BIOINFORMATICS ORIGINAL PAPER

Vol. 30 no. 19 2014, pages 2717–2722
doi:10.1093/bioinformatics/btu395

Genome analysis

Advance Access publication June 19, 2014

Omega: an Overlap-graph *de novo* Assembler for Metagenomics

Bahlul Haider[1], Tae-Hyuk Ahn[1], Brian Bushnell[2], Juanjuan Chai[1], Alex Copeland[2] and Chongle Pan[1,*]

[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831 and [2]U.S. Department of Energy, Joint Genome Institute, Walnut Creek, CA, 94598 USA

Associate Editor: John Hancock

ABSTRACT

Motivation: Metagenomic sequencing allows reconstruction of microbial genomes directly from environmental samples. Omega (overlap-graph metagenome assembler) was developed for assembling and scaffolding Illumina sequencing data of microbial communities.

Results: Omega found overlaps between reads using a prefix/suffix hash table. The overlap graph of reads was simplified by removing transitive edges and trimming short branches. Unitigs were generated based on minimum cost flow analysis of the overlap graph and then merged to contigs and scaffolds using mate-pair information. In comparison with three de Bruijn graph assemblers (SOAPdenovo, IDBA-UD and MetaVelvet), Omega provided comparable overall performance on a HiSeq 100-bp dataset and superior performance on a MiSeq 300-bp dataset. In comparison with Celera on the MiSeq dataset, Omega provided more continuous assemblies overall using a fraction of the computing time of existing overlap-layout-consensus assemblers. This indicates Omega can more efficiently assemble longer Illumina reads, and at deeper coverage, for metagenomic datasets.

Availability and implementation: Implemented in C++ with source code and binaries freely available at http://omega.omicsbio.org.

Contact: panc@ornl.gov

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on January 26, 2014; revised on June 12, 2014; accepted on June 13, 2014

1 INTRODUCTION

Metagenome assemblers attempt to reconstruct genomes of microorganisms in a community from its metagenomic sequencing data. In recent years, many isolate genome assemblers have been developed for Illumina sequencing data using de Bruijn graphs [e.g. ABySS (Simpson *et al.*, 2009), IDBA (Peng *et al.*, 2010), ALLPATH (Butler *et al.*, 2008), Velvet (Zerbino and Birney, 2008) and SOAPdenovo (Li *et al.*, 2010)] and overlap graphs [e.g. SGA (Simpson and Durbin, 2012) and PEGASUS (Haider, 2012)]. However, they cannot be directly applied to metagenome assembly for the following reasons. First, isolate genome assemblers typically assume a uniform coverage depth across a genome. This assumption is used for identifying repeat regions in a genome and estimating the size of a genome. In metagenome assembly, however, genomes may have vastly different coverage depths depending on their relative abundances in a community. Second, isolate genome assembly only needs to resolve repeat regions within a single genome, while metagenome assembly also has to handle repeat regions between multiple genomes. Third, sequencing errors significantly convolute the assembly by introducing false overlaps between reads and disrupting true overlaps. Error correction can be performed for isolate genome assembly using consensus sequences. However, it is difficult to separate sequencing errors from single nucleotide polymorphisms (SNPs) in metagenome assembly. To address these challenges, some of the de Bruijn graph assemblers have been upgraded for Illumina metagenomic sequencing data, including MetaVelvet (Namiki *et al.*, 2012) and IDBA-UD (Peng *et al.*, 2012).

In this study, the Omega (overlap-graph metagenome assembler) algorithm was developed specifically for metagenome assembly. Omega followed the general overlap graph (string graph) approach described in BOA (Myers, 2005) and PEGASUS (Haider, 2012). Here, the overlap graph approach was adapted to metagenome assembly by addressing its differences from isolate genome assembly described above. The assembly performance of Omega was compared with SOAPdenovo, IDBA-UD and MetaVelvet on Illumina HiSeq 100-bp data and MiSeq 300-bp data. SOAPdenovo was selected because it was used for metagenome assembly in the human microbiome project (Pop, 2011) and many Joint Genome Institute studies. IDBA-UD and MetaVelvet were designed specifically for metagenome assembly. A widely used overlap-layout-consensus assembler, Celera (Myers *et al.*, 2000), was also compared using the MiSeq 300-bp data.

