De-Duper Part #1

Alexa Dowdell

Wed Oct 18 17:06:27 2017

Directions: Write up a strategy for writing a Reference based PCR duplicate removal tool. Be sure to include an example input and output SAM file

Part 1

The Problem

A PCR duplicate is any additional read to one already present that was derived from the same molecule. Therefore, any PCR duplicate has the same chromosome, alignment position, UMI and strand (assuming the library was stranded) as the original mapped read. PCR duplicate removal is a recommended process regardless of the method used in next generation sequencing. The current problem is finding a PCR duplicate removal tool that is unbias and can also balance accuracy, efficiency, and cost.

Our goal is to attempt to successfully remove PCR duplicates from single end RNA-seq libraries, in which the records contain UMIs at the ends of the header lines. However, the problem is a little more complicated once the notion of soft clipping is introduced. Soft clipping is the idea nucleotide bases in a sequence are present in the SAM file but do not appear to align to the genome. Likewise, the possiblility of mismatching UMIs can arise due to sequencing error. Bottom line: PCR duplicates need to be removed since their presence can result in false conclusions regarding the data being drawn, especially in the case of RNA-seq differential expression. A layout of a potential strategy to remove PCR duplicates can be found below:

Assume we began with a SAM file of uniquely mapped reads

Strategy Walk Through

1. SAMtools sort: Sort the SAM file for proper POS alignment.
2. Create a dictionary of all known UMIs and associated counts. Given the file of 96 known UMIs, concstruct a reference dictionary with all UMIs saved as keys and values initialized to zero.
3. Read in the first read of SAM file and extract alignment POS from the fourth column.
4. Continue reading in additional reads with POS values within one untrimmed read length from alignment POS obtained above (Step 3), assuming the alignment was defined and fixed before sequencing.
5. Declare the POS saved from the first read as the reference POS.
6. Compare next read that passed (Step 4) to determine whether read aligns by checking for presence of soft clipping using CIGAR flag. If soft clipping == TRUE, adjust alignment POS from read and re-compare sequences.
7. If POS of reference read established in Step 5 and adjusted POS of read from Step 6 are the same, check for matching UMI.

8. If reference POS == adjusted POS, extract the UMI from header corresponding to each read. If the UMIs are identical, remove the first duplicate that is crossed and revert back to Step 6.

9. If the starting read in the section and the read being compared do not match on means of POS or UMI, iterate through the section of reads initially read in as having a similar reference POS until a read passes Step 6. Otherwise, if no reads remain, revert back to reading in the next read with a differing starting alignment POS and continue at Step 4.

Step 1: Pre-Algorithm - SAMtools Sort

Given a SAM file, the very first thing we want to do is sort the file into nine determined columns, aligned by left most position (in the fourth column) for easier data manipulation down the road.

```
samtools view -u input.sam > output.bam
samtools sort output.bam > output2.bam
samtools view -h output2.bam > output.sam
```

