Evaluating Metagenome Assembly on a Complex Community

Sherine Awad [1], Luiz Irber [1], C. Titus Brown [1,*]
1 Population Health and Reproduction University of California, Davis, Davis, CA, USA
∗ E-mail: ctbrown@ucdavis.edu

Abstract

NEEDS ENHANCEMENTS

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 [**?**], SPAdes [**?**], IDBA-UD [**?**], and MEGAHIT [**?**].

Velvet [**?**] 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 [?] 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 [?] 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 [?] 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 containing 64 known species, sequenced with Illumina HiSeq, yielding 109,629,496 paired-end sequences with an untrimmed length of 11.07 Gbp and an estimated insert size of $\sim$ 380 [?].

We received the reference genomes from the original authors (posted on j) and the original reads are available through the NCBI Sequence Read Archive at Accession SRX200676. Figure **??** shows the coverage profile of the reference genome, and the percentage of read with that coverage.

Quality Filtering

We removed adapters with Trimmomatic v0.30 in paired-end mode with the Truseq adapters [?]. We next used the fastq_quality_filter from the FASTX-Toolkit v0.0.13.2 [?] to remove sequences using the parameters -Q33 -q 30 -p 50, which keeps all sequences with 50% or more bases with quality score greater than or equal to 30.

Mapping

We aligned all quality-filtered reads to the reference metagenome with bwa aln (v0.7.7.r441) [?]. We aligned paired-end and orphaned reads separately using bwa aln samse. We then used samtools (v0.1.19) [?] to convert SAM files to BAM files for both paired-end and orphaned reads. To count the unaligned reads, we find the records with the “4” flag in the SAM files [?].

We found chimeric alignments with the bwa mem aligner using the default parameters (v0.7.7.r441). Chimeric alignments cut reads in two (or more). For each chimeric alignment, in the SAM file, there will be a primary alignment and at least one secondary alignment tagged SA. To count the chimeric alignments, we count the records with the “SA” flag in the SAM file [?].

To extract the reads that contribute to unaligned contigs, we mapped the quality filtered reads to the unaligned contigs using bwa aln (v0.7.7.r441) [?]. Then we used samtools [?] to retrieve the reads that are mapped to the unaligned contigs.

Reference Coverage and Coverage Profile

To evaluate how much of the reference genome was contained in the read data, we used bwa aln to map reads to the reference genome. We then calculated how many reference bases are contained in at least one mapped read (script `sam-calc-refcov-cmp.py`).

Digital Normalization

We applied the `normalize-by-median.py` script from khmer v1.1 to execute abundance normalization (“digital normalization”, [?]) on the data, retaining paired reads and using a k-mer size of 20 (-p -k 20). We executed digital normalization with 4 hash tables, each 1 GB in size (-N 4 -x 1e9). After read normalization, we used the `filter-abund.py` script to trim high-abundance reads (estimated k-mer coverage ≥ 20) at low-abundance k-mers (k-mer abundance ≤ 2) to remove erroneous k-mers [?] [?].

Partitioning

We next applied partitioning to the data [?, ?]. We first eliminated high-abundance k-mers that could join multiple species bins using the `filter-below-abund.py` script from khmer v1.1 using an abundance cutoff of 50 or higher. We then ran `do-partition.py` with a k-mer size of 32 and 4 Bloom filters each of size 1 gigabit for partitioning (-k 32 -N 4 -x 1e9). After partitioning, partitions were extracted to groups using the `extract-partitions.py` script with a maximum group size of 100,000 (`-X 100000`).

Metagenomes Assembly and evaluation

We assembled the reads using four different assemblers: Velvet [?], IDBA-UD [?], SPAdes [?], and MEGAHIT [?].