2 SYSTEM AND METHODS

The performance of assemblers on Illumina HiSeq 100-bp data was benchmarked using a real-world sequencing dataset of genomic DNA mixture of 64 diverse bacterial and archaeal microorganisms (Shakya *et al.*, 2013). The dataset is available at National Center for Biotechnology Information (NCBI) Sequence Read Archive (SRA) (accession number: SRX200676). The 64 microorganisms are listed in Supplementary Table S1. Fastq sequences were extracted from the SRA format raw dataset using NCBI SRA Toolkit (version

*To whom correspondence should be addressed.

2.3.4). This dataset contains 108.7 million paired-end and 0.4 million single-end 100-bp reads. Sickle (https://github.com/najoshi/sickle) was used to trim reads using a 20-Phred quality threshold, to filter out reads shorter than 60 bp and to discard reads containing many Ns. BBNorm (https://sourceforge.net/projects/bbmap/) was then used for error correction with default settings.

The HiSeq 100-bp dataset was assembled using SOAPdenovo, IDBA-UD, MetaVelvet and Omega. The *k*-mer length or minimum overlap length was optimized for each assembler based on the N50 size: SOAPdenovo (best *k*-mer length = 51 of 31, 41, 51 and 61), IDBA-UD (*k*-mer length range = 30–60 with a step size of 10), MetaVelvet (best *k*-mer length = 51 of 31, 41, 51 and 61) and Omega (best minimum overlap length = 50 of 30, 40, 50, 60 and 70). SOAPdenovo was run in a metagenome configuration as described (Pop, 2011). IDBA-UD, MetaVelvet and Omega were run with default parameters.

The performance of assemblers on Illumina MiSeq 300-bp data was tested using a simulated dataset of a nine-genome synthetic community. Ten million paired-end 300-bp reads with an average insert size of 900 bp were simulated based on an empirical error model using MetaSim (Richter *et al.*, 2008). Supplementary Table S2 lists the nine genomes and their relative abundances ranging from 3 to 20%. The simulated reads were preprocessed using Sickle and error-corrected using BBNorm as described above.

The maximum *k*-mer length of MetaVelvet was increased to 171 by changing a constant parameter in its source code. We were unable to increase the maximum *k*-mer lengths of SOAPdenovo and IDBA-UD, which were hard-coded at 127 and 124, respectively. The *k*-mer length or minimum overlap length was optimized for each assembler: SOAPdenovo (best *k*-mer length = 121 of 41, 61, 81, 101, 121 and 127), IDBA-UD (*k*-mer length range = 40–120 with a step size of 20), MetaVelvet (best *k*-mer length = 151 of 121, 131, 141, 151 and 161) and Omega (best minimum overlap length = 150 of 120, 130, 140, 150 and 160). The assemblers were run as described above. Celera was also tested for the MiSeq 300-bp dataset using a default setting and a metagenomics setting (http://sourceforge.net/apps/mediawiki/wgs-assembler/index.php?title=RunCA_Examples_-_454_%2B_Sanger_Metagenomic). The Celera assembly using the default setting was substantially better than that using the metagenomics setting and, therefore, was used for performance comparison.

