2 Extract_IGR.pl

#!/usr/bin/Perl

#Title: Perl script to extract IGRs from embl format of different contigs of whole genome of *N. meningitidis*

#Description: the main idea of script is to extract index of starting points and ending points of each CD in forward and reverse strands with full sequence and gene names then save them in an array. Then subtracting the ending point of first CD from starting point of second CD refers to IGR. If the sequence of any IGR is in opposite direction comparable with the rest of sequences of IGRs, the script is going to make reverse complement then print it. Finally, the script is going to identify regions located in head to head within adjacent two genes and print the gene name with (-D). While script printed (-sh)with the gene name for the IGRs that located in tail to tail within adjacent two genes and for the rest of IGR printed only gene name without any addition.

#Date: 29/June/2015

#UpDate: 9/Dec/2016

```
# All the variables, arrays were defined
use strict;
use warnings;
my $FullSequence1;
my $FinalResult = "";
my $FullSequence;
my $seq = 0;
my @GenName;
my @RF;
my @StartPoints;
my @EndPoints;
```

Define and open the directory that will hold all the files in embl format, an array to hold all the files name in the directory and loop to read each file within the folder then check if the file is .embl file, open them otherwise files will be ignored. All the variables and arrays reinitialized to avoid overwritten in the final output file. The script functions in this block are adapted from (http://perldoc.perl.org/functions/readdir.html).

```
my $directory = $ARGV[0];
die "Need the path of your directory that hold the embl files \n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
opendir(DIR,$directory);
my @files = readdir(DIR);
closedir(DIR);
foreach(@files){
        if ($_ =~ /.embl$/) {
                my $filename = $directory . '/'.$_;
                open(my $fh, '<:encoding(UTF-8)', $filename)or die "Could not open file '$filename' $!";
                @StartPoints = ();
                @EndPoints = ();
                @GenName = ();
                @RF = ();
                $seq = 0;
                $FullSequence = "";
```

This bit match when the sequence matched and then reach into ending of sequence with (//), match the sequence lines in embl (small pieces of sequence separated by spaces), replace space with nothing, then concatenated full sequence and assign it into variable. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).

```
while (my $row = <$fh>) {
    if($seq > 0){
        if ( $row =~ /(\s+)((\w+\s+)+)(\s+)(\d+)/){
            $FullSequence1 = $2;
            $FullSequence1 =~ s/\s+//g;
            $FullSequence = $FullSequence . $FullSequence1;

        }
    }
```

#Match lines contained starting points and ending points in forward strands, push starting points into an array, push ending points into another an array, specify the CD is forward strands. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 136).

```
    if ($row =~ /^(FT   CDS)\s+(\d+)(..)(\d+)/){
        push(@StartPoints, $2);
        push(@EndPoints, $4);
        push(@RF, "F");

    }
```

#Match lines contained starting points and ending points in reverse strands, push starting points into an array, push ending points into an array, specify the CD is reverse strands. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 136).

```
    elsif ($row =~ /^(FT   CDS)(\s+)(complement\()(\d+)(..)(\d+)/){
        push(@StartPoints, $4);
        push(@EndPoints, $6);
        push(@RF, "R");
    }
```

Match gene name and save it in specific array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 136).

```
    if ($row =~ /(\S+)(gene=")(\S+)/){
        push(@GenName, $3);
    }
```

#Setting the Sequence to true (seq become 1) at file starts with gene sequence The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 124).

```
    if ($row =~ /^SQ   Sequence/){
        $seq = 1;
    }
}
```

Check the length of Starting point in an array, loop to go through all points, check if there is some value between ending index of current gene and starting index of next gene, if there is some value extracts the resulting value, if the contigs ordered in reverse (ie starting with *NEIS1906* then *NEIS1905* then *NEIS1904* so on, make reverse complement then go to print function with four situations explained below. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 166) and (https://perldoc.perl.org/perlsub.html).

```
		my $TNumber = scalar @StartPoints;
		for (my $b = 0; $b < $TNumber-1; $b++){
			my $sequ = "";
			my $revcomp = "";
			if(($EndPoints[$b] - $StartPoints[$b+1]) < -1){
				$sequ = substr($FullSequence, $EndPoints[$b], (($StartPoints[$b+1]-$EndPoints[$b]))-1);
				if(($GenName[$b] gt $GenName[$b+1])){
					$revcomp = reverse($sequ);
					$revcomp =~ tr/ACGTacgt/TGCAtgca/;
# Situation_1 if the name of current gene and next gene end with (R), both on reverse strands, print the name of current gene for each IGR sequence.
					if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /R/){
						$FinalResult = $FinalResult . '>' . "$GenName[$b]\n" . $revcomp . "\n";
	}


# Situation_2 if the name of current gene and next gene end with (F), both on forward strands, print the name of next gene for each IGR sequence.
					if($RF[$b] =~ /F/ && $RF[$b+1] =~ /F/){
						$FinalResult = $FinalResult . '>' . "$GenName[$b+1]\n". $revcomp . "\n";
					}


# Situation_3 if the name of current gene end with (R) (it's on a reverse strand) and next gene end with F (it's on a forward sequence), print twice (once print current gene name with IGR sequence, another print next gene name with IGR sequence).

					if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /F/){
						$FinalResult = $FinalResult . '>' . "$GenName[$b+1]-sh\n" . $revcomp . "\n";
						$FinalResult = $FinalResult . '>' . "$GenName[$b]-sh\n" . $revcomp . "\n";

					}

# Situation_4 if the name of current gene end with (F) (it's on a forward strand) and next gene end with R (it's on a reverse sequence), print twice (once print current gene name with IGR sequence, another print next gene name with IGR sequence).

					if($RF[$b] =~ /F/ && $RF[$b+1] =~ /R/){
						$FinalResult = $FinalResult . '>' . "$GenName[$b+1]-D\n". $revcomp . "\n";
						$FinalResult = $FinalResult . '>' . "$GenName[$b]-D\n". $revcomp . "\n";
					}
				}else {

# All the situation above will be repeated for the IGRs sequence in the same direction without complement reverse

					if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /R/){
```

```
                                        $FinalResult = $FinalResult . '>' . "$GenName[$b]\n" . $sequ . "\n";
        }

                                        if($RF[$b] =~ /F/ && $RF[$b+1] =~ /F/){
                                                $FinalResult = $FinalResult . '>' . "$GenName[$b+1]\n". $sequ . "\n";
                                        }

                                        if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /F/){
                                                $FinalResult = $FinalResult . '>' . "$GenName[$b+1]-sh\n" . $sequ . "\n";
                                                $FinalResult = $FinalResult . '>' . "$GenName[$b]-sh\n" . $sequ . "\n";

                                        }

                                        if($RF[$b] =~ /F/ && $RF[$b+1] =~ /R/){
                                                $FinalResult = $FinalResult . '>' . "$GenName[$b+1]-D\n". $sequ . "\n";
                                                $FinalResult = $FinalResult . '>' . "$GenName[$b]-D\n". $sequ . "\n";
                                        }
                                }

                        }

                }
        }
# For writing into file and print the result then close it. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 171).
my $FileName = 'intergenic.fasta';
open(my $fh, '>', $FileName) or die "Could not Open File '$FileName' $!";
print $fh $FinalResult;
close $fh;
}
```

3 Identify_var_IGR.pl

```
#!/usr/bin/Perl
```

Title: Perl script to extract the same identifier and sequences of each two variable IGRs from two multifasta files and aligned them

Description: the main goal of script is to compare the DNA sequences for each IGR in the same identifier from two different multifasta files. Then, the variable sequence between each two IGRs from two different multifasta files will be printed out in one file. Finally, the script is going to align variable sequence in each IRG belongs into two different files and print them in separate files.

#Date: 29/June/2015
#UpDate: 9/Dec/2016

```
# All the variables and hashes were defined
use warnings;
use strict;
my $id1;
my $id2;
my %fasta_hash1;
my %fasta_hash2;
my ($filename1, $filename2) = @ARGV;
die "Need two multifasta files hold the variable IRGs of each 25 pair \n" if (@ARGV != 2); # Test whether there were two arguments by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file
```

In each file in the two multifasta files while loop is to loop through all the IGR names and sequences, chomp them and matches header of fasta, save strain name in key of hash and push them into an array, in case it did not match header of fasta, it will match the sequence then save sequence in a value of hash and push them into another an array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 182, 166).

```
while ( <File1> ) {
        chomp;
   if($_ =~ /^>(.+)/){
                $id1 = $1;
        }else{
        $fasta_hash1{$id1} .= $_;
        }
}
close File1;

while ( <File2> ) {
        chomp;
   if($_ =~ /^>(.+)/){
           $id2 = $1;
         }else{
      $fasta_hash2{$id2} .= $_;
        }
```

```
}
close File2;

# Loop through key-value paired belongs into two multifasta files, then comparison of keys was carried out, the same keys from two different files, their values (sequences) were compared. If different sequence in one of each two different files was detected, the sequences from two different files were aligned and were printed. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 182, 166).
foreach my $id1 ( keys %fasta_hash1 ) {
        foreach my $id2 ( keys %fasta_hash2 ) {
                if ($id1 eq $id2 ) {
                        if ($fasta_hash1{$id1} ne $fasta_hash2{$id2}) {
                                my $outfile = "$id1". ".fasta";
                                open(my $out, " > $outfile") or die "error creating $outfile. $!";
                                print $out ">$id1-1\n", "$fasta_hash1{$id1}\n";
                                print $out ">$id2-2\n", "$fasta_hash2{$id2}\n";
                                my $filename = "$id1".".fasta";
                                my $string = 'muscle -in '.$filename.' -clw -out muscle.ma'.$id1 ;
        system ($string);

                        }
                }
        }
}
```

4 align_format.pl

```
#!/usr/bin/Perl
# Title: Perl script to insert the gene name in each line of alignment of BIGSdb

# Description: the main goal of script is to insert the gene name in each line of alignment file of variable genes that was produced from BIGSdb (strain name|gene name DNA sequence)

#Date: 29/June/2015
#UpDate: 19 Dec 2016

# All variables and hash were defined
use strict;
use warnings;
my $key;
my $id1;
my $id2;
my %Similarity;
my $outfile = "concatenatedseq.clustal";
open(my $out, " > $outfile") or die "error creating $outfile. $!"; #open  output file
my $filename1= $ARGV[0]; #command line argument to open file
die "Need alignment file output of BIGSdb\n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV

open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file


# Match gene name then save it in variable. Then match name of each isolate and DNA sequence by which sequence hold in a value of hash and name of isolates hold in scalar variable $key. Finally, print (strain name|gene name DNA sequence). The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).
while (<File1>) {
        if ($_ =~ /(NMB\d+)/){
                $id1=$1;
        }
        if ($_ =~ /^(\d+)(\S+)\s+([GTCAgtca-]+)/){
                $Similarity{$1} ="$3";
                $key= $1 ;
                print $out "$key" .'|'. "$id1" . "  " . "$Similarity{$1}\n";
        }
        if ($_ =~ /(\*+)/){
                $id2=$1;
                print $out "            $id2\n"."\n";
        }
}
close File1;
```

