Assembly course at CSC, September 5, 2012

Panu Somervuo

All programs used in this exercise are freely available which means you can download them also to your own computer. Most programs give short help if you just type the name of the program or name with argument -h. More information can be found in their web pages. In hippu.csc.fi, we must sometimes use module command first in order to make software available.

During this exercise, try to find out what each program does, what kind of parameters you can define, and what kind of output they produce. Take a look on both instructions and questions and in the end try to summarize what you have done. Commands to type on terminal are written in `Courier`. Good luck for assemblies!

Login to hippu.csc.fi (enable X, e.g. if login with ssh, use ssh -X), and define some variables first:

```
set COURSEDIR=/fs/lustre/wrk/somervuo/ngskurssi
set DATDIR=$COURSEDIR/data
set REF=$DATDIR/NC_000913.fna
set SCRIPTS=$COURSEDIR/scripts
```

Our genome assembly data is Illumina PE, 600x E.coli (SRA: ERP000092). We can take small subset of it by

```
zcat $DATDIR/ERR022075pe.fasta.gz | head -1500000 > subset.fasta
```

Data quality checking

It's always good idea to first check the quality of data. In this excercise we skip data filtering and possible error correction, but in order to get some statistics for the data, you can use e.g. FASTX package or Java based FASTQC. Notice that in data directory there are both fastq and fasta files, for quality values we need fastq. Here's an example of FASTX:

```
module load fastx
zcat $DATDIR/ERR022075_1.fastq.gz | head -1500000 > reads1.fq
zcat $DATDIR/ERR022075_2.fastq.gz | head -1500000 > reads2.fq
fastx_quality_stats -i reads1.fq -o reads1.qv
fastx_quality_stats -i reads2.fq -o reads2.qv
```

Task. How average quality goes towards the end of reads?

Genome denovo assembly using velvet

http://www.ebi.ac.uk/~zerbino/velvet/

Velvet is denovo assembler suitable for small genomes. It is based on deBruijn graph and we must define kmer length when using it. In practice you should try several assemblies with different values and see which gives best results. Typically larger value of k will give longer contigs, but only if there are enough reads. There is a relation between read depth and optimal k value which can be found empirically.

Example 1: velvet assembly using only single read information, for this data set we define expected coverage to be 15 and minimum contig length to be 1000nt:

```
module load velvet
velveth vdirS 23 -fasta -short subset.fasta
velvetg vdirS -exp_cov 15 -min_contig_lgth 1000
```

Calculate contig lengths:

```
$SCRIPTS/fastalen.pl vdirS/contigs.fa > contigsS.len
```

Calculate summary of contig lengths:

```
$SCRIPTS/n50.pl contigsS.len
```

Example 2: velvet assembly using pairend information. When using Velvet with pairend read information, r1 and r2 reads must be interleaved (our data file is already in this format).

```
velveth vdirP 23 -fasta -shortPaired subset.fasta
velvetg vdirP -exp_cov 15 -ins_length 500 -min_contig_lgth 1000
```

Calculate contig lengths and N50:

```
$SCRIPTS/fastalen.pl vdirP/contigs.fa > contigsP.len
$SCRIPTS/n50.pl contigsP.len
```

We can plot contig lengths e.g. using R which is a software for general purpose statistics and data analysis. It provides good tools to make graphical plots. Here we are going to see what N50 really means. First start R program by typing 'R' in terminal.

```
module load R
R
```

Note that the data files "contigsS.len" and "contigsP.len" must be available in working directory of R

```
a=read.table("contigsS.len", header=F)
b=read.table("contigsP.len", header=F)

hist(a[,2],col="yellow",main="contigs without Pairend info",xlab="contig length")
x11()
hist(b[,2],col="green",main="contigs with Pairend info",xlab="contig length")

d=b[,2]
x=sort(d,dec=T)
cx=cumsum(x)

plot(x,main="sorted contig lengths")
dev.set(2)
plot(cx,main="cumulative contig lengths")
v=sum(d)/2
abline(h=v,col="red",lwd=3)
i50=min(which(cx>v))
# this is N50:
x[i50]
dev.set(3)
abline(v=i50,col="blue",lwd=3)
```

You can quit R by command q()

N50 is the length of shortest contig which corresponds to half of the total sum, i.e., half of the genome size, when summing sorted contig lengths. Notice that by default, velvet produces scaffolds although the output name is called 'contigs.fa'. Scaffolding can be switched on/off by argument -scaffolding yes/no.

