

args=get_arguments()
umidict = {}

with args.umifile as umi:
  for line in umi:
    umidict[line.strip()]=0

umiref=""
posref=0

with args.samfile as sam:
  with args.outfile as out:
    for line in sam:
      stripped=line.strip()
      # If line is a @ header line, output to file
      if stripped starts with @:
        print(stripped, file=out)
      # else if it's not, do duplicate checks
      else:
        # Split line and read UMI
        splitline=stripped.split("\t")
        lineumi=splitline[0] plus a bunch of regex to extract the UMI from the end of the header
        # Check if UMI exists in dict, if not, read next line
        if lineumi not in umidict:
          break

        # Read rest of variables
        linepos=splitline[3]
        linecigar=splitline[4]
        # Adjust position with cigar string
        adjpos=positionchange(linepos, linecigar)

        # If this line's UMI and adjusted position is equal to the reference, it's a duplicate, so thro
        if lineumi==umiref and adjpos==posref:
          break
        elif lineumi!=umiref or adjpos!=posref:
          # If it's not a duplicate, store variables as new references and write to file
          lineumi = umiref
          adjpos = posref
          print(stripped, file=out)
```

Functions

get_arguments()

Description: Pretty typical argparse statement. Will take in the UMI input file, SAM input file, and the name of the output file.

Function header: `def get_arguments():`

Test examples: ???

Return statement: `return parser.parse_args()`

positionchange

Description: A function which will take in a position and cigar string, and, if there is soft clipped on the 5' end, will take that into account and return the true start position of the read

Function header: `def positionchange(pos, cigar):`

Test examples:

```
pos = 644, cigar=101M
return 644

pos = 644, cigar = 10S91M
return 634

pos = 644, cigar = 91M10S
return 644
```

Return statement:

```
# If soft clipped reads are detected at 5' end of cigar string
return pos - num1

# If there are no soft clipped reads (at 5' end)
return pos
```

~