BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 2 2011, pages 167–174
doi:10.1093/bioinformatics/btq626

Sequence analysis

Advance Access publication December 12, 2010

PE-Assembler: *de novo* assembler using short paired-end reads

Pramila Nuwantha Ariyaratne[1] and Wing-Kin Sung[1,2,*]

[1]Computational & Mathematical Biology Group, Genome Institute of Singapore, 138672 and [2]School of Computing, National University of Singapore, Singapore 117543

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Many *de novo* genome assemblers have been proposed recently. The basis for most existing methods relies on the de bruijn graph: a complex graph structure that attempts to encompass the entire genome. Such graphs can be prohibitively large, may fail to capture subtle information and is difficult to be parallelized.

Result: We present a method that eschews the traditional graph-based approach in favor of a simple 3′ extension approach that has potential to be massively parallelized. Our results show that it is able to obtain assemblies that are more contiguous, complete and less error prone compared with existing methods.

Availability: The software package can be found at http://www.comp.nus.edu.sg/~bioinfo/peasm/. Alternatively it is available from authors upon request.

Contact: ksung@comp.nus.edu.sg; sungk@gis.a-star.edu.sg

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

Received on June 4, 2010; revised on October 18, 2010; accepted on October 29, 2010

1 INTRODUCTION

De novo genome assembly has been a fundamental problem in bioinformatics since the advent of DNA sequencing. The second-generation sequencing technologies such as Illumina Solexa and ABI SOLiD have introduced a new sense of vigor to the field. The short length of the sequences coupled with high coverage and high level of noise has transformed *de novo* assembly to a tractable yet challenging proposition. The ease at which paired-end read libraries can be generated on these platforms is an added advantage.

A number of works have been proposed to assemble short reads. The first few *de novo* assemblers developed to handle high-throughput short reads were based on base-by-base 3′ extension. SSAKE, VCAKE and SHARCGS (Dohm *et al*., 2007; Jeck *et al*., 2007; Warren *et al*., 2007) are examples using this principle. To resolve ambiguities, these methods adapted trivial heuristics such as 'selecting the base with maximum overlap' or 'selecting the base with the highest consensus'. Such arbitrary criteria results in substandard assemblies that were often a compromise between contiguity and error rate. Furthermore, the approaches were not scalable to handle medium or large genomes; therefore, their use is restricted to assembling BAC clones or small bacteria genomes. They were also not designed to make use of paired-end reads, thus greatly limiting their usefulness in assembling high-throughput data.

*To whom correspondence should be addressed.

The more practical approaches for assembling high-throughput short reads have spawned based on de Bruijn graph approach. Velvet (Zerbino and Birney, 2008) is perhaps the most widely used method for *de novo* genome assembly today. It is very fast in execution, fairly memory efficient and produces reasonably accurate assemblies. Similar to all other methods based on de Bruijn graph, Velvet requires the entire genome to be stored in a graph structure. In the presence of noise, the graph may be too large to be stored on system memory. Furthermore, resulting assembly generated from Velvet tends to contain many errors at small repeat regions. Another approach, Euler-USR (Chaisson *et al*., 2009) is very similar in concept to Velvet, but employs more sophisticated error detection and correction steps. However, in practice, we noted Velvet produces more contiguous and complete assemblies in comparison with Euler-USR. Both Velvet and Euler-USR take full advantage of paired-end read libraries.

One of the major shortcomings of de Bruijn graph approaches is the inability to parallelize the assembly process. This is a critical requirement as many powerful computers utilize multiple processors where numerous threads can be run seamlessly in parallel. Introduction of ABySS (Simpson *et al*., 2009) tackled this issue. The core assembly algorithm of ABySS is very similar to that of Velvet, but it allows de Bruijn graph to be distributed across multiple cores/nodes, and each core/node can operate on the graph independently to a certain extent. The assembly result of ABySS is similar to that of Velvet. However, we noticed that when executed in parallel in a multi-core single computer, Abyss does not offer any advantage over Velvet in term of execution time or memory usage. To utilize Abyss efficiently, it requires a multi-node computing cluster that may seem a disadvantage in an era where computers are increasingly made faster by adding more cores within a single CPU. SOAPdenovo (Li *et al*., 2010) addressed many of these issues by introducing a de Bruijn graph-based method that can seamlessly takes advantage of multi-core systems.

