Manual of SOAPdenovo-V2.04

Ruibang Luo, 2012-7-10
Zhenyu Li, 2012-7-10

****

Introduction

SOAPdenovo is a novel short-read assembly method that can build a de novo draft assembly for the human-sized genomes. The program is specially designed to assemble Illumina GA short reads. It creates new opportunities for building reference sequences and carrying out accurate analyses of unexplored genomes in a cost-effective way.

****

System Requirement

SOAPdenovo aims for large plant and animal genomes, although it also works well on bacteria and fungi genomes. It runs on 64-bit Linux system with a minimum of 5G physical memory. For big genomes like human, about 150 GB memory would be required.

****

Update Log

1) 63mer and 127mer versions were merged.

2) A new module named "sparse-pregraph" was added which can reduce considerable computational comsumption.

3) "Multi-kmer" method was introduced in "contig" step which allows the utilization of the advantages of small and large kmer.

4) Algorithm of scaffolding was improved to get longer and more accuracy scaffolds.

5) AIO (Asynchronous Input/Output) was introduced to boost the performance of reading files.

6) Information for visualization purpose was available after scaffolding.

7) Several bugs fixed.

****

Installation

1. You can download the pre-compiled binary according to your platform, unpack using "tar -zxf ${destination folder} download.tgz" and execute directly.

2. Or download the source code, unpack to ${destination folder} with the method above, and compile by using GNU make with command "make" at ${destination folder}/SOAPdenovo-V2.04. Then install executable to ${destination folder}/SOAPdenovo-V2.04/bin using "make install"

****

How to use it

1. Configuration file

For big genome projects with deep sequencing, the data is usually organized as multiple read sequence files generated from multiple libraries. The configuration file tells the assembler where to find these files and the relevant information. "example.config" is an example of such a file.

The configuration file has a section for global information, and then multiple library sections. Right now only "max_rd_len" is included in the global information section. Any read longer than max_rd_len will be cut to this length.

The library information and the information of sequencing data generated from the library should be organized in the corresponding library section. Each library section starts with tag [LIB] and includes the following items:

1) avg_ins

This value indicates the average insert size of this library or the peak value position in the insert size distribution figure.

2) reverse_seq

This option takes value 0 or 1. It tells the assembler if the read sequences need to be complementarily reversed.

Illumima GA produces two types of paired-end libraries: a) forward-reverse, generated from fragmented DNA ends with typical insert size less than 500 bp; b) reverse-forward, generated from circularizing libraries with typical insert size greater than 2 Kb. The parameter "reverse_seq" should be set to indicate this: 0, forward-reverse; 1, reverse-forward.

3) asm_flags

This indicator decides in which part(s) the reads are used. It takes value 1(only contig assembly), 2 (only scaffold assembly), 3(both contig and scaffold assembly), or 4 (only gap closure).

4) rd_len_cutof

The assembler will cut the reads from the current library to this length.

5) rank

It takes integer values and decides in which order the reads are used for scaffold assembly. Libraries with the same "rank" are used at the same time during scaffold assembly.

6) pair_num_cutoff

This parameter is the cutoff value of pair number for a reliable connection between two contigs or pre-scaffolds. The minimum number for paired-end reads and mate-pair reads is 3 and 5 respectively.

7) map_len

This takes effect in the "map" step and is the minimun alignment length between a read and a contig required for a reliable read location. The minimum length for paired-end reads and mate-pair reads is 32 and 35 respectively.

The assembler accepts read file in three kinds of formats: FASTA, FASTQ and BAM. Mate-pair relationship could be indicated in two ways: two sequence files with reads in the same order belonging to a pair, or two adjacent reads in a single file (FASTA only) belonging to a pair. If a read in bam file fails platform/vendor quality checks(the flag field 0x0200 is set), itself and it's paired read would be ignored.

In the configuration file single end files are indicated by "f=/path/filename" or "q=/pah/filename" for fasta or fastq formats separately. Paired reads in two fasta sequence files are indicated by "f1=" and "f2=". While paired reads in two fastq sequences files are indicated by "q1=" and "q2=". Paired reads in a single fasta sequence file is indicated by "p=" item. Reads in bam sequence files is indicated by "b=".