For Velvet v1.2.07 [?], we used k-mer values from 19 to 51 incremented by 2. We also used -fastq.gz for fastq format, -shortPaired for the pe files and -short for the se files. Also, we asked Velvet to automatically calculate expected coverage and coverage cutoff (`-exp_cov auto -cov_cutoff auto`). From among the many assemblies, one for each k-mer size, we then chose the assembly that had the most bases in contigs longer than 500 bp (script `calc-best-assembly.py`).

For IDBA-UD v1.1.1 [?], we used –pre_correction to perform pre-correction before assembly and -r for the pe files. For SPAdes v3.1.1 [?], we used –sc –pe1-12 where –sc is required for MDA (single-cell) data and –pe1-12 for file with interlaced reads for the first paired-end library.

For MEGAHIT [?], we used -l 101 -m 3e9 –cpu-only where -l is for maximum read length , -m is for max memory in byte to be used, and –cpu-only to use CPU not GPU.

We examined the assembly quality of each assembler and treatment using QUAST v2.3 [?] using quast.py and we use the default minimum contig length equal to 500.

Results

Cost of Assembly

We estimated time and memory requirements for each of them. We also estimated the running time and memory utilization for each assembler under both treatments and compared to assemblers time and memory requirements using quality filtered reads.

Digital normalization utilized 74.93 GB of memory and took around 3 hours and 53 minutes to run. Partitioning utilized 21.78 GB and around 2 hours and a half to run.

For Velvet assemblies, table **??** row 3, it took ~ 60 hours using quality filtered reads, while it took only ~ 6 hours using digital normalizations and ~ 4 hours using partitioning. For IDBA-UD assemblies, table **??** row 6, it took ~ 33 hours using quality filtered reads, while it took ~ 6 hours using digital normalization and ~ 8 hours using partitioning. SPAdes assemblies utilized ~ 67 hours using quality filtered reads while it took ~ 15 hours and ~ 7 hours using digital normalization and partitioning respectively, table **??** row 9.

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, table **??** row 12.

For Velvet assemblies, table **??** row 4, it used 98.40 GB of memory using quality filtered reads, while it used one 52.67 GB and 35.23 GB of memory when applying digital normalization and partitioning respectively. For IDBA-UD assemblies, table **??** row 7, it used used 123.84 GB of memory using quality filtered reads, while it used one 99.88 GB and 76.53 GB of memory when applying digital normalization and partitioning respectively. For SPAdes assemblies, table **??** row 10, it used 381.79 GB of memory using quality filtered reads, while it used one 121.52 GB and 94.70 GB of memory when applying digital normalization and partitioning respectively. For MEGAHIT, table **??** row 13 it utilizes 33.41 GB, 18.89 GB, 13.17 GB for quality-filtered reads, digital normalization, and partitioning respectively. See Table **??** for more details.

In terms of compute cost, Megahit is the fastest and utilizes less memory. In conclusion, quality filtering utilizes more resources digital normalization and the later utilizes more resources than partitioning.

Assembly Comparisons

Evaluation Against the Reference Genome

Assembly alignment

We aligned each assembly to the reference genome using nucmer cite here. The we used our script analyze_assembly.py to analayze the results. IDBA quality filtered assembly has 27,444 contigs, out of which 71.36 % is totally aligned to the reference genome. For SPAdes quality filtered assembly, it has 33,704 contigs and 72.22% is totally aligned to the reference. Finally, Megahit has 75,497 contigs and 74.97% is totally aligned to the reference.

Read Incorporation

To evaluate how much of the reads are captured by the assembly, we mapped the quality filtered reads to each assembly. Then we extracted the unaligned sequences.

More about unalignments

We extracted the reads that contributed to unaligned contigs of each assembler (see Methods:Mapping). We mapped those reads to the reference genome. We find that most of the reads that mapped to the unaligned contigs don't exist in the reference genome. Figure **??** shows a histogram for the percentage of reads contributed to unaligned contigs (orange columns) and the portion of those reads that mapped to the reference genome (green columns).