To measure assembly accuracy, contigs and scaffolds >200 bp produced by each assembler were aligned with reference genomes using Burrow–Wheeler Aligner (BWA) (Li and Durbin, 2010). The alignments were used to generate a list of correct contigs containing <5% of substitutions and indels. The scaffolding of two adjacent contigs was considered to be correct if their alignments to the same reference genome were in correct orientation and separated apart by less than the mean plus one standard deviation (SD) of the mate inner distances of the paired-end sequencing data. The performance of assemblers was compared by N80, N50, N20, largest contig length and genome sequence coverage for each reference genome. N80, N50 and N20 are the minimum size thresholds for length-sorted contigs that covers 80, 50 and 20% of the total length of all contigs, respectively. Genome sequence coverage is the percentage of a reference genome sequence covered by the assembled contigs. N80, N50, N20 and largest contig length measure the contiguity of the correct contigs at different levels. Genome sequence coverage measures the completeness of the correct contigs. Different types of assembly errors were identified based on the BWA alignment between the contigs and the reference genomes, including the number of base pairs of insertion, deletion and substitution and the number of chimeric contigs. Chimeric contigs were identified by their fragmented alignments to multiple non-contiguous regions of a reference genome or multiple reference genomes.

3 ALGORITHM

Omega was developed in C++ using object-oriented programming. Omega can accept multiple input datasets with different insert sizes and variable read lengths in fasta or fastq format. The assembly and scaffolding were performed in eight steps (Fig. 1):

(1) **Hash table construction.** All unique reads are loaded to the memory and indexed in a hash table. Let K be the user-defined minimum overlap length. The keys of the hash table are DNA sequence substrings of length $K-1$. Each read is inserted to the hash table with four keys: prefix and suffix of length $K-1$ of both forward sequence and reverse complement sequence of the read. A value in the hash table is an array of pointers to the reads associated with the corresponding key. The hash table is initialized to be eight times of the total read number. Hash collision is resolved using linear probing. The hash table allows a nearly constant time search of all reads by their prefixes or suffixes. A read that is a substring of another read is called a *contained read*. To identify all contained reads of read r, every proper substring s of length K in read r is searched in the hash table. This produces a short list of reads that contains s as a prefix or suffix, which is then compared with read r to identify the contained reads of

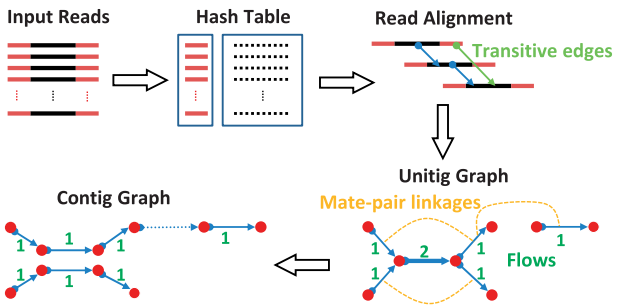

Fig. 1. Overview of Omega. The prefix and suffix (red sections) of every read are indexed in a hash table. As reads are aligned using the hash table, transitive edges (green arrows) are removed. In the unitig graph, edges (blue arrows) represent unambiguous series of overlapping reads, vertices (red dots) represent branching points and flows (green numbers) estimate the copy numbers of strings in the edges. The mate-pair linkages (orange dotted lines) are used to build contigs and then scaffolds containing gaps (blue dotted arrows). The repeat region between two different genomes (the edge with 2 units of flow) may be resolved using mate-pair supports (as shown here) or coverage depth information

read r. The contained reads are used for coverage depth calculation and mate-pair linkage analysis below.

(2) **Overlap graph construction.** Each read is represented by a vertex in a bi-directed overlap graph. An edge is inserted between two vertices if the two corresponding reads have an exact-match overlap of at least K bp. The bi-directed edges represent the four different orientations in which two reads can overlap: suffix with prefix (•→——→•), suffix of the reverse complement with prefix (•←——→•), suffix with prefix of the reverse complement (•→—←•) and suffix of the reverse complement with prefix of the reverse complement (•←——←•). To efficiently find all reads overlapping with a read r, every proper substring s of length $K-1$ in read r is searched in the hash table, and all retrieved reads are compared with the read r. If a read has the exact match with read r for their remaining overlap, an edge is inserted between the two reads' corresponding vertices. After inserting all edges of read r, all transitive edges incident on read r are removed using a linear algorithm as described (Haider, 2012; Myers, 2005). Briefly, suppose that r is connected with two other reads, a and b. If there is also an edge between a and b to form a triangle with r and the sequence represented by the edge (r, b) is the same as the sequence represented by the path through (r, a) and (a, b), then (r, b) is identified as a transitive edge and is deleted. Removing all transitive edges significantly simplifies the overlap graph without losing any information.