Steps 2-9: Python Pseudocode

```
Arg parse takes in two arguments -f output.sam file from above and -l defined
untrimmed read length

referenceFlag == TRUE #set a flag to determine the first read of a new section
length = args.length #set the untrimmed determined read length from arg parge
                     # as length
```

```
with open samFile of interest as fh:
  for each line in file:
    function sectionChecker(line) {
      If first read {
        refPOS = 0 # initialize a variable refPOS
        Extract alignment POS from column 4 and assign to  variable "refPOS"
        Add line to sectionBuddy, where sectionBuddy is an list containing lines
        with similar POS, each line is a new element in the list.
        referenceFlag == FALSE #all subsequent reads that make it into the
                               #sectionBuddy will not be the reference
      }
      continue #on next read from samFile
      else {
        If POS of current read is within one 'length' of the reference read {
          Add read to sectionBuddy
        }
        else {
        Signifies end of sectionBuddy, and halt reading in new lines momentarily
        and move to softClippingChecker()
        referenceFlag == TRUE #revert back to default since a new section
                              # must begin
        }
    }
}
```

```
function softClippingChecker(sectionBuddy){
  refLine = sectionBuddy[0] # set first line to reference line
  firstBudLine = sectionBuddy[1] #set second line to firstBuddy or
                                 # first line of comparison
  refPOS = extract 4th column alignment POS from refLine
  firstBudPOS = extract 4th column alignment POS from firstBudLine
  if refPOS == firstBudPOS{
    Continue on to UMIchecker(refLine, firstBudLine)
  }
  elif refPOS != firstBudPOS{
    firstBudCigar = Extract CIGAR flag of firstBudLine
    If firstBudCigar.contain('S'){
      firstBudPOSadj = Adjust firstBudPOS by number in front of 'S'
    }
    if refPOS == firstBudPOS{
      Continue on to UMIchecker(refLine, firstBudLine)
    }
    else{
      Pass line through function validUMIchecker(firstBudLine)
      Read in buddyLine = sectionBuddy[i]
    }
  }
  else{
    Read in buddyLine = sectionBuddy[i] until duplicate POS is found and moved
    onto UMIchecker(refLine, buddyLine)
  }
  If refLine != firstBudLine at any sectionBuddy[i], no PCR duplicates found {
    Write sectionBuddy lines to revisedFile.sam
    Revert back to sectionChecker(line)
  }
}



function UMIstripper(header) {
  UMIofInterest = Extract out UMI at the end of the header after the colon
  return UMIofInterest
}



function UMIchecker(refLine, buddyLine){
  refheader = extract 1st column UMI header from refLine
  buddyheader = extract 1st column UMI header from buddyLine
  refUMI = UMIstripper(refheader)
  buddyUMI = UMIstripper(buddyheader)
  Check if refheader is a key in knownUMIdict{
```

```
  value = refPOS
  Update value of 0 in knownUMIdict[refUMI] = value
  return TRUE/FALSE
}
Check if buddyheader is a key in knownUMIdict{
  return TRUE/FALSE
}
if either check returns FALSE{
  Discard the corresponding line
}
elif refUMI == buddyUMI {
  Discard buddyLine
  Write refLine to revisedFile.sam
}
else {
  if value in knownUMIdict[buddyUMI] == 0 {
    value = buddyPOS
    Update value of knownUMIdict[buddyUMI] = value
    Write refLine, buddyLine to revisedFile.sam
  }
  else {
    Discard buddyLine
    Write refLine to revisedFile.sam
  }
}
```

```
function validUMIchecker(line){
  header = extract 1st column UMI header from line
  UMI = UMIstripper(header)
  linePOS = extract fourth column alignment POS
  if UMI in knownUMIDict {
      Write line to revisedFile.sam
      }
    else {
      Discard line
    }
}
```

Below is an example SAM input file that may be passed in the `-f` arg parse argument:

```
<QNAME> <FLAG> <RNAME> <POS> <MAPQ> <CIGAR> <MRNM> <MPOS> <ISIZE> <SEQ>

SRR035022.2621862:AACGCCAT 163 16 50 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCCT
AACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AACGCCAT 163 16 50 37 22S54M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AACGCCAT 163 16 60 37 10S66M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AACGCCAT 163 16 72 37 22S54M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AAGGTACG 163 16 72 37 22S54M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AAGGTACG 163 16 75 37 25S54M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:ACACAGAG 163 16 159 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:ACACAGAG 163 16 167 37 8S68M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AATTCCGG 163 16 180 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AATTCCGG 163 16 185 37 5S71M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:ACGAAGGT 163 16 306 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
```

After the PCR duplicate removel code is run on the input SAM file, an output file called `revisedFile.sam` would be created containing the following:

```
<QNAME> <FLAG> <RNAME> <POS> <MAPQ> <CIGAR> <MRNM> <MPOS> <ISIZE> <SEQ>

SRR035022.2621862:AACGCCAT 163 16 50 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCCT
AACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AAGGTACG 163 16 72 37 22S54M = 156 179 CCAACCCAACCCTAACCCTAACCCTAAC
CCTAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:ACACAGAG 163 16 159 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:AATTCCGG 163 16 180 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
SRR035022.2621862:ACGAAGGT 163 16 306 37 76M = 156 179 CCAACCCAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCTAACCCTAACCCTAACCGACCCTCACCCTCACCC
```