All the above items in each library section are optional. The assembler assigns default values for most of them. If you are not sure how to set a parameter, you can remove it from your configuration file.

2. Get it started

Once the configuration file is available, a typical way to run the assembler is:

```
${bin} all -s config_file -K 63 -R -o graph_prefix 1>ass.log 2>ass.err
```

User can also choose to run the assembly process step by step as:

step1:

```
${bin} pregraph -s config_file -K 63 -R -o graph_prefix 1>pregraph.log 2>pregraph.err
```

OR

```
${bin} sparse_pregraph -s config_file -K 63 -z 5000000000 -R -o graph_prefix 1>pregraph.log 2>pregraph.err
```

step2:

```
${bin} contig -g graph_prefix -R 1>contig.log 2>contig.err
```

step3:

```
${bin} map -s config_file -g graph_prefix 1>map.log 2>map.err
```

step4:

```
${bin} scaff -g graph_prefix -F 1>scaff.log 2>scaff.err
```

3.Options

3.1 Options for all (pregraph-contig-map-scaff)
 -s <string> configFile: the config file of solexa reads
 -o <string> outputGraph: prefix of output graph file name
 -K <int> kmer(min 13, max 63/127): kmer size, [23]
 -p <int> n_cpu: number of cpu for use, [8]
 -a <int> initMemoryAssumption: memory assumption initialized to avoid further reallocation, unit G, [0]
 -d <int> KmerFreqCutoff: kmers with frequency no larger than KmerFreqCutoff will be deleted, [0]
 -R (optional) resolve repeats by reads, [NO]
 -D <int> EdgeCovCutoff: edges with coverage no larger than EdgeCovCutoff will be deleted, [1]
 -M <int> mergeLevel(min 0, max 3): the strength of merging similar sequences during contiging, [1]
 -m <int> max k when using multi kmer
 -e <int> weight to filter arc when linearize two edges(default 0)
 -r (optional) keep available read(*.read)
 -E (optional) merge clean bubble before iterate
 -f (optional) output gap related reads in map step for using SRkgf to fill gap, [NO]
 -k <int> kmer_R2C(min 13, max 63): kmer size used for mapping read to contig, [K]
 -F (optional) fill gaps in scaffold, [NO]
 -u (optional) un-mask contigs with high/low coverage before scaffolding, [mask]
 -w (optional) keep contigs weakly connected to other contigs in scaffold, [NO]
 -G <int> gapLenDiff: allowed length difference between estimated and filled gap, [50]
 -L <int> minContigLen: shortest contig for scaffolding, [K+2]
 -c <float> minContigCvg: minimum contig coverage (c*avgCvg), contigs shorter than 100bp with coverage smaller than c*avgCvg will be masked before scaffolding unless -u is set, [0.1]
 -C <float> maxContigCvg: maximum contig coverage (C*avgCvg), contigs with coverage larger than C*avgCvg or contigs shorter than 100bp with coverage larger than 0.8*C*avgCvg will be masked before scaffolding unless -u is set, [2]
 -b <float> insertSizeUpperBound: (b*avg_ins) will be used as upper bound of insert size for large insert size (> 1000) when handling pair-end connections between contigs if b is set to larger than 1, [1.5]
 -B <float> bubbleCoverage: remove contig with lower cvoerage in bubble structure if both contigs' coverage are smaller than bubbleCoverage*avgCvg, [0.6]
 -N <int> genomeSize: genome size for statistics, [0]
 -V (optional) output visualization information of assembly, [NO]

3.2 Options for sparse_pregraph
 Usage: ./SOAPdenovo2 sparse_pregraph -s configFile -K kmer -z genomeSize -o outputGraph [-g maxKmerEdgeLength -d kmerFreqCutoff -e kmerEdgeFreqCutoff -R -r runMode -p n_cpu]
 -s <string> configFile: the config file of solexa reads

