Bioinformatics, 32(9), 2016, 1301–1307
doi: 10.1093/bioinformatics/btw011
Advance Access Publication Date: 10 January 2016
Original Paper

OXFORD

Genome analysis

Integration of string and de Bruijn graphs for genome assembly

Yao-Ting Huang* and Chen-Fu Liao

Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan

*To whom correspondence should be addressed.

Associate Editor: Gunnar Ratsch

Received on 7 April 2016; revised on 23 December 2015; accepted on 7 January 2016

Abstract

Motivation: String and de Bruijn graphs are two graph models used by most genome assemblers. At present, none of the existing assemblers clearly outperforms the others across all datasets. We found that although a string graph can make use of entire reads for resolving repeats, de Bruijn graphs can naturally assemble through regions that are error-prone due to sequencing bias.
Results: We developed a novel assembler called StriDe that has advantages of both string and de Bruijn graphs. First, the reads are decomposed adaptively only in error-prone regions. Second, each paired-end read is extended into a long read directly using an FM-index. The decomposed and extended reads are used to build an assembly graph. In addition, several essential components of an assembler were designed or improved. The resulting assembler was fully parallelized, tested and compared with state-of-the-art assemblers using benchmark datasets. The results indicate that contiguity of StriDe is comparable with top assemblers on both short-read and long-read datasets, and the assembly accuracy is high in comparison with the others.
Availability and implementation: https://github.com/ythuang0522/StriDe
Contact: ythuang@cs.ccu.edu.tw
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Although next- and third-generation sequencing technologies have been widely used to sequence and assemble genomes of many species in the biosphere (Haussler *et al.*, 2009; Metzker, 2010), many assembled genomes are still fragmented due to complex repetitive structure (Phillippy *et al.*, 2008). A fragmented assembled genome often introduces extra complexity into downstream processing, e.g. estimation of gene family size and comparative analysis. Recently, several evaluation projects (e.g. GAGE, GAGE-b and Assemblathon 1/2) have been conducted to assess the accuracy, contiguity and speed of state-of-the-art assemblers (Bradnam *et al.*, 2013; Earl *et al.*, 2011; Magoc *et al.*, 2013; Salzberg *et al.*, 2012). None of the existing assemblers clearly outperforms all the others across all benchmarks.

The overlap-layout-consensus (OLC) and de Bruijn graphs are two models used by most assemblers. These two models represent the overlapping relation between reads in different ways. The OLC-based assemblers, including MIRA, Newbler and Celera assemblers, first identify all pairs of overlapping reads and construct a graph with vertices representing reads and with edges denoting two overlapping reads (Miller *et al.*, 2008). Next, the genome sequence is assembled by figuring out a feasible layout of reads from the graph. The OLC-based methods make good use of the entire read length for resolving repeats and chimeras; this approach is beneficial for sequencing platforms generating long reads (e.g. Sanger sequencing, Roche 454 and Pacific Biosciences). These assemblers, however, are computationally inefficient at assembling a massive amount of short reads because of the time-consuming overlap computation.

Nowadays, de Bruijn graph assemblers, including Velvet, SOAPdenovo, ABySS and ALLPATHS, are the preferred choice for most sequencing projects (Butler *et al.*, 2008; Luo *et al.*, 2012; Simpson *et al.*, 2009; Zerbino and Birney, 2008). These assemblers break each read into fixed-size *k*-mers, which do not require the overlap computation, and a graph is directly constructed where each vertex is a *k*-mer and each edge indicates two adjacent *k*-mers

overlapping by $k-1$ letters. The construction of a de Bruijn graph is much more efficient. Nonetheless, the graph structure is more complex owing to repeats larger than k-mer. Recently, a paired de Bruijn graph assembler (SPAdes) overcame this limitation by building a graph by means of paired k-mers from paired-end reads (Bankevich *et al.*, 2012).

Aside from these two graph models, there is a variant (called string graph) that is similar to the OLC graph without transitive edges (Myers, 2005). The construction of a string graph from reads can be computed in linear time using an FM-index (Ferragina and Manzini, 2000; Simpson and Durbin, 2010). The first such assembler, called the String Graph Assembler (SGA), is capable of assembling mammalian-size genomes, but its contiguity is not better than that of de Bruijn graph assemblers according to several benchmark tests. The string graph shares many properties with the OLC and de Bruijn graphs, but their equivalence in terms of real sequencing data remains a subject of debate (Simpson and Durbin, 2012). We found that the major difference lies in the ability to assemble through regions that are error-prone due to sequencing bias (e.g. high-GC regions; see Supplementary Fig. S1; Schirmer *et al.*, 2015). This article presents a novel assembler called StriDe, which adaptively decomposes reads within error-prone regions and extends paired-end reads into long reads using an FM-index. In addition, an improved error correction algorithm, overlap computation, specialized layout algorithms and full parallelization were implemented to make the assembler more practical.

