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 4 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.

Treating Ambiguous Mapping

We define ambiguous mapping as any base that maps to more than one position in the reference genome. We used two approaches to dealing with ambiguous mapping: Best-hit and ambiguous approaches. In the best-hit approach, we consider only the mapping that has high confidence and discard the other mapping completely, so every base in the reference genome is hit once or zero times. In the ambiguous approach, we use all mapping, and whenever a base in the reference genome is hit more than one time, we count it's coverage once.

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 6 row 3, 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. For IDBA-UD assemblies, table 6 row 6, it took $\sim$ 33 hours using quality filtered reads, while it took $\sim$ 6 hours using digital normalization and $\sim$ 8 hours using partitioning. 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, table 6 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 6 row 12.

For Velvet assemblies, table 6 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 6 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 6 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 6 row 13 it utilizes 33.41 GB, 18.89 GB, 13.17 GB for quality-filtered reads, digital normalization, and partitioning respectively. See Table 6 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 5 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

Using quality filtering treatment and 25% coverage, we counted the genome coverage using both best-hit and ambiguous approaches. Using ambiguous approach, genome coverage is 77.38%, 82.53%, and 81.30% and with duplication ratio 1.38%, 1.24%, and 1.26% for IDBA, SPAdes, and Megahit respectively. Using best-hit approach, genome coverage is 63.10%, 70.6%, and 71.75% and with duplication ratio 0.27%, 0.40%, and 0.34 % for IDBA, SPAdes, and Megahit respectively.

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

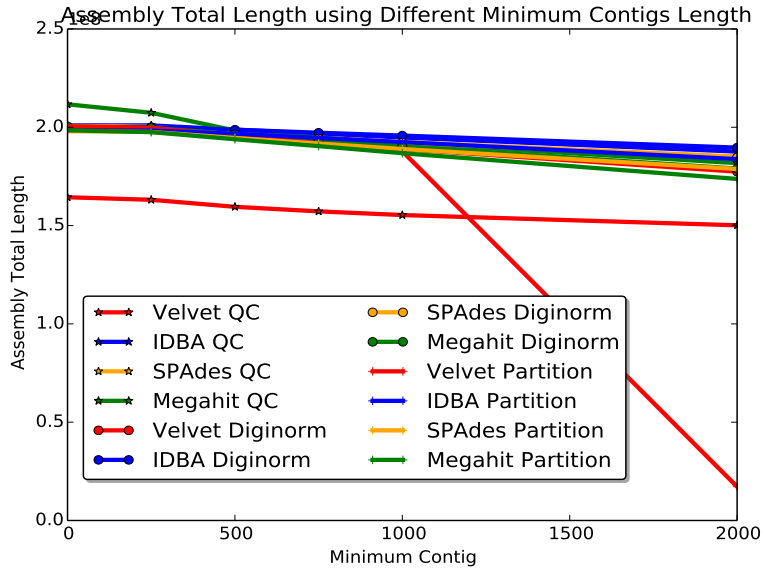

Figure 1. *Total Length of assemblies in basepairs based on different min contigs length.*

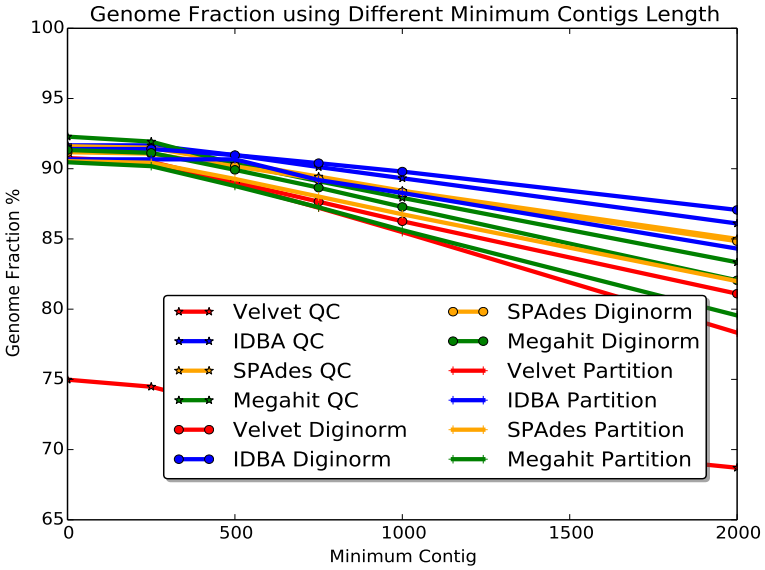

Figure 2. *Genome Fraction of assemblies in basepairs based on different min contigs length.*