Allpaths/Allpaths2 (Butler *et al*., 2008; MacCallum *et al*., 2009) appears to be the most accurate method at present. It introduces an interesting hybrid approach where the genome is still stored as a large graph; however, the graph is separated into different segments and assembly of these segments can be carried independently. This makes it possible to run some stages of Allpaths algorithm in parallel. The high accuracy of Allpaths is contributed by the fact that it tries all possible ways to assemble every segments; however, this comes at a tremendous cost in terms of time and memory usage, and therefore it will not augment well for larger genomes.

We propose the method PE-Assembler that is capable of handling large datasets and produces highly contiguous and accurate assemblies within reasonable time. Our approach is based on simple

3′ extension approach and does not involve representing the entire genome in the form of a graph. Fundamentally, it is similar to other 3′ extension approaches such as SSAKE, VCAKE and SHARCGS. However, it improves upon such early approaches in multiple ways. The extensive use of paired-end reads ensures that the dataset is localized within the region. Hence, our method can be run in parallel to greatly speedup the execution while staying within reasonable system requirements. Ambiguities are resolved using a multiple path extension approach, which takes into account sequence coverage, support from multiple paired libraries and more subtle information such as the span distribution of the paired-end reads.

2 METHODS

Paired-end reads are also known as paired reads or mate pairs (depending on some technical differences) in different literature. Essentially, they all refer to a pair of short reads that originates from 5′ to 3′ ends of a DNA fragment whose length is known approximately. The length of the fragment is referred to as the insert size. For every paired-end read, its two reads are called the mates of each other. The length of each read is denoted as *ReadLength*. It could be of any length from 25 to 100 bp. The insert size is not exact. It may vary from *MinSpan* to *MaxSpan.*

Our program is called PE-Assembler, which aims to reconstruct the sample genome from a paired-end read library. PE-Assembler can also accept multiple paired-end read libraries of different insert sizes, which can facilitate to resolve ambiguities that cannot be conclusively resolved using a single paired-end read library.

PE-Assembler is fundamentally based on 3′ *overlap extension,* similar to SSAKE and VCAKE. The procedure is illustrated in Figure 1. Given a sequence, PE-Assembler extracts all reads whose prefix aligns with the suffix of the sequence. We define this as an *overlap*. The suffix of each read, which overhangs from the 3′ of the sequence, forms a feasible extension to the contig. If there is a clear consensus for a single base, then that base is appended to the end of the sequence and the process is iterated. Multiple feasible extensions are handled differently in various stages of the algorithm and are described in following sections.

PE-Assembler is implemented as a series of five steps, which are briefly described as follows (also see Supplementary Fig. 1). First, the read screening step selects a set of reads (called 'solid' reads) as starting points for extending the assembly. This step specifically avoids reads containing sequencing errors and reads occurring in repeat regions in the genome. The second step then extends these 'solid' reads using single end reads to make them longer than *MaxSpan*. Those successfully extended regions are called seeds. Seeds are long enough for extension using paired-end reads. Our third step (called contig extension) tries to extend all these seeds using paired-end reads. The resulting sequences are called contigs. The fourth step links those contigs using paired-end reads to form scaffolds (i.e. ordered set of contigs with gaps in between). Finally, the last step tries to fill-in the gaps in between scaffolded contigs. Below, we will detail the five steps.