2 Method

We would like to emphasize two major features of this assembler before presenting the details (see Fig. 1). First, the original reads are adaptively decomposed into overlapping subreads within error-prone regions, in which the correct subreads (collected from different reads) can still be assembled (i.e. just as in a de Bruijn graph). Second, the paired-end reads are extended into long reads using an FM-index, which can resolve repeats longer than read length. In summary, the input reads are converted into short (overlapping) subreads, original reads, or extended long reads (Fig. 2). Therefore, a string graph that is constructed from these variable-length reads implicitly integrates de Bruijn and string graphs for assembly through error-prone and repetitive regions. Efficient construction of this assembly graph requires an FM-index built from variable-length reads; this problem was recently solved by Li's ropebwt2 algorithm (Li, 2014). The StriDe assembler uses Li's ropebwt2 for FM-index construction (see Supplementary Methods, Section 1.1). Below, we present the main ideas behind the major components of this assembler (see Supplementary Fig. S2). The other components and details can be found in Supplementary Methods.

2.1 Correction of errors by frequency turbulence

Conventional methods for identification of errors usually select a k-mer frequency cutoff depending on the underlying k-mer frequency spectrum, which is largely affected by repeats and the sequencing bias (see Supplementary Fig. 3). We found that the frequency differences between two adjacent k-mers are much smaller and are stable regardless of repetitive and low-coverage regions (e.g. for >98% of the regions, the frequency difference is less than 10). On the other hand, the large frequency differences are mainly due to sequencing errors (see Fig. 3(a)). Suppose an error occurs at the $(i+k)$th position. The frequency difference between the ith and $i+1$th k-mer drops significantly because the rightmost base of the $(i+1)$th k-mer represents the error base. In addition, the frequency difference between the $(i+k)$- and $(i+k+1)$th k-mer increases significantly because the leftmost base of the $(i+k+1)$th k-mer just left the error base. As a consequence, the error base is identified if any two adjacent k-mer frequencies differ by more than t (default: 10), which aims to match the expected error rate of Illumina platforms (see Supplementary Method, Section 1.2).

After identifying the exact position of the error, we first try to replace the error with alternative alleles A, T, C, G and see whether the frequency turbulence is eliminated. On the other hand, indel errors and clustered errors cannot be corrected this way. In this case, an overlap correction algorithm is performed via alignment of overlapping reads using a conventional seed-and-extend approach (Fig. 3(b)). The seeds are selected from regions flanking the errors,

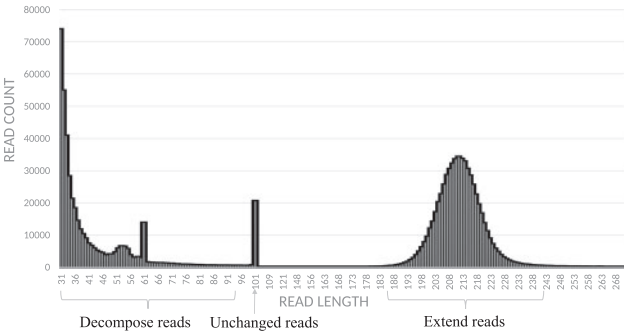

Fig. 2. An example of a read length distribution of the StriDe assembler. The left-hand group is composed of variable-length subreads decomposed from error-prone regions. The right-hand group is composed of variable-length reads extended from paired-end reads, and its size roughly equals the insert size. Only a few unchanged reads are neither decomposed nor extended

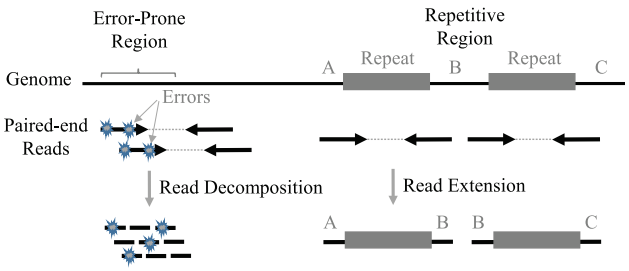

Fig. 1. Illustration of adaptive decomposition and extension of paired-end reads by the StriDe assembler across error-prone and repetitive regions. Note that the decomposed subreads still overlap with each other, and the correct subreads (collected from different reads) can still be assembled

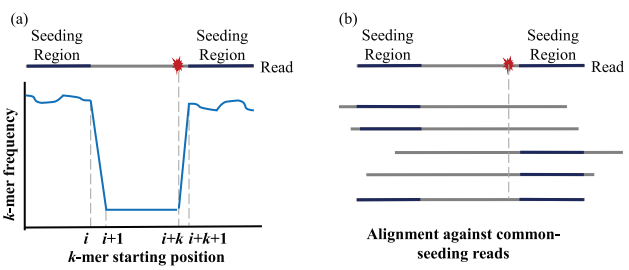

Fig. 3. (**a**) The k-mer frequency drops when the rightmost base reaches the error and increases after leaving the error; (**b**) The k-mer seeds are selected from the regions flanking the low-frequency region and are used for identifying potentially overlapping reads

and banded dynamic programming is used to compute the exact overlap alignment (see Supplementary Method, Section 1.2). Note that in high-GC regions, the high density of sequencing errors still cannot be corrected by either method. The reads within these regions will be decomposed into subreads.

