Case study 2: Transcript-based RNA-seq pipeline

de novo assembly and differential expression analysis

updated: September 5, 2015

Pipeline overview

transcript base のRNA-seq解析の基本的なパイプラインを学ぶ。イルミナMiSeqのショートリードを用いて、de novo RNA-seq アセンブリと得られたコンティグの簡易アノテーション、発現量推定と発現比較解析までのプロトコルを具体的なスクリプトやコマンドを追って説明する。

実験デザイン：シロイヌナズナ *Arabidopsis thaliana* を明条件(L)と暗条件(D)で育て、それぞれ3個体からRNAを抽出して (three biological replicates)、illumina TruSeq kitでRNA-seqライブラリを作製し、イルミナシーケンサーMiSeqでシーケンスした(片側76base シーケンス)。明暗 (L vs D) の条件の差で発現の異なる遺伝子を同定したい。

モデル植物シロイヌナズナはゲノムシーケンスは既知だが、ここではあえてゲノム情報は使わず、de novo RNA-seqのアプローチで解析する。

```
[Strategy]
1. de novo assembly to build transcriptome reference
   tool: Trinity

2. mapping reads to transcriptome reference
   tool: bowtie2

3. abundance estimation
   tool: eXpress

4. differential expression analysis
   tool: edgeR

5. annotation of reference sequences
   tool: blastx
```

Setup

Software

本解析に必要なソフトウェアは以下の通り。すべて演習用のMacにインストール済み。

- bowtie2, eXpress, edgeR, MS Excel, NCBI BLAST

Data set

```
~/data/EX/practice2/
|-- Data
|   |-- Trinity.fasta
|   |-- blastx_results.txt
|   `-- TAIR10_pep_20110103_representative_gene_model_updated
|
|-- IlluminaReads
|   |-- D1_R1.fastq
|   |-- D2_R1.fastq
|   |-- D3_R1.fastq
|   |-- L1_R1.fastq
|   |-- L2_R1.fastq
```

```
|   `-- L3_R1.fastq
|-- Scripts
|   |-- compile_results.rb
|   `-- merge_express_results.rb
```

de novo RNA-seq assembly using Trinity

Trinity でRNA-seq readsをde novo assembling.

(注：TrinityはLinux上でしか稼働しないので、本講習ではskipする。)

Input readsの準備

```
$cat *.fastq > left_all.fq
```

Run Trinity (example)

```
# prepare input reads
$ cat *.R1.fastq > left_all.fq
$ cat *.R2.fastq > right_all.fq

# Run Trinity
$ Trinity --seqType fq --left left_all.fq --right right_all.fq
          --CPU 8 --max_memory 20G
```

Result: ''Trinity.fasta''

Quality assessment

Trinityソフトウェアに含まれる `TrinityStats.pl` でassembly statisticsをチェック。

```
$TRINITY_HOME/util/TrinityStats.pl Trinity.fasta
```

Mapping Illumina Reads to Trinity contigs using bowtie2

bowtie2 を使ってリードをTrinity.fasta にマッピングする。

Build bowtie2 index

まず、reference (Trinity.fasta) をindexing。この作業は一度やればよい。

```
$ bowtie2-build Trinity.fasta Trinity.fasta
```

Mapping

```
$ bowtie2 -x Trinity.fasta -U IlluminaReads/D1_R1.fastq -p 8 -a -S D1.sam
```

bowtie2の実行が終わると、mapping rateなどのサマリーが表示されるので保存しておくとよいだろう。

(例)

```
382799 reads; of these:
  382799 (100.00%) were unpaired; of these:
    21064 (5.50%) aligned 0 times
    322103 (84.14%) aligned exactly 1 time
```

```
    39632 (10.35%) aligned >1 times
94.50% overall alignment rate
```