(3) **Composite edge contraction.** While the bi-directed edges can be traversed in both directions, the vertices can be traversed only by entering a vertex in an in-arrow and exiting in an out-arrow (—→•→–) or by entering a vertex in an out-arrow and exiting in an in-arrow (–←•←—). A valid path in the overlap graph represents an assembled DNA sequence containing proper overlapping reads with appropriate orientation and sufficient overlap length. After removing transitive edges, simple vertices have exactly one in-arrow and one out-arrow, representing only one possible way to traverse such simple vertices. A read in a simple vertex uniquely overlaps with one other read in either direction. To simplify the overlap graph, a simple vertex, r, along with its in-arrow edge (u, r) and out-arrow edge (r, w), are replaced by a composite edge (u, w) in the overlap graph. The composite edge (u, w) contains the read r and all ordered reads in edge (u, r) and (r, w). The edge (u, w) has the same arrow types to u and w as the original edges, (u, r) and (r, w), respectively. Simple vertices are merged into composite edges iteratively, until there is no simple vertex remaining in the overlap graph.

(4) **Sequence variation removal.** Sequence variations originate from uncorrected sequencing errors and natural sequence polymorphisms in microbial communities. Many reads with sequence variations do not overlap with any other reads and are represented as isolated vertices in the overlap graph. Reads with the same sequence variation may overlap with one another, which creates small branches and bubbles in the overlap graph. Small branches are short dead-end paths that contain <10 reads. Bubbles are two edges that connect the same two vertices with the same arrow types. The overlap graph is systematically traversed to trim off small branches and remove the edges containing less reads in bubbles. This may create new simple vertices that are then removed by repeating the composite edge contraction.

(5) **Minimum cost flow analysis.** Each edge in the overlap graph is associated with a string copy number, representing how many times the edge's sequence is present in the metagenome. String copy numbers of edges are estimated based on the topology of the overlap graph using minimum cost flow analysis as described (Haider, 2012; Myers, 2005). Composite edges with sequences >1000 bp are set to have a minimum flow of 1, requiring such edges' sequences to be present in the metagenome at least once. The minimum flow for short edges (<1000 bp) is set to 0. The CS2 algorithm (Goldberg, 1997) is used to optimize the amount of flow passing through every edge such that the total cost of the flow network in the overlap graph is minimized. Edges with more than one unit of flow correspond to repeat regions shared among multiple genomes or multiple places in a single genome. Edges with zero flow represent short sequences that are not needed to connect long sequences together and are ignored. Tree structures in the overlap graph are simplified using the flows. A tree comprises two edges, (p, t) and (q, t), merging to a third edge (t, r), and the flow on (t, r) is equal to the total flow on (p, t) and (q, t). Such a tree is reduced to two new edges (p, r) and (q, r) that both contain the reads in vertex t and edge (t, r).

(6) **Merging of adjacent edges with mate-pair support.** The insert size of each paired-end dataset is estimated separately to accommodate a mixture of datasets with different insert sizes. The overlap graph at this stage has long composite edges that contain both reads of many mate-pairs. The insert sizes of such pairs are determined from their relative locations on the long edges and are pooled to estimate the mean μ and SD σ of all mate-pairs' insert sizes in each dataset. Mate-pairs that span multiple edges are used to merge adjacent edges in the overlap graph. For each of such mate-pairs, all possible paths of length within range $(\mu-3\sigma, \mu+3\sigma)$ are enumerated. If all paths of a mate-pair travel through two adjacent edges, (m, r) and (r, n), the connection between these two edges is considered to be supported by this mate-pair. After processing all mate-pairs, if the connection between (m, r) and (r, n) is supported by more than three mate-pairs, these two edges are merged to one edge (m, n) containing a duplicated r.