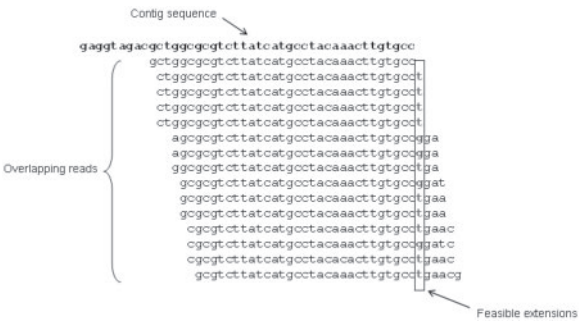

Fig. 1. Overview of 3′ *overlap extension.* Both *t* and *g* are feasible extensions.

2.1 Read screening

Many short read assemblers perform error correction/detection steps prior to the assembly. While it is generally effective in detecting and fixing random sequencing errors, it treats each read as a single read and therefore fails to utilize the pairing information. This may result in overcorrecting the reads coming from low coverage regions as the actual location of the paired-end read is not taken into account.

Our approach does not perform error correction. However, we require a pool of error-free and non-repetitive reads as starting points for the seed building step (Section 2.2). These reads are isolated by carrying out a read screening step.

The idea behind the screening step is similar to the *k*mer frequency based error correction method proposed by Pevzner *et al*. (2001). Its details are as follows. A *k*mer is a length *k* DNA sequence. Provided the genome is sampled at a high coverage, a *k*mer that occurs in the genome is likely to occur multiple times in the input reads. Suppose a particular *k*mer occurs once (or very sparingly) in the input reads, such *k*mer is unlikely to occur in the target genome and is likely to be a result of a sequencing error. Similarly, if a *k*mer occurs at a higher frequency than expected, we can conclude that it may have originated from a repeat region in the genome. A *k*mer that is expected to occur in the actual genome is called a 'solid' *k*mer while a *k*mer that is expected to occur within a repeat region is called a 'repeat' *k*mer. To classify a read as either a solid *k*mer or a repeat *k*mer, we scan the entire dataset of reads to extract the set of *k*mers and their frequencies. A *k*mer frequency histogram is plotted. Then, we identify the solid *k*mer threshold and the repeat *k*mer threshold from the troughs on either side of the main peak (Fig. 2). A read is said to be 'solid' if the frequencies of all its *k*mers are higher than the solid *k*mer threshold and lower than the repeat *k*mer threshold. Only solid reads are chosen as the start points for the next step. Note that this stage does not discard or correct any data. The entire dataset is used in the assembly as it is.

2.2 Seed building

A 'seed' is defined as a contiguous region in the target genome which is of length at least *MaxSpan*. To assemble a seed, we start with an unused solid read as the initial seed and carry out 3′ overlap extension as described above. However, due to the presence of small repeats or sequencing errors, there may be multiple feasible candidates as the next 3′ base.

Ambiguities arising due to repeats can be resolved with the help of paired-end reads. Throughout the seed assembly, we maintain a pool of reads whose

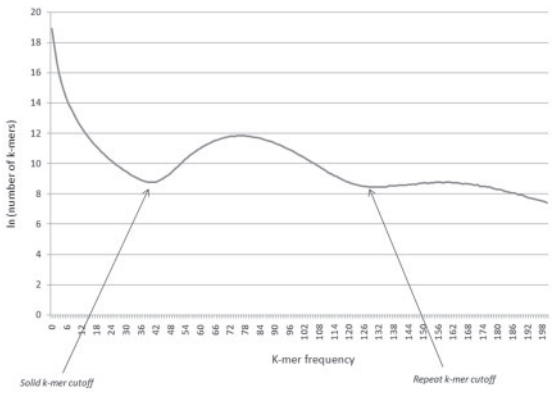

Fig. 2. The *k*mer frequency histogram. We can determine the solid *k*mer cutoff and repeat *k*mer cutoff from the two troughs.