2.2 Extension of Paired-end reads by FM-index walk

The paired-end reads are extended inward into long reads by searching for feasible overlapping sequences between two ends. We consider the FM-index (constructed from all reads) as multiple (implicit) de Bruijn graphs of various k-mers. Given a paired-end read, the algorithm (called the FM-index walk) is aimed at finding feasible {A, T, C, G}-extensions from the first end to the other end by updating the suffix array (SA) intervals (Fig. 4). The major advantage of this approach is that each one-base extension becomes a simple update of an existing SA interval, which can be completed in $O(1)$ time. The details of this algorithm are described below.

The entire search space from first end (source) to second end (destination) is maintained by a search tree, with each tree node storing two SA intervals (for forward/reverse strands) of any feasible extension. Initially (see Fig. 4), the forward/reverse SA intervals of the source k-mer (e.g. ATC) are computed using the backward-search algorithm (Ferragina and Manzini, 2000). Subsequently, we recursively extend the implicit sequences for all possible {A,T,C,G}-extensions by updating the SA intervals of leaves using the backward-search algorithm (see Supplementary Method, Section 1.3), until the arrival at the destination k-mer (e.g. AAC). Note that the destination k-mer is also a suffix of any feasibly extended sequence (e.g. GCAAC). This notion implies that the SA interval of any feasibly extended sequence is included within that of the destination k-mer (e.g. $[772, 796] \subseteq [682, 886]$). Because the SA interval of destination k-mer can be computed in advance, the arrival at the destination for each newly extended sequence can be checked within $O(1)$ time. The search process aborts if the search depth exceeds the maximum insert size, or the number of extensions exceeds the maximum leaf number (default: 32).

Note that each update of the SA interval increases the implicit sequence as well as the implicit overlap between two reads. The implicit sequence and overlap will eventually exceed the maximum read length of the input reads, thus yielding no feasible extensions. Therefore, whenever none of the leaf nodes can be extended, we have to double-check whether the implicit sequence and overlap are too big. This task can be done by refining the SA interval that exactly corresponds to the k-mer suffix extracted from each leaf sequence. If any feasible SA interval is found after the refinement, we continue the extension using the new SA intervals. Otherwise, the entire search process is aborted because no feasible extensions can be found even though we shrunk the implicit sequence/overlap.

The refinement of SA intervals when the implicit sequence/overlap exceeds read length takes extra $O(k)$ time, whereas the remaining extensions take $O(1)$ time. Because the read length is generally much greater than the k-mer size (e.g. 250-bp/300-bp reads are common in Illumina systems), the $O(k)$ refinement cost can be amortized into the majority of $O(1)$ extensions in the entire searching process. Therefore, the overall amortized cost of this algorithm is $O(IL)$, where I is the search depth (corresponding to insert size) and L is the maximum number of leaves allowed. MaSurCa and ABySS implemented similar de Bruijn walks using hashtable and bloom-filter (Supplementary Fig. S4; Simpson *et al.*, 2009; Zimin *et al.*, 2013). The implicit overlap of de Bruijn walks is fixed to the k-mer size. On the other hand, the implicit overlap of FM-index walks is dynamic and can be larger than k, which is able to walk through repeats larger than k.

2.3 Adaptive read decomposition in Error-prone regions

The paired-end reads that failed to be extended into long reads are often prone to errors due to the sequencing bias (e.g. high-GC regions), which is common on the Illumina platforms. We decompose these reads into smaller overlapping subreads at each potential error base. Therefore, the corrected subreads (collected from different reads) can still be assembled at the final graph layout stage (Fig. 5 and Supplementary Fig. S5). The details are described below.