5 fasta_format.pl

```
#!/usr/bin/Perl
# Title: Perl script to convert alignment file of BIGSdb into multifasta file

# Description: the main goal of script is to concatenate sequence of each particular variable gene of each isolate from alignment file of BIGSdb and print them as a fasta format (header : >name of isolates|name of gene) and (DNA sequence)

#Date: 29/June/2015
#UpDate: 19 Dec 2016

# All variables and hash were defined
use strict;
use warnings;
my $id1;
my %Similarity;
my $outfile = "concatenatedseq.fasta"; # variable for output file
open(my $out, " > $outfile") or die "error creating $outfile. $!"; #open  output file
my $filename1= $ARGV[0]; #command line argument to open file
die "Need output file of align_format as input\n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV

open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file


# Match  each (isolate name| gene name) and DNA sequence, then assign each (isolate name| gene name) into a key of hash and concatenates sequence to appropriate name of (isolate name| gene name) with new line character in value of hash. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).
while (<File1>) {
        if ($_ =~ /(\S+)\s+([-GTCAgtca]+)/){
                $id1= $1 ;
                $Similarity{$id1} .="\n$2";
        }
}
close File1;


# Loop through key-value pair of hash and print them in fasta format. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 182, 166).
foreach my $id1 ( keys %Similarity ) {
        print $out ">$id1\n" . "$Similarity{$id1}\n";
}
```

6 extr_gen_isolate.pl

```
#!/usr/bin/Perl
# Title: Perl script to extract variable genes for each particular isolate from output of fasta_format.pl script


# Description: the main goal of script is to extract variable genes for each particular isolate from output of fasta_format.pl script and print them in a separated file

#Date: 29/June/2015
#UpDate: 19 Dec 2016


# All variable and hash were defined
use warnings;
use strict;
my $id1;
my $id2;
my $id3;
my $flaq2=0;
my %fasta_hash1;
my ($filename1, $filename2) = @ARGV;    #command line argument to open file
die "Need multifasta files of many variable genes of many isolates (output of fasta_format.pl) \n" if (@ARGV != 2); # test whether there is one argument by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open  file
open (File2, $filename2) or die "Couldn't open $filename2: $!";
# Loop through file, remove end line character (\n), match header, key hold header or match DNA sequence, value hold sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 182, 166).
while ( <File1> ) {
                chomp;
                if($_ =~ /^>(.+)/){
                $id1 = $1;
                 }else{
                $fasta_hash1{$id1}.= $_;
    }
}
close File1;
# Loop through file2, match header, scalar variable hold header and set the  $flaq2 to 1
while (my $row = <File2>) {
                if ($row =~ /(\S+)/){
                                $id2=$1;
                                $flaq2=1;
                }


# Loop through each key-value pair, match a particular isolate name of all variable genes, then unpack function used to break string into chunks using specific template and each chunk is separately converted to a value. The template is "(A60)*" which means for each 60 characters (A) unpack with spaces. Then join each 60 character of string with new line character. Finally, print gene name and their sequence for a particular
```

isolate and reinitialize $flaq2 for each isolate matched. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 182, 166) and (https://perldoc.perl.org/perlsub.html).

```
        if ($flaq2==1) {
                foreach my $id1 ( sort keys %fasta_hash1 ) {
                        if ($id1 =~ /(\d+)(.+)/){
                                $id3 = $1;
                                if ($id3 eq $id2) {
                                        my $outfile1 = "$id3.fasta";
                                        open(my $out1, " >>$outfile1") or die "error creating $outfile1. $!";
                                        $fasta_hash1{$id1} = join("\n", unpack("(A60)*", $fasta_hash1{$id1}));
                                        print $out1 ">$id1\n" , "$fasta_hash1{$id1}\n";
                                }
                        }
                }
        }
        $flaq2=0;
}
close File1;
```

7 identify_nu_copy.pl

```
#!/usr/bin/Perl
# Title: Perl script to identify variable gene with more than one copy in the genome

# Description: the main goal of script is to identify variable genes with more than one copy in the genome by blast them against MC58. The output of blast result will be checked if any gene has more than one score with statistical significant, this gene considered as found with more than one copy.

#Date: 29/June/2015
#UpDate: 19 Dec 2016
# All variables were defined
use warnings;
use strict;
my $id1;
my $string;
my $id2;
my $hit;
my $idE;
my $Ld;
my $idp;
my $hit2=0;
my $outfile2= $ARGV[0]; #output of gene has more than one copy
die "Need name of output, MC58.fasta and 27497.fasta should be in the directory with script \n" if (@ARGV != 1); # test if one argument by checking the length of @ARGV
my $filename2 = "BLAST_result1.txt"; #output blast result
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open  output file for writting

# Blast MC58 against all variable genes of reference genome (27497). The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 206).
system ('blastn -query MC58.fasta -subject 27497.fasta > BLAST_result1.txt');

# Open output file of blast result, loop in file, save the percentage of length of alignment, identifier, E-value and percentage of identity in different variables. If the percentages of length of alignment, percentage of identity are more than 85% and 90% respectively, with more than two hits for the same gene means these genes have more than one copy in the genome and print them with their criteria. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).
open (File2, $filename2) or die "Couldn't open $filename2: $!";
while (my $row = <File2>) {
        if ($row =~ /Length=(\d+)/){
                $hit2++;
                if ($hit2 > 2) {
                        $Ld = $1;

                }
        }elsif ($row =~ /(Subject= )(.+)/){
                $hit2++;
                $hit =0;
```

```
		$id2 = $2;

	}elsif ($row =~ /( Score =)\s+(\d+)( bits \(\d+\),  )(Expect = )(\S+)/){
		$idE = $5;
		$hit++;
		$hit2 =0;

	}elsif ($row =~ /( Identities =)\s+(\d+)(\/)(\d+)\s+\((\d+)%\)(.+)/){
		my $idI = $5;
		my $idL = $2;
			$idp =($idL*100)/$Ld;
			if ($idp > 85){
				if ($idI > 90){
					my  $rounded = sprintf("%.2f", $idp);
					print $out2 "$id2: percentage of alignment length:$rounded%\tidentity:$idI\tE-
value:$idE\n";

				}
			}

		}
	}
close File2;
```

8 check_omittedgene.pl

```
#!/usr/bin/Perl
# Title: Perl script to identify variable gene listed in appendix 10 for our isolates

# Description: the main goal of script is to identify if variable genes among our isolates CC-174 contained some genes that mostly predicted as phase variable genes therefore those genes were omitted from analysis

#Date: 29/June/2015
#UpDate: 19 Dec 2016

# All variables were defined
use warnings;
use strict;
my $id1;
my $string;
my $id2;
my $hit;
my $idE;
my $Ld;
my $idp;
my $hit2=0;
my $filename1= $ARGV[0]; #command line argument to open file
die "Need 27497gen.fasta contain varied genes detected by genome comparator, omitted_gene_seq.fasta should be in the same directory\n" if (@ARGV != 1); # test if one argument by checking the length of @ARGV
my $outfile= "one_copy.fasta";
my $outfile2= "Nu_copy.txt";
my $filename2 = "BLAST_result1.txt";
open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file for second file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open  output file for extracted seq of whole contigs for each isolate
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file


# Loop through file, remove end line character (\n), match header, variable hold header, print concatenated sequence in one_copy.fasta file, close the file handle. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).
 while ( <File1> ) {
                chomp;
                if($_ =~ /^>(.+)/){
                                $id1 = $1;
                 }else{
                                print $out1 $string.= $_;
                }
}
close File1;
```

Blast omitted genes against our variable genes hold in reference sequence ID: 27497. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 206).

```
system ('blastn -query one_copy.fasta -subject omitted_gene_seq.fasta > BLAST_result1.txt');

#open blast output file
open (File2, $filename2) or die "Couldn't open $filename2: $!";
```

#Open output file of blast result, loop in file, save the percentage of length of alignment, identifier, E-value and percentage of identity in different variables. If the percentage of length of alignment, percentage of identity are more than 85% and 90% respectively, means these genes have found in the list of 46 genes and print them with their criteria. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).

```
while (my $row = <File2>) {
        if ($row =~ /Length=(\d+)/){
                $hit2++;
                if ($hit2 > 2) {
                        $Ld = $1;

                }
        }elsif ($row =~ /(Subject= )(.+)/){
                $hit2++;
                $hit =0;
                $id2 = $2;

        }elsif ($row =~ /( Score =)\s+(\d+)( bits \(\d+\),  )(Expect = )(\S+)/){
                $idE = $5;
                $hit++;
                $hit2 =0;

        }elsif ($row =~ /( Identities =)\s+(\d+)(\/)(\d+)\s+\((\d+)%\)(.+)/){
                my $idI = $5;
                my $idL = $2;
                if ($hit==1){
                        $idp =($idL*100)/$Ld;
                        if ($idp > 85){
                                if ($idI > 90){
                                        my  $rounded = sprintf("%.2f", $idp);
                                        print $out2 "$id2: percentage of alignment length:$rounded%\tidentity:$idI\tE-value:$idE\n";

                                }
                        }

                }
        }
}
close File2;
```

10 Extract_CD.pl

```
#!/usr/bin/Perl
#Title: Perl script to extract genic regions of embl format of different contigs of whole genome of N. meningitidis

# Description: the main idea of script is to extract index of starting points and ending points of each CD in forward and reverse strands with full sequence and gene names then save them in an array. Then subtracting the ending point of first CD from starting point of same CD refers into genic region. If the sequence of any genic region was in opposite direction comparable with the rest of sequences of genic regions, the script is going to make reverse complement then print it.

#Date: 29/June/2015
#UpDate: 9/Dec/2016

# All the variables, arrays were defined
use strict;
use warnings;
my $FullSequence1;
my $FinalResult = "";
my $FullSequence;
my $seq = 0;
my $line=0;
my @StartPoints;
my @EndPoints;
my @GenName;
my @RF;


# Define and open the directory that will hold all the files in embl format, an array to hold all the files name in the directory and loop to read each file within the folder then check if the file is .embl file, open them otherwise files will be ignored. All the variables and arrays reinitialized to avoid overwritten in the final output file. The script functions in this block are adapted from (http://perldoc.perl.org/functions/readdir.html).
my $directory = $ARGV[0];
die "Need the path of your directory that hold the embl files \n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
opendir(DIR,$directory);
my @files = readdir(DIR);
closedir(DIR);
foreach(@files){
        if ($_ =~ /.embl$/) {
                my $filename = $directory . '/'.$_;
                open(my $fh, '<:encoding(UTF-8)', $filename)or die "Could not open file '$filename' $!";
                @StartPoints = ();
                @EndPoints = ();
                @GenName = ();
                @RF = ();
                $seq = 0;
                $FullSequence = "";
```