More about misassemblies

We extracted all the partially aligned contigs. Using IDBA and quality filtering, 11.99% of the contigs were partially aligned. 49.73% of the partially aligned contigs has less than 50 bases only unaligned while 50.27% has 50 bases or more unaligned. Using SPAdes and quality filtering, 10.82% of the contigs were partially aligned. 51.44%of the partially aligned contigs has less than 50 bases only unaligned while 48.56% has 50 bases or more unaligned. Using Megahit and quality filtering, 10.45% of the contigs were partially aligned. 58.49%of the partially aligned contigs has less than 50 bases only unaligned while 41.51% has 50 bases or more unaligned.

More about uncovered regions

We used the same scrit to analyze the uncovered regions. Approximately 2.02% of the reference bases are uniquely covered by IDBA quality filtered assembly, ∼9.75% and ∼6.54% of the reference bases are uniquely covered by SPAdes and Megahit quality filtered assembly respectively using Besthit approach. Approximately ∼10.60% of reference bases are uniquely uncovered by IDBA quality filtered assembly, ∼7.62% and ∼3.44% of the reference bases are uniquely uncovered by SPAdes and Megahit quality filtered assemblies respectively using Besthit approach. Meanwhile, there are 31,446,810 (∼15.29%) of the reference bases commonly uncovered by IDBA, SPAdes, and Megahit quality filtered assemblies.

Subsambling

Furthermore, we aligned the unaligned contigs of each assembly to the unaligned contigs of IDBA-UD quality filtered assembly. Using quality filtered reads, the reference fraction (the reference here is the unaligned contigs of IDBA-UD quality filtered assembly) equals to 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. See Table **??** for more details. Velvet unalignments show the highest percentages of IDBA-UD QC unalignments. IDBA-UD diginorm unalignments shows the less percentage of IDBA-UD QC unalignments. This shows that not all reads are common among unaligned contigs.

Discussion

Assembly works well

Much of the reference is covered by the assembly. Most contigs are broadly accurate. (Q: How do we measure this?) Most genes within the reference are recovered. The assembly represents the majority of the reads (90%?) and the majority of the k-mers (XX).

Acknowledgments

Figure Legends

Tables

Supporting Information Legends

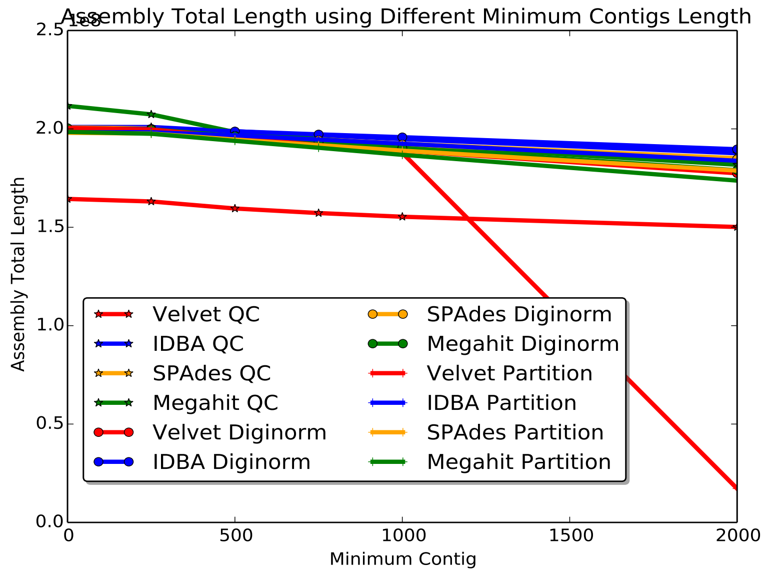

Figure 1. *Total Length of assemblies in basepairs based on different min contigs length.*

