Evaluating Metagenome Assembly on a Complex Community

Sherine Awad [1], Titus Brown [1,*]

1 Population Health and Reproduction University of California, Davis, Davis, CA, USA

∗ **E-mail: ctbrown@ucdavis.edu**

Abstract

SUPER WEAK SO FAR

Motivation: With the emergence of de novo assembly, several work have been to done to assemble metagenomic data from de novo. Several assemblers exist that are based on different assembly techniques. However, we still lack a study that analyze different assemblers behavior on metagenomic data .

Problem statement: In this paper, we performed analytical study for metagnome assembly using different assemblers and different preprocessing treatments. The aim of the analysis is studying how well metagenome assembly works, and which assembly works best. In addition, the study analyzes the resource requirements of the assembly.

Approach: We used a mock community dataset for the analysis, and used its reference genome for benchmark evaluation. We quality filtered the reads, then we applied 2 other preprocessing steps: digital normalization and partitioning. We used 4 different assembler: Velvet, IDBA-UD, SPAdes, and megahit to assemble the reads using each treatment. We used Quast to analyze assemblies accuracy.

Results: Results show that assembly works well. Velvet is the worst assembler in terms of accuracy and recourses utilizations. The results also showed that assembly counts to most of the reads.

Conclusions: Except for Velvet, assemblers works well. Further analysis is required to study which assembler is better used with each specific dataset. This step is left for our future work,

Author Summary

WHAT SHOULD BE WRITTEN HERE

Introduction

Metagenome is the sequencing of DNA in an environmental sample. While whole genome sequencing (WGS) usually targets one genome, metagenome targets several ones which introduces complexity to metagenome analyis due to genomic diversity and variable abundance within populations. Metagenomic assembly means the assembly of multiple genomes from mixed sequences of reads of multiple species in a microbial community. Most approaches for analyzing metagenomic data rely on mapping to reference genomes. However, not all microbial diversity of many environments are covered by reference databases. Hence, the need for de novo assembly of complex metagenomic data rises. Several assemblers exist that can be used for de novo assembly. In order to decide which assembly works best, we need to evaluate metagenome assembly generated by each assembler. In this paper, we provide, an evaluation for metegnome assembly generated by several assemblers and using different preprocessing treatments. We use a reference genome as a benchmark for the evaluation. The evaluation is based on assembly accuracy, and time and memory requirements. This evaluation shed light on doability of metagenome assembly and the minimum requirements needed for the assembly. In addition, knowing how each assembler works, helps deciding which assembler to use prior to assembly. However, the later point is left for our future work.

The comparative study in this paper is based on four different assemblers; Velvet [1], SPAdes [2], IDBA-UD [3], and Megahit [4].

Velvet [1] is a group de Bruin graph-based sequence assembly methods for very short reads that can both remove errors. It also uses read pair information to resolve a large number of repeats. The

error correction algorithm merges the sequences that belongs together. Then the repeat solver algorithm separates parts that share overlaps.

Spades [2] is an assembler for both single-cell and standard (multicell) assembly. SPAdes generates single-cell assemblies and provides information about genomes of uncultivatable bacteria that vastly exceeds what may be obtained via traditional metagenomics studies.

IDBA-UD [3] is a de Bruijn graph approach for assembling reads from single cell sequencing or metagenomic sequencing technologies with uneven sequencing depths. IDBA-UD uses multiple depth-relative thresholds to remove erroneous k-mers in both low-depth and high-depth regions. It also uses paired-end information to solve the branch problem of low-depth short repeat regions. It applies and error correction step to correct reads of high-depth regions that can be aligned to high confident contigs.

Megahit [4] is a new approach that constructs a succinct de Bruijn graph using multiple k-mers, and uses a novel "mercy k-mer" approach that preserves low-abundance regions of reads. It also uses GPUs to accelerate the graph construction.

Materials and Methods

Datasets

We used a diverse mock community data set [5]. Raw reads contains 5536.29 megabases and 54,814,748 sequences for each file. In total, 11072.58 megabases and 109,629,496 sequences. The set also has a reference genome downloaded from XX. Using a reference genome makes the evaluation process easier and much better in terms of accuracy measures.

Quality Filtering

We trimmed low-quality bases using trimmomatic-0.30.jar [6] and using PE and using TruSeq3 adapter. We also used Fastq quality filter from FASTX-Toolkit to filter sequences with low qualities with these parameters -Q33 -q 30 -p 50. The fastx utilities that arent paired-end aware; theyre removing individual sequences. Because the paired-end files are interleaved, this means that there may now be some orphaned sequences in there. Downstream, we will want to pay special attention to the remaining paired sequences, so we use "extract-paired-reads.py" script in khmer/scripts to separate out the paired-end and single-end files.