D1_R1.fastq D2_R1.fastq D3_R1.fastq L1_R1.fastq L2_R1.fastq L3_R1.fastq ６つのシーケンスファイルすべてについて、同様にマッピングを行なう。

```
$ bowtie2 -x Trinity.fasta -U IlluminaReads/D2_R1.fastq -p 8 -a -S D2.sam
$ bowtie2 -x Trinity.fasta -U IlluminaReads/D3_R1.fastq -p 8 -a -S D3.sam
$ bowtie2 -x Trinity.fasta -U IlluminaReads/L1_R1.fastq -p 8 -a -S L1.sam
$ bowtie2 -x Trinity.fasta -U IlluminaReads/L2_R1.fastq -p 8 -a -S L2.sam
$ bowtie2 -x Trinity.fasta -U IlluminaReads/L3_R1.fastq -p 8 -a -S L3.sam
```

samファイルの中身を確認する。

Abundance estimation using eXpress

eXpress はSAM/BAM fileを読み込んで、コンティグごとにリードをカウントする。multiple mapなどのマッピング結果のあいまいさを考慮して、真のカウントをEMアルゴリズムで推定する。

```
$ express -o L1 Trinity.fasta L1.sam
```

L1.sam　の解析結果が、L1 ディレクトリ以下に保存される。 `results.xprs` にカウント推定結果が出力されている。

多の5つのサンプルも同様に処理。

```
$ express -o L2 Trinity.fasta L2.sam
$ express -o L3 Trinity.fasta L3.sam
$ express -o D1 Trinity.fasta D1.sam
$ express -o D2 Trinity.fasta D2.sam
$ express -o D3 Trinity.fasta D3.sam
```

''results.xprs'' の中身を確認する。

Differential expression analysis using edgeR

Prepare count matrix

このあとの解析がしやすいように、サンプルごとに別のファイルに記録されているeXpressのカウントデータを、ひとつのファイルにまとめる。edgeRは、FPKMでなくカウントデータを入力としなければいけない。各、results.xprsファイルの、est_countsカラムを抜き出す。この作業にはやや煩雑なテキストデータ処理を要するので、筆者が用意したRubyスクリプト `merge_express_results.rb` を使って欲しい。

merge_express_results.rb

```
(使い方)
$ruby merge_express_result.rb dir1 dir2 dir3 ...

(例)
$ruby merge_express_result.rb D1 D2 D3 L1 L2 L3 > eXpress_est_count_merged.txt
```

Scatter plot

複雑な統計計算で発現変動解析をあれこれ行なう前に、scatter plotを描くなどの、簡単なデータチェックをしておく。

以下、R環境で。

```
> dat <- read.delim("eXpress_est_count_merged.txt", comment.char="#", row.name=1)

(example of scatter plot)
> plot(dat$D1 + 1, dat$D2+1, log="xy")

(example of all-vs-all scatter plot)
> pairs(dat, log="xy")

(example of comparison between D1vsD2 and D1vsL1)
> par(mfrow=c(1,2))
> plot(dat$D1 + 1, dat$D2+1, log="xy")
> plot(dat$D1 + 1, dat$L1+1, log="xy")
```

edgeR: data import

```
> library(edgeR)

> category <- c("D", "D", "D", "L", "L", "L")

> D <- DGEList(dat, group=group)           # import table into edgeR

> D <- calcNormFactors(D, method="TMM")    # TMM normalization

> D$samples
   group lib.size norm.factors
D1     D   361691    0.9436719
D2     D   311297    1.0367666
D3     D   410178    0.8524095
L1     L   455588    0.9706589
L2     L   378548    1.0408683
L3     L   349357    1.1868267
```

TMM normalization

```
# dump normalized count data
> D.cpm.tmm <- cpm(D, normalized.lib.size=T)
> write.table(D.cpm.tmm, file="cpm.tmm.txt", sep="\t", quote=F)
```