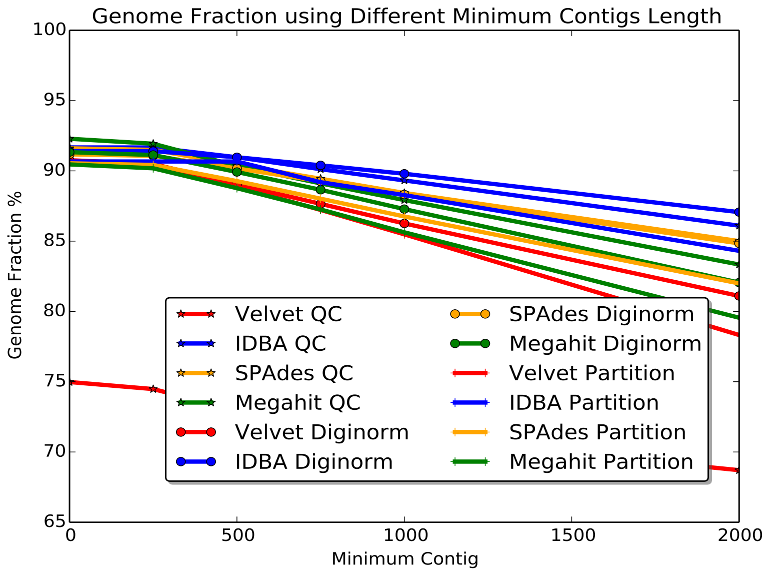

Figure 2. *Genome Fraction of assemblies in basepairs based on different min contigs length.*

