Does subsampling improve assembly quality for very high coverage datasets?

Umer Zeeshan Ijaz

I assembled Nanoarchaeum Equitans for which there were 1639910 trimmed forward and reverse reads (Trimming was done using Sickle version 1.200 with a minimum window quality score of 20. Reads shorter than 10 bp. after trimming were removed). This dataset was sequenced on MISEQ machine using Nextera library.
I used all 1639910 reads in VelvetOptimiser (http://bioinformatics.net.au/software.velvetoptimiser.shtml) with n50 score as an optimisation criteria and kmer range as [21,99]. In the final assembly, I found optimum kmer size=99 and obtained 13795 contigs with an n50 score of 201 which was NOT ACCEPTABLE. Neil Hall (Liverpool) suggested that perhaps, the problem is that we have too much coverage which is propagating errors because of depth of assignments and may be subsampling is the way to go.
So I subsampled from the reads taking 10% (163991), 20% (327982), 30% (491973), 40% (655964), and 50% (819955), 60% (983946), and 70% (1147937) of the reads respectively.
I then wrote a fast paired-end fastq files subsampler as a bash one-liner (given below) since with Illumina we are playing with millions of reads and execution time is important. The optimal algorithm is to use reservoir sampling which has space complexity of O(n) instead of O(N) where we want to sample n elements from a pool of N.

Reservoir sampling:

A simple random sampling strategy to produce a sample without replacement from a stream of data - that is, in one pass: O(N)

Want to sample s instances - uniformly at random without replacement - from a population size of n records, where n is not known.

Figuring out n would require 2 passes. Reservoir sampling achieves this in 1 pass.
A reservoir R here is simply an array of size s. Let D be data stream of size n

Algorithm:

```
Store first s elements into R.
for each element in position k = s+1 to n ,
   accept it with probability s/k
   if accepted, choose a random element from R to replace.
```

Partial analysis:

Base case is trivial. For the k+1st case, the probability a given element i with position <= k is in R is s/k. The prob. i is replaced is the probability k+1st element is chosen multiplied by i being chosen to be replaced, which is: s/(k+1) * 1/s = 1/(k+1), and prob that i is not replaced is k/k+1.

So any given element's probability of lasting after k+1 rounds is: (chosen in k steps, and not removed in k steps)

= s/k * k/(k+1), which is s/(k+1).

So, when k+1 = n, any element is present with probability s/n.

Reference: http://blogs.msdn.com/b/spt/archive/2008/02/05/reservoir-sampling.aspx

Here are the commands used:

Subsampling 10% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=163991 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p10.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p10.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p10.fastq N54_130624_L001_R2_p10.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 20% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=327982 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p20.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p20.fastq"}'
```

```
nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p20.fastq N54_130624_L001_R2_p20.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 30% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=491973 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p30.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p30.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p30.fastq N54_130624_L001_R2_p30.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 40% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=655964 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p40.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p40.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p40.fastq N54_130624_L001_R2_p40.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 50% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=819955 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p50.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p50.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p50.fastq N54_130624_L001_R2_p50.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 60% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=983946 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p60.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p60.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p60.fastq N54_130624_L001_R2_p60.fastq' -t 10 --optFuncKmer 'n50' &
```

Subsampling 70% reads:

```
paste N54_130624_L001_R1.fastq N54_130624_L001_R2.fastq | awk '{ printf("%s",$0); n++; if(n%4==0) { printf("\n");}
else { printf("\t");} }' | awk -v k=1147937 'BEGIN{srand(systime() + PROCINFO["pid"]);}{s=x++<k?x-
1:int(rand()*x);if(s<k)R[s]=$0}END{for(i in R)print R[i]}' | awk -F"\t" '{print $1"\n"$3"\n"$5"\n"$7 >
"N54_130624_L001_R1_p70.fastq";print $2"\n"$4"\n"$6"\n"$8 > "N54_130624_L001_R2_p70.fastq"}'

nohup /home/opt/VelvetOptimiser/VelvetOptimiser.pl -s 21 -e 99 -f '-shortPaired -fastq -separate
N54_130624_L001_R1_p70.fastq N54_130624_L001_R2_p70.fastq' -t 10 --optFuncKmer 'n50' &
```

Here are the results:

% of reads	Optimum kmer size	**No of contigs**	**n50 score**	Length of longest contig	Number of contigs > 1k
100%	99	**13795**	**201**	1950	28
70%	99	**2386**	**1564**	288878	18
60%	99	**611**	**396479**	396479	11
50%	95	**151**	**489315**	489315	14
40%	95	**265**	**451202**	451202	16
30%	95	**661**	**457328**	457328	10
20%	99	**330**	**101099**	131609	16
10%	95	**1277**	**2263**	13314	121

To validate assembly:

We will use the following paper:

"Read Length and Repeat Resolution: Exploring Prokaryote Genomes Using Next-Generation Sequencing Technologies"
Reference: http://www.plosone.org/article/info%253Adoi%252F10.1371%252Fjournal.pone.0011518

Table S1 is an excel sheet that gives read length and gap benchmarks for 818 bacterial genomes.
From there, for the closely-related strain Nanoarchaeum equitans Kin4-M (genome length: 490885), with 125bp reads, 2 gaps are predicted, and with 250bp reads, 0 gaps are predicted in final assembly.

It can be seen that with 50% of the reads, the longest contig 489315 is 99.68017% of the genome length of Nanoarchaeum equitans Kin4-M.

Verdict: Yes SUBSAMPLING does improve assembly quality for very high coverage datasets.