Digital Normalization

First, we normalize everything to a coverage of 20, starting with the (more valuable) paired-ended reads by running "normalize-by-median.py" script in khmer/scripts. We used -p to keep pairs, k=20 for k-mer size , cutoff of coverage C=20, N=4 for number of k-mer counting tables, x =1e9 for lower bound on tablesize and with option savetable for further use. Then we used to the same script to normalize the single-ended reads. We then use "filter-abund.py" script in khmer/scripts to trim off any k-mers that are abundance-1 in high-coverage reads. The -V option is used for variable coverage. We ran "extract-paired-reads.py" in khmer/scripts to seperate paired-ended and single-ended reads. By now, weve eliminated many more erroneous k-mers. We then ditch some more high-coverage data, so we normalize the paired-end reads and single-end reads down to coverage 5 using "normalize-by-median.py" again.

Partitioning

First, eliminate highly repetitive k-mers that could join multiple species using "filter-below-abund.py" script found in khmer/sandbox. Then we run partitioning using do-partition script found in khmer/scripts. We used k =32 for k-mer size, x=1e9 for lower bound on tablesize to use and threads=4 for number of

simultaneous threads to execute. We then extracted partitions to groups using "extract-partitions.py" script in khmer/scripts/ with X=100000 for maximum group size. We finally, ran "extract-paired-reads.py" in khmer/scripts to seperate paired-ended and single-ended reads.

Metagenomes Assembly and evaluation

We assembled the reads using four different assemblers; Velvet [1], Idba [3], Spades [2], and Megahit [4] in combination with different preprocessing treatments; quality filtering, digital normalization, and partitioning. For Velvet [1], we used kmers values from 19 to 51 incremented by 2. We also used -fastq.gz for fatsq format, -shortPaired for the pe files and -short for the se files. Also, we used exp_cov auto cov_cutoff auto. For Idba [3], we also used –pre_correction and -r for the pe files.

We examined the assembly quality of each assembler and treatment using Quast [7] using quast.py with these parameters -R for the reference genome and -o for the output.

Results

Quality filtering did not change the number of reads

After trimming, we got 11024.50 megabases and 109,153,498 sequences for the pair-end reads and 14.86 megabases and 235,966 sequences for the single-end reads. In total, 11039.36 megabases and 109,389,464 sequences. After fastx, we got 10547.80 megabases and 104,433,622 sequences for the pair-end reads and 184.44 megabases and 106,326,865 sequences for the single-end reads. In total, 10732.23 megabases and 106,326,865 sequences. In summary, the quality filtering eliminates 3.01% of the reads.

Digital normalization and partitioning decreased the total number of reads

After digital normalization, the pair ended file contains 1687.59 megabases and 16,853,716 sequences while the single ended file contains 5.86 megabases and 64,638 sequences. After partitioning, we got 28 partitions. The largest partition has 1379.27 megabases. The smallest partition has 7.14 megabases. The total base pairs in all partitions is 1651.53 megabases. Clearly, digital normalizatation and partitioning decrease the total number of reads.

Metagenome assemblers recover the great majority of the known content

Table 1 shows various quality metrics for the results of the assembly using combinations of four different assemblers and different preprocessing treatments. The unaligned length for assembly is 8,977,149 bp, 10,709,716 bp, 10,597,529 bp, and 10,686,421 bp using quality filtered reads for Velvet, IDBA, SPAdes, and Megahit respectively. The genome fraction % is the percentage of aligned bases in the reference. A base in the reference is aligned if there is at least one contig with at least one alignment to this base. The genome fraction percentage is 72.949 %, 90.969 %, 90.424%, and 90.358% using quality filtered reads for Velvet, IDBA, SPAdes, and Megahit respectively. Misassembled contigs length is the total number of bases in misassembled contigs. Misassembles contigs length is 631, 1032, 752, and 648 using quality filtered reads for Velvet, IDBA, SPAdes, and Megahit respectively. Clearly, SPAdes, IDBA, and Megahit have fairly similar results and they all outperformed Velvet.

Digital normalization and partitioning reduced memory and time requirements for assembly

In this section, we aim to explore time and memory requirements of metagenome assembly using each preprocessing treatments. Table 2 shows the running time and memory utilizations for four assemblers

and different reads treatments.