Task 1. Compare assemblies with and without pairend read information (number & length of contigs, N50).

Task 2. How the choice of kmer length affects the assembly?

Task 3. Make subsets of data: 10x, 30x, 100x, 300x. How sequencing depth affects contigs? Number of reads required for coverage x is N=xG/L, where here G: 5Mb and L=100. Notice that fasta format includes also header for each read so number of lines from fasta file must be double.

Task 4. How parameters exp_cov and cov_cutoff affect the assembly?

Task 5. Can you think reasons why there are gaps in assembly, i.e. why we don't get a single long contig representing the entire genome even when using high sequencing depth? How we could improve assembly?

Example 3: assembly with 60X coverage data:

```
zcat $DATDIR/ERR022075pe.fasta.gz | head -5400000 > datape.fasta
velveth vdirP23 23 -fasta -shortPaired datape.fasta
velvetg vdirP23 -ins_length 500 -exp_cov 60 -min_contig_lgth 1000 -scaffolding no

$SCRIPTS/fastalen.pl vdirP23/contigs.fa > len1
$SCRIPTS/n50.pl len1
```

sort contig names based on coverage:

```
grep '^>' vdirP23/contigs.fa | sort -n -k6 -t"_"
```

sort names based on contig length:

```
grep '^>' vdirP23/contigs.fa | sort -n -k4 -t"_"
```

Let's produce .afg file, this is needed in assembly visualization

```
velvetg vdirP23 -ins_length 500 -exp_cov 60 -min_contig_lgth 1000 -scaffolding
yes -amos_file yes

$SCRIPTS/fastalen.pl vdirP23/contigs.fa > len2
$SCRIPTS/n50.pl len2
```

Assembly validation

In practice, we don't know what the outcome should be. N50 measures only contig lengths but it doesn't take into account possible assembly errors. In this exercise we are in a lucky situation to know the truth. In order to see how good contigs/scaffolds are, we can map them against finished, complete genome:

```
module load bwa
bwa index -a is $REF
bwa bwasw -t 4 $REF vdirP23/contigs.fa > contigs.sam
```

Look contig coverage along reference and examine gaps. In data directory, there is genbank file `$DATDIR/NC_000913.gb` with annotations so it is possible to check in which genomic regions gaps occur.

`tablet` (Open assembly: contigs.sam and NC_010473.fna)

Tablet can be also used for visualizing .afg-files, but hawkeye is more informative for that purpose.

Assembly visualization

In order to use hawkeye, we must first convert .afg file into AMOS format

```
bank-transact -m vdirP23/velvet_asm.afg -b velvet.bnk -c
hawkeye -t velvet.bnk/
```

Assembly with SOAPdenovo

http://soap.genomics.org.cn/soapdenovo.html

SOAPdenovo is deBruijn graph based assembler suitable for large genomes, e.g., it has been used for human genomes. It is more efficient compared to velvet in terms of memory usage and also we can use several cpus at the same time.

