Extracting GFF features from Sequences

Introduction to PERL

Marcos Cámara Donoso

— December 3, 2016—

MBI1617_PER

1st quarter - 2016-2017

Contents

1 Introduction

Nowadays, sequencing and annotation projects are producing a lot of data about genomic features of whole genome sequences in a huge amount of formats. The GFF (General Feature Format) is the commonly used and the most accepted in scientific community. This is because of his portable and flat file format that allows the GFF to be used as genome annotation storage format. With the increase in the interest in genome annotation projects and other further analysis, the demand of new tools to extract GFF features from sequences has been increased.

In order to fulfill this demand, several Bioinformaticians have developed tools that are very useful in order to extract information from fa (FASTA) files and locate some annotations in a sequence-dependent manner. As many others, this project tries to develop another tool to succeed in this task. In this case, we want to retrieve those substrings from a DNA sequence that are delimited by the start and end coordinates of a GFF record. In order to do that, we need to read the GFF file and then, add a while-loop to read a sequence in fasta format from another file. Finally, iterate through each start-end coords pair from the GFF records and use the `substr` function appropriately, in order to extract the corresponding sequence and printing to standard output along with the coordinates and the feature identifier. Once sequences have been extracted, we can check if the GC content for the CDS features was different from that of the introns. An example of its command-line is shown below:

```
sh$ ./getseqfeatures.pl drome_seq.fa drome_genes.gff
exon1.group 100 150 CGTTGGAT...ATTGAT
exon2.group 340 410 ATTGGATG......CTGTTG
...

sh$
```

2 Methodology: Implementing the code

As it has been said before, we want to retrieve those substrings from the sequence that are delimited by the start and end coordinates of a GFF record and associate them with some features annotated for that specific sequence. And also calculate some parameters that are unique for the sequence, as it could be the GC content of the CDS and intron sequences. For this task we use a bunch of functions that have their important features described below.

2.1 Reading GFF files and .fa files

For this task, we've implemented two functions one for the GFF and one for the .fa files. In first place, the GFF is a tabular file that we need to read line by line selecting the important information for us. In this case, we need the ID of the gene (column 10), the feature that we are going to associate with the ID (column 3), the start coordinate (column 4), the end coordinate (column 5) and the strand (column 7). To achieve that, we use the subroutine `opening_gff` that opens the file and creates 4 arrays, each one with the information we want from each column. This subroutine is an hybrid subroutine because the 4 arrays resulting are placed in the global environment in order to work with them whereas the variables that uses for doing the task are locally placed. This means that if we want to implement this subroutine in another code, we need to establish in that code the four arrays that we are going to use to store the data. Also if we want to change some parameters in order to get other columns indexed in the arrays or we want to work with more arrays we should modify the function given adding more lines following the next architecture:

```
$name = (split(/\s/, $_))[numberofcolumn - 1];
push(@array, $name);
```

In second place, the .fa is just a sequence given by segments with a header. The subroutine `opening_fa` will do the job solving some problems at the moment of work with this .fa file. The main problem here is to avoid the header information and to join all the lines in a single string variable. But for that we use the next block of commands inside the function:

```
next if /^>/o;              #jumps directly to the sequence
chomp $_;
$sequence=substr($_, 0, length($_));
unless (defined $seq){          #The first time introduces the first sequence
    $seq="$sequence";   #and then always else to avoid "use of uninitialized..." issue
}
else {
    $seq="$seq"."$sequence";
};
```

2.2 Associating GFF features its sequence

As soon as we have our files read and our data organized, we need to extract sequences from our string that fulfills the features annotated. For that we will use the subroutine `asso_fa_GFF` which using parameters determined by the start and end coordinates given as arguments and other information as the whole sequence, the feature + ID and the strand computes the substring that belongs to the given coordinates. Then, it pays attention to the strand to decide how to work with the sequence. If the strand is "+" it will compute the GC content using the subroutine `GCcontent`, which simply counts how many Gs and Cs are in the sequence and divides the number by the length of the sequence. After computing that it will print out the feature + ID, the start coordinate, the end coordinate, the GC content (expressed by a number between 0-1) and finally the sequence associated with those features. If the strand is "-" then it computes the reverse complementary using the subroutine `com_rev` and then, it just calculates the GC content and prints the same data before but using the reverse complementary sequence instead the original ones. This entire module works inside of a for-loop that creates a tabular output that can be stored in a text file if we want by using the standard output channel.

```
$dnaseq = substr($faseq, $jfa, $kfa);
    if ($strandsfa eq "+"){
        $GC = &GCcontent($dnaseq);
        print STDOUT "$IDsfa\t$startsfa\t$endsfa\t$GC\t$dnaseq\n";
        }
        else{
        $comrev = &com_rev($dnaseq);
        $GC = &GCcontent($comrev);
        print STDOUT "$IDsfa\t$startsfa\t$endsfa\t$GC\t$comrev\n";
```

2.3 Extracting intron features from sequence

This is the last module of this project. In order to extract intron features associated with the exon features we can use the subroutine `intronizer`. This subroutine consists in a calculator of intron start and end coordinates using the end of the CDS before the exon in the table and the start of the CDS after the exon in the table. Then, it captures the sequence between the two CDS that defines the intron sequence, in most of cases. Once we evaluated that we are in the right position of the table to extract the intron features, we work on the sequences extracted and compute the GC content if it's direct stranded, if not, we use the subroutine `com_rev` to get the reverse complementary and then compute the GC content. In this way, we'll get the GC content for our intron sequences. And we'll have printed a tabular output with the ID of the intron and its GC content. In this print we can also add the sequence of the intron just adding the variable in which we have stored the sequence at the moment of work with it.

```
if (defined $endint && defined $IDint2){
    $m = $endint-1;             #defines the start pos of intron
    $n = ($startint-$endint)+1;
    $intseq = substr($faint, $m, $n);       #defines the length of the intron
    if ($IDint eq $IDint2 && $IDint =~ /^exon*/){   #searches for exon... lines
        if ($strandint eq "+"){
            $gici= &GCcontent($intseq);
            print STDOUT "intron.$IDint\t$gici\n"; #If you want to see the sequence add \t$intseq
        }
        else{
            $intcomrev = &com_rev($intseq);
            $gici = &GCcontent($intcomrev);
            print STDOUT "intron.$IDint\t$gici\n"; #If you want to see the sequence add \t$intcomrev
        };
```