For Velvet assemblies, it took $\sim$ 60 hours using quality filtered reads, while it took only $\sim$ 6 hours using digital normalizations and $\sim$ 4 hours using partitioning which is approximately 10% and less of time utilized using quality filtered reads. For IDBA assemblies, it took $\sim$ 33 hours using quality filtered reads, while it took $\sim$ 6 hours using digital normalization and $\sim$ 8 hours using partitioning, approximately less than $\sim$ 7% of time utilized using quality filtered reads. SPAdes assemblies utilized $\sim$ 67 hours using quality filtered reads while it took $\sim$ 15 hours and $\sim$ 7 hours using digital normalization and partitioning respectively less than 5% of time utilized using quality filtered reads. Finally, for megahit, it took $\sim$ 2 hours, $\sim$ half an hour, and $\sim$ hour and a half using quality-filtered reads, digital normalization, and partitioning respectively.

For Velvet assemblies, it used used 1594.85 GB of memory using quality filtered reads, while it used one 827.41 GB and 1156.73 GB of memory when applying digital normalization and partitioning respectively. For IDBA assemblies, it used used 129.85 GB of memory using quality filtered reads, while it used one 104.74 GB and 93.58 GB of memory when applying digital normalization and partitioning respectively. For SPAdes assemblies, it used used 400.34 GB of memory using quality filtered reads, while it used one 127.42 GB and 129.71 GB of memory when applying digital normalization and partitioning respectively. For megahit, it utilizes 35.03 GB,419.80 GB,198.76 GB for quality-filtered reads, digital normalization, and partitioning respectively.

See Table 2 for more details. We conclude that Digitial normalization and partitioning treatments reduced time and memory requirements of assembly while they didn't affect assemblies quality as we shown in previous section.

Something about misassemblies

As shown in Table 1, using quality filtered reads, misassemblies contigs length are 16,566,891, 21,777,032, 28,238,787 and 11,927,502 for Velvet, SPAdes, IDBA, and Megahit respectively. Clearly, Velvet and Megahit has fewer misassemblies.

Velvet/quality filtering shows the least mismatches percentage 0.06%. IDBA/quality filtering, SPAdes/quality filtering, and Megahit/quality filtering have 0.08%, 0.09%, and 0.08% mismatches percentages respectively. Digital normalization and partitioning slightly increased the mismatches percentages. For Velvet, mismatches increased from 0.06% using qualitfy filtering to 0.09% and 0.12% using digital normalization and partitioning respectively. For IDBA, the percentage increased from 0.08% using quality filtering, to 0.12% for both digital normalization and partitioning. For SPAdes, the precentage increased from 0.09% using quality filtering to 0.12% for both digital normalization and partitioning. For Megahit, it increased from 0.08% using quality filtering to 0.10% for both digital normalization and partitioning. Indels percentage is 0.03% for Velvet, 0.01% for IDBA, SPAdes, and megahit. Clearly, the mismatches percentage is very low. Digital normalization and partitioning unsignificantly increased the mismatches.

See Table 3 for more details about misassemblies contigs, the types of missassembly events, mismatches and indels lengths.

Metagenome assemblies account for the majority of reads

In order to find out how many reads are covered by each assembly, we mapped the quality filtered reads to each assembly using bwa [8]. Then we extracted the unaligned sequences. Table 4 shows the number and percentages of unaligned sequences from mapping quality filtered reads to each assembly treatment using the four assemblers under study. For all treatments assemblies, the full set of trimmed reads were used for mapping. Default parameters were used, and both paired ends and singletons were mapped. Samtools [9] was used for format conversion from SAM to BAM format, and also to calculate the percentage of mapped reads. We conclude that assemblies account for the majority of reads. For quality-filtered assembly,

the number of unaligned reads is 5,553,831, 124,846, 81,775, and 80,002 for Velvet, IDBA, SPAdes, and megahit respectively. This represents 5.22 %, 0.12%, 0.08%, and 0.08% of the quality filtered sequences. This reflects that assemblies account for the majority of reads. See Table 4 for more details.

As mentioned, digital normalization and partitioning decreased the number of total reads. Meanwhile, the genome fraction percentage and unaligned length didn't significantly change, which shows that digital normalization and partitioning throw unnecessary reads. For velvet, genome fraction is 89.043% and 88.879% using digital normalization and partitioning respectively (versus 72.949% using quality filtered reads). For IDBA, genome fraction is 91.003% and 90.082% using digital normalization and partitioning respectively (versus 90.969% using quality filtered reads). For SPAdes, genome fraction is 90.173% and 89.272% using digital normalization and partitioning respectively (versus 90.424% using quality filtered reads). For megahit, genome fraction is 89.92% and 88.769% using digital normalization and partitioning respectively (versus 90.358% using quality filtered reads). See Table 1 for more details.