Differential expression analysis

```
>D <- estimateCommonDisp(D)               # estimate common dispersion
> D$common.dispersion
[1] 0.05574236

>D <- estimateTagwiseDisp(D)              # estimate tagwise dispersion
> summary(D$tagwise.dispersion)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000871 0.000871 0.026900 0.059340 0.067680 1.029000

>de <- exactTest(D, pair=c("D", "L"))      # significance test to find differentially expressed genes
>topTags(de)                               # view the most significant genes


# dump DE analysis result
> de.sorted <- topTags(de, n=nrow(de$table))
> write.table(de.sorted$table, "de.txt", sep="\t", quote=F)
```

Result evaluation

有意に発現変動している遺伝子はいくつあるのか。例えば、Lで高発現し (logFC > 0)、FDR < 0.01 の遺伝子の数は、以下で求め

られる。

```
> sum(de.sorted$table$FDR<0.01 & de.sorted$table$logFC > 0)
```

これに答えるためには、edgeRの command `decideTestsDGE` も便利。

```
使い方例
> summary(decideTestsDGE(de, p.value=0.05))
   [,1]
-1    49
0  25903
1    269
```

plotSmear (edgeRに含まれるMA描画ツール) でMA plotを描いてみよう。

```
de.names <- row.names(D[decideTestsDGE(de, p.value=0.01) !=0, ])
plotSmear(D, de.tags=de.names)
```

MDS plotを行なうと、ライブラリ間の発現パターンの類似性をおおざっぱにとらえることができる。

```
plotMDS(D)
```

Quick annotation of Trinity contigs using BLAST

Trinityで得られたコンティグそれぞれがどのような遺伝子をコードしているだろうか？BLASTによる相同性検索はおおまかなアノテーションを行なうのに便利な手法である。ここでは、シロイヌナズナのタンパク質データベースを検索することにより、各コンティグがシロイヌナズナのどのタンパク質に対応するかを調べる。今回は、シロイヌナズナのシーケンスがqueryとなるので、シロイヌナズナのタンパク質データベースに対して検索をかけるとほぼ100%ヒットする。非モデル生物のde novo RNAseqでは、de novoアセンブリで得られたコンティグをqueryに、近縁種やモデル生物のタンパク質データベースや、nrデータベースに対して検索をかけることになる。

Build BLAST DB

国際コンソーシアムの運営するシロイヌナズナデータベース TAIRから、シロイヌナズナのタンパク質アミノ酸配列セットをダウンロードする。

ダウンロード

- ftp://ftp.arabidopsis.org/home/tair/Genes/TAIR10_genome_release/TAIR10_blastsets/TAIR10_pep_20110103_representative_gene_model_updated

（このファイルは、~/data/EX/practice2/Data/ ディレクトリにもコピーしておきました）。

ダウンロードしたファイルを''TAIR10.pep''の名前に変更。

```
$mv TAIR10_pep_20110103_representative_gene_model_updated TAIR10.pep
```

BLAST DBをビルド。

```
$ makeblastdb -in TAIR10.pep -dbtype prot -parse_seqids
```

（BLAST検索例）

```
$ blastx -query Trinity.fasta -db TAIR10.pep -num_threads 8 -max_target_seqs 1 \
       -evalue 1.0e-8 -outfmt 6 > blastx_results.txt
```

上の例では、トップヒットだけをテーブル形式で出力している。（参考のため結果ファイルをDataディレクトリに保存しておいた。）

Compile results

モデル生物の場合、大半の遺伝子に詳細なアノテーションがついているので、それらの情報と紐づけするとさらに利便性は上がる。シロイヌナズナの場合、各遺伝子のfunctional annotationは以下のファイルにまとめられており、TAIRのウェブサイトからダウンロードすることができる。

```
ftp://ftp.arabidopsis.org/home/tair/Genes/TAIR10_genome_release/TAIR10_functional_descriptions
```

これらの、DE analysis, annotation data, をひとつのテーブルにまとめると、見やすく、またさらなる下流解析を行なうのにも便利である。その処理には簡単なプログラミングが必要である。今回は、compile_results.rb (Scriptsディレクトリに含まれる) を使って下さい。

使い方

```
$ruby compile_results.rb > result_merged.txt
```

結果をMS Excelで吟味しよう。

(例)