Task 1. Create config file soap.config with text editor (e.g. emacs):

```
#maximal read length
max_rd_len=100
[LIB]
#average insert size
avg_ins=500
#if sequence needs to be reversed
reverse_seq=0
#in which part(s) the reads are used
asm_flags=3
#use only first 75 bps of each read
rd_len_cutoff=75
#in which order the reads are used while scaffolding
rank=1
# cutoff of pair number for a reliable connection (default 3)
pair_num_cutoff=3
#minimum aligned length to contigs for a reliable read location (default 32)
map_len=32
#fastq file for read 1
#q1=/path/**LIBNAMEA**/fastq_read_1.fq
#fastq file for read 2 always follows fastq file for read 1
#q2=/path/**LIBNAMEA**/fastq_read_2.fq
#fasta file for read 1
#f1=/path/**LIBNAMEA**/fasta_read_1.fa
#fastq file for read 2 always follows fastq file for read 1
#f2=/path/**LIBNAMEA**/fasta_read_2.fa
#fastq file for single reads
#q=/path/**LIBNAMEA**/fastq_read_single.fq
#fasta file for single reads
#f=/path/**LIBNAMEA**/fasta_read_single.fa
```

```
#a single fasta file for paired reads
#p=/path/**LIBNAMEA**/pairs_in_one_file.fa
p=datape.fasta
```

Assembly can be done in four steps. Number of cpus is defined by -p, and since we all share hippu, let's be kind to other users (here we use 4 threads).

```
module load soapdenovo

SOAPdenovo-31mer pregraph -s soap.config -K 23 -p 4 -o soap23
SOAPdenovo-31mer contig -g soap23
SOAPdenovo-31mer map -s soap.config -g soap23 -p 4
SOAPdenovo-31mer scaff -g soap23 -p 4
```

Contigs are in file soap23.contig and scaffolds in soap23.scafSeq.
You can also run all four steps in a single command:

```
SOAPdenovo-31mer all -s soap.config -K 23 -o soap23b -p 4
```

and contig and scaffold length summaries can be obtained:

```
$SCRIPTS/fastalen.pl soap23b.contig | $SCRIPTS/n50.pl
$SCRIPTS/fastalen.pl soap23b.scafSeq | $SCRIPTS/n50.pl
```

Task 2. Try different parameters for rd_len_cutoff (e.g. 50,75,100) in config file and -K in command line. If K is greater than 31, you have to use `SOAPdenovo-63mer`.

Task 3. Add matepair libraries to assembly, there are two files in data directory, matepairs2K.fasta corresponds to insertsize of 2K and matepairs10K.fasta insertsize of 10K. Create two config files soap.config2 and soap.config3.

soap.config2:

```
max_rd_len=100
[LIB]
avg_ins=500
reverse_seq=0
asm_flags=3
rd_len_cutoff=75
rank=1
pair_num_cutoff=3
map_len=32
p=datape.fasta
[LIB]
avg_ins=2000
reverse_seq=1
asm_flags=2
rd_len_cutoff=75
rank=2
pair_num_cutoff=3
map_len=32
p=/fs/lustre/wrk/somervuo/ngskurssi/data/matepairs2K.fasta
```

soap.config3:

```
max_rd_len=100
[LIB]
avg_ins=500
reverse_seq=0
asm_flags=3
rd_len_cutoff=75
rank=1
pair_num_cutoff=3
map_len=32
p=datape.fasta
[LIB]
avg_ins=2000
reverse_seq=1
asm_flags=2
rd_len_cutoff=75
rank=2
pair_num_cutoff=3
map_len=32
p=/fs/lustre/wrk/somervuo/ngskurssi/data/matepairs2K.fasta
[LIB]
avg_ins=10000
reverse_seq=1
asm_flags=2
rd_len_cutoff=75
rank=3
pair_num_cutoff=3
map_len=32
p=/fs/lustre/wrk/somervuo/ngskurssi/data/matepairs10K.fasta
```

Assemblies are done by

```
SOAPdenovo-31mer all -s soap.config2 -K 23 -o soap2 -p 4
SOAPdenovo-31mer all -s soap.config3 -K 23 -o soap3 -p 4
```