This bit match when the sequence matched and then reach into ending of sequence with (//), match the sequence lines in embl (small pieces of sequence separated by spaces), replace space with nothing, then concatenated full sequence and assign it into variable. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 191, 166).

```
while (my $row = <$fh>) {
    if($seq > 0){
        if ( $row =~ /(\s+)((\w+\s+)+)(\s+)(\d+)/){
            $FullSequence1 = $2;
            $FullSequence1 =~ s/\s+//g;
            $FullSequence = $FullSequence . $FullSequence1;

        }
    }
```

Match lines contained starting points and ending points in forward strands, push starting points into an array, push ending points into another an array, specify the CD is forward strands. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 136, 166).

```
    if ($row =~ /^(FT   CDS)\s+(\d+)(..)(\d+)/){
        push(@StartPoints, $2);
        push(@EndPoints, $4);
        push(@RF, "F");

    }
```

Match lines contained starting points and ending points in reverse strands, push starting points into an array, push ending points into an array, specify the CD is reverse strands. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 136, 166).

```
    elsif ($row =~ /^(FT   CDS)(\s+)(complement\()(\d+)(..)(\d+)/){
        push(@StartPoints, $4);
        push(@EndPoints, $6);
        push(@RF, "R");
    }
```

Match gene name and save it in specific array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 136, 166).

```
    if ($row =~ /(\S+)(gene=")(\S+)/){
        push(@GenName, $3);
    }
```

#Setting the Sequence to true (seq become 1) at file starts with gene sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 124, 166).

```
    if ($row =~ /^SQ   Sequence/){
        $seq = 1;
    }
}
```

Check the length of Starting point in an array, loop is to go through all points, check if there is some value between ending index of current gene and starting index of same gene, if there is some value extracts the resulting value, if the contigs ordered in reverse (ie starting with

NEIS1906 then NEIS1905 then NEIS1904 so on, make reverse complement then go to print function with four situations explained below. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 166) and (https://perldoc.perl.org/perlsub.html).

```
my $TNumber = scalar @StartPoints;
for (my $b = 0; $b < $TNumber-1; $b++){
my $sequ = "";
my $revcomp = "";
if(($EndPoints[$b] - $StartPoints[$b]) > -1){
$sequ = substr($FullSequence, $StartPoints[$b], ($EndPoints[$b]-$StartPoints[$b]));
$line = $line+1;
if(($GenName[$b] gt $GenName[$b+1])){
$revcomp = reverse($sequ);
$revcomp =~ tr/ACGTacgt/TGCAtgca/;
# In all situation print the gene name
if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /R/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n" . $revcomp . "\n";
}

if($RF[$b] =~ /F/ && $RF[$b+1] =~ /F/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n". $revcomp . "\n";
}

if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /F/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n" . $revcomp . "\n";

}

if($RF[$b] =~ /F/ && $RF[$b+1] =~ /R/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n". $revcomp . "\n";
}
}else {
# In all situation print the gene name
if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /R/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n" . $sequ . "\n";
}

if($RF[$b] =~ /F/ && $RF[$b+1] =~ /F/){
$FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n". $sequ . "\n";
```

```
                                                }

                                                if ($RF[$b] =~ /R/ && $RF[$b+1] =~ /F/){
                                                        $FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n" . $sequ . "\n";
                                                }

                                                if($RF[$b] =~ /F/ && $RF[$b+1] =~ /R/){
                                                        $FinalResult = $FinalResult . '>' . "$GenName[$b]_$line\n". $sequ . "\n";
                                                }
                                        }

                                }

                        }
                }
 # For writing into file and print the result then close it. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 171)
my $FileName = 'genic.fasta';
open(my $fh, '>', $FileName) or die "Could not Open File '$FileName' $!";
print $fh $FinalResult;
close $fh;
}
```

11 allele_comp.pl

```
#!/usr/bin/Perl
# Title: Perl script to detect if the variable genes identified by AC method had already been picked up by GC method

# Description: the main goal of script is to detect if the variable genes identified by AC had already been picked up by GC. The script is going to blast each variable gene detected by AC for reference sequence (ID: 27497) against concatenated sequence of varied genes detected by GC for the same reference sequence (ID: 27497). Then varied genes did not show significant match which means detected only by AC.

#Date: 29/June/2015
#UpDate 28/Jan/2017

# All variables were defined
use warnings;
use strict;
my $id1;
my $string;
my $id2;
my $hit;
my $idE;
my $Ld;
my $idp;
my $hit2=0;
my $filename1= $ARGV[0]; #command line argument to open file
die "Need 27497gen.fasta containing varied genes detected by genome comparator\n" if (@ARGV != 1); # test if one argument by checking the length of @ARGV
my $outfile= "one_copy.fasta";
my $outfile2= "Nu_copy.txt";
my $filename2 = "BLAST_result1.txt";
open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file for second file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open  output file for extracted seq of whole contigs for each isolate
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file


# Loop through file, remove end line character (\n), match header, variable hold header, and print concatenated sequence in one_copy.fasta, close the file handle. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).
while ( <File1> ) {
        chomp;
        if($_ =~ /^>(.+)/){
                $id1 = $1;
         }else{
                print $out1 $string.= $_;
        }
}
close File1;
```

Blast concatenated sequence of varied genes detected by GC against varied genes detected by AC. The script functions in this block are adapted from (https://perldoc.perl.org/perlsub.html).

```
system ('blastn -query one_copy.fasta -subject allele_comp_data25CC174.fasta > BLAST_result1.txt');

# Open blast output file
open (File2, $filename2) or die "Couldn't open $filename2: $!";


# Loop in file, save the percentage of length of alignment, identifier, E-value and percentage of identity in different variables. If the percentage of length of alignment, percentage of identity are more than 85% and 90% respectively, means these genes have already been detected in genome comparator and print them with their criteria.  The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).
while (my $row = <File2>) {
	if ($row =~ /Length=(\d+)/){
		$hit2++;
		if ($hit2 > 2) {
			$Ld = $1;


		}
	}elsif ($row =~ /(Subject= )(.+)/){
		$hit2++;
		$hit =0;
		$id2 = $2;

	}elsif ($row =~ /( Score =)\s+(\d+)( bits \(\d+\),  )(Expect = )(\S+)/){
		$idE = $5;
		$hit++;
		$hit2 =0;

	}elsif ($row =~ /( Identities =)\s+(\d+)(\/)(\d+)\s+\((\d+)%\)(.+)/){
		my $idI = $5;
		my $idL = $2;
		if ($hit==1){
			$idp =($idL*100)/$Ld;
			if ($idp > 85){
				if ($idI > 90){
					my  $rounded = sprintf("%.2f", $idp);
					print $out2 "$id2: percentage of alignment length:$rounded%\tidentity:$idI\tE-value:$idE\n";

				}
			}

		}
	}
}
close File2;
```

12 concate_embl.pl

```
#!/usr/bin/Perl
# Title: Perl script to concatenate all the sequence from all EMBL files of each isolate
```

Description: main idea of script is to concatenate all the sequence from all EMBL files of each isolate to get the full genome sequence of each isolate.

```
#Date: 29/June/2015
#UpDate: 9/Dec/2016

# All variables were defined
use strict;
use warnings;
my $line=0;
my $FullSequence1;
my $FullSequence;
my $seq = 0;
my $outfile = "concatenatedseq.fasta"; # variable for output file
open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file
```

Define and open the directory that will hold all the files in embl format, an array to hold all the files name in the directory and loop to read each file within the folder then check if the file is .embl file, open them otherwise files will be ignored. All the variables reinitialized to avoid overwritten in the final output file. The script functions in this block are adapted from (http://perldoc.perl.org/functions/readdir.html).

```
my $directory = $ARGV[0];
die "Need the path of your directory that hold the embl files \n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
opendir(DIR,$directory);
my @files = readdir(DIR);
closedir(DIR);
foreach(@files){
        if ($_ =~ /.embl$/) {
                my $filename = $directory . '/'.$_;
                open(my $fh, '<:encoding(UTF-8)', $filename)or die "Could not open file '$filename' $!";
                $seq = 0;
                $FullSequence = "";
```

This bit match when the sequence matched and then reach into ending of sequence with (//), match the sequence lines in embl (small pieces of sequence separated by spaces), replace space with nothing, then concatenated full sequence and assign it into variable. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).

```
                while (my $row = <$fh>) {
                        if($seq > 0){
                                if ( $row =~ /(\s+)((\w+\s+)+)(\s+)(\d+)/){
                                        $FullSequence1 = "$2";
                                        $FullSequence1 =~ s/\s+//g;
```

```
                                        $FullSequence = $FullSequence . $FullSequence1;
                                    }
                                }
```

Setting the Seq to true (seq become 1) at file starts with Gene sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 124).

```
                                if ($row =~ /^SQ   Sequence/){
                                    $seq = 1;
                                }
                            }
# Print the concatenated sequence of all embl files
print $out1 "$FullSequence\n";
                }
}
```

13 ext_seq.pl

```
#!/usr/bin/Perl
# Title: Perl script to extract sequence of varied genes detected by AC method for each isolate belongs into CC-174

# Description: the main goal of script is to extract sequence of varied genes detected by AC for each isolate belongs into CC-174. The full sequence of each isolates was extracted from embl file using (concate_embl.pl script). Then script is going to blast full sequence of each isolates against sequence of varied genes detected by AC for reference genome (ID: 27497). Finally, from blast output, sequence of varied genes detected by AC for each isolate was extracted.

#Date: 29/June/2015
#UpDate 28/Jan/2017

# All variables were defined
use warnings;
use strict;
my $id2;
my $id3;
my $hit;
my $hit2=0;
my $filename1= $ARGV[0]; #command line argument to open file
die "Need allele_notcomp.fasta containing varied genes detected by allele comparator for reference genome (ID: 27497) \n" if (@ARGV != 1); # test if one argument by checking the length of @ARGV
my $outfile2= "Nu_copy.fas";
my $filename2 = "BLAST_result1.txt";
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; # Open  for output file

# Blast full sequence of each isolates that contained real variation against sequence of varied genes detected by AC for reference genome (ID: 27497). The script functions in this block are adapted from (https://perldoc.perl.org/perlsub.html).
system ('blastn -query fullseq_each_isolate.fasta -subject allele_notcomp.fasta > BLAST_result1.txt');

# Open blast output file
open (File2, $filename2) or die "Couldn't open $filename2: $!";


# Loop in a file, match gene name and expected value of alignment then set counter to zero ($hit =0) and increases counter by one for each gene name and expected value of alignment ($hit2++), assign gene name and expected value of alignment to scalar variable $id2 = $2. Match score of each particular gene in alignment then if ($hit2==1) print gene name, set counter to zero ($hit2 =0) and increases counter by one ($hit++). Finally, match query part to extract their sequence and concatenates sequence of alignment to scalar variable $id3. = $4; then if ($hit==1) print concatenates sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).
while (my $row = <File2>) {
        if ($row =~ /(Subject= )(.+)/){
                $hit =0;
                $hit2++;
                $id2 = $2;

        }elsif ($row =~ /( Score =.+)/){
```

```
                if ($hit2==1){
                        print $out2 ">$id2\n";
                }
                $hit2 =0;
                $hit++;
        }elsif ($row =~ /( Query  )(\d+)(\s+)([-GTCAgtca]+)(\s+)(\d+)/){
                my $id3.= $4;
                if ($hit==1){
                        print $out2 "$id3\n";
                }
        }
}
```