The decomposition algorithm identifies the breakpoints of subreads by checking all pairs of adjacent k-mers of a read (e.g. $k = 9$ in Fig. 5). For each pair of adjacent k-mers, a breakpoint is identified if any of them contain alternative overlaps with other k-mers. Consider the example shown in Figure 5. The only correct k-mer in R3 is also present in R2 and thus has two possible overlapping paths to the left (and to the right). Therefore, this k-mer subread is decomposed from R3 in order to ensure flexibility at the final graph layout stages, where erroneous subreads (in black boxes) are usually dead ends and can be easily detected/removed. The decomposed subreads must retain the overlap $(k - 1)$ bp with adjacent subreads in the original read because the correct layout of subreads is still uncertain at this stage. This algorithm may falsely decompose reads within different repeat copies, because such ambiguous k-mer overlap paths are encountered often due to independent mutations in each repeat copy. Therefore, we set an additional requirement that the k-mer frequencies for any breakpoint must be lower than a threshold (≤ 3 by default) because these error-prone regions are mainly due to the sequencing bias, where the sequencing coverage is relatively lower.

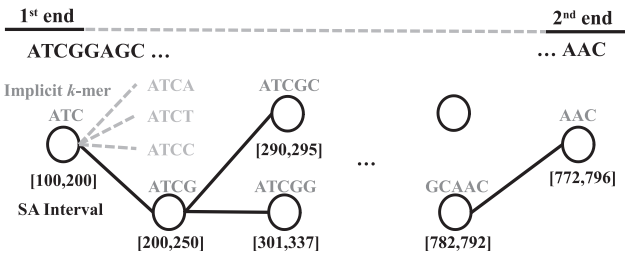

Fig. 4. Illustration of the FM-index walk within a paired-end read. The implicit/initial k-mer size is 3 and shown only for illustration purposes. Each tree node internally stores forward and reverse suffix array (SA) intervals. The solid lines represent successful extensions of the existing SA interval and creation of new leaf nodes, where the dashed lines represent infeasible SA intervals: no new nodes are created

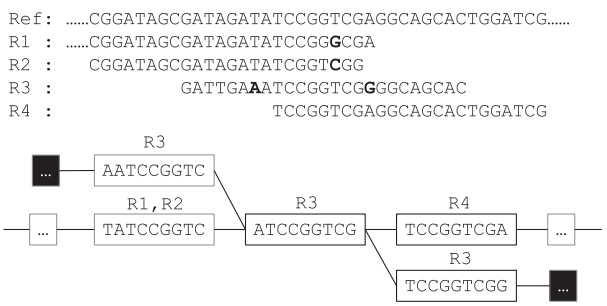

Fig. 5. Illustration of decomposition across four reads (R1–R4). The error is boldfaced, and error subreads are shown in black boxes, which are often tips in the graph. Correct subreads are thus overlapped and can be assembled at a later stage

The worst result of the algorithm is decomposition of each read into $(R_L - k + 1)$ k-mers, where the R_L is read length; this situation is equivalent to a de Bruijn graph. In reality, only the reads within error-prone regions tend to be decomposed into short subreads, whereas the majority of reads still retain the contiguous information of entire sequences.

2.4 Overlap computation with SA-interval pruning

The string graph is composed of vertices and edges, where each vertex stands for a (decomposed or extended) read and each edge represents an overlap between two reads. The overlap computation is the most time-consuming step during graph generation. The exact overlap with respect to any read can be computed in linear time using an FM-index (Simpson and Durbin, 2010). Nonetheless, to assemble a genome in error-prone regions in our algorithm, the error-prone reads are decomposed into shorter subreads with a $(k-1)$-mer overlap. The minimum overlap length has to be reduced to $(k-1)$ in order to maintain the connectivity among the subreads. This small-overlap requirement leads to a huge amount of edges in the graph owing to repeats $\geq (k-1)$-mer; this situation greatly affects the disk/memory usage and reduces the efficiency.

We introduced a pruning procedure into the overlap algorithm to reduce the number of false edges due to decomposed subreads. We found that if a long-extended read (e.g. 200 bp) possesses both a large and small overlap with other reads (e.g. overlaps of 160 and 30 bp), the small overlap can be usually discarded. These small-overlap edges are often transitive (i.e. replaceable by other larger-overlap edges) or form small repeats, which are not informative at the final graph layout stage. Therefore, if the length difference between the largest overlap and any smaller overlap is greater than a threshold L_{gap} (default: half of read length), the smaller overlap is discarded (Fig. 6). In addition, because the sequencing coverage is limited, the influence of small repeats can be controlled by retaining the largest L_{max} overlaps only, where L_{max} equals sequencing coverage. This pruning procedure can effectively skip many repeat edges, while retaining the edges between decomposed subreads. In the implementation, multiple string graphs are concurrently constructed using the overlap-computation algorithm. The race conditions are resolved and described in Supplementary Method (Section 1.5).