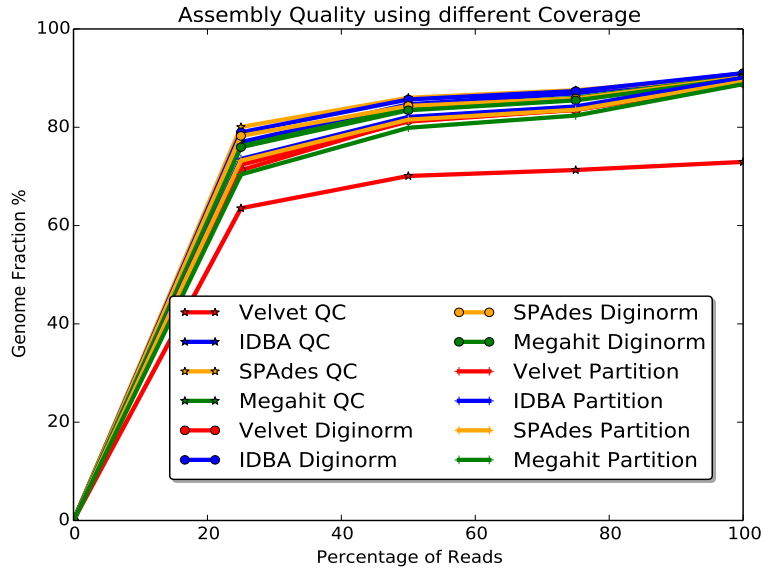

Figure 3. *Genome Fraction of assemblies using different read coverage.*

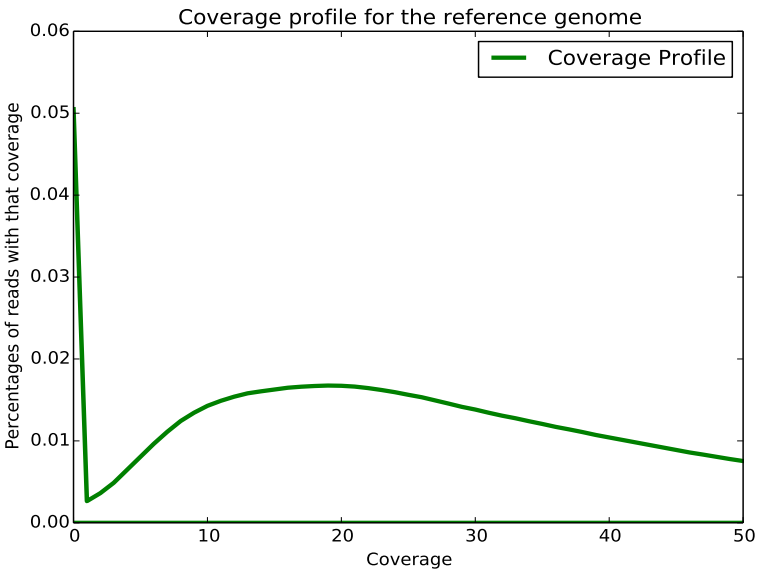

Figure 4. *Reference genome coverage.*

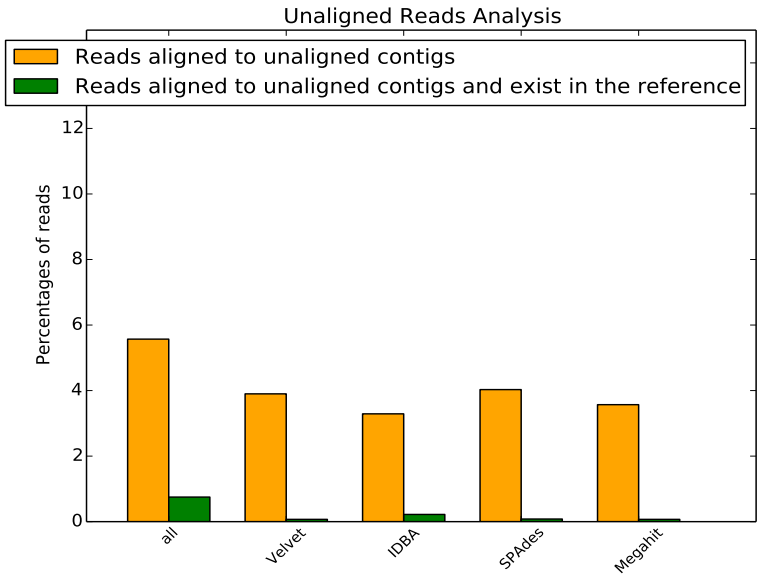

Figure 5. *Histogram for unaligned reads.*

Table 1. Reference Genome Coverage and Duplication Ratio

Minimum IDY	Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition	
(1) BEST HIT APPROACH					
(1) IDBA-UD					
99.0	**Genome Coverage%**	57.81%			
	Duplication Ratio	0.49			
95.0	**Genome Coverage%**	59.11%			
	Duplication Ratio	0.78			
(2) SPAdes					
99.0	**Genome Coverage%**	68.52%			
	Duplication Ratio	0.38			
95.0	**Genome Coverage%**	70.00 %			
	Duplication Ratio	0.52			
(3) MEGAHIT					
99.0	**Genome Coverage%**	69.50%			
	Duplication Ratio	4.25			
95.0	**Genome Coverage%**	70.99%			
	Duplication Ratio	4.70			
(2) AMBIGUOUS APPROACH					
(1) IDBA-UD					
99.0	**Genome Coverage%**	89.56%			
	Duplication Ratio	1.05			
95.0	**Genome Coverage%**	95.47%			
	Duplication Ratio	2.30			
(2) SPAdes					
99.0	**Genome Coverage%**	89.49%			
	Duplication Ratio	0.99			
95.0	**Genome Coverage%**	96.11%			
	Duplication Ratio	1.58			
(3) MEGAHIT					
99.0	**Genome Coverage%**	90.60%			
	Duplication Ratio	5.04			
95.0	**Genome Coverage%**	95.85%			
	Duplication Ratio	6.73			

Table 2. Contigs Analysis

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) BEST HIT APPROACH			
(1) IDBA-UD			
No. of Contigs	27,444		
Totally Aligned Contigs %	71.36%		
Partial Aligned Contigs %	11.99%		
Unaligned Contigs %	16.65%		
(2) SPAdes			
No. of Contigs	33,704		
Totally Aligned Contigs%	72.22%		
Partial Aligned Contigs%	10.82%		
Unaligned Contigs%	16.96%		
(3) MEGAHIT			
No. of Contigs	75,497		
Totally Aligned Contigs %	74.97%		
Partial Aligned Contigs%	10.45%		
Unaligned Contigs%	14.58%		
(2) AMBIGUOUS APPROACH			
(1) IDBA-UD			
No. of Contigs	27,444		