-K <int> kmer(min 13, max 63/127): kmer size, [23]
-g <int> maxKmerEdgeLength(min 1, max 25): number of skipped intermediate kmers, [15]
-z <int> genomeSize(mandatory): estimated genome size
-d <int> kmerFreqCutoff: delete kmers with frequency no larger than,[1]
-e <int> kmerEdgeFreqCutoff: delete kmers' related edge with frequency no larger than [1]
-R (optional) output extra information for resolving repeats in contig step, [NO]
-r <int> runMode: 0 build graph & build edge and preArc, 1 load graph by prefix & build edge and preArc, 2 build graph only, 3 build edges only, 4 build preArcs only [0]
-p <int> n_cpu: number of cpu for use,[8]
-o <int> outputGraph: prefix of output graph file name

4. Output files

4.1 These files are output as assembly results:
a. *.contig
contig sequences without using mate pair information.
b. *.scafSeq
scaffold sequences (final contig sequences can be extracted by breaking down scaffold sequences at gap regions).

4.2 There are some other files that provide useful information for advanced users, which are listed in Appendix B.

5. FAQ

5.1 How to set K-mer size?

The program accepts odd numbers between 13 and 31. Larger K-mers would have higher rate of uniqueness in the genome and would make the graph simpler, but it requires deep sequencing depth and longer read length to guarantee the overlap at any genomic location.

The sparse pregraph module usually needs 2-10bp smaller kmer length to achieve the same performance as the original pregraph module.

5.2 How to set genome size(-z) for sparse pregraph module?

The -z parameter for sparse pregraph should be set a litter larger than the real genome size, it is used to allocate memory.

5.3 How to set library rank?

SOAPdenovo will use the pair-end libraries with insert size from smaller to larger to construct scaffolds. Libraries with the same rank would be used at the same time. For example, in a dataset of a human genome, we set five ranks for five libraries with insert size 200-bp, 500-bp,

2-Kb, 5-Kb and 10-Kb, separately. It is desired that the pairs in each rank provide adequate physical coverage of the genome.

****

APPENDIX A: an example.config

```
#maximal read length
max_rd_len=100
[LIB]
#average insert size
avg_ins=200
#if sequence needs to be reversed
reverse_seq=0
#in which part(s) the reads are used
asm_flags=3
#use only first 100 bps of each read
rd_len_cutoff=100
#in which order the reads are used while scaffolding
rank=1
# cutoff of pair number for a reliable connection (at least 3 for short insert size)
pair_num_cutoff=3
#minimum aligned length to contigs for a reliable read location (at least 32 for short insert size)
map_len=32
#a pair of fastq file, read 1 file should always be followed by read 2 file
q1=/path/**LIBNAMEA**/fastq1_read_1.fq
q2=/path/**LIBNAMEA**/fastq1_read_2.fq
#another pair of fastq file, read 1 file should always be followed by read 2 file
q1=/path/**LIBNAMEA**/fastq2_read_1.fq
q2=/path/**LIBNAMEA**/fastq2_read_2.fq
#a pair of fasta file, read 1 file should always be followed by read 2 file
f1=/path/**LIBNAMEA**/fasta1_read_1.fa
f2=/path/**LIBNAMEA**/fasta1_read_2.fa
#another pair of fasta file, read 1 file should always be followed by read 2 file
f1=/path/**LIBNAMEA**/fasta2_read_1.fa
f2=/path/**LIBNAMEA**/fasta2_read_2.fa
#fastq file for single reads
q=/path/**LIBNAMEA**/fastq1_read_single.fq
#another fastq file for single reads
q=/path/**LIBNAMEA**/fastq2_read_single.fq
#fasta file for single reads
f=/path/**LIBNAMEA**/fasta1_read_single.fa
#another fasta file for single reads
f=/path/**LIBNAMEA**/fasta2_read_single.fa
#a single fasta file for paired reads
```