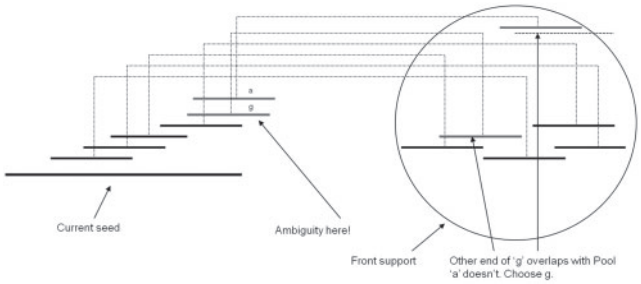

Fig. 3. Resolving ambiguities in the seed building step: suppose the current seed can be extended by two possible candidates 'a' and 'g'. Assume that, for reads extending 'g', their mates overlap with the reads in the pool, while the reads extending 'a' do not have such support. Then, we can safely select candidate 'g' for extension.

Fig. 4. A paired-end read is said to overlap the 3′ end of a seed if the 3′ read of the paired-end read overlaps the 3′ end of the seed and the 5′ read maps on the seed within the expected region, as determined by MinSpan and MaxSpan of the library.

mates map on to the current seed. In case of any ambiguity, for every read overlapping with the seed, we check if its mate overlaps with any reads in the maintained pool (Fig. 3). Those without overlap support are assumed to be noise and thus discarded.

The above method cannot resolve ambiguities arising due to sequencing errors. In such case, we extend every candidate base up to a distance of *ReadLength*. Any extension path arising due to sequencing errors is likely to be terminated prematurely. If only one candidate path can reach the full distance, then that path is assumed to be the correct extension.

At any stage, if there is no candidate for extension (likely due to low sequencing coverage) or multiple candidates for extensions (possibly due to longer repeats), the extension is terminated. Seed will then be extended from the other side. The extension will be 'successfully' terminated once the seed reaches the length of *MaxSpan.*

For every successfully terminated seed, a seed verification step is performed to ensure that the seed represents a contiguous region in the target genome. Precisely, to verify the 3′ end of a seed, we require at least one paired-end read overlaps with 3′ end of the seed (Fig. 4). Similarly, we can verify the 5′ end of a seed. All verified reads are immediately subjected to contig extension step (Section 2.3). Seeds which fail the verification step are discarded.

2.3 Contig extension

The contig extension step aims to extend each verified seed to form a longer contig iteratively. Again, this step relies on *overlap extension* to elongate the current contig; but with some differences. Since a contig is longer than MaxSpan, instead of using single reads to extend the contig, we try to identify feasible extensions from paired-end reads that overlap with the contig. Moreover, when no paired-end read is found overlapping with the contig, we identify feasible extensions from overlapping reads instead.

Fig. 5. Minimum span distance of this chimeric mapping is $a+b$. Actual span may vary depending on the gap size between contigs X and Z.

If a clear consensus is found among the feasible extensions, then that base is appended to the end of the contig and process is repeated. Occasionally, there are multiple feasible candidates to extend the contig. Such scenario may arise due to three reasons. The first reason is sequencing errors. These errors can be dealt similar to the seed building step. The second reason is due to short tandem repeat regions. In such case, we stop the extension and we will try to estimate the correct number of tandem repeats during the gap filling step. The third reason is due to long repeats. In such case we also terminate the extension. Note that when the repeat is longer than *MaxSpan*, we cannot theoretically resolve the ambiguity using the given paired-end read information. A paired-end read library of longer insert size is required to resolve such ambiguity.

The contig extension step is performed until we cannot extend the contig from both ends. Then, the resultant contig is kept to be used in scaffolding.

2.4 Scaffolding

The objective of the scaffolding step is to find the correct ordering of the resulting set of contigs.

As the scaffolding step is very sensitive to the presence of repeat regions, the first step is to demarcate all repeat regions within assembled contigs. In this step, all individual reads are mapped back to the contigs and read density across all the contigs is calculated. The mode of the read density is assumed to be the expected read coverage across the genome. Any region with read density higher than 1.5 times of the mode is considered as a repeat region. Any reads mapped onto such repeat region are discarded.

During this step, additional statistics such as average span and standard deviation for each library is calculated. This information is used during the gap filling stage.