14 identify3_orgi.pl

```
#!/usr/bin/Perl
# Title: Perl script to extract the same identifier and sequences of each three variable genes from three multifasta files and aligned them

# Description: the main goal of script is to compare the DNA sequences for each gene of the same identifier from three different multifasta files. One of the multifast file used as a reference genome that is (27497) and compared with other two multifasta file of isolates of CC-174. Then, the variable sequence between each three genes from three different multifasta files will be printed out in one file. Finally, the script is going to align variable sequence in each gene belongs into three different files and print them in separate files.

#Date: 29/June/2015
#UpDate: 9/Dec/2016

# All variables and hashes were defined
use warnings;
use strict;
my $id1;
my $id2;
my $id3;
my %fasta_hash1;
my %fasta_hash2;
my %fasta_hash3;
my ($filename1, $filename2, $filename3) = @ARGV;
die "Need three multifasta files reference genome (ID: 27497) with two CC-174 isolates \n" if (@ARGV != 3); # test whether there were three arguments by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file
open (File3, $filename3) or die "Couldn't open $filename3: $!"; #open first file


# In each file in the three multifasta files, while loop is to loop through all the gene names and sequences and chomp them. Then match header of fasta, save strain name in a key of hash and push them into an array, in case it did not match header of fasta, it will match the sequence then save sequence in a value of hash and push them into another an array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

while ( <File1> ) {
                chomp;
   if($_ =~ /^>(.+)/){
                                $id1 = $1;
                }else{
                $fasta_hash1{$id1} .= $_;
                }
}
close File1;

while ( <File2> ) {
                chomp;
```

```
   if($_ =~ /^>(.+)/){
                $id2 = $1;
              }else{
      $fasta_hash2{$id2} .= $_;
            }
}
close File2;

while ( <File3> ) {
            chomp;
   if($_ =~ /^>(.+)/){
                $id3 = $1;
   }else{
      $fasta_hash3{$id3} .= $_;
   }
}
close File3;
```

Loop through key-value pair belongs into different multifasta files, the reference genome (27497) and two other isolates of CC-174, then comparison of keys was carried out, the same keys from three different files, their values (sequences) were compared. If different sequence in one of each three different files was detected, the sequence from three different files were aligned and printed. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

```
foreach my $id1 ( keys %fasta_hash1 ) {
            foreach my $id2 ( keys %fasta_hash2 ) {
                        foreach my $id3 ( keys %fasta_hash3 ) {
                                    if ($id1 eq $id2 && $id1 eq $id3  ) {
                                                if ($fasta_hash1{$id1} ne $fasta_hash2{$id2} || $fasta_hash1{$id1} ne $fasta_hash3{$id3} ) {
                                                            my $outfile = "$id1". ".fasta";
                                                            open(my $out, " > $outfile") or die "error creating $outfile. $!";
                                                            print $out ">$id1-27497\n", "$fasta_hash1{$id1}\n";
                                                            print $out ">$id2-27506\n", "$fasta_hash2{$id2}\n";
                                                            print $out ">$id3-27517\n", "$fasta_hash3{$id3}\n";
                                                            my $filename = "$id1".".fasta";
                                                            my $string = 'muscle -in '.$filename.' -clw -out muscle.ma'.$id1 ;
                                                system ($string);
                                                }
                                    }
                        }
            }
}
```

15 final_gap_remov.pl

```
#!/usr/bin/Perl
# Title: Perl script to identify gaps at the beginning and at the end of alignment and remove them

# Description: the main goal of script is to identify gaps at the beginning and at the end of alignment and remove them while keep other gaps in other positions and SNPs. The script is going to take each gene name from file and doing the work on alignment files that specified in a particular directory

#Date: 29/June/2015
#UpDate: 15/Jan/2017

# All variables, arrays and hashes were defined
use warnings;
use strict;
my $id2;
my $flag=0;
my $flaq2=0;
my $value;
my $valueq;
my $valuee;
my $value2;
my $key;
my $value3;
my $value4;
my $TNumber2;
my $TNumber1;
my @check1;
my @check2;
my @check3;
my @check4;
my %Similarity;
my $outfile= "IGR2.clustal"; #output file
# Open output file for writing, define the directory that will hold all the files in alignment format, open gene name file
open(my $out1, " > $outfile") or die "error creating $outfile. $!";
my $directory = $ARGV[0];
die "Need the path of your directory that hold the muscle.ma files \n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
my $filename1= "28253_20072c.txt";
open (File1, $filename1) or die "Couldn't open $filename1: $!";


# Loop through gene name file, match gene name, variable hold gene name, print gene name, set the flaq2 to one for each gene name. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 1191).

while (my $row = <File1>) {
        if ($row =~ /(\S+)/){
```

```
		$id2=$1;
		print $out1 "#$id2\n\n";
		$flaq2=1;
}
```

Define and open the directory that will hold all the files in alignment format, array to hold all the files name in the directory and loop to read each file within the folder then check if the file is muscle.ma$id2 file, open them otherwise files will be ignored. All the variables, an arrays and hash reinitialized to avoid overwritten in the final output file. The script functions in this block are adapted from (http://perldoc.perl.org/functions/readdir.html).

```
if ($flaq2==1) {
		opendir(DIR,$directory);
		my @files = readdir(DIR);
		closedir(DIR);
		foreach(@files){
			if ($_ =~ /muscle.ma$id2/) {
				my $filename = $directory . '/'.$_;
				open(my $fh, '<:encoding(UTF-8)', $filename)or die "Could not open file '$filename' $!";
				$value="";
				$value2="";
				$value3="";
				$value4="";
				$valueq="";
				$valuee="";
				%Similarity=();
				$key="";
				$flag=0;
				@check1=();
				@check2=();
				@check3=();
				@check4=();
				$TNumber2="";
				$TNumber1="";
```

Loop through alignment file for the first three lines of alignment of each gene, match name of each isolate and DNA sequence, sequence hold in value of hash, assign name of gene-isolate into variable key, Then push value into array, after the first three lines get matched , the flaq became 1. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
				while (<$fh>) {
					if ($flag==0) {
						if ($_ =~ /(\S+)\s+([-GTCA]+)/){
							$Similarity{$1} =$2;
							push (@check3, $Similarity{$1});
							push (@check4, $1);
```

```
                }
        }
        if ($_ =~ /(\*+)/) {
                $flag=1;
        }
```

Match name of each isolate and DNA sequence for the rest of lines in the alignment file, sequence hold in value of hash, assign name of gene-isolate name into variable key, push value into array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
        if ($flag==1) {
                if ($_ =~ /(\S+)\s+([-GTCA]+)/){
                        $Similarity{$1} =$2;
                        push (@check1, $Similarity{$1});
                        push (@check2, $1);

                }

        }
}
```

Check the length of key for the first three lines of alignment, if any one of three lines has gap at the beginning, remove first character for first line, remove first character for second line, remove first character for third line then print gene-isolate name and sequence without first character else print gene-isolate name and sequence with first character. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 166) and (https://perldoc.perl.org/perlsub.html).

```
$TNumber1 = scalar @check4;
for (my $b = 0; $b < $TNumber1; $b++){
        if ($check3[0]=~ /^(-)/  || $check3[1]=~ /^(-)/ || $check3[2]=~ /^(-)/ ){
                $value=     substr($check3[0], 1);
                $valueq=    substr($check3[1], 1);
                $valuee=    substr($check3[2], 1);
                print  $out1 "$check4[0]" ."   ". "$value\n\n";
                print  $out1 "$check4[1]" ."   ". "$valueq\n\n";
                print  $out1 "$check4[2]" ."   ". "$valuee\n\n";
                last,
        }else {
        print  $out1 "$check4[$b]" ."   ". "$check3[$b]\n\n";
        }
}
```

Check the length of key for the rest lines of alignment, print gene-isolate name and sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 166).

```
$TNumber2 = scalar @check2;
for (my $b = 0; $b < $TNumber2-3; $b++){
        print  $out1 "$check2[$b]" ."   ". "$check1[$b]\n\n";
}
```

If any one of last three lines has gap at the end, last character for the last three lines was remove, gene-isolate names and sequences without last character were printed else gene-isolate names and sequences with last character were printed. The script functions in this block are adapted from (Bradnam and Korf, 2012) (page 166) and (https://perldoc.perl.org/perlsub.html).

```
if ($check1[-1]=~ /-$/  || $check1[-2]=~ /-$/ || $check1[-3]=~ /-$/ ){
        my $value2=         substr($check1[-1],0, -1);
        my $value3=         substr($check1[-2],0, -1);
        my $value4=         substr($check1[-3],0, -1);
        print $out1 "$check2[-1]" ."   ". "$value2\n\n";
        print $out1 "$check2[-2]" ."   ". "$value3\n\n";
        print $out1 "$check2[-3]" ."   ". "$value4\n\n";
}else {
        print $out1 "$check2[-1]" ."   ". "$check1[-1]\n\n";
        print $out1 "$check2[-2]" ."   ". "$check1[-2]\n\n";
        print $out1 "$check2[-3]" ."   ". "$check1[-3]\n\n";
}
close $fh;
                }
        }
    }
$flaq2=0;
}
```

16 replace_fullseq.pl

```
#!/usr/bin/Perl
# Title: Perl script to edit the reference genome ID: 27497 by inserting the real variable SNPs of each CC-174 isolates
```