(7) **Scaffolding of long edges with mate-pair support.** Scaffolding uses mate-pairs that have no valid path between their paired reads in the overlap graph because of a gap in genome coverage. Scaffolding is attempted for every pair of non-adjacent edges >1000 bp. A mate-pair is considered to support the scaffolding of two edges if its two reads are uniquely mapped to the two edges at an

appropriate distance apart. After processing all mate-pairs, the scaffolds of long edges with support of more than three mate-pairs are accepted.

(8) **Resolving ambiguity by coverage depth.** Many unresolved vertices in the overlap graph have two incoming edges and two outgoing edges, which often originate from a short repeat region between two different genomes. The two genomes may have different coverage depths to separate their edges. The coverage depth is calculated for every position along an edge to estimate the mean δ and SD θ of coverage depth along the edge. Only unique reads in an edge are considered for coverage depth calculation. A pair of adjacent edges on an unresolved vertex are merged if $|\delta_1 - \delta_2| < \theta_1 + \theta_2$.

Finally, Omega reports contigs and scaffolds based on the edges of the overlap graph.

4 RESULTS AND DISCUSSION

After trimming and filtering, the HiSeq 100-bp dataset contained 101 million paired-end reads. To find the error rate, reads were aligned to the concatenated 64 reference genomes using Bowtie2 (Langmead and Salzberg, 2012) allowing up to three mismatches per read. We defined sequencing errors as mismatches supported by less than three reads to exclude consistent substitutions attributable to SNPs. Before error correction, 93.8 million reads were aligned to at least one reference genome, and an average of 0.12 sequencing error per read was found. After error correction with BBNorm, 97.5 million reads were aligned with 0.02 sequencing error per read.

The error-corrected HiSeq 100-bp dataset was assembled using SOAPdenovo, IDBA-UD, MetaVelvet and Omega. The CPU usage and peak memory usage were, respectively, 13 h and 29 GB for SOAPdenovo, 49 h and 112 GB for IDBA-UD, 8 h and 21 GB for MetaVelvet and 15 h and 105 GB for Omega.