For the scaffolding step, we only consider the paired-end reads whose two reads map uniquely to two different contigs. Such a mapping is referred as a *chimeric* mapping. Although we cannot estimate the exact span of a chimeric mapping, the minimum span for a chimeric mapping can be calculated by the distance that it has covered on the two contigs (Fig. 5). Every paired-end read mapping whose minimum span exceeds *MaxSpan* is discarded.

Two contigs of specific orientation are said to be linked by an edge if there is at least a certain number of chimeric mappings between the two contigs in that orientation. The weight of the edge is the total number of such chimeric mappings, normalized by the total number of paired-end reads in the library. The maximum gap size is estimated by subtracting *MaxSpan* by the average of minimum spans of all chimeric paired-end reads of that edge. Multiple fragment libraries of different insert sizes may be used at this point. Each library will result in its own distinct set of edges.

A potential scaffold is a linear ordering of contigs. An edge between two contigs X and Z is deemed *satisfied* if both contigs X and Z occur within the same scaffold in a correct orientation and the total length of all contigs between X and Z is less than the maximum gap size estimated by that edge; otherwise, if X and Z cannot be arranged so that they are within the expected span, the edge is said to be *contradicted*. The score for each scaffold is calculated by totaling the weights of all satisfied edges and subtracting the weights of all contradicted edges.

The aim of the scaffolding algorithm is to produce a set of scaffolds such that the above score is maximized. However, exact solution to this

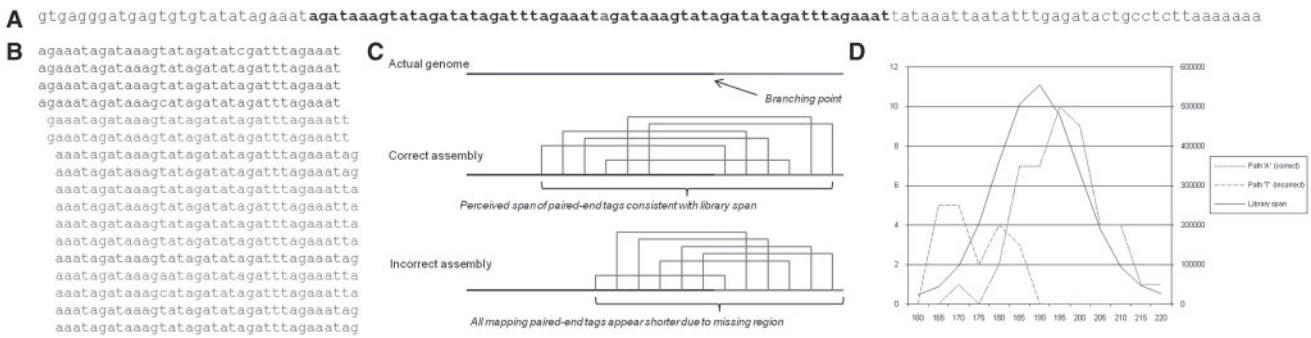

Fig. 6. Use of average span and standard deviation to resolve ambiguities in *Staphylococcus aureus* assembly: (**A**) Reference sequence from region in question. Bolded segments are identical. Both 'a' and 't' seems a valid choice after that region. (**B**) Sequence overlap shows both 'a' and 't' as potential candidates. Both paths are extended up to a distance of '*TagLength*'. (**C**) Illustration of the two different extensions. For each path, spans of paired-end reads mapping across the branching point is kept. Spans resulting from the incorrect assembly are noticeably shorter due to missing region. (**D**) Histograms of perceived paired-end read span of two different paths and that of the entire library. Note that span distribution of correct path closely follows that of the library. Path with the span distribution closest to the library span distribution is chosen.

is computationally prohibitive. Therefore, we employ the following greedy heuristic approach.