2.5 Graph layout algorithms

The layout algorithms are designed to identify unique paths of vertices in the graph for maximization of the assembly contiguity and accuracy. The major challenge at this stage is distinguishing true overlapping edges from chimeric/random edges for vertices with two or more ambiguous edges. Three features were selected from the OLC/string graphs for edge classification: overlap lengths, overlap ratio (overlap length normalized to read length) and overlap ratio differences on the same edge (Supplementary Fig. S7). Each feature requires determination of a reasonable cutoff (e.g. removal of edges with overlap length less than 80 bp). Because the sequencing quality varies a lot across different datasets, these cutoffs have to be adaptively determined. In the text below, we present the major novelties of our layout algorithms, i.e. the simple-path statistic in conjunction with the three features. Other details (e.g. the conventional bubble/tip removal) are described in Supplementary Methods (Section 1.6).

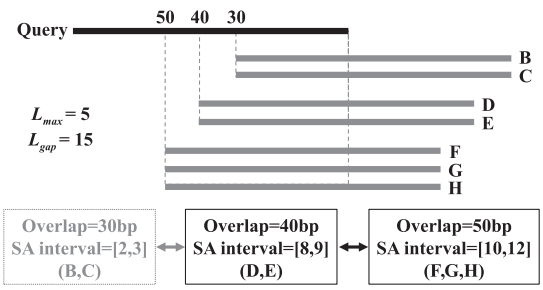

Fig. 6. Collection and pruning of a suffix array (SA) interval list during overlap computation. In this example, only the top five SA indices (set by L_{max}) are retained, which are from the top two SA intervals ([10,12] and [8,9]). That is, only the overlap to reads D, E, F, G and H are retained

2.5.1 Construction of the Simple-path statistic

In most OLC/string graph assemblers, vertices along the same simple paths are contracted to reduce the graph size; this approach throws away valuable information about these features, especially within the initial simple paths. In fact, the assembled sequences on the initial simple paths are usually correct (due to a lack of repeats/chimeras/errors). Therefore, the statistics on these features that are collected from initial simple paths form an empirical distribution for subsequent determination of a cutoff (Supplementary Fig. S7). For instance, if 95% of overlap lengths on the initial simple paths are greater than 80 bp, we can estimate that the removal of edges with an overlap less than 80 bp may yield a 5% error rate. Although any cutoff still faces a tradeoff, we can at least make decisions with statistical confidence using the simple-path statistic. The current implementation is designed to maintain (at least) 95% confidence for each cutoff.

2.5.2 Removal of apparent chimeric vertices

The graph structure surrounding the majority of chimeric vertices is usually complicated owing to small repeats within a chimera; these repeats cannot be easily removed by the conventional tip/bubble removal algorithms (Supplementary Fig. S9). Nevertheless, we noticed three features of chimeras in the graph. First, chimeric vertices have relatively shorter overlap length connecting to their neighbors because the chimeric read is composed of two distant DNA fragments, and the overlap cannot be greater than that of nonchimeric reads. Second, after tip/bubble removal and graph contraction, most vertices on simple paths are merged into larger vertices (i.e. longer sequences). On the other hand, due to the complex graph structure surrounding the chimeric vertices, they usually do not merge with others and remain as small vertices. Third, the k-mer frequency at the chimeric vertices is also relatively lower due to infeasible k-mers across the chimeric junction. Therefore, in conjunction with the simple-path statistic, the small (default: ≤read length) and low-frequency vertices (default: ≤ 3) with short overlap length (default: >95% confidence) to their neighbors are removed (Fig. 7).

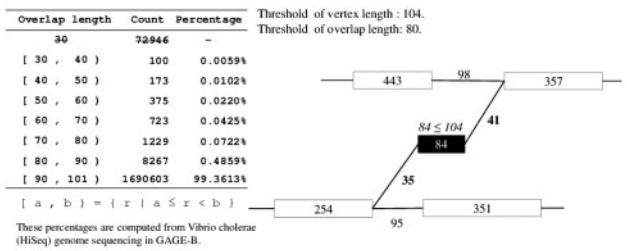

Fig. 7. Simple path statistic of overlap lengths collected from the *V.cholerae* dataset. The small and low-frequency vertices with short overlap are apparent chimeras

The tips/bubble removal algorithms can then be re-applied to further simplify the graph (Supplementary Fig. S8).

2.5.3 Removal of edges with low overlap ratios

Because the input reads were decomposed or extended, the overlap length alone is insufficient for distinguishing true overlapping edges. Instead of removing the entire vertex, we gradually remove less-confident edges using the simple-path statistic of overlap ratios. For the example shown in (Fig. 8), overlap ratios of more than 95% of simple-path edges are greater than 0.8 in the dataset of *A.hydrophila*. Consequently, by requiring the overlap ratio greater than 0.8, we have 95% confidence that the removal of these unreliable edges is safe. After this removal operation, the complexity of the graph structure around the chimeric vertices can be further reduced and may turn into simple tips or bubbles. The algorithms of tip/bubble removal can be applied to simplify the graph.