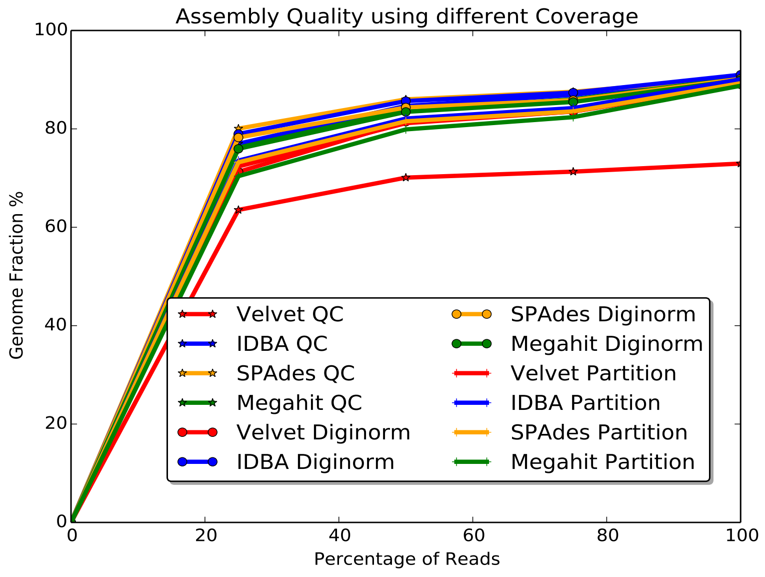

Figure 3. *Genome Fraction of assemblies using different read coverage.*

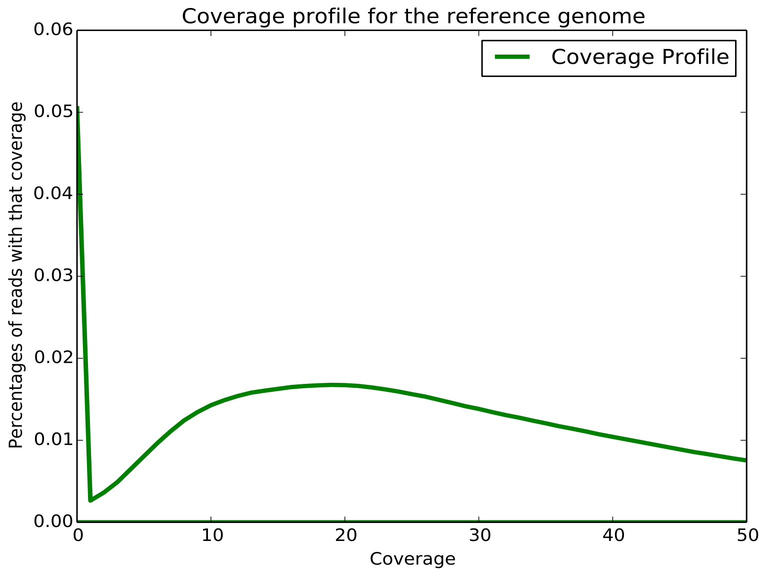

Figure 4. *Reference genome coverage.*

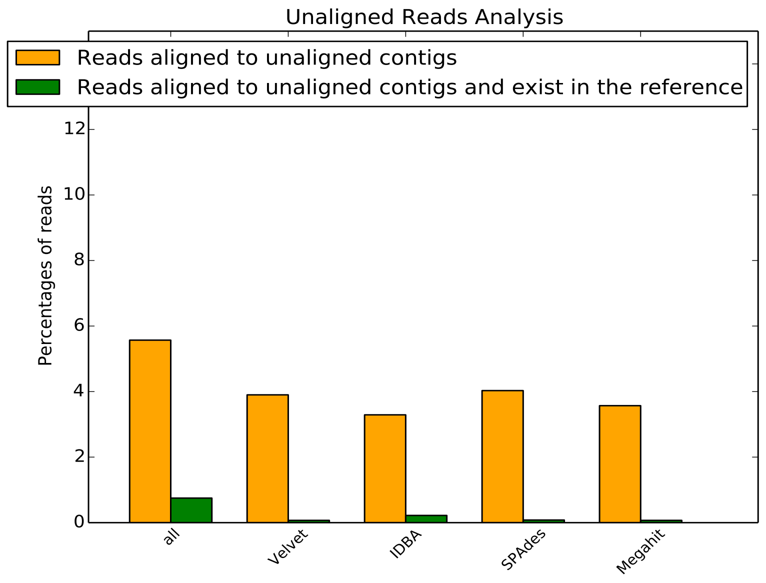

Figure 5. *Histogram for unaligned reads.*

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
NG50	22223	17212	7905
(2) IDBA-UD			
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	49773	47828	26575
NG50	45748	44351	24326
(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	42773	35580	22319
NG50	38841	32598	19909
(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	35,136	27,302	17,492
NG50	32251	25248	15393

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	98.40	52.67	35.23
(2) IDBA-UD			
Running Time	33:53:46	6:34:24	8:30:29
‘ **Memory Utilization in GB**	123.84	99.88	89.25
(3) SPADes			
Running Time	67:02:16	15:53:10	7:54:26
Memory Utilization in GB	381.79	121.52	123.7
(4) MEGAHIT			
Running Time	1:52:55	0:30:23	1:23:28
Memory Utilization in GB	33.41	18.89	189.55