HOW TO COMPUTE PERCENTAGE WE HAVE 104433622 seqs in SRR606249.pe.qc.fq.gz and 1893243 seqs; in SRR606249.se.qc.fq.gz

Something about unknown and new assembly

To examine how close assemblies are, we aligned the unaligned contigs of each assembly with different treatments to the unaligned contigs of IDBA assembly using quality filtered treatment. Using quality filtered reads, the genome fraction equals 80.613%, 91.922%, and 92.715% for Velvet, SPAdes, and Megahit respectively.

The unaligned length is 2,475,529, 2,174,574, and 916,247 for Velvet, SPAdes, and Megahit respectively using quality filtered reads, representing 37.06%, 32.56% and 13.72% of the reference length which is IDBA unaligned contigs using quality filtered reads.

The misassembly contigs length is 360,191,1,962,380, and 775,917 for Velvet, SPAdes, and Megahit respectively using quality filtered reads. See Table 5 for more details. The percentages are fairly close except for Velvet, which show that unalignments are common among the four different assemblers and the unalignments are likely to be unknowns, new assembly, or contamination.

Discussion

Assembly works pretty well

Except for Velvet assembly using quality filtered reads, the genome fraction percentage is 88% or higher. Unaligned length is less than 1% for all assemblers and using different treatments. Misassembled length is less than 1.3% for all assemblers and using different treatments. We conclude that assembly works well although there are some rooms for improvements including enhancing accuracy, and decreasing time and memory requirements. Velvet shows the least performance in terms of accuarcy and time, and memory utlizations. However, the difference between other assemblers are not significant. Hence, more investigations are needed to decide what assembler to use prior assembly. Such analysis is left for our future work.

Digital normalization and partitioning significantly reduce running time and memory utilizations

The difference between genome fraction percentage using quality filtered reads versus digital normalizations and partitioning doesn't exceed 1%. However, the time and memory resource are reduced a lot using digital normalization and partitioning. We conclude that digital normalization and partitioning are beneficial steps for assembly to reduce time and memory utilities.

Digitial normalization and partitioning do not affect misassemblies and unalignments

Except for Velvet assemblies, misassemblies are not affected by digital normalization and partitioning. Mapping the unaligned contigs of different assemblies and different treatments to the unaligned contigs of IDBA assembly using quality filtered , shows genome fraction percentage is 91% or higher. This means the unaligned contigs are common among assemblers with various treatments and they are likely to be unknowns, new assemblies, or contamination. XXX MAP TO UNALIGN INTERPRETATION This indicates that digital normalization and partitioning enhance assembly time and memory requirements without affecting assembly accuracy.

Acknowledgments

References

1. Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de bruijn graphs. Genome Research 18: 821-829.
2. Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, et al. (2012) Spades: A new genome assembly algorithm and its applications to single-cell sequencing. Journal of Computational Biology 19: 455-477.
3. Peng Y, Leung HC, Yiu S, Chin FY (2012) Idba-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. Bioinformatics 28: 1420-1428.
4. Li D, Liu CM, Luo R, Sadakane K, Lam TW (2014) Megahit: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. Bioinformatics .
5. Migun S, Quince C, Campbell J, Yang Z, Schadt C, et al. (2013) Comparative metagenomic and rrna microbial diversity characterization using archaeal and bacterial synthetic communities. Enivromental Microbiology 15: 1882-1899.
6. Anthony B, Marc L, Bjoern U (2014) Trimmomatic: a flexible trimmer for illumina sequence data. Bioinformatics : btu170.
7. Gurevich A, Saveliev V, Vyahhi N, Tesler G (2013) Quast: quality assessment tool for genome assemblies. Bioinformatics 28: 1072-1075.
8. Li H (2013) Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. arXiv preprint arXiv:13033997 .
9. Heng L, Handsaker B, Wysoker A, Fennell T, Ruan J, et al. (2009) The sequence alignment/map format and samtools. Bioinformatics 25: 2078-2079.