2.5.4 Removal of edges with large overlap differences

We found that the overlap ratio can still be biased to the size of a vertex because two vertex sizes flanking each edge may differ substantially (e.g. a decomposed subread and an extended long read). In other words, the overlap ratios that were computed using either vertex may be quite different although both of them are on the same edge. Again, during the analysis of the initial simple-path edges, we found that the two overlap ratios on the same edge are quite small, regardless of the size of the vertex. Therefore, a third feature, called overlap-ratio difference, is used to distinguish true overlapping edges (Fig. 9). For instance, the overlap differences in more than 95% of simple-path edges are less than 0.2. Similarly, we can remove more unreliable edges and break down the complex graph structure into tips and bubbles. Finally, the tip/bubble removal algorithms are invoked again to further simplify the graph.

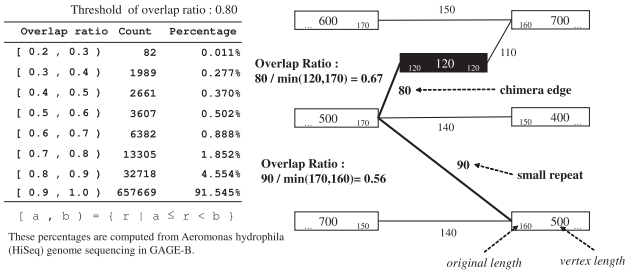

Fig. 8. The simple-path statistic of overlap ratios collected from the *A.hydrophila* dataset in GAGE-b. We can thus remove edges with low overlap ratios and reapply bubble/tip removal algorithms to further simplify the graph

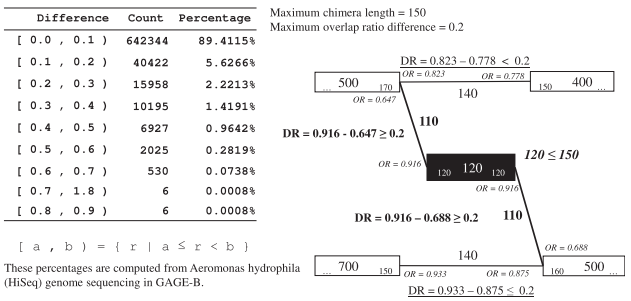

Fig. 9. Simple path statistic of overlap differences collected from the *A.hydrophila* dataset. Chimeric edges have large differences of overlap ratios and tend to be removed

3 Results

The new assembler (StriDe) was tested using GAGE-B benchmark datasets Magoc *et al.* (2013). GAGE-B provides 12 datasets of genome sequencing of different bacteria using Illumina HiSeq and MiSeq platforms. StriDe was compared with seven other well-known assemblers: ABySS (v1.9.0), CABOG (6.0), MaSurCa (v3.1.3), SOAPdenovo (v2.04), SGA (v0.10.13), SPAdes (v3.5.0) and Velvet (v1.2.08). The QUality ASessment Tool (QUAST) was used to compute and compare various assembly metrics (e.g. N50 and misassembly; Gurevich *et al.*, 2013). The parameters k (k-mer size) and T (min. k-mer frequency) used by StriDe are fixed to 31 and 3 across all GAGE-b experiments, respectively.

3.1 Percentages of extended and decomposed reads

The two major features of the StriDe assembler are the decomposition and extension of paired-end reads prior to assembly. Figure 10 shows a comparison of the percentages of extended, decomposed, or unchanged reads in the 12 datasets. The median ratios of extended, decomposed and unchanged reads are 52%, 13% and 25%, respectively. An individual ratio varies a lot depending on the sequencing quality of different datasets. The extended ratio can be more than 70% for high-quality sequencing (e.g. *A.hydrophila* HiSeq). The majority of reads are extended to the length of the expected insert size (i.e. 200–600 bp), and StriDe can take advantage of long reads, just as OLC/string graph assemblers can. On the other hand, the decomposed ratio can be as high as 90% for low-quality sequencing (e.g. *R.sphaeroides* MiSeq). This observation implies that most reads are decomposed into short subreads, and StriDe will act like de Bruijn graph assemblers. In general, the MiSeq datasets are of lower quality and are frequently decomposed in comparison with HiSeq datasets. Note that even in a single dataset, the sequencing quality still varies substantially across the entire genome because of the sequencing bias. The adaptive decomposition or extension of reads by StriDe is robust in the face of such quality variance within or between datasets.