Description: the main goal of script is to replace sequence of each particular gene from reference genome ID: 27497 by the sequence of gene from each CC-174 isolate for the same identifier. In practical, the genes in the first file (ID: 27497) are matched with full genome sequence of reference genome (ID: 27497) in the third file then they were substituted by the genes from second file (each CC-174 isolates) for the same identifier. After substitution, the full genome sequence of reference genome was printed therefore in this case the full genome sequence of reference genome will keep only the real variation that came from second file.

```
#Date: 29/June/2015
#UpDate: 9/Dec/2016



# All variables and hashes were defined
use warnings;
use strict;
my $id1;
my $id2;
my $id3;
my %fasta_hash1;
my %fasta_hash2;
my $outfile = "seq_var_.fasta"; # variable for output file
open(my $out, " > $outfile") or die "error creating $outfile. $!"; #open each output file
my ($filename1, $filename2, $filename3) = @ARGV;
die "Need first file ( variable gene ID: 27497) second file (variable gene each isolate CC-174) reference file (whole genome sequence ID: 27497\n" if (@ARGV != 3); # test whether there were three arguments by checking the length of @ARGV
my $outfile2 = "not_math.fasta"; # variable for output file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open each output file
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file
open (File3, $filename3) or die "Couldn't open $filename3: $!"; #open first file
```

In first and second files while loop is to loop through all the gene names and sequences, chomp them and matches header of fasta, save strain name in key of hash and push them into an array, in case it did not match header of fasta, it will match the sequence then save sequence in value of hash. In the third file while loop is to match whole sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182, 191).

```
while ( <File1> ) {
        chomp;
   if($_ =~ /^>(.+)/){
                $id1 = $1;
        }else{
        $fasta_hash1{$id1} .= $_;
        }
}
```

```
close File1;

while ( <File2> ) {
        chomp;
   if($_ =~ /^>(.+)/){
          $id2 = $1;
        }else{
      $fasta_hash2{$id2} .= $_;
        }
}
close File2;

while ( <File3> ) {
        chomp;
        if($_ =~ /(\S+)/){
          $id3.= $1;
        }
}
close File3;
```

Loop through each key and value of first and second hashes, convert the sequence to lower case, for each key of first hash replace sequence of first hash with sequence of second hash for same identifier, print success if the replaced was done. Otherwise print the name of keys and the sequences of second hash for the genes that were not replaced. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182, 191).

```
foreach my $id1 ( sort keys %fasta_hash1 ) {
        $fasta_hash1{$id1}=lc $fasta_hash1{$id1};
        $fasta_hash2{$id1}=lc $fasta_hash2{$id1};
        if ($id3 =~ s/$fasta_hash1{$id1}/$fasta_hash2{$id1}/g) {
                print "success for $id1\n";
        } else {print $out2 ">$id1\n" . "$fasta_hash2{$id1}\n"; }
}
```

Unpack function used to break string into chunks using specific template and each chunk is converted separately to a value. The template is "(A60)*" which means for each 60 characters (A) unpack with spaces. Then join each 60 character of string with new line character. Finally, print gene name and their sequence for a particular isolate. The script functions in this block are adapted from (https://perldoc.perl.org/perlsub.html).

```
$id3 = join("\n", unpack("(A60)*", $id3));
print $out ">ref\n"."$id3";
```

17 extr_contg.pl

```
#!/usr/bin/Perl
# Title: Perl script to extract starting points and ending points of each contig of isolate of reference genome (ID: 27497).

# Description: the main goal of script is to extract starting points and ending points of different contigs for each isolate belong into CC-174. The full sequence of reference genome (ID: 27497) was extracted from embl file using (concate_embl.pl script), then the Mummer alignment was run. Output of Mummer alignment was used as input to ABACAS script. ABACAS script ordered the reference sequence for each contig depending on MC58 as a reference genome. It inserted N to fill the gaps among contigs therefore the sequence of all contigs of reference genome separated by N hosted in first file. The script is going to remove N between gaps and assign each name of contig to the key and sequence to the value, then print them in a second file. Finally, script is going to blast full sequence of (ID: 27497) isolate that contained real variation against second file. Then all the starting points and ending points of each contig were extracted.

#Date: 29/June/2015
#UpDate: 19 Dec 2016

# All variables and hashes were defined
use warnings;
use strict;
my $id1;
my $string;
my $id2;
my $id3;
my $id4;
my $id5;
my $id8;
my $id9;
my $sta_pot2;
my $end_pot2;
my $hit;
my $hit2;
my $line=0;
my %fasta_hash8;
my %fasta_hash9;
my $filename1= $ARGV[0]; #command line argument to open file
die "Need file which is the output of  ABACAS script on reference genome 27497 (all the contigs separated by NNNNN) and reference genome ID:27497 file with real variation in the same directory\n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
my $outfile= "one_copy.fasta";
my $outfile2= "Nu_copy.fas";
my $outfile3= "points.fas";
my $filename2 = "BLAST_result1.txt";
open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file for second file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open  output file for extracted seq of whole contigs for each isolate
open(my $out3, " > $outfile3") or die "error creating $outfile3. $!";
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
```

Loop through file, remove end line character (\n), match header, key hold header or match DNA sequence, value hold sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).

```
while ( <File1> ) {
        chomp;
        if($_ =~ /^>(.+)/){
                $id1 = $1;
         }else{
                $string.= $_;
        }
}
close File1;
```

Split function used to split string into a list of strings using multi N as a pattern and returns the list in an array (remove N among contigs), In array, match sequence, $line is a counter to identify contigs number, unpack function used to break string into chunks using specific template and each chunk is converted separately to a value. The template is "(A60)*" which means for each 60 characters (A) unpack with spaces. Then join each 60 character of string with new line character. Finally, convert it to upper lower case and print the name of isolate and contig with their numbers. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191) and (https://perldoc.perl.org/perlsub.html).

```
my @string = split /[N]+/, $string;
foreach(@string) {
        if ($_ =~ /([GTCAgtca]+)/){
                $line=$line+1;
                $_ = join("\n", unpack("(A60)*", $_));
                $_ = uc $_;
                print $out1   ">27497_contig_$line\n". "$_\n";
        }
}
```

Blast full sequence of each isolates that contained real variation against second file. The script functions in this block are adapted from (https://perldoc.perl.org/perlsub.html).

```
system ('blastn -query 27497.fasta -subject one_copy.fasta > BLAST_result1.txt');
```

Open blast output file, loop in file, match identifier of each contig and their significant of alignment, once counter $hit became 1 save the identifier of each contig in a key. Then match starting points and ending points of alignment of each contig and save the identifier of each contig in a key and their starting points and ending points in values of hashes. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182, 191).

```
open (File2, $filename2) or die "Couldn't open $filename2: $!";
```

```
while (my $row = <File2>) {
	if ($row =~ /(Subject= )(.+)/){
		$id2 = $2;
		$hit =0;
		$hit2++;
	}elsif ($row =~ /( Score =.+)/){
		if ($hit2==1){
			print $out2 ">$id2\n";
		}
		$hit2 =0;
		$hit++;
	}elsif ($row =~ /(Query  )(\d+)(\s+)([-GTCA]+)(\s+)(\d+)/){
		$sta_pot2 = $2;
		$end_pot2 = $6;
		my $id3.= $4;
		if ($hit==1){
			$id8 = $id2;
			$fasta_hash8{$id8} .= "::$sta_pot2";
			$id9 = $id2;
			$fasta_hash9{$id9} .= "::$end_pot2";
			print $out2 "$id3\n";
		}
	}
}
close File2;


# Loop in hashes, extract the first value of starting point and the last value of ending points of each contig, then print the identifier of each contig in a key and their starting points and ending points in values of hashes. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).
foreach my $id8 ( sort keys %fasta_hash8 ) {
	foreach my $id9 ( sort keys %fasta_hash9 ) {
		if ($id8 eq $id9){
			if ($fasta_hash8{$id8} =~ /(::)(\d+)(.+)/ ) {
				$id4 = $2;
				if ($fasta_hash9{$id9} =~ /(\d+$)/ ) {
					$id5 = $1;
					print  $out3 "$id8\t$id4\t$id5\n";
				}

			}
		}
	}
}
```

18 extractespoint.pl

```
#!/usr/bin/Perl
# Title: Perl script to extract recombination fragments from output of ClonalFrameML program for the isolates CC-174
```

Description: the main goal of script is to extract recombination fragments from output of ClonalFrameML program, the DNA sequence of isolates of CC-174 from multifast file was assigned into value of hash to use it as offset in the substring function, while the output of ClonalFrameML program contains starting points and ending points of each recombination fragments of isolates CC-174. The script is going to loop through starting points and ending points of each recombination fragments of isolates CC-174 then doing substring function (offset) DNA sequence of isolates of CC-174 from multifast file hold in value of hash, starting points of each recombination fragments of isolates CC-174, length of sequence (ending points - starting points) of each recombination fragments of isolates CC-174. Then print them in a file with their identifier (name of isolates).

```
#Date: 29/June/2015
#UpDate 29/Jan/2017
```

```
# All variables, arrays and hashes were defined
use warnings;
use strict;
my $id1;
my %fasta_hash1;
my @StartPoints;
my @EndPoints;
my @RF;
my $filename1= $ARGV[0];
my $filename2= $ARGV[1];
die "Need two files(output of ClonalFrameML program and multifasta of isolates CC-174)\n" if (@ARGV != 2);#test whether two arguments by checking the length @ARGV
my $outfile= "contig.fasta";
open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file for second file
```

Open the file, loop through all lines of file, default of split command is splitting ($_) by space. $name, $start and $end assigned into first, second and third values in each line of output of ClonalFrameML program file, then push all the three different values (identifier isolates name, starting point and ending points of recombination fragments) into three different arrays. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 136) and (https://perldoc.perl.org/perlsub.html).

```
open (File1, $filename1) or die "Couldn't open $filename1: $!";
while (<File1>) {
        my ($name, $start, $end) = split;
        push (@RF, $name);
        push (@StartPoints, $start);
        push (@EndPoints, $end);
}
```

Open the file, loop through all lines of file, assign key into identifier isolate name and value into DNA sequence, close the file. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
open (File2, $filename2) or die "Couldn't open $filename2: $!";
while ( <File2> ) {
        chomp;
        if($_ =~ /^>(.+)/){
                $id1 = $1;
        }else{
                $fasta_hash1{$id1} .= $_;
        }
}
close File2;
```

Check the length of starting point array, loop to go through all points, $sequ: initializing, then substring function to extract the sequences of all the recombination fragments (offset sequences, starting points, and length (starting points-ending points). Then print the sequences of all the recombination fragments of isolate CC-17 in a file. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182) and (https://perldoc.perl.org/perlsub.html).

```
my $TNumber = scalar @StartPoints;
for (my $b = 0; $b < $TNumber; $b++){
        foreach my $id1 ( keys %fasta_hash1 ) {
                my $sequ = "";
                $sequ = substr($fasta_hash1{$id1}, $StartPoints[$b], ($EndPoints[$b] - $StartPoints[$b]));
                print $out1 ">$RF[$b]\n"."$sequ\n";
        }
}
```

19 extractespoint2.pl

```
#!/usr/bin/Perl
#Title: Perl script to extract mutation fragments from output of ClonalFrameML program for the isolates CC-174

# Description: Same principle as extractpoint.pl but the output file of ClonalFrameML program containing starting points and ending points of recombination fragment was modified to extract mutation fragments.