The scaffolding process starts by selecting a contig at random as the initial scaffold. The process extends the scaffold iteratively by including contigs to the right. A contig X is said to be a right neighbor of a scaffold if there exists some contig Z in the scaffold such that (Z, X) is an edge and the total length of contigs to the right of Z in the scaffold is less than the maximum gap size of (Z, X). All right neighbors of the scaffold are potential candidates to extend the scaffold from its 3′ end. Each candidate right neighbor is temporally added to the 3′ end of the scaffold, and all permutations of remaining right neighbors are appended after it to obtain multiple possible orderings. Each such potential ordering is evaluated. The candidate right neighbor that results in the ordering with the highest score is permanently added to the 3′ end of the scaffold. This process is repeated until any of the following occurs: neighborhood is empty; best ordering score is negative or the current region of the scaffolding has already been ordered elsewhere. If scaffolding is terminated from the 3′ end, we try to extend the scaffold from the 5′ end. Once both ends are not extendable, we obtain one scaffold and the entire procedure is repeated with an unused contig as the start point to identify other scaffolds.

2.5 Gap filling

The scaffolding step reports a list (or lists) of contigs in the same order as they would be in the actual genome. The adjacent contigs are usually separated by an unknown sequence. The objective of the gap filling step is to assemble the gap region between two adjacent contigs to form a longer contig. Note that the length of the gap can be estimated using paired-end reads, which map across two adjacent contigs.

For every read that occurs in the gap, its mate must map to either the left or the right contig of the gap. Hence, the gap can be filled in using such reads. As we are dealing with a localized set of data, gap filling step can use a less stringent minimum overlap length, thus facilitating assembly of low-coverage regions.

A key difference between the gap filling step and the seed building step is that the former can resolve convoluted repeat regions by exploiting span information of paired-end reads to a greater degree. Similar to seed building step, the assembly is carried out using *overlap extension.* Whenever there are multiple extension paths due to multiple candidate bases, each path is extended up to a distance of *ReadLength*. Moreover, for each extension path, we can obtain the span histogram of all paired-end reads, which map on this extension path. The distribution of this 'perceived' span for each extension path is compared against the span distribution of the entire library. The span distribution of the correct extension will be inline with that of the entire library, whereas distributions of incorrect extensions will exhibit a noticeable shift. This idea is demonstrated in Figure 6.

The adjacent contigs whose gap can be successfully bridged are merged as a single contig. The resulting set of contigs from this step represents the final output of the assembly.

2.6 Parallelization

This section discusses the issue of parallelization for the five steps. For the read screening step, since it is largely disk bound, parallelizing this step does not improve the performance noticeably. All remaining steps can be run as threads on multiple cores sharing the same memory space almost independent of each other.

For the seed building step and the contig extension step, the solid reads and the seeds are distributed to different threads for parallel execution. Provided the genome is reasonably large and the number of threads is not impractically high, we can assume most of the threads assemble different regions of the genome. Every thread will mark the reads which are so far used in the assembly. Periodically, every thread will refer to this information to detect if the region it is currently assembling has been previously assembled by other threads. If a read is detected to be marked by other threads, the thread will rewind the assembly to the last unmarked read and terminated.

Scaffolding step involves mapping back each paired-end read to assembled contigs and forming a graph comprising of contigs as nodes and 'chimeric' paired-end reads as edges. The graph building step is carried out in parallel. Actual scaffolding is carried out in a single thread; however, this step is not very time consuming.

Gap filling can be executed in parallel since gap filling is localized and is independent from one another.

For the entire assembly process, the time taken is roughly inversely proportional to the number of cores/threads utilized. (Please refer to the Section 3.)

