Pseudocode_PCR_Dup

Dane Dewees

Tue Oct 17 16:49:54 2017

Part 1

Stating the problem with PCR duplication and how to go about parsing through them in SAM file

1. Once PCR amplification of the fragments with adapters occurs, there is a strong possibility of PCR duplicates to occur (since we are literally making multiple 'copies' of the genetic code of interest). This in turn induces a bias output that indicates false clusters of duplicates which takes away from the true amount of transcript abundance. In essence, this process of amplification creates multiple copies of each molecule. Thus, this can have an adverse effect on the overall outcome when it comes to next-step process like RNA seq analysis.
2. In order to fix this, one must filter out these PCR duplicates post amplifcation via specific parameters in ones algorithm i.e., targeting the 'tags' or 'barcodes' that are appended on the end of the library fragments. By isolating these within the region of interest, one can parse through each read and filter out repeated sequences/UMIs/randomers, etc. Only problem is, only factoring these parameters will not account for all of the 'replicated' sequences. Sequening error can induce an inconsistent read of UMIs, but comparing said read to the alignment position could potentially resolve this problem. However, there are processes called 'soft-clipping' which occurs when nucleotide bases are simply not present in the final alignment, but are still found in the SAM file (part of the sequence). It simply shifts the base position, this process needs to be corrected. After combating for UMIs, randomers, soft-clipping, etc., one should be able to parse through the SAM file and keep those that not duplicated and of those filter through the UMI and soft-clipped regions.
3. Generating an algorithm using the psuedo code pasted below, it will take in a SAM file (uniquely mapped) and remove any PCR duplicates found in said file. I will use samtools sort via command line to sort the aligned reads by chromosome followed by <- leftmost position. From there, it will check for UMIs (extract and compare to 96 expected UMIs given from Leslie). If found, store said UMI and then find the corresponding chromosome. From there the alogorithm should check the duplicates strand, and if duplicate is on same strand (like the UMI), save the said strand and then locate the corresponding position (this part of the code factors in CIGAR string annotation - if CIGAR contains an S, save new start position. If it does not, save starting position as listed in said read). While each parameter is parsed and filtered through, a global dictionary will be set so each of these said parameters can be incremented into dictionary. Simply tally up how many times the tuple occurs in dictionary (PCR duplicate), and if tupile does not exist - set count to 1 and generate an output file of 'other' aligned reads.
4. In inpurt and output SAM files, once the it was read through, all duplicates (simple, complex, and reverse) should be removed and only the non-duplicates, same position, same position and same UMI (different chromosome) and duplicated sequence but with different UMI should be leftover in said output files.

Pseudocode used for PCR duplicate algorithm

```
Set argparse argument in order to call input/output file names /
as well as set directory

Functions

def 'function name' (i.e., cig_finder)(cig_str_val, pos_of_aligned_read):

    if soft_clip is present in read?

        correct said alignment_pos (pos_of_aligned_read)

    else

        ignore highlighted read

    return changed_position that was aligned


def 'function name' (i.e., alignment check) (changed/corrected_alignment, list_
of_aligned_reads):

    look for changed/corrected alignment of said read in list_of_aligned_reads
list

    return True/False depending on what is found when parsing through each read

def 'function name' (i.e., UMI_check_from_sequence(sequence_reads):

    pull out UMI substring from the sequence reads

    return UMI

def UMI_check_in_UMI_List(UMI, list_of_UMI)

    look for UMI's in list_of_UMI

    return True/False depending on what is found when parsing through each read


>Set a global dictionary that establishes UMI as keys and alignment positions w
ith strandess
info as the values

>set global list to use the UMIs in order to locate any sequencing error in fil
e


Once argparse and global settings are made, open file and being to parse throug
```

```
h said file


    > Parse through file (by each individual read of SAM file)

    > Be sure to establish regions of interest while parsing (i.e., call for ju
st header /
      line, seq line, etc.)

    > Establish the CIGAR string and set alignment parameters

        - factor in 'soft clipping' in this part of the script

            > CIGAR indicates whether or not the starting position left most po
sition or,
            depending on if soft clipped, then the position will start where al
ignment begins

        - Establish an alignment parameter if 'soft clipping' is found

    > Post alignment, establish if the position has been seen/read through befo
re hand begin
      a if/else statment

        - if position has not been read, add this position to the global list m
ade above.

            > do this by setting a return = True or False at end of if/else loo
p.

    > Want to do the same thing as done with the alignment position for the UMI
& Randomers as well

        - IF UMI/Randomers are found, add to global list made above.

            > do this by setting a return = True/False at end of if/else loop (
same as above)




    > Once each read has been parsed through for both alignment position as wel
l as UMI/Randomers
      and they either are returned as True statements --> output the particular
read to a new
      file, if not returned True, go onto the next read of the SAM file.
```