Assembly validation

Let's check results by creating dotplot against our reference

```
mummer -mum -b -c $REF soap2.scafSeq > mummer2.mums
mummerplot -postscript -p mummer2 -Q soap3.scafSeq mummer2.mums

mummer -mum -b -c $REF soap3.scafSeq > mummer3.mums
mummerplot -postscript -p mummer3 -Q soap3.scafSeq mummer3.mums
```

Compare mummer2.ps and mummer3.ps, red dots correspond to forward strand matches and blue dots correspond to reverse strand matches.

```
evince mummer2.ps &
evince mummer3.ps &
```

Task. Look the gaps in assembly using 2K matepair library. Which genomic regions they correspond? Use genbank file `$DATDIR/NC_000913.gb` to get annotations.

OLC assembler

We can use Minimo to join contigs, either from one assembly or combining several assemblies. Stringency can be controlled by two parameters: minimum overlap length (`MIN_LEN`) and identity percentage (`MIN_IDENT`).

```
module load amos
Minimo vdirP23/contigs.fa -D FASTA_EXP=1 -D OUT_PREFIX=comb -D MIN_LEN=40 -D
MIN_IDENT=90
```

Task. Use velvet contigs from several assemblies with different values of k. Can you improve N50?

RNA-Seq denovo assembly

Our data is mouse RNA-Seq, Illumina PE 2x100nt.

For baseline, let's make first assembly using genome assembler:

```
module load velvet
shuffleSequences_fastq.pl $DATDIR/rnaseq.left.fq $DATDIR/rnaseq.right.fq perna.fq
velveth vrna 21 -fastq -shortPaired perna.fq
velvetg vrna -ins_length 280 -exp_cov auto -cov_cutoff auto
```

Although N50 is not necessarily proper measure for RNA-Seq assembly quality, let's check contig summary:

```
$SCRIPTS/fastalen.pl vrna/contigs.fa > vrnactg.len
$SCRIPTS/n50.pl vrnactg.len
```

131 sequences, sum_length: 63188
min_length: 41, max_length: 3431, N50: 1346

Tasks: How many transcript sequences you get? How many locuses and how many transcripts within each locus? Can you find any alternative splicing?

Oases

http://www.ebi.ac.uk/~zerbino/oases/

oases reconstructs transcripts based on velvet's deBruijn graph.

```
module load oases
oases vrna
```

```
$SCRIPTS/fastalen.pl vrna/transcripts.fa > vrnatr.len
$SCRIPTS/n50.pl vrnatr.len
```

68 sequences, sum_length: 174510
min_length: 103, max_length: 8728, N50: 5398

Calculate number of loci (49) and transcripts (1-6) within each locus:

```
grep '^>' vrna/transcripts.fa | cut -f1,2 -d "_" | uniq -c
```

In order to see if transcripts are correct, we can blast them against mouse transcripts.

http://blast.ncbi.nlm.nih.gov

In order to get more information, go to UCSC genome browser http://genome.ucsc.edu/ and:

- Select BLAT and paste your assembled transcript sequence
- Select Mouse genome and click "submit"
- Click on the top scoring hit
- You can add tracks, e.g. Alt-Splicing in 'mRNA and EST Tracks'