3 RESULTS

3.1 Simulated data

To evaluate the goodness of our approach, experiments were carried out on simulated datasets based on *Escherichia coli* and *Schizosaccharomyces pombe* reference genomes. Three libraries of paired-end reads with varying 'fragment lengths' were simulated

Table 1. Details of the simulated dataset

Organism	*Escherichia coli*			*Schizosaccharomyces pombe*			HG 18-Chr 10		
No. of contigs/chromosomes	1			3			1		
Genome length (bp)	4 639 658			12 571 820			135 374 737		
Library	200 bp	1 kb	10 kb	200 bp	1 kb	10 kb	200 bp	1 kb	10 kb
Read length (bp)	35	35	35	35	35	35	75	75	75
Average insert size (bp)	235	1035	10035	235	1035	10035	275	1075	10075
Insert size range (average ± bp)	±40	±200	±2000	±40	±200	±2000	±40	±200	±2000
No. of paired reads (millions)	3.31	3.31	3.31	8.98	8.98	8.98	45.12	9.02	9.02
Coverage	50×	50×	50×	50×	50×	50×	50×	10×	10×
Seq. error rate, %	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
Ligation error rate, %	0.0	2.0	2.0	0.0	2.0	2.0	0.0	2.0	2.0

Table 2. Comparison of simulated data results

	Escherichia coli					*Schizosaccharomyces pombe*					HG18 chr10		
	PA	Velvet	Allpaths2	Abyss	SOAP	PA	Velvet	Allpaths2	Abyss	SOAP	PA	Abyss	SOAP
Parameters	−minol = 25	−*k* = 27 −cov = auto −exp = auto	−*k* = 21	−*k* = 25 −*j* = 3,−*n* = 10 −np = 8	−*k* = 27 −pair_num = 3 −*p* = 8	−minol = 25	−*k* = 29 −cov = auto −exp = auto	−*k* = 21	−*k* = 25 −*j* = 3 −*n* = 10 −np = 8	−*k* = 29 pair_num = 3 −*p* = 8	−minol = 50	−*k* = 45 −*j* = 3 −*n* = 10	−*k* = 31 −*p* = 20
Contig statistics													
No. of contigs (>200 bp)	6	56	44	283	199	31	181	164	650	348	4262	49015	18238
Average length (kb)	777.4	82606	107.6	22.3	22.8	394.7	67.9	75.3	23.1	35.0	30.2	2.9	6.6
Maximum length (kb)	2492.6	708.6	593.7	163.0	232.0	3519.6	856.1	851.0	297.3	468.7	403.5	65.2	155.8
Contig N50 size (kb)	2492.6	398.3	373.3	63.8	49.9	1487.7	273.0	226.8	80.1	99.8	62.4	5.3	13.0
Contig N90 size (kb)	2146.0	109.9	115.4	33.9	12.4	363.6	54.4	59.5	36.7	19.0	11.1	1.7	3.6
Coverage (%)	100.00	99.59	99.85	99.00	98.87	97.78	99.35	98.60	98.38	98.91	90.89	92.04	87.14
Evaluation													
Large misassemblies	0	11	0	1	1	0	17	0	1	4	5	171	605
Segment maps (%)	99.68	94.74	99.18	93.31	93.66	96.42	94.44	96.83	92.72	94.31	86.17	63.61	32.2
Performance[a]													
Total execution time (min)	21	10	227	43	5	101	40	734	98	11	748	N/A[b]	240
Peak memory usage (gb)	2.3	2.9	29.7	2.9	5.9	4.5	7.7	66	6	8.1	15.1	N/A[b]	48.0

[a]*Escherichai coli* and *S.pombe* datasets were run using eight threads for PE-Assembler, SOAPdenovo and ABySS. HG18 Chr10 dataset was run using 20 threads for PE-Assembler and SOAPdenovo. For this dataset ABySS was run across four nodes in a cluster, each running two separate threads.
[b]Execution time and memory usage not available for ABySS. See Supplementary Material.

from each genome (Table 1); a short fragment library of average span 200 bp, a medium fragment library of average span 1000 bp and a long fragment library of average span 10 000 bp. All reads were assumed to be 35 bp long. Precise criteria for simulation are detailed in Supplementary section. For comparison, we executed all popular *de novo* assembly programs such as Allpaths2, Velvet, ABySS and SOAPdenovo in addition to PE-Assembler (denoted by 'PA' in tables). Each program was run with multiple parameters and the best result for each program is quoted below. The summary results for all experiments and parameters are available in Supplementary section.