Totally Aligned Contigs%	77.56%		
Partial Aligned Contigs%	10.15%		
Unaligned Contigs %	12.28%		
(2) SPAdes			
No. of Contigs	33,704		
Totally Aligned Contigs	75.65%		
Partial Aligned Contigs%	8.12%		
Unaligned Contigs%	16.23%		
(3) MEGAHIT			
No. of Contigs	75,497		
Totally Aligned Contigs%	78.20%		
Partial Aligned Contigs%	9.22%		
Unaligned Contigs%	12.58%		

Table 3. More Coverage Analysis

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) BEST HIT APPROACH			
(1) IDBA-UD			
Uniquely Covered%	2.02%		
Uniquely Uncovered	10.60%		
(2) SPAdes			
Uniquely Covered%	9.75%		
Uniquely Uncovered%	7.62%		
(3) MEGAHIT			
Uniquely Covered%	6.54%		
Uniquely Uncovered%	3.44%		
(2) AMBIGUOUS APPROACH			
(1) IDBA-UD			
Uniquely Covered%	0.88%		
Uniquely Uncovered%	1.62%		
(2) SPAdes			
Uniquely Covered%	0.99%		
Uniquely Uncovered%	1.81%		
(3) MEGAHIT			
Uniquely Covered%	1.52%		
Uniquely Uncovered%	1.22%		

Table 4. Mapping assembly to Reads: Unmapped Reads

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) IDBA-UD			
Paired ends	1,375,436	1,650,015	2,481,486
Orphan Reads	273,943	279,632	313,468
Total %	0.016%	0.018%	0.026%
(2) SPAdes			
Paired ends	2,006,905	2,397,503	2,972,792
Orphan Reads	284,370	290,669	320,037
Total %	0.022%	0.025%	0.031%
(3) MEGAHIT			
Paired ends	1,435,006	1,991,470	2,936,122
Orphan Reads	263,879	285,330	320,941
Total %	0.016%	0.021%	0.030%

Table 5. Mapping Unaligned Contigs To Quality Filtered Reads: Unmapped Reads

Treatment/Quality Metric	Quality Filtering	Digital Normalization	Partition
(1) BEST HIT APPROACH			
Paired ends	32,267,775		
Orphan Reads	32,371,583		
Total %	%	%	%
(1) AMBIGUOUS APPROACH			
Paired ends	89,623,212		
Orphan Reads	1,610,872		
Total %	%	%	%

Table 6. 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 7. Comparision between N50 and NG50

Treatment	Quality Filtering	Digital Normalization	Partition
(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