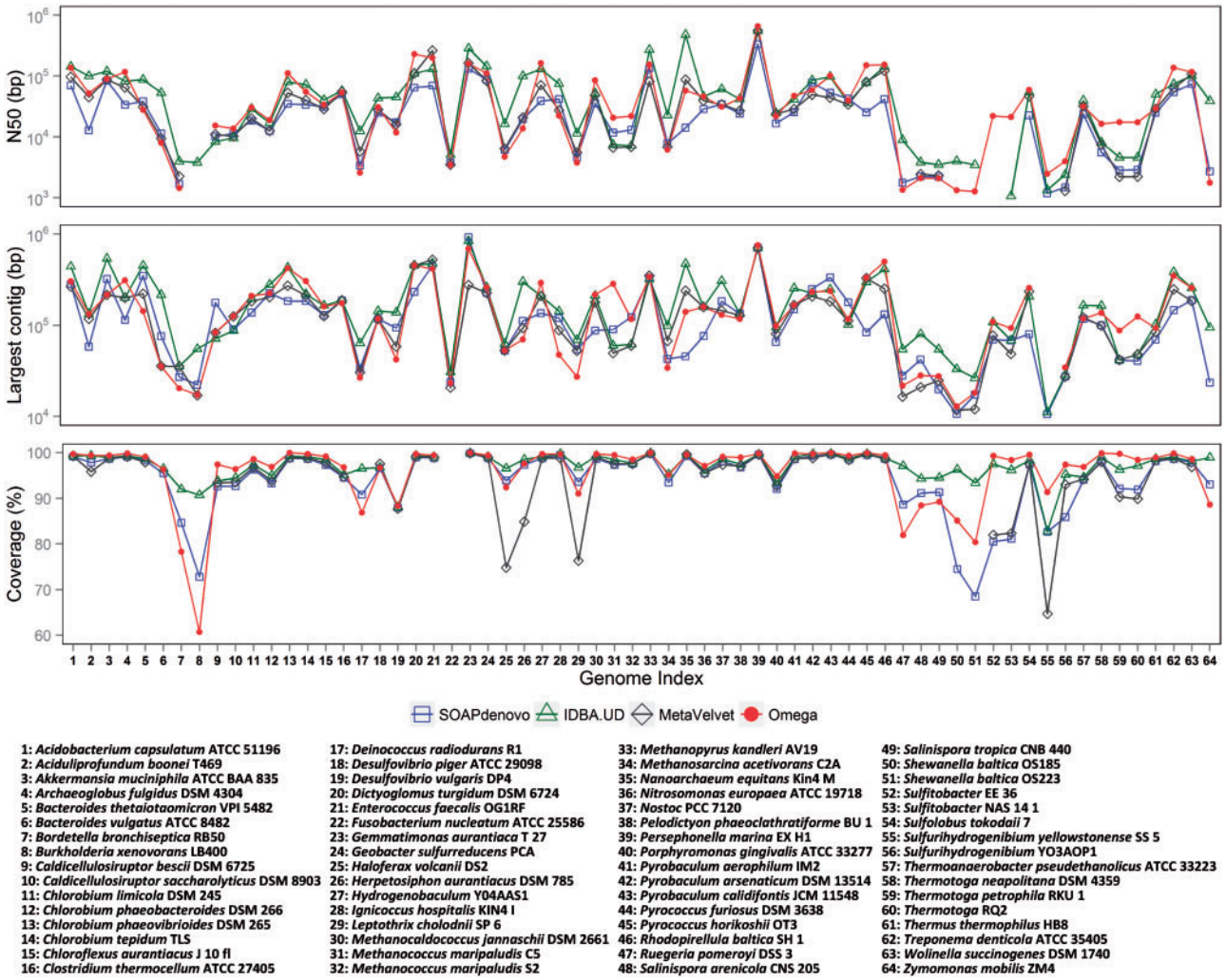

Fig. 2. Scaffold assembly statistics for individual genomes from the HiSeq 100-bp dataset. The *x*-axis lists the 64 genomes in alphabetic order, and the indices and organism names of the genomes are shown in the legend. The assemblies were provided by SOAPdenovo (blue squares), IDBA-UD (green triangles), MetaVelvet (black diamonds) and Omega (solid red circles). Outliers below the minimum threshold of a performance statistics are not shown

SOAPdenovo, MetaVelvet and Omega were efficient in CPU usage, but SOAPdenovo and MetaVelvet used much less memory. Omega spent 1.5 h building the hash table from reads, 2.7 h identifying contained reads, 7.1 h constructing the overlap graph and 3.5 h simplifying the overlap graph. The peak memory usage of Omega was at the end of overlap graph construction when Omega stored all reads (~50 GB), the hash table (~5 GB) and the completed overlap graph (~50 GB) in memory.

The assembly results were verified by aligning contigs and scaffolds with the reference genomes. The four assemblers all produced some misassembled contigs. The common causes for misassemblies included homologous repeat regions among the 64 genomes, undersampled regions of the genomes and remaining sequencing errors. For each reference genome, we generated standard assembly statistics of correct scaffolds, including N80, N50, N20, largest contig length and genome sequence coverage (Fig. 2 and Supplementary Table S1). On average across all genomes, more contiguous assemblies were provided by IDBA_UD, Omega, MetaVelvet and SOAPdenovo in this order (Table 1). However, the four assemblers performed similarly for many genomes, and IDBA-UD and Omega provided clearly improved assembly results for different subsets of genomes (Fig. 2). The assembly of *Fusobacterium nucleatum* (genome 22 in Fig. 2) was poor because of its low abundance in the mock community.