#Date: 29/June/2015
#UpDate 29/Jan/2017
#All variables, arrays and hashes were defined
use warnings;
use strict;
my $id1;
my %fasta_hash1;
my @StartPoints;
my @EndPoints;
my @RF;
my $filename1= $ARGV[0];
my $filename2= $ARGV[1];
my $outfile= 'mut.fas';
die "Need two files(modified output of ClonalFrameML program and multifasta of isolates CC-174)\n" if (@ARGV != 2);#test whether two arguments by checking the length @ARGV
my %fasta_hash1;

open(my $out1, " > $outfile") or die "error creating $outfile. $!"; #open  output file for second file

# Open the file, loop through all lines of file, default of split command is splitting ($_) by space, $name, $start and $end assigned into first, second and third values in each line of output of ClonalFrameML program file, then push all the three different values (identifier isolates name, starting point and ending points of mutation fragments) into three different arrays. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 136) and (https://perldoc.perl.org/perlsub.html).


open (File1, $filename1) or die "Couldn't open $filename1: $!";
while (<File1>) {
        my ($name, $start, $end) = split;
        push (@RF, $name);
        push (@StartPoints, $start);
        push (@EndPoints, $end);

}


# Open the file, loop through all lines of file, assign key into identifier isolate name  and value into DNA sequence, close the file. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open first file
```

```
while ( <File2> ) {
        chomp;
        if($_ =~ /^>(.+)/){
                $id1 = $1;
        }else{
                $fasta_hash1{$id1} .= $_;
        }
}
close File2;
```

Check the length of starting point array, loop to go through all points, $sequ: initializing, then substring function to extract the sequences of all the mutation fragments (offset sequences, starting points, and length (starting points-ending points). Then print the sequences of all the mutation fragments of isolate CC-17 in a file. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182) and (https://perldoc.perl.org/perlsub.html).

```
my $TNumber = scalar @EndPoints;
for (my $b = 0; $b < $TNumber; $b++){
        foreach my $id1 ( keys %fasta_hash1 ) {
                my $sequ = "";
                $sequ = substr($fasta_hash1{$id1}, $EndPoints[$b], ($StartPoints[$b+1] - $EndPoints[$b]));
                print $out1 ">$id1\n"."$sequ\n";
        }
}
```

20 extrfasta.pl

```
#!/usr/bin/Perl
#Title: Perl script to concatenate all the extracted mutation and recombination fragments of the isolates CC-174 in a separate file.

# Description: The main idea of script is to concatenate all the recombination fragments or mutation fragments that have been extracted by the extractespoint.pl or extractespoint2.pl scripts, then print them in a separate file.



#Date: 29/June/2015
#UpDate 29/Jan/2017

#All variables and hash were defined
use warnings;
use strict;
my $id2;
my %fasta_hash2;
my $outfile2= "contig2.fasta";
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open  output file for second file
my $filename3= $ARGV[0]; #command line argument to open file
die "Need multifasta file with mutation and recombination fragments)\n" if (@ARGV != 1);#test whether one argument by checking the length @ARGV
open (File3, $filename3) or die "Couldn't open $filename3: $!"; #open input file



# Loop through file, assign key-hash pair into identifier and concatenated sequence respectively. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

while ( <File3> ) {
        #chomp;
        if($_ =~ /^>(\d+$)/){
                        $id2 = $1;

                }else{
                $fasta_hash2{$id2} .= $_;

                }
        }


# Loop through hash and print the identifier and sequence in fasta format. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

foreach my $id2 (sort keys %fasta_hash2 ) {

        print $out2 ">$id2\n"."$fasta_hash2{$id2}\n";
}
```

21 index_var.pl

```
#! /usr/bin/Perl
#Title: Perl script to extract the index of variables from two sequences in multifasta formats

# Description: the main goal of script is to compare the DNA sequences from two different files, and then the position of each varied base pair between two different files was printed.

#Date: 22/Aug/2016
#UpDate: 9/Dec/2016

# All variables and arrays were defined
use strict;
use warnings;
my $str_source;
my $str;
my @ind_var;
my $outfile = "comp.txt";
open(my $out, " > $outfile") or die "error creating $outfile. $!"; #open orginal seq file
my ($filename1, $filename2) = @ARGV; # two command-line arguments
die "Need file with first sequence (original sequence) and file with second sequence (editing original sequence with real variation)\n" if (@ARGV != 2); # test whether there are two arguments by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file

# Loop through file, remove end line character (\n), match header, scalar variable hold header, scalar variable hold sequence, close first file handle. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 199, 182).

while ( <File1> ) {
        chomp;
        if($_ =~ /^>(.+)/){
   my $id1 = $1;
        }else{
        $str_source.= $_;
        }
}
close File1;

# Loop through file, remove end line character (\n), match header, scalar variable hold header, scalar variable hold sequence, close second file handle. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 199, 182).
while ( <File2> ) {
        chomp;
        if($_ =~ /^>(.+)/){
   my $id2 = $1;
        }else{
        $str.= $_;
        }
```

```
}
close File2;
```

Initialize subroutine, send sequence from first file and second file for comparison between them, receive positions of each varied base pair in an array, loop in the array and print the positions of each varied base pair and the value one for each varied base pair in the form of (position of each varied base 1). The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 136, 211).

```
@ind_var = diff_index ($str_source, $str);
foreach (@ind_var) {
        my $var_pos="$_\t\t"."1\n";
        print $out "$var_pos";
}
```

Subroutine is to make comparison between two sequences and return positions of varied base pair, subroutine receives sequence from two files for comparison. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 199, 136, 215) and (https://perldoc.perl.org/perlsub.html).

```
sub diff_index{
        my @st1;
        my @st2;
        my $hit =-1;
        my $hit2 =-1;
        my $index =0;
        my $index2 =0;
        my @cmp_ind;
        my($a,$b) = @_;
        while ($a) {
                if ($a =~  /^[ATGC]/i){
                        $hit++;
                        if ($hit<=length($a)){
                                $index = substr($a,$hit,1);
                                push (@st1, $index);
                        }else {
                                last;
                        }
                }
        }
        while ($b) {
                if ($b =~  /^[ATGC]/i){
                        $hit2++;
                        if ($hit2<=length($b)){
                                $index2 = substr($b,$hit2,1);
                                push (@st2, $index2);
                        }else {
                                last;
                        }
                }
        }
        for (my $c = 0; $c < length($a); $c++){
                if ($st1[$c] ne  $st2[$c]){
                        my $posi =$c+1;
                        push (@cmp_ind, $posi);
                }
        }
return @cmp_ind;
}
```

22 sliding_win.pl

```
#!/usr/bin/Perl
# Title: Perl script to detect different recombinant fragments within sliding window approach

# Description: the main goal of script is to identify recombination fragments depending on statistical analysis of (Wong et al., 2013).Statistical analysis of variation realized that at least six point mutations were required for a length of 500 base pair to infer recombination pattern. Therefore, the script was designed as follows
# Firstly, the script is going to calculate the number of varied sequence in sliding window approach (0-500), (1-501), (2-502) so on
#Secondly, the script is going to print (stating point: number of varied sequence) for windows passed the threshold (6 varied base pair for length of 500) example (0:8:500) (1:7:501) (2:7:502) (3:7:503) (4:7:504) (5:6:505) (6:6:506) (7:6:507) (8:6:508)  (100:6:600) so on
#Thirdly, The script is going to detect the length of each recombinant fragment as follows
#the script is going to detect starting point of recombinant fragment which is the first element passed the threshold in the example above is zero. On other hand,  if the ending point that passed the threshold incremented by more than one, this is considered ending point of recombinant fragment, in the above example is 508 because ending point 600 passed the threshold after 508 which is belong into another fragment, therefore the length of first fragment starting (0-508).


#written_date: 15/Aug/2016
#UpDate: 10/Nov/2016
#UpDate : 20/Nov/2016
#UpDate: 9/Dec/2016

# All variables and arrays were defined
use warnings;
use strict;
my $pos;
my $id1;
my $starting_point2=0;
my $line2=0;
my $fraq2=0;
my $flaq2=0;
my $str_source;
my $leng;
my $posi=0;
my @poscutoff;
my @names2;
my @end2
my %count2;
my ($filename1, $filename2, $outfile1)  = @ARGV;
# test whether there are three arguments by checking the length of @ARGV
die "Need first file with position of varied seqs (output of index_var.pl script)  and second file with seq in fasta file (one of compared seq in index_var.pl script) and third file for output file (printing length of putative recombination fragments)\n" if (@ARGV != 3);
my $outfile2 = "final500.txt";      #output file for all ending point in window 500 with number of varied seq
open(my $out1, " > $outfile1") or die "error creating $outfile1. $!"; #open output file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open output file
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
```

```
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file


# While loop to loop through all (position-values) pair, scalar variable hold header, push the corresponding position of the ending point in each window into array. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 136).
while (<File1>) {
	if ($_ =~ /(\S+)/){
		$pos = $1;
		push(@poscutoff, $pos);

	}
}
close File1; #close first file

# Length of elements of array
my $TNumber = scalar @poscutoff;

# Loop through file, remove end line character (\n), match header, scalar variable hold header, scalar variable hold sequence, scalar variable hold length of DNA sequence. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).
while ( <File2> ) {
	chomp;
	if($_ =~ /^>(.+)/){
		$id1 = $1;
	}else{
		$str_source.= $_;
		$leng=length ($str_source);
	}
}
close File2; #close second file


# Loop in concatenated sequence, if each base pair is matched with DNA letters then the script is going to do the following; base pair is subtracted using (substr) function, looping through elements of position of the ending point in each window is achieved, elements of position of the ending point in each window is pushed in an array if it located between (0-500),(1-501) (2-502)so on. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 199, 136) and (https://perldoc.perl.org/perlsub.html).
while ($str_source){
	if($str_source =~ /^[ATGCatcg]/i){
		substr($str_source,0,1)='';
		for (my $b = 0; $b < $TNumber; $b++){
			if ($posi < $poscutoff[$b] && $posi+500 > $poscutoff[$b]){
				push(@names2, $posi+500);
			}

		}
			$posi ++; #counter start at 0,1,2,3

	}
```

```
}

# Count the number varied sequence (SNPs) in each ending window 500, 501, 502 so on
foreach my $element2( @names2 ) {
	$count2{$element2}++;
}

# Check if the number of varied sequence (SNPs) is larger than 6 for each ending window do the following;  get the starting point of each window 0,1,2 so on, print starting: number of varied sequence passed threshold 6. Then scalar variable $line2 is used as a counter to print the starting point of each recombinant fragment of each 500 window size. Finally, if ($starting_point2 == $fraq2-1 checking if the ending point of each window incremented by more than one, it is printed as the ending point of each recombinant fragment. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 136).
foreach my $element2( sort {$a <=> $b} keys %count2 ) {
	if ($count2{$element2} >= 6 ) {
		$starting_point2=$element2-500;
		print $out2 "$starting_point2\t$count2{$element2}\n";
		$line2 ++;
		if ($line2 == 1) {
			print $out1 "putative recombination block is detected within length starting at  $starting_point2 for 500 window\n";
			$fraq2=$starting_point2;
		}
		$fraq2=$fraq2+1;
		if ($starting_point2 == $fraq2-1){