An example of alternatively spliced transcript:

```
>Locus_48_Transcript_1/5_Confidence_0.429_Length_1816
CGTAAATGCCACATTTATTTGTGGGAGGCTCCAAGCCTCTCTAAGCACGGCACAGCAGGG
CTCTCCCACATGCTGCCCTGAGCCCAGCTCCTTAAACATGCAGGATGGAGCCAGAGGATA
GGACCCGGTGTCTTCAGGAGCAGACCCATAAGAAAAGCAAATTGTTCTATCACATGCCCT
CCCCACCCAGCCTGTTCTGGCAGGGCCGCCTGGATCAGCTGCAGTAACCCTTTCTTGAGG
GCCACCCTTCCTCACCTTCCCTTCCCTGTGATCTCCCTGAATGTCAGAGCCATGATCCAT
GTGCTTTGACCCTTCACATGCTGCCATTCAGAAGCTTCTGGGCTTCTCGTCTCTCCGGTC
GGTCAGGTTATCAGCTGGGTAGGGCAGGCCAGGCTCTCTGTCTTCACTCTGTGTTCCGAC
TCAGCACTAAGCATTTGTTGGGGATTGAACGAGCAAAGGCCTGTCCAGAGGACTACCACA
GGCTAAGGGGTGGTACACCCGACTCAGCTGTGACCAGCAGTAGATGTCTCCCAGCCTCCC
GGGCTCCTCGTTGGGACTGAAACATCTTCAGGCCTCAGTTTTCCATGGCCCAAACCAACG
CCTCCTATTCCAAGAGGCTTCTCTGGGTTGGCACCAGGTTCAGGGCTAGGTGTCTGATGG
ACAACCAACACGGCTACTTCAAGGCCTTGAGTACAGCCTCCAGCTTCATGACCACGGCCA
GGTCACCCTTCACCTTCAGCCGCCCACTCATGTAGGCCCCTAAGGGCCGAAGTTCTTTGC
TCAACAAGGCCTGCAAGTCTGCTTCAGCCATCTCCACCACCACGTCAGGGATGCCATCAG
GCTCTCCATGTCCCACGCGGCCTTGTCCTGTAGTAAGGTCCAGGAAGTAGATACTCTGGG
TGCCACTGGGCAAGATGACATTGAACTGGTAGCAGGCCCCAACCTGGCTGACCAGTGCCT
CTGACAGGAAGGGTTGCAGGGCTGTGAGGAGCCCCTCAGCCACGGGCTGCTTCGGGCTGG
GCTCAGGCCCGGCCCCAGCAAGAGAGGCAGGAGGTTCCACTTCATTTATCATCTCCAAGG
TGTCTGGCCCTGGGGATGGAACATGTCCGACCGCTGAGTTCATACTTCCCCCTAGCAAGT
GGAGGGCCAGCTGTTGGAGGGTGCTGTCCAGGCTGGGCACAGTCAGGCTGTCTTGCGGTG
GCTGCAACACAGCCTCTACTGCCAGCTCTACACGGTCGACCTCCAGTCCCCAAGCTCTGG
TCACGTCATTGATCTCCAGCAGGAGCTGGTCACCGATCTTGAGCTTCTCCATCTGGATCT
CCTGCAGTGGCCTTCTGAGCAGGGCCTTAGTCATGGCGTTGTGGGCTGTCATGCGAGTAG
CTGTGTTCAGGTCCTTCACAGCCATCACGGATAGCACTGGGTCCCAGATGCGGAACTGGA
CATCAGCTCCCACCGACAGCACAGCCCCATCCTTAGAGGCCAGCTTGCAAGGAGGAACGT
TGAAGGCTCGGGTCCTCAAATCTACCCTCTGGAAGGAGTCAATAAAGGGCAGGAGAAGAA
CCATGCCAGGCCCCTGGGGGTTACGGATCCGGCCCAGTCGAAACACAATCATCCTCTCAT
AGGTGGGCACAATCTTCAGAGCAAACCAGCCGGAAATGGGGAAGGTGAGCAGCAGCAGCA
AGAACCCCAGGACACTGACGAGGCCATGGCAGAGACAGGAGGGCCAGCTCTCCGGCGCGT
CTTCCAAGCATGACTTCTAACCGGGACACGCCCCGATCCTCTGCTGGGAGACTGCCCTGC
CCAGCCTGTGGCAGCT
```

Trinity

http://trinityrnaseq.sourceforge.net/

Trinity is denovo assembler for RNA-Seq data based on deBruijn graphs. It is suitable for large scale eukaryotic transcriptomes and tries to resolve also different isoforms. Trinity consists of three programs which can be run by a single Perl script.

```
module load trinity
```

```
Trinity.pl --seqType fq --kmer_method meryl --left $DATDIR/rnaseq.left.fq --right
$DATDIR/rnaseq.right.fq --CPU 2
```