The four assemblers, as well as Celera, were then compared using a simulated MiSeq 300-bp dataset. The aggregate raw and verified assembly statistics of contigs and scaffolds are shown in Table 2. The assembly results for individual genomes are listed in Supplementary Table S2. Celera used much more CPU hours than the other four assemblers (>20 times more than Omega; Table 3). The assembly from Omega was more contiguous than the assemblies from the three de Bruijn graph assemblers and Celera. The assemblers had different error profiles (Table 3). For example, Omega generated more chimeric contigs and substitution errors than Celera, but less insertion and deletion errors. MetaVelvet had less insertion, deletion and substitution errors than Omega and Celera, but more chimeric contigs. Generally, it was difficult to generate more contiguous assembly while minimizing all types of errors.

Here, the performance of assemblers was benchmarked using a real-world HiSeq 100-bp dataset and a simulated MiSeq 300-bp dataset, both of which had reference genomes for result verification. However, computationally simulated sequencing data cannot reproduce many complications of Illumina sequencing. Even real-world sequencing data of artificially mixed genomic DNAs cannot replicate the true complexity of natural microbial

Table 1. Average genome assembly statistics across all genomes in the HiSeq 100-bp dataset

Assembler	N80 (10^3bp)	N50 (10^3bp)	N20 (10^3bp)	Largest Contig (10^3bp)	Coverage (%)
SOAPdenovo	11	33	73	144	92.81
MetaVelvet	17	46	92	147	82.10
Omega	25	61	111	174	94.50
IDBA_UD	35	70	136	203	95.65

Table 2. Comparison of overall assembly statistics on the MiSeq 300-bp dataset*

Assembly	Statistics	Assembler	Total contigs	N50 contigs	N80 (10^3bp)	N50 (10^3bp)	N20 (10^3bp)	Largest contig (10^3bp)	Sum (10^6bp)	Coverage (%)
Contigs	Raw	SOAPdenovo	7815	361	8	20	51	358	29	–
		IDBA_UD	1683	72	38	102	217	965	29	–
		MetaVelvet	1097	52	48	136	340	1389	29	–
		Celera	**435**	48	53	151	483	1406	29	–
		Omega	537	**40**	**64**	**159**	**490**	**2572**	29	–
	Verified	SOAPdenovo	7817	361	8	20	51	358	29	97.95
		IDBA_UD	1775	79	36	95	199	547	29	97.73
		MetaVelvet	1104	54	46	135	313	1389	29	98.06
		Celera	**448**	48	52	151	483	1406	29	**99.16**
		Omega	578	**43**	**55**	**158**	**486**	**2091**	29	99.05
Scaffolds	Raw	SOAPdenovo	5586	50	45	138	379	1404	30	–
		IDBA_UD	1570	63	40	116	302	965	29	–
		MetaVelvet	996	44	51	156	489	1389	29	–
		Celera	**429**	48	53	151	483	1406	29	–
		Omega	434	**36**	**67**	**188**	**556**	**2572**	29	–
	Verified	SOAPdenovo	6926	160	11	38	149	593	29	95.11
		IDBA_UD	1733	75	37	100	213	547	29	97.84
		MetaVelvet	1065	50	48	138	479	1389	29	98.03
		Celera	**450**	48	52	151	483	1406	29	**99.16**
		Omega	562	**42**	**55**	**160**	**486**	**2091**	29	99.02

*Best assembly statistics in each category is highlighted in bold.

Table 3. Comparison of error profiles and computational costs of the MiSeq 300-bp dataset assembly

Assembler	Verified N50 (10^3bp)	Verified N50 / Raw N50	Small Insertion < 5bp (bp)	Large Insertion ≥ 5bp (bp)	Small Deletion < 5bp (bp)	Large Deletion ≥ 5bp (bp)	Substitution (bp)	Chimeric Contigs	CPU (hours)	Memory (GB)
SOAPdenovo	20	100%	0	0	0	0	2485	2	3	19
IDBA_UD	95	93.1%	9	502	35	1443	732	101	8	24
MetaVelvet	135	99.3%	10	0	2	0	298	65	1	8
Celera	151	100%	11	220	5	122	1384	19	67	19
Omega	158	99.4%	13	35	23	28	1958	43	3	22

communities with high strain-level variations and large abundance differences. As there was no complex natural community composed of microorganisms with known genome sequences, it was still a challenge to accurately benchmark the real-world performance of metagenome assemblers.