			@end2=();
			push(@end2, $element2);
			$flaq2=1;
		}else{
			print $out1 "putative recombination block is detected within length ending at $end2[-1] for 500 window\n";
			$line2 = 0;

		}
	}
}

# Printing ending point of last recombinant fragment
if ($flaq2==1){
	print $out1 "putative recombination block is detected within length ending at  $end2[-1] for 500 window\n";

}
```

23 extract_aa.pl

```
#! /usr/bin/Perl
# Title: Perl script to extract the amino acids from GenBank format of MC58

# Description: the main goal of script is to extract header (gene name) and amino acids for each gene from GenBank format and print them as a fasta file. This file was submitted into blast search in KEGG to calculate the proportion effect of each functional scheme.
#Date: 22/Aug/2016
#Updat: 28/Jan/2017
# All variables and hash were defined
use strict;
use warnings;
my $flaq= 0;
my $row;
my $str;
my $str2;
my $str3;
my $id1;
my %fasta_hash1;
my $outfile12 = "comp12.fasta";
open(my $out12, " > $outfile12") or die "error creating $outfile12. $!";
my ($filename1) = $ARGV[0]; #command line argument for MC58 GeneBank entery
die "Need file MC58 GenBank format\n" if (@ARGV != 1); # test whether one argument provided by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
# Loop through MC58 file, $flaq became 1 once the CDS line matched, $str hold identifier while $str2 and $str3 hold amino acids sequence, all identifier and their amino acids printed in the comp12.fasta file as fasta file, #close first file handle. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191).
while ( my $row = <File1> ) {
        if ($row =~ /(FT   CDS             )(.+)/){
                $flaq= 1;
        }elsif ($row =~ /(FT                   \locus_tag=)(\S+)/){
                $str= $2;
                if ($flaq==1){
                        print $out12 ">$str\n";
                        $flaq= 0;
                }

        }elsif ($row =~ /(FT                   \translation=")([A-Z]+)/){
                $str2= $2;
                print $out12 "$str2\n";
        }elsif ($row =~ /(FT                   )([A-Z]+)/){
                $str3= $2;
                print $out12 "$str3\n";
        }
}
close File1;
```

24 Significant_CE.pl

```
#!/usr/bin/Perl
# Title: Perl script to identify the presence of CEs within the (IGRs) and the starting point and ending point of CEs pattern within the DNA sequence of the compared IGRs

# Description: there are two aims for this script, firstly, script is going to achieve BLAST search between compared IGRs individually with CE template then save the identifier of each IGR as a key and the significant of alignment as a value in key-value pair of hash. Then the script is going to loop through key-value pair of hash and print identifier of each IGR and their significant of alignment. Secondly, script is going to pull out the starting point and ending point of alignment from BLAST result as it refers into starting point and ending point of CE in the DNA sequence of each IGR regions.

#Date: 22/Aug/2016
#UpDate: 6/Feb/2016

# All variables and hashes were defined
use warnings;
use strict;
my $hit;
my $id2;
my $id3;
my $id10;
my $id11;
my $id20;
my $id21;
my $id23;
my $id22;
my $idE2;
my $idv;
my $idv2;
my $id14;
my $id15;
my $idE;
my $id12;
my $flaq1=0;
my $sta_pot;
my $end_pot;
my $id1;
my $id4;
my $id5;
my $hit2;
my $id6;
my $id7;
my $sta_pot2;
my $end_pot2;
my $id0;
my $id8;
```

```
my $id9;
my %fasta_hash1;
my %fasta_hashv2;
my %fasta_hashv;
my %fasta_hash12;
my %fasta_hash23;
my %fasta_hash11;
my %fasta_hash14;
my %fasta_hash15;
my %fasta_hash4;
my %fasta_hash5;
my %fasta_hash21;
my %fasta_hash2;
my %fasta_hash0;
my %fasta_hash8;
my %fasta_hash9;
my ($outfile1, $outfile2,$outfile3) = @ARGV; #three command-line arguments
die "Need two multifasta files for IGRs in first and second strains and file contain CE templete hosted in a same directory of script with three name of outputs files\n" if (@ARGV != 3); # test whether there are three arguments by checking the length of @ARGV

my $filename2 = "BLAST_result1.txt";
my $filename3 = "BLAST_result2.txt";

open(my $out1, " > $outfile1") or die "error creating $outfile1. $!"; #Open  output file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #Open  output file
open(my $out3, " > $outfile3") or die "error creating $outfile3. $!"; #Open  output file
```

First run BLAST search between template CE as a query and multifasta file carrying IGRs of first isolate, second run between template CE and multifasta file carrying IGRs of second isolate, The script functions in this block are adapted from (https://perldoc.perl.org/perlsub.html).

```
system ('blastn -query CE.fasta -subject 1.fasta > BLAST_result1.txt');
system ('blastn -query CE.fasta -subject 2.fasta > BLAST_result2.txt');

open (File2, $filename2) or die "Couldn't open $filename2: $!"; #Open first file
open (File3, $filename3) or die "Couldn't open $filename3: $!"; #Open second file
```

Loop in the first BLAST result, match identifier of each IGRs and their significant of alignment, once counter $hit became 1 save the identifier of each IGRs in a key and their significant of alignment in a value of hash. Then match starting points and ending points of alignment of each IGR and save the identifier of each IGR in a key and their starting points and ending points in values of hashes. This is for IGRs in first isolates. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
while (my $row = <File2>) {
        if ($row =~ /Subject=(.+)/){
                $id2 = $1;
```

```
		$hit=0;
		$flaq1=1;
}elsif ($row =~ /( Score =)\s+(\d+)( bits \(\d+\),  )(Expect = )(\S+)/){
			$id3 = $2;
			$idE = $5;
			$hit++;

			if ($idE < 1e-20){
				$id1 = $id2;
				$idv = $id2;
				$fasta_hash1{$id1} = $id3;
				$fasta_hashv{$idv} = $idE;
			}


}elsif ($row =~ /( Identities =)\s+(\d+)(\/)(\d+)\s+\((\d+)%\)(.+)/){
		$id10 = $5;
		$id20 = $2;
		$id11 = $id2;
		$id21 = $id2;
		$fasta_hash11{$id11} = $id10;
		$fasta_hash21{$id21} = $id20;

}elsif ($row =~ /(Sbjct  )(\d+)(\s+)([-GTCA]+)(\s+)(\d+)/){
		$sta_pot = $2;
		$end_pot = $6;

			if ($idE < 1e-20){
				if ($hit == 1){
					if ($idE < 1e-20){

						$id4 = $id2;
						$fasta_hash4{$id4} .= "::$sta_pot";
						$id5 = $id2;
						$fasta_hash5{$id5} .= "::$end_pot";
					}
				}elsif ($hit == 2){
					if ($idE < 1e-20){
						$id14 = $id2;
						$fasta_hash14{$id14} .= "::$sta_pot";
						$id15 = $id2;
						$fasta_hash15{$id15} .= "::$end_pot";
					}

				}elsif ($hit > 2){
					if ($idE < 1e-20){
						print "$id2 contains more than 2 CEs with significant match";
```

```
                                                }
                                        }
                                }

        }
}
```

The same process occurs for the IGRs for second isolate, see previous step of annotation. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
while (my $row = <File3>) {
        if ($row =~ /Subject=(.+)/){
                $id6 = $1;
                $hit2=0;
        }elsif ($row =~ /( Score =)\s+(\d+)( bits \(\d+\),  )(Expect = )(\S+)/){
                $id7 = $2;
                $idE2 = $5;
                $hit2++;
                if ($idE2 < 1e-20){
                        $id0 = $id6;
                        $idv2 = $id6;
                        $fasta_hash0{$id0} = $id7;
                        $fasta_hashv2{$idv2} = $idE2;
                }

        }elsif ($row =~ /( Identities =)\s+(\d+)(\/)(\d+)\s+\((\d+)%\)(.+)/){
                $id11 = $5;
                $id22 = $2;
                $id12 = $id6;
                $id23 = $id6;
                $fasta_hash12{$id12} = $id11;
                $fasta_hash23{$id23} = $id22;
                #print "$id12\t$fasta_hash12{$id12}\n";

        }elsif ($row =~ /(Sbjct  )(\d+)(\s+)([-GTCA]+)(\s+)(\d+)/){
                $sta_pot2 = $2;
                $end_pot2 = $6;
                if ($hit2==1){
                        $id8 = $id6;
                        $fasta_hash8{$id8} .= "::$sta_pot2";
                        $id9 = $id6;
                        $fasta_hash9{$id9} .= "::$end_pot2";

                }

        }
```

```
}
```

Loop in first hash of first isolate and second hash of second isolate and print (the identifier of each IGR in a key and their significant of alignment in a value) The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

```
foreach my $id1 ( sort keys %fasta_hash1 ) {
	foreach my $id11 ( sort keys %fasta_hash11 ) {
		foreach my $idv ( sort keys %fasta_hashv ) {
			foreach my $id21 ( sort keys %fasta_hash21 ) {
				if ($id1 eq $id11 && $id1 eq $idv && $id1 eq $id21) {
					print $out1 "$id1 st1\tscore: $fasta_hash1{$id1}\tidentity: $fasta_hash11{$id11}%\tE-value:$fasta_hashv{$idv}\tlength:$fasta_hash21{$id21}\n";
				}
			}
		}
	}
}

foreach my $id0 ( sort keys %fasta_hash0 ) {
	foreach my $id12 ( sort keys %fasta_hash12 ) {
		foreach my $idv2 ( sort keys %fasta_hashv2 ) {
			foreach my $id23 ( sort keys %fasta_hash23 ) {
				if ($id0 eq $id12 && $id0 eq $idv2 && $id0 eq $id23) {
					print $out1 "$id0 st2\tscore:$fasta_hash0{$id0}\tidentity: $fasta_hash12{$id12}%\tE-value:$fasta_hashv2{$idv2}\tlength:$fasta_hash23{$id23}\n";
				}
			}
		}
	}
}
```

Loop in first hashes for first isolate and print the identifier of each IGR in a key and their starting points and ending points in values of hashes. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

```
foreach my $id4 ( sort keys %fasta_hash4 ) {
	foreach my $id5 ( sort keys %fasta_hash5 ) {
		if ($id4 eq $id5 ) {

				print $out2 "$id4\tst$fasta_hash4{$id4}\n";
				print $out2 "$id5\tend$fasta_hash5{$id5}\n";

		}
	}
}
```

```
foreach my $id14 ( sort keys %fasta_hash14 ) {
        foreach my $id15 ( sort keys %fasta_hash15 ) {
                if ($id14 eq $id15 ) {

                                print $out3 "$id14\tst$fasta_hash14{$id14}\n";
                                print $out3 "$id15\tend$fasta_hash15{$id15}\n";

                }
        }
}
```

Loop in first hashes for second isolates and print the identifier of each IGRs in a key and their starting points and ending points in values of hashes. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 182).