Figure Legends

Tables

Supporting Information Legends

Table 1. Assembly Quality Metrics

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) Velvet			
Genome Fraction	72.949	89.043	88.879
Unaligned Length	8,977,149	10,909,693	11,317,834
Misassembled contigs length	16,566,891	25,594,315	16,922,852
N50	38,028	18,944	8,504
(2) Idba			
Genome Fraction	90.969	91.003	90.082
Unaligned Length	10,709,716	10,637,811	10,644,357
Misassembled contigs length	21,777,032	27,668,818	18,440,791
N50	4,977,3	4,782,8	2,657,5
(3) Spades			
Genome Fraction	90.424	90.173	89.272
Unaligned Length	10,597,529	10,621,398	10,500,235
Misassembled contigs length	28,238,787	23,103,154	14,338,099
N50	4,277,3	3,558,0	2,231,9
(4) Megahit			
Genome Fraction	90.358	89.92	88.769
Unaligned Length	10,686,421	10,581,435	10,564,244
Misassembled contigs length	11,927,502	17,319,534	11,814,070
N50	35,254	35,427	17,492

Table 2. Running Time and Memory Utilization

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) Velvet			
Running Time	60:42:52	6:48:46	4:30:36
Memory Utilization in GB	1594.85	827.41	1156.73
(2) Idba			
Running Time	33:53:46	6:34:24	8:30:29
‘ Memory Utilization in GB	129.85	104.74	93.58
(3) Spades			
Running Time	67:02:16	15:53:10	7:54:26
Memory Utilization in GB	400.34	127.42	129.71
(4) Megahit			
Running Time	1:52:55	0:30:23	1:23:28
Memory Utilization in GB	35.034	19.80	198.76

Table 3. Misassemblies

Assembly	Quality Filtering	Digital Normalization	Partition
(1) Velvet			
Misassemblies	917	5271	5202
Relocations	592	998	1036
Translocations	309	4262	4153
Inversions	16	11	13
Misassembled Contigs	631	3104	3337
Mismatches	104,740	174,446	178,348
Percentage of Mismatches	0.06%	0.09%	0.12%
Indels Length	50,190	181,453	346,988
Indels Percentage	0.03%	0.09%	0.18%
(3) IDBA			
Misassemblies	1223	1094	960
Relocations	613	668	578
Translocations	580	398	350
Inversions	30	28	32
Misassembled Contigs	1032	916	828
Mismatches	162,733	231,432	230,840
Percentage of Mismatches	0.08%	0.12%	0.12%
Indels Length	30,433	43,358	42,523
Indels Percentage	0.01%	0.02%	0.02%
(2) SPAdes			
Misassemblies	894	997	753
Relocations	608	613	496
Translocations	267	368	239
Inversions	19	16	18
Misassembled Contigs	752	881	654
Mismatches	184,630	244,849	235,396
Percentage of Mismatches	0.09%	0.12%	0.12%
Indels Length	27,328	32,783	21,516
Indels Percentage	0.01%	0.02%	0.01%
(4) Megahit			
Misassemblies	738	880	748
Relocations	448	593	513
Translocations	172	274	222
Inversions	118	13	13
Misassembled Contigs	648	780	677
Mismatches	152,964	207,349	203,515
Percentage of Mismatches	0.08%	0.10%	0.10%
Indels Length	15,298	18,195	16,517
Indels Percentage	0.01%	0.01%	0.01%

Table 4. Mapping quality-filtered reads to assemblies

Treatment	Quality Filtering	Digital Normalization	Partition
(1) Velvet			
No. of Unaligned Sequences	5,553,831	521,126	667,471
Percentage	5.22%	0.49%	0.63%
(2) Idba			
No. of Unaligned Sequences	124,846	115,323	512,264
Percentage	0.12%	0.11%	0.48%
(3) Spades			
No. of Unaligned Sequences	81,775	95,338	423,833
Percentage	0.08%	0.09%	0.40%
(4) Megahit			
No. of Unaligned Sequences	80,002	88,616	468,032
Percentage	0.08%	0.08%	0.44%

Table 5. Mapping unaligned contigs to Idba quality-filtered unaligned contigs

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) Velvet			
Genome Fraction	80.613	92.034	98.013
Unaligned Length	2,475,529	3,192,491	64,539,560
Misassembled contigs length	360,191	1,202,869	854,604
Percentage of unaligned	37.06%	47.80%	**966.31%**
(2) Idba			
Genome Fraction	-	91.53	94.738
Unaligned Length	-	498,299	37,437,754
Misassembled contigs length	-	2,214,766	2,164,075
Percentage of unaligned	-	7.46%	**560.53%**
(3) Spades			
Genome Fraction	91.922	93.959	94.826
Unaligned Length	2,174,574	1,951,911	2,398,664
Misassembled contigs length	1,962,380	1,575,890	1,282,230
Percentage of unaligned	32.56%	29.22%	35.91%
(4) Megahit			
Genome Fraction	92.715	91.838	92.219
Unaligned Length	916,247	1,569,436	3,832,050
Misassembled contigs length	775,917	1,585,005	1,192,449
Percentage of unaligned	13.72%	23.50%	**57.37%**