The overall performance of Omega was comparable with SOAPdenovo, IDBA_UD and MetaVelvet on the HiSeq 100-bp dataset and superior on the 300-bp MiSeq dataset, although each assembler provided the best assembly for some individual genomes in the two synthetic communities. Our benchmarking indicated the unique advantages of the five assemblers: Omega was generally more suitable for datasets with longer reads and higher coverage depth using larger overlaps; SOAPdenovo and MetaVelvet were efficient in memory and CPU usage, which is critical for large datasets; IDBA-UD automatically iterated through a *k*-mer range and provided better assembly for more genomes in the HiSeq 100-bp dataset; and Celera performed well for long reads but was computationally very expensive for Illumina datasets. It is important for users to select an assembler based on test assembly results from their actual metagenome datasets. It may also be advantageous to use a dedicated metagenome scaffolding algorithm, such as Bambus 2 (Koren *et al*., 2011), or to combine multiple assembly tools using the metAMOS pipeline (Treangen *et al*., 2013).

In conclusion, our results indicated the effectiveness of the overlap graph approach for metagenome assembly. We believe the overlap graph approach will become more useful for future Illumina technologies with longer reads and higher throughput.

Funding: This work was supported by Laboratory Directed Research and Development (LDRD) funding from Oak Ridge National Laboratory and the Emerging Technologies Opportunity Program (ETOP) from DOE Joint Genome Institute. The contribution of J.C. was sponsored by the Office of Advanced Scientific Computing Research. Oak Ridge National Laboratory and DOE Joint Genome Institute are supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 and DE-AC02-05CH11231, respectively.

Conflict of interest: none declared.

REFERENCES

Butler,J. *et al*. (2008) ALLPATHS: *de novo* assembly of whole-genome shotgun microreads. *Genome Res*., **18**, 810–820.

Goldberg,A.V. (1997) An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, **22**, 1–29.

Haider,B. (2012) *A New Algorithm for De Novo Genome Assembly*. Department Computer Science, The University of Western Ontario.

Koren,S. *et al*. (2011) Bambus 2: scaffolding metagenomes. *Bioinformatics*, **27**, 2964–2971.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Li,H. and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, **26**, 589–595.

Li,R. *et al*. (2010) *De novo* assembly of human genomes with massively parallel short read sequencing. *Genome Res*, **20**, 265–272.

Myers,E.W. *et al*. (2000) A whole-genome assembly of Drosophila. *Science*, **287**, 2196–2204.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinformatics*, **21** (**Suppl. 2**), ii79–ii85.

Namiki,T. *et al*. (2012) MetaVelvet: an extension of Velvet assembler to *de novo* metagenome assembly from short sequence reads. *Nucleic Acids Res*., **40**, e155.

Peng,Y. *et al*. (2010) IDBA—A Practical Iterative de Bruijn Graph *De Novo* Assembler. In: Berger,B. (ed.) *Research in Computational Molecular Biology*. Berlin Heidelberg, Springer, pp. 426–440.

Peng,Y. *et al*. (2012) IDBA-UD: a *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, **28**, 1420–1428.

Pop,M. (2011) HMP Whole Metagenome Assembly.

Richter,D.C. *et al*. (2008) MetaSim: a sequencing simulator for genomics and metagenomics. *PloS One*, **3**, e3373.

Shakya,M. *et al*. (2013) Comparative metagenomic and rRNA microbial diversity characterization using archaeal and bacterial synthetic communities. *Environ. Microbiol*., **15**, 1882–1899.

Simpson,J.T. and Durbin,R. (2012) Efficient *de novo* assembly of large genomes using compressed data structures. *Genome Res*., **22**, 549–556.

Simpson,J.T. *et al*. (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res*., **19**, 1117–1123.

Treangen,T.J. *et al*. (2013) MetAMOS: a modular and open source metagenomic assembly and analysis pipeline. *Genome Biol*., **14**, R2.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Res*., **18**, 821–829.