```
foreach my $id8 ( sort keys %fasta_hash8 ) {
        foreach my $id9 ( sort keys %fasta_hash9) {
                if ($id8 eq $id9 ) {

                                print $out2 "$id8\tst$fasta_hash8{$id8}\n";
                                print $out2 "$id9\tend$fasta_hash9{$id9}\n";

                }
        }
}
```

25 variation_location.pl

```
#! /usr/bin/Perl
# Title: Perl script to extract the index of variables from two sequences in alignment files

# Description: the main goal of script is to compare the DNA sequences from two different sequences in alignments files, and then the positions of each varied base pair between two different sequence alignments were printed. The script is going to work on many files but they should be specified in a directory.

#Date: 22/Aug/2016
#UpDate: 6/Feb/2016

# All variables, arrays and hashes were defined
use warnings;
use strict;
my $id1;
my $id2;
my $var_pos;
my $final_result;
my $finalresult1;
my %fasta_hash1;
my %fasta_hash2;
my @ind_var;
my $outfile1= "variation_location.txt";
open(my $out1, " > $outfile1") or die "error creating $outfile1. $!"; #open  output file

# Define and open the directory that will hold all the files in clustal format, array to hold all the files name in the directory and loop to read each file within the folder then check if the file is .clustal file, open them otherwise files will be ignored. All the variables, arrays, and hash reinitialized to avoid overwritten in the final output file. The script functions in this block are adapted from (http://perldoc.perl.org/functions/readdir.html).
my $directory=$ARGV[0];
die "Need the path of your directory that hold the clustal files \n" if (@ARGV != 1); # test whether there is one argument by checking the length of @ARGV
opendir(DIR,$directory);
my @files = readdir(DIR);
closedir(DIR);
foreach(@files){
        if ($_ =~ /.clustal$/) {
                my $filename = $directory . '/'.$_;
                open(my $fh, '<:encoding(UTF-8)', $filename)or die "Could not open file '$filename' $!";
                $id1 = "";
                $id2 = "";
                $var_pos = "";
                %fasta_hash1 = ( );
                %fasta_hash2 = ( );
                @ind_var = ( );
                $final_result="";
```

Loop through each sequence alignment file, save the identifier as a key and their sequence as a value in key-value pairs of hash. Push the identifier and sequences into two different arrays then assign the length of array into scalar content. The IGR name of each clustal file saved in $finalresult1 after substr the extension of full name of each input file (clustal file). The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182).

```
while (my $row = <$fh>) {
        chomp;
        if ($row =~ /(\S+)\s+([-GTCA]+)/){
                $id1 = $1;
                $fasta_hash1{$id1} .= $2;
        }
}
foreach my $id1 ( keys %fasta_hash1 ) {
        push (@seq, $fasta_hash1{$id1});
        push (@id, $id1);
}
my $TNumber = scalar @id;
$final_result=$_;
$finalresult1 = substr($final_result, 0, index($final_result, '.'));
```

Loop through array hold sequence, Initialize subroutine, send first element of array sequence (used as a reference and will be compared with other sequences in the same array) and all other sequences from same array for comparison between them, receive positions of each varied base pair in an array, loop in the array, and print the positions of each varied base pair and the identifier. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 136).

```
for (my $c = 0; $c < $TNumber; $c++){
        @ind_var = diff_index ($seq[0], $seq[$c]);
        foreach (@ind_var) {
                $var_pos="$_\t\t"."$finalresult1". ":" . "$id[$c]\n";
                print $out1 "$var_pos";
        }
}
    }
}
```

Subroutine is to make comparison between two sequences and return positions of varied base pair, subroutine receives sequence from two files for comparison. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 199, 136, 215) and (https://perldoc.perl.org/perlsub.html).

```
sub diff_index{
    my @st1;
    my @st2;
    my $hit =-1;
    my $hit2 =-1;
    my $index =0;
    my $index2 =0;
    my @cmp_ind;
    my($a,$b) = @_;
    while ($a) {
        if ($a =~  /^[ATGC]/i){
```

```
                                        $hit++;
                                        if ($hit<=length($a)){
                                                $index = substr($a,$hit,1);
                                                push (@st1, $index);
                                        }else {
                                                last;
                                        }
                        }
        }
        while ($b) {
                        if ($b =~  /^[ATGC]/i){
                                        $hit2++;
                                        if ($hit2<=length($b)){
                                                $index2 = substr($b,$hit2,1);
                                                push (@st2, $index2);
                                        }else {
                                                last;
                                        }
                        }
        }
        for (my $c = 0; $c < length($a); $c++){
                        if ($st1[$c] ne  $st2[$c]){
                                        my $posi =$c+1;
                                        push (@cmp_ind, $posi);
                        }
        }
return @cmp_ind;
}
```

26 Locate_CE.pl

```
#!/usr/bin/Perl
# Title: Perl script to identify the presence of varied SNPs in the CEs patterns for the DNA sequence of the compared IGRs

# Description: the script is going to save the identifier of compared IGR and position of varied SNPs from first file as key-value pair of hash. Then the script also save the identifier of compared IGR and the starting point and ending point of each CEs from second file as key-value pair of hash. Finally, the script is going to check if the position of varied SNPs located within the starting point and ending point of each CE and print them in a file


#Date: 22/Aug/2016
#UpDate: 6/Feb/2016


# All variable, arrays and hashes were defined
use strict;
use warnings;
my $str_source;
my $id1;
my $id2;
my $id3;
my $id4;
my $id6;
my $id5;
my $id7;
my %fasta_hash6;
my %fasta_hash4;
my %fasta_hash3;
my %fasta_hash2;
my %fasta_hash1;
my %count3;
my %value;
my $outfile = "CE1_state.txt"; # variable for output of original sequence file
my $outfile2 = "CE2_state.txt"; # variable for output of original sequence file
open(my $out, " > $outfile") or die "error creating $outfile. $!"; #open original sequence file
open(my $out2, " > $outfile2") or die "error creating $outfile2. $!"; #open original sequence file
my ($filename1, $filename2, $filename3, ) = @ARGV; #capture three command-line arguments
die "Need variation_location (output of variation_location.pl), b2 and b3 files (output of Significant_CE.pl) \n" if (@ARGV != 3); # test whether there are three arguments by checking the length of @ARGV
open (File1, $filename1) or die "Couldn't open $filename1: $!"; #open first file
open (File2, $filename2) or die "Couldn't open $filename2: $!"; #open second file
open (File3, $filename3) or die "Couldn't open $filename3: $!"; #open second file


# Loop through file, save identifier and position of SNPs in key-value pair of hash, to assign multiple values to one key in a hash , push values into an array reference stored in a another hash value. The push function needs an array therefore an array dereferences will be achieved later in
```

another code in the script. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182, 262) and (https://stackoverflow.com/questions/3779213/how-do-i-push-a-value-onto-a-perl-hash-of-arrays).

```
while (my $row = <File1>) {
	if ($row =~ /(\d+)\s+(\S+)/){
		$id1 = $2;
		$fasta_hash1{$id1} = $1;
		push @{ $value{$id1} }, $fasta_hash1{$id1};

	}
}
close File1;
```

Loop through file containing starting and ending point of first hits of CEs, save identifier and starting, ending points as key-value pair of hash. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 136).

```
while (my $row = <File2>) {
	if ($row =~ /(\S+)\s+(st)(::)(\d+)(.+)/) {
		$id3 = $1;
		$fasta_hash3{$id3} = $4;

	}elsif ($row =~ /(\S+)\s+(end)(::)(.+)/) {
		$id4 = $1;
		$id5 = $4;
		if ($id5 =~ /(\d+$)/) {
			$fasta_hash4{$id4} = $1;
		}
	}
}
 close File2;
```

Loop through file containing starting and ending point of second hits of CEs, save identifier and starting, ending points as key-value pair of hash. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 136).

```
while (my $row = <File3>) {
	if ($row =~ /(\S+)\s+(st)(::)(\d+)(.+)/) {
		$id2 = $1;
		$fasta_hash2{$id2} = $4;

	}elsif ($row =~ /(\S+)\s+(end)(::)(.+)/) {
		$id6 = $1;
		$id7 = $4;
		if ($id7 =~ /(\d+$)/) {
```

```
                                $fasta_hash6{$id6} = $1;

                        }
                }
}
close File3;
```

For first hit of CEs, For each key in my value hash, $value{$group} this means search for the key $group and get its value (reference of array), Then check if the position of varied SNPs located within the starting point and ending point of each CEs and print them in a file. The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182, 262) and (https://perldoc.perl.org/perlsub.html and perldoc.perl.org/perlreftut.html).

```
for my $group ( sort keys %value ) {
   my @value_group = @{ $value{$group} };
   for my $value ( @value_group ) {
                        foreach my $id3 ( keys %fasta_hash3 ) {
                                foreach my $id4 ( keys %fasta_hash4 ) {
                                        if ($group eq $id3 && $group eq $id4) {
                                                if ( $fasta_hash3{$id3} >= $value &&  $value >= $fasta_hash4{$id4} ||
$fasta_hash3{$id3} <= $value &&  $value <= $fasta_hash4{$id4}) {
                                                        print  $out  "$group\tCE\t$fasta_hash3{$id3}\t$value\t$fasta_hash4{$id4}\n";
                                                }
                                        }
                                }
                        }
   }
}
```

For second hit of CEs, same as in the first hits of CEs (See previous annotation). The script functions in this block are adapted from (Bradnam and Korf, 2012) (pages 166, 191, 182, 262) and (https://perldoc.perl.org/perlsub.html and perldoc.perl.org/perlreftut.html).

```
for my $group ( sort keys %value ) {
   my @value_group = @{ $value{$group} };
   for my $value ( @value_group ) {
                        foreach my $id2 ( keys %fasta_hash2 ) {
                                foreach my $id6 ( keys %fasta_hash6 ) {
                                        if ($group eq $id2 && $group eq $id6) {
                                                if ( $fasta_hash2{$id2} >= $value &&  $value >= $fasta_hash6{$id6} ||
$fasta_hash2{$id2} <= $value && $value <= $fasta_hash6{$id6}) {
                                                        print $out2 "$group\tCE\t$fasta_hash2{$id2}\t$value\t$fasta_hash6{$id6}\n";
                                                }
                                        }
                                }
                        }
                }
}
```