Table 3. mis-assemblies

Assembly	**Quality Filtering**	**Digital Normalization**	**Partition**
(1) Velvet			
mis-assemblies	917	5271	5202
Relocations	592	998	1036
Translocations	309	4262	4153
Inversions	16	11	13
Misassembled Contigs Length	16,566,891	25,594,315	16,922,852
Mismatches	104,740	174,446	178,348
Percentage of Mismatches	0.05%	0.08%	0.09%
Indels Length	50,190	181,453	346,988
Indels Percentage	0.02%	0.09%	0.17%
(3) IDBA-UD			
mis-assemblies	1223	1094	960
Relocations	613	668	578
Translocations	580	398	350
Inversions	30	28	32
Misassembled Contigs Length	21,777,032	27,668,818	18,440,791
Mismatches	162,733	231,432	230,840
Percentage of Mismatches	0.08%	0.11%	0.11%
Indels Length	30,433	43,358	42,523
Indels Percentage	0.01%	0.02%	0.02%
(2) SPAdes			
mis-assemblies	894	997	753
Relocations	608	613	496
Translocations	267	368	239
Inversions	19	16	18
Misassembled Contigs Length	28,238,787	23,103,154	14,338,099
Mismatches	184,630	244,849	235,396
Percentage of Mismatches	0.09%	0.12%	0.11%
Indels Length	27,328	32,783	21,516
Indels Percentage	0.01%	0.02%	0.01%
(4) MEGAHIT			
mis-assemblies	738	880	748
Relocations	448	593	513
Translocations	172	274	222
Inversions	118	13	13
Misassembled Contigs Length	11,927,502	17,319,534	11,814,070
Mismatches	152,964	207,349	203,515
Percentage of Mismatches	0.07%	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	6,801,329	3,375,222	3,890,205
Percentage	6.40%	3.17%	3.66%
(2) IDBA-UD			
No. of Unaligned Sequences	1,490,609	1,738,371	2,297,377
Percentage	1.40%	1.63%	2.16%
(3) SPAdes			
No. of Unaligned Sequences	2,100,555	2,439,158	2,804,006
Percentage	1.98%	2.29%	2.64%
(4) MEGAHIT			
No. of Unaligned Sequences	1,559,300	2,082,881	2,747,427
Percentage	1.47%	1.96%	2.58%

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

Treatment/Quality Metric	**Quality Filtering**	**Digital Normalization**	**Partition**
(1) Velvet			
Unaligned IDBA Fraction	80.613 %	92.03%	92.982%
Unaligned Length	2,475,529	3192491	3,868,558
Percentage of unaligned	37.06%	47.8%	57.92%
(2) IDBA-UD			
Unaligned IDBA Fraction	-	91.53%	94.72%
Unaligned Length	-	498,299	1,320,036
Percentage of unaligned	-	7.46%	19.76%
(3) SPAdes			
Unaligned IDBA Fraction	91.92%	93.959%	94.826%
Unaligned Length	2,174,574	1,951,911	2,398,664
Percentage of unaligned	32.56%	29.22%	35.91%
(4) MEGAHIT			
Unaligned IDBA Fraction	92.715%	91.838%	92.219%
Unaligned Length	916,247	1,569,436	3,832,050
Percentage of unaligned	13.72%	23.5%	57.37%

Table 6. Supplementary Table: More Assembly Quality Metrics

Treatment/Quality Metric	**Quality Filtering**	**Digital Normalization**	**Partition**
(1) Velvet			
N75	13301	6084	3771
NG75	1186	4805	3214
L50	1013	2455	6037
LG50	1806	2740	6641
L75	2777	7026	14734
LG75	11087	8460	16867
(2) IDBA-UD			
N75	11693	12154	7834
NG75	9617	10221	6461
L50	828	896	1536
LG50	899	970	1712
L75	2986	3025	5062
LG75	3467	3484	6002
(2) SPAdes			
N75	11263	10554	6900
NG75	9005	8379	5401
L50	974	1192	1846
LG50	1078	1325	2108
L75	3276	3768	5840
LG75	3908	4495	7198
(4) MEGAHIT			
N75	8166	7230	5271
NG75	6601	5632	4030
L50	1199	1582	2490
LG50	1306	1757	2848
L75	4164	5063	7670
LG75	4907	6147	9581

Table 7. Comparision between N50 and NG50

Treatment	**Quality Filtering**	**Digital Normalization**	**Partition**
(1) Velvet			
N50	38,028	18,944	8,504
NG50	22,223	17,212	7,905
(2) IDBA-UD			
N50	49,773	47,828	26,575
NG50	45,748	44,351	24,326
(3) SPAdes			
N50	42,773	35,580	22,319
NG50	38,841	32,598	19,909
(4) MEGAHIT			
N50	35,136	27,302	17,492
NG50	32,251	25,248	15,393