3.2 Assembly results of Short-read datasets

StriDe was compared with the seven assemblers using eight short-read datasets in GAGE-b. Table 1 lists the N50 values of eight short-read datasets (≈ 100 bp) assembled by all the assemblers. The results indicate that the StriDe assembler outperforms most assemblers in terms of contiguity. In general, MaSurCa, SPAdes and StriDe usually output the largest assembly across all datasets. The number of misassemblies, indel rates and other accuracy metrics are

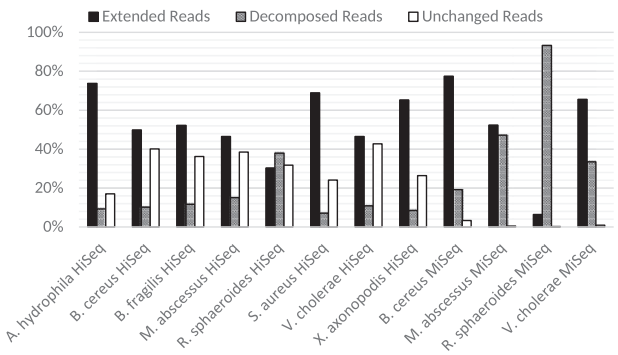

Fig. 10. The percentages of extended, decomposed and unchanged reads across the 12 datasets. The percentages of decomposed reads can be as high as 90% and as low as 10% owing to various sequencing quality

Table 1. Assembly contiguity (N50 in kb) of eight short-read (HiSeq) and four long-read (MiSeq) datasets from GAGE-b

Assembler	Platform	ABySS	CABOG	MaSuRCa	SOAPdenovo	SGA	SPAdes	Velvet	StriDe
Aeromonas hydrophila	HiSeq	237.7	278.4	**838.5**	243.9	67.1	237.6	184.4	**827.8**
Bacillus cereus VD118	HiSeq	41.6	61.1	75.2	57.9	20.5	78.6	38.9	**90.6**
Bacteroides fragilis	HiSeq	116.3	94.2	99.7	116.1	45.0	127.4	125.2	**151.5**
Mycobacterium abscessus	HiSeq	128.5	78.2	147.4	147.2	28.7	**278.4**	60.3	**298.0**
Rhodobacter sphaeroides	HiSeq	115.8	11.2	36.4	10.5	4.8	**173.3**	13.1	**175.1**
Staphylococcus aureus	HiSeq	99.2	102.8	**228.9**	146.3	39.9	148.1	122.5	**222.2**
Vibrio cholerae	HiSeq	172.6	48.8	167.9	106.5	23.8	**344.0**	39.5	**356.0**
Xanthomonas axonopodis	HiSeq	74.1	105.8	**115.7**	74.2	48.9	**117.2**	83.0	**113.0**
Bacillus cereus ATCC	MiSeq	139.1	150.5	270.1	246.3	18.9	**311.1**	24.5	**340.7**
Mycobacterium abscessus	MiSeq	68.5	8.3	18.2	113.3	26.5	**343.8**	41.5	233.6
Rhodobacter sphaeroides	MiSeq	21.4	30.5	**130.6**	33.5	9.2	**132.2**	24.2	**130.5**
Vibrio cholerae	MiSeq	60.3	32.5	46.3	106.5	46.2	**356.1**	67.1	**344.1**

Table 2. The number of misassemblies by each assembler on short-read (HiSeq) or long-read (MiSeq) datasets

Assembler	Platform	ABySS	CABOG	MaSuRCa	SGA	SOAPdenovo	SPAdes	Velvet	StriDe
B.cereus VD118*	HiSeq	1 (140.14)	0 (281.24)	3 (235.23)	0 (97.50)	1 (127.95)	1 (115.88)	0 (108.65)	1 (115.44)
M.abscessus	HiSeq	3 (0.98)	7 (5.81)	7 (3.58)	1 (0.43)	9 (0.67)	5 (0.48)	4 (0.74)	5 (0.43)
R.sphaeroides	HiSeq	9 (5.37)	4 (1.55)	5 (2.13)	1 (0.30)	2 (1.94)	2 (1.19)	2 (0.31)	3 (0.70)
V.cholerae	HiSeq	3 (3.98)	20 (6.84)	16 (6.00)	3 (2.60)	21 (3.67)	7 (3.03)	5 (3.37)	3 (2.85)
B.cereus ATCC	MiSeq	6 (4.60)	4 (2.37)	8 (2.13)	5 (2.60)	2 (2.13)	0 (2.72)	3 (2.64)	1 (2.16)
M.abscessus	MiSeq	2 (0.44)	122 (0.74)	70 (0.47)	7 (0.33)	5 (0.69)	6 (0.40)	72 (0.56)	5 (0.49)
R.sphaeroides	MiSeq	12 (4.71)	6 (0.46)	12 (1.56)	2 (0.40)	1 (0.40)	3 (1.20)	2 (0.62)	3 (0.39)
V.cholerae	MiSeq	2 (2.64)	17 (3.35)	24 (3.47)	3 (2.75)	16 (3.25)	8 (2.83)	14 (2.80)	5 (2.69)

The misassembled-indel rate (per 100 kb) is shown in parentheses. A complete set of metrics is provided in Supplementary Tables S4–S15. Only a few species are available with the reference of the same strain.