As it was described before, in most of the cases this works well like this, but not always. Sometimes there're some cases in which the sequence of n(CDS-exon-CDS) is interrupted by a start codon followed by an exon or other thing. For those cases we have to assume and prepare our algorithm to read and solve those positions avoiding errors that can result in skipping a possible intron. For the case shown below it is necessary a solution adapted to the problem here. For that, we have the second `if` block shown after this.

The problem:

```
chr3R   dm3_refGene exon    19030   19177   0.000000    -   .   gene_id "NM_142034"; transcript_id "NM_142034";
chr3R   dm3_refGene CDS 19771   19995   0.000000    -   0   gene_id "NM_142034"; transcript_id "NM_142034";
chr3R   dm3_refGene start_codon 19993   19995   0.000000    -   .   gene_id "NM_142034"; transcript_id "NM_142034";
chr3R   dm3_refGene exon    19771   20705   0.000000    -   .   gene_id "NM_142034"; transcript_id "NM_142034";
```

The solution:

```
if (defined $IDint3 && $IDint eq $IDint3 && $IDint2=~ /^start*/){
    if ($strandint eq "+"){
        $gici= &GCcontent($intseq);
        print STDOUT "intron.$IDint\t$gici\n"; #If you want to see the sequence add \t$intseq
    }
    else{
        $intcomrev = &com_rev($intseq);
        $gici = &GCcontent($intcomrev);
        print STDOUT "intron.$IDint\t$gici\n"; #If you want to see the sequence add \t$intcomrev
};
```

There's an possible issue with this function and a possible intron placed before the start codon for the gene "NM_057743". The problem is that the GFF has annotated a exon in a place that is not mapped inside a CDS and also is not expressed as protein. For this reason, this exon is not added as an special case to the algorithm in order to calculate the possible intron associated between the two exons that belongs to this gene.

At last, we have to consider that this block of code under the subroutine `intronizer` is also inside a for-loop that creates a tabular output that can be stored in the output text file that can be demanded from the shell command.

2.4 Other subroutines: A reverse complementary traductor and a GC content calculator

These other subroutines are those that are involved directly with the main functions of this project and even they are small functions, they are very useful in order to get the final result of this script. In first place, we have the reverse complementary translator (code below) given by the subroutine `com_rev` which takes the substring from the sequences and using the `transliteration` and `reverse` functions changes each nucleotide from the sequence to its complementary and then change the sense of the string, respectively.

```
$dna =~ tr/ACGT/tgca/;      #complementary
$comrev = reverse($dna);    #reverse
```

In second place, we had a GC content calculator given by the subroutine `GC_content` that measures how many Gs and Cs we have in a given string and dividing them by the length of the sequence we obtain the GC content.

```
my $length = length($dna_seq);
    for (my $i = 0; $i < $length; $i++) {
        my $char;
        $char = uc(substr($dna_seq,$i,1));
      SWITCH: {
          $char eq 'G' && ($G++, last SWITCH);
          $char eq 'C' && ($C++, last SWITCH);
          $char eq 'g' && ($G++, last SWITCH);
          $char eq 'c' && ($C++, last SWITCH);
          };

    };
    my $GC = (($G+$C)/$length);
```

3 Discussion

The main aim of this project was to extract GFF features from sequences relating those features with the sequence that they define. With that objective achieved, as we can see in the output below if we compare it with the output shown in the introduction, we can know start wondering about further points that this script reaches by itself. For instance, the calculation of the GC content and its printing in the final output of the extracted features from sequences, this advantage allows us to compare this new information with another that it's also generated in this script. Both GC contents from the exons and the introns of each gene can now be compared in order to determine differences between both types of sequences. Although further calculations are need to be performed, we may notice that there're differences between exon and intron sequences. In this way, exon sequences are enriched in GC if we compare them with introns ones. This could be interpreted as a kind of strategy in the recognition of splice sites in the RNA processing.

Another fact that can be highlighted is the possible applications of this code. For instance, the question of if this code it only work with DNA sequences or can be adapted to another biomolecules that follow sequence patterns is a good one to be explore. Maybe, doing some changes in the GC content and modifying some parameters we can get a good recogniser of signal sequences in a protein or even in a RNA for different processing of the molecules or for its functions. Also we could study with this module the quimical composition of the sequence of a protein in a case given. But these are only cases exploring the functions of a single module. The real answer is that we can adapt the entire code to extract features of proteins in sequences of aminoacids if it was necessary just by doing some changes in the script.

```
sh$ ./getseqfeatures.pl drome_seq.fa drome_genes.gff

start_codon.NM_142033   5996    5998    0.333333333333333   ATG
CDS.NM_142033   5996    6911    0.46943231441048    ATGGGCGT...CAAGTG
...
CDS.NM_142034   15328   15492   0.527272727272727   aaaactc...ttgtaa

...
intron.exon.NM_142033   0.327759197324415
intron.exon.NM_142033   0.262626262626263
intron.exon.NM_142033   0.327272727272727
intron.exon.NM_142034   0.378823529411765

sh$
```