gapGlu

gapGlu is python based package for inserting unfilled elements into the reference sequence. It required a partially completed draft reference sequence, and small size pair reads.

System requirement:
Python, perl, Bowtie, BWA

This package contains 2 shell and 14 python scripts. Shell scripts are used for echoing python scripts and facilitating input, output exchange between scripts.

A brief of each file is given here:

13_20110929_gapfillRE.sh - shell script processing other python scripts data
13_20110929_positive_control.sh - for running posistive control- a bit different as input comes from blast
13a.py - for filtering of reads where both ends map to rep elements
13b.py - for making reference compatible, i.e. adding headers, removing small letters
13c.py - for taking out all gap positions
13d.py - take out genomic sequence with flanking regions
13e.py - remove additional N regions around targeted gap
13f.py - take out hanging reads mapping on the flanking region
13g.py - filter out pairs mapped to the flanking region
13h.py - filter out diretional reads i.e. for 5'&3', 5',3'
13i.py - taking out top four condidate suitable for replacement of gap and make score table
13j.py - reporting for gap regions which have no appropriate rep element for gap
13k.py - pick out best possible element from scores
13l.py - print final list of elements with score
13m.py - count correctly inserted elements (only for positive controls)

Installing prerequisite:

BWA: http://bio-bwa.sourceforge.net/bwa.shtml

Bowtie: http://bowtie-bio.sourceforge.net/index.shtml

Perl and python: inbuilt with any *nix system and MacOS, for Windows: http://www.python.org/getit/windows/ and http://www.activestate.com/activeperl for python and perl respectively

Installing package: Package requires no installation.

Running package: It can be run with shell command:
sh 13_20110929_positive_control.sh
sh 13_20110929_gapfillRE.sh

Once configure file is given with required inputs.

Details on pipeline:

This section provides more insight in the pipeline and user is not expected to modify code within.

1. We start with `. "configuration_file.sh"`
   This commands run the configuration file as a shell scripts and creates a working environment with all required input variable.

2. In very next step we define flanking regions for each library, which is double in length of insert size.

```
### define flanking regions
nb_1=`expr $insert_1 '*' 2`
```

3. Depending on that fact that filtering of raw reads is required or not, this part of the script can be used as it can be a time consuming step it should be run only when reads contains too many sequences. This section uses pair end mapping and then filters out all the sequences where only one of the read in pair is mapping to gap elements. We refer these as hanging reads.

4. This step is for making reference file compatible to the package. This basically does two things. A) add proper tags to multi-fasta sequences B) replacing any lower case nucleotide character to uppercase.

```
nice -n 19 python 13b.py $ref >"$ref"_replaced
```

5. Program uses this reference sequence to create a database for genome, which facilitate the access of any sub-sequence.

```
nice -n 19 formatdb -i $ref_file -o T -p F
```

6. Take out all the gaps in the reference sequence with their chromosomal location.

```
nice -n 19 python 13c.py $ref_file > $ref_file.N_regions.txt_im
```

7. gap_count in the further part of the script is a variable for running script in the batch of gaps. By default this value is same as the number of gaps in the reference.

```
if [ "$gap_count" -eq "$number_of_gaps" ]
```

8. Reading start, end chromosome of the gaps

```
st=`echo $line|awk '{split($0,a," ");  print a[1]}'`
en=`echo $line|awk '{split($0,a," ");  print a[2]}'`
chro=`echo $line|awk '{split($0,a," ");  print a[3]}'`
```

```
### make sure start < End otherwise screws up blast
if [ "$st" -lt "$en" ]
then
        st=$st
        en=$en
else
        st_im=$st
        st=$en
        en=$st_im
fi
```

9. Taking out sequence with the flanking region

```
echo nice -n 19 fastacmd -d "$ref_file" -p F -s "$chro" -L "`expr $st '-' $nb_1`"','"`expr $en '+' $nb_1`">check.txt
```

10. Unique gap identifier

```
st=`expr 10000000000 '*' $st`
st=`expr $st '+' $count`
count=`expr $count '+' 1`
```

11. making sure that no other N region in to +/- 10 Kb

```
nice -n 19 python 13e.py $pair_read_1.gene_gap_out.fa $nb_1 $st>> $pair_read_1.Nfil_gene_gap_out.fa
```

12. Doing mapping on genome using bwa

    - Index the genome

```
bowtie-build $pair_read_1.Nfil_gene_gap_out.fa $pair_read_1.Nfil_gene_gap_out.fa
```

    - align the reads

```
nice -n 19 bowtie -q --solexa1.3-quals --seedmms 0 --maqerr 30 --seedlen 28 -k 1 --time --offbase 0 --sam $pair_read_1.Nfil_gene_gap_out.fa "$rep_dir"$pair_read_1 | nice -n 19 perl -lane 'print $_ if ($F[3] > 0)' >$pair_read_1.sam.im
```

13. Filter out hanging reads, which are directing towards the targeted gap region.

```
nice -n 19 python 13f.py $pair_read_1.sam.im $pair_read_2.sam.im $nb_1 F >out.txt
nice -n 19 python 13g.py out.txt >$pair_read_1.sam
```

14. Taking out upstream, downstream or reads mapped from both side of the gap region and giving output all hanging reads as pairs.

```
nice -n 19 python 13h.py $pair_read_1.sam
"$rep_dir"$pair_read_1 $pair_read_1.fil.sam $nb_1 F
$pair_read_1.unmapped.unique "$x"> $pair_read_1.unmapped
```

15. After merging all the hanging read files do the mapping on the gap element to find the best match.

```
### single end mapping
nice -n 19 bowtie -q --solexa1.3-quals --seedmms 0 --maqerr 30 --
seedlen 28 -k 1 --time --offbase 0 --sam $rep_file
$pair_read_1.unmapped.fq.unique_all | nice -n 19 perl -lane 'print
$_ if ($F[3] > 0)'
>$pair_read_1.unmapped.fq.unique_all_rep_aln.sam
cat $pair_read_1.unmapped.fq.unique_all_rep_aln.sam| nice -n 19
perl -lane 'print $_ if ($F[1] == 0)'|sort|cut -f 1,3,4  >check.txt
```

16. Taking out top four candidate suitable for replacement of gap and for making score table

```
nice -n 19 python 13i.py check.txt out.txt $rep_file $x $strand
```

17. Reporting for gap regions which have no appropriate rep element for gap

```
nice -n 19 python 13j.py out.txt rep_region >"$input"direct_rep.$x
```

18. Pick out best possible element from scores and print final list of elements with score

```
python 13l.py "$input"direct_rep.$x "$input"reverse_rep.$x >
"$dat"_summary.txt.$x
nice -n 19 python 13k.py rep_pos "$dat"_summary.txt.1
"$dat"_summary.txt.2 "$dat"_summary.txt.3
>"$input"summary_"$dat"
```