If everything went fine, transcripts are in trinity_out_dir/Trinity.fasta

```
$SCRIPTS/fastalen.pl trinity_out_dir/Trinity.fasta > tritr.len
$SCRIPTS/n50.pl tritr.len
```

76 sequences, sum_length: 145418
min_length: 204, max_length: 8816, N50: 5399

Let's grep header lines

```
grep '^>' trinity_out_dir/Trinity.fasta
```

Fasta entry names have this form:

```
>comp56_c0_seq1 len=2433 path=[1:0-587 589:588-1076 5644:1077-1145 1078:1146-1689
1622:1690-1840 1773:1841-2432]
```

Format of output transcript names is explained in
http://trinityrnaseq.sourceforge.netadvanced_trinity_guide.html#Butterfly_reconstruction

Let's grep first two columns based on delimiter '_' and take only unique names and count how many there are, this is the number of loci:

```
grep '^>' trinity_out_dir/Trinity.fasta | cut -f1,2 -d"_" | uniq | wc -l
```

This shows how many transcripts there are within each locus:

```
grep '^>' trinity_out_dir/Trinity.fasta | cut -f1,2 -d"_" | uniq -c
```

```
   2 >comp0_c0
   1 >comp1_c0
   4 >comp2_c0
   2 >comp3_c0
   4 >comp4_c0
   8 >comp5_c0
   3 >comp6_c0
   1 >comp7_c0
   2 >comp8_c0
   2 >comp9_c0
   2 >comp10_c0
   1 >comp11_c0
  ...
```

Example of alternative splicing:
>comp0_c0_seq1 len=3739 path=[3675:0-394 7367:395-436 4070:437-565 4199:566-3738]
>comp0_c0_seq2 len=3697 path=[3675:0-394 4070:395-523 4199:524-3696]

are identical except one indel:

```
comp0_c0_seq1    251 GTCCCGGTCCACAGCAGGATTCCCCCTCTGTGAAAAGGCACGCTGATCTG    300
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    251 GTCCCGGTCCACAGCAGGATTCCCCCTCTGTGAAAAGGCACGCTGATCTG    300

comp0_c0_seq1    301 TCTGGATAAGTGTGGCCGGCCCCATGTATCCGGAATCAACCACGGGGTCC    350
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    301 TCTGGATAAGTGTGGCCGGCCCCATGTATCCGGAATCAACCACGGGGTCC    350

comp0_c0_seq1    351 CCAGCTCGACTCTCCCTGCGGCAGACAGGCTCCCCCGGGATGATCTACAG    400
                     |||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    351 CCAGCTCGACTCTCCCTGCGGCAGACAGGCTCCCCCGGGATGATC-----    395

comp0_c0_seq1    401 TACTCGTTATGGGAGTCCCAAAAGACAGCTCCAGTTTTACAGGAATCTGG    450
                                                          |||||||||||||
comp0_c0_seq2    396 -------------------------------------TACAGGAATCTGG    408

comp0_c0_seq1    451 GCAAATCTGGCCTTCGGGTCTCCTGCCTGGGGCTTGGAACATGGGTGACC    500
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    409 GCAAATCTGGCCTTCGGGTCTCCTGCCTGGGGCTTGGAACATGGGTGACC    458

comp0_c0_seq1    501 TTCGGGGGCCAGATCACGGATGAGATGGCAGAGCACCTAATGACCTTGGC    550
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    459 TTCGGGGGCCAGATCACGGATGAGATGGCAGAGCACCTAATGACCTTGGC    508

comp0_c0_seq1    551 CTACGATAATGGCATCAACCTGTTCGATACGGCGGAGGTCTACGCTGCTG    600
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
comp0_c0_seq2    509 CTACGATAATGGCATCAACCTGTTCGATACGGCGGAGGTCTACGCTGCTG    558
```