*The reference genome of *B.cereus* VD118 is not available and was replaced with that of *B.cereus* ATCC.

listed in Table 2 and Supplementary Tables S4–S15. Note that only a subset of species is available with reference genomes. The results indicated that most assemblers (including StriDe) control misassembly (misassembled contigs) quite well, except for CABOG, MaSurCa and SOAPdenovo, which are slightly worse than the others. In terms of smaller assembly errors (e.g. mismatches/indels), the accuracy of most assemblers is high in general (Supplementary Tables S4–S15). Although ABySS, CABOG, MaSurCa and SOAPdenovo are slightly worse on a few datasets (e.g. Vibrio cholerae HiSeq), the accuracy does not differ much.

3.3 Assembly results of long-read datasets

The GAGE-B also provides four datasets of long reads (≈ 250 bp) from the Illumina MiSeq platform. We compared StriDe with the other assemblers on these datasets. Table 1 lists the N50 values of each bacterial genome assembled by different methods. The results indicate that SPAdes and StriDe outperform the others across the four datasets. The Bacillus cereus dataset is from a longer library of 600 bp, and both SPAdes and StriDe perform quite well on this dataset. This observation meets our expectations because StriDe generates longer reads from paired-end reads, and SPAdes constructs a paired de Bruijn graph from paired-end reads. Both methods make good use of the long library for resolving larger repeats. On the other hand, SGA and CABOG are OLC/string graph assemblers but do not perform better than StriDe over these long reads and perform even worse than de Bruijn assemblers. This result is mainly due to lower sequencing quality of long reads on the MiSeq platform, where a greater number of error-prone reads is generated. Consequently, the read decomposition of StriDe greatly facilitates the assembly of error-prone reads.

3.4 Computation resource

All the experiments were conducted on a Dell R816 server with 48 cores and 256 GB of RAM. The running time (including FM-index construction, error correction and overlap computation) was approximately 20–40 min for completion of the assembly of each GAGE-B dataset. The memory consumption ranged from 1 to 2 GB across 12 datasets, due to compression of the FM-index and assembly graph (see Supplementary Methods, Section 1.1). The extra disk usage is much lower than that of input reads. The constructed FM-indices range from 27 to 237 MB, and graph sizes from 6 to 40 MB.

4 Conclusion

This article presents a novel assembler (StriDe) with advantages from string and de Bruijn graphs. Many essential components of an assembler were developed or improved. An initial test over various Illumina short-/long-read datasets showed that StriDe can assemble through error-prone and repetitive regions using decomposed and extended reads, respectively. The entire implementation is largely inherited from two open-source projects: Li's ropebwt2 and Simpson's SGA. The proposed assembler is still lacking a scaffolding module although third-party scaffolding programs can be incorporated.

Acknowledgements

We thank the reviewers for helping improve the presentation quality of this article. The authors sincerely thank KMC, TJC and HKT for their encouragement along the way.

Funding

This work has been supported by MOST grants 104-2221-E-194-048-MY2 and 103-2923-E-194-001-MY3.

Conflict of Interest: none declared.

References

Bankevich,A. *et al.* (2012) Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, **19**, 455–477.

Bradnam,K. *et al.* (2013) Assemblathon 2 assemblies. *GigaScience Datab.*, **2**, 1–32.

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

Earl,D. *et al.* (2011) Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Res.*, **21**, 2224–2241.

Ferragina, P. and Manzini, G. (2000). Opportunistic Data Structures with Applications. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp. 390–398.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Haussler,D. *et al.* (2009) Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *J. Hered.*, **100**, 659–674.

Li, H. (2014) Fast construction of fm-index for long sequence reads. *Bioinformatics*, **30**, 3274–3275.

Luo,R. *et al.* (2012) Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**, 18.

Magoc,T. *et al.* (2013) Gage-b: an evaluation of genome assemblers for bacterial organisms. *Bioinformatics*, **29**, 1718–1725.

Metzker,M.L. (2010) Sequencing technologies – the next generation. *Nat. Rev. Genet.*, **11**, 31–46.

Miller,J.R. *et al.* (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.

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

Phillippy,A. *et al.* (2008) Genome assembly forensics: finding the elusive misassembly. *Genome Biol.*, **9**, R55.

Salzberg,S. *et al.* (2012) Gage: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.

Schirmer,M. *et al.* (2015) Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. *Nucleic Acids Res.*, **43**, e37.

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

Simpson,J.T. and Durbin,R. (2010) Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, **26**, i367–i373.

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

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

Zimin,A.V. *et al.* (2013) The MaSuRCA genome assembler. *Bioinformatics*, **29**, 2669–2677.