p=/path/**LIBNAMEA**/pairs1_in_one_file.fa
#another single fasta file for paired reads
p=/path/**LIBNAMEA**/pairs2_in_one_file.fa
#bam file for single or paired reads, reads 1 in paired reads file should always be followed by reads 2
# NOTE: If a read in bam file fails platform/vendor quality checks(the flag field 0x0200 is set), itself and it's paired read would be ignored.
b=/path/**LIBNAMEA**/reads1_in_file.bam
#another bam file for single or paired reads
b=/path/**LIBNAMEA**/reads2_in_file.bam
[LIB]
avg_ins=2000
reverse_seq=1
asm_flags=2
rank=2
# cutoff of pair number for a reliable connection (at least 5 for large insert size)
pair_num_cutoff=5
#minimum aligned length to contigs for a reliable read location (at least 35 for large insert size)
map_len=35
q1=/path/**LIBNAMEB**/fastq_read_1.fq
q2=/path/**LIBNAMEB**/fastq_read_2.fq
f1=/path/**LIBNAMEA**/fasta_read_1.fa
f2=/path/**LIBNAMEA**/fasta_read_2.fa
p=/path/**LIBNAMEA**/pairs_in_one_file.fa
b=/path/**LIBNAMEA**/reads_in_file.bam

****

Appendix B: output files

1. Output files from the command "pregraph"

a. *.kmerFreq

Each row shows the number of Kmers with a frequency equals the row number. Note that those peaks of frequencies which are the integral multiple of 63 are due to the data structure.

b. *.edge

Each record gives the information of an edge in the pre-graph: length, Kmers on both ends, average kmer coverage, whether it's reverse-complementarily identical and the sequence.

c. *.markOnEdge & *.path

These two files are for using reads to solve small repeats.

e. *.preArc

Connections between edges which are established by the read paths.

f. *.vertex

Kmers at the ends of edges.

g. *.preGraphBasic

Some basic information about the pre-graph: number of vertex, K value, number of edges, maximum read length etc.

2. Output files from the command "contig"

a. *.contig

Contig information: corresponding edge index, length, kmer coverage, whether it's tip and the sequence. Either a contig or its reverse complementry counterpart is included. Each reverse complementary contig index is indicated in the *.ContigIndex file.

b. *.Arc

Arcs coming out of each edge and their corresponding coverage by reads

c. *.updated.edge

Some information for each edge in graph: length, Kmers at both ends, index difference between the reverse-complementary edge and this one.

d. *.ContigIndex

Each record gives information about each contig in the *.contig: it's edge index, length, the index difference between its reverse-complementary counterpart and itself.

3. Output files from the command "map"

a. *.peGrads

Information for each clone library: insert-size, read index upper bound, rank and pair number cutoff for a reliable link. This file can be revised manually for scaffolding tuning.

b. *.readOnContig

Reads' locations on contigs. Here contigs are referred by their edge index. Howerver about half of them are not listed in the *.contig file for their reverse-complementary counterparts are included already.

c. *.readInGap

This file includes reads that could be located in gaps between contigs. This information will be used to close gaps in scaffolds if "-F" is set.

4. Output files from the command "scaff"

a. *.newContigIndex

Contigs are sorted according their length before scaffolding. Their new index are listed in this file. This is useful if one wants to corresponds contigs in *.contig with those in *.links.

b. *.links

Links between contigs which are established by read pairs. New index are used.

c. *.scaf_gap

Contigs in gaps found by contig graph outputted by the contiging procedure. Here new index are used.

d. *.scaf

Contigs for each scaffold: contig index (concordant to index in *.contig), approximate start position on scaffold, orientation, contig length, and its links to others contigs.

e. *.gapSeq

Gap sequences between contigs.

f. *.scafSeq

Sequences of each scaffolds.

g. *.contigPosInscaff

Contigs' positions in each scaffold.

h. *.bubbleInScaff

Contigs that form bubble structures in scaffolds. Every two contigs form a bubble and the contig with higher coverage will be kept in scaffold.

i. *.scafStatistics

Statistic information of final scaffold and contig.