We adapted the following approach to evaluate each assembly result. All contigs were aligned against the reference genome using BLAT (Kent, 2002). Any contig which does not completely align to the reference genome, while allowing for small indels and mismatches, is deemed a 'large misassembly'. To evaluate the accuracy at micro level, we segmented the reference genome into continuous, non-overlapping sequences of 1000 bp and check if they can be mapped on the assembly's contigs without errors. The number of error-free segments that can be mapped on the contigs is reflective of the accuracy of the assembly. Contiguity of the assembly is measured by the N50 and N90 sizes. The completeness of the assembly is evaluated by calculating the percentage of reference genome covered by the assembled contigs. The computational complexity of each assembler is measured by its running time and memory usage. These evaluation steps are detailed in the Supplementary section.

The results for simulated data are summarized in Table 2. The experiments demonstrate that PE-Assembler can generate highly contiguous assemblies at a very low error rate using less system resources. While Velvet is fast in execution, the number of misassemblies shows that it lags behind PE-Assembler and Allpaths in terms of accuracy. Both ABySS and SOAPdenovo produces highly fragment results with relatively small N50 sizes.

To demonstrate that PE-Assembler is scalable to handle large genomes, we simulated three paired-end read libraries of

Table 3. Performance of PE-Assembler using different read lengths

	Escherichia coli							
	35 bp reads		50 bp reads		75 bp reads		100 bp reads	
	PA	Velvet	PA	Velvet	PA	Velvet	PA	Velvet
No. of contigs (>600 bp)	73	90	64	89	67	83	53	80
Contig N50 size (kb)	124.7	111.8	133.0	132.6	144.0	132.8	178.3	171.8
Contig N90 size (kb)	31.9	35.0	35.2	31.5	35.2	40.1	41.4	40.1
Large misassembiles	0	1	0	1	1	0	1	1
Coverage (%)	98.78	98.76	99.11	99.08	98.91	99.37	98.73	99.36
Execution time[a] (min)	7	6	7	6	6	6	6	7
Peak memory usage (g)	1.4	1.7	1.4	2.1	1.3	2.7	1.3	3.5

[a]PE-Assembler was run using 20 parallel threads.
Velvet was run with following *k*-values, respectively, 23, 31, 43 and 47. –cov_cutoff and –exp_cov was set to auto.

Table 4. Details of the experimental datasets

Organism	*Staphylococcus aureus*		*Escherichia coli*		*Schizosaccharomyces pombe*		*Neurospora crassa*	
No. of contigs/chromosomes	3		1		4		251	
Genome length	2 903 107		4 638 902		12 554 318		39 225 835	
Library (bp)	200	3000	200	3000	200	3000	200	3000
Read length (bp)	35	26	35	26	35	26	35	26
Average insert size (bp)	224	3845	210	3771	208	3658	210	3650
Insert size range (bp)	195–255	3175–4725	180–260	3026–4626	195–265	2935–4535	175–245	2875–4675
No. of paired reads (millions)	5.52	3.89	15.04	5.46	27.58	25.62	95.66	61.88
Approximate coverage	130×	35×	230×	60×	150×	110×	170×	80×

aforementioned fragment sizes from chromosome 10 of HG18 and assembled using PE-Assembler. PE-Assembler can cover 90% of the original chromosome with N50 size exceeding 60 000. ABySS and SOAPdenovo produces a large number of contigs with very low N50 value. We failed to execute both Allpaths2 and Velvet for this dataset due to their high memory usage.

Furthermore, we simulated four libraries of 500 bp fragment paired-end data of four different read lengths at 60× coverage to assess the impact of increase in read length on PE-Assembler. The results (Table 3) show that PE-Assembler benefits from increase in read length and compares favorably against Velvet for all read lengths.

3.2 Experimental data

To assess our approach against wet lab data, we used four datasets provided with Allpaths2. Each dataset contains two paired-end read libraries; one of approximate fragment length 200 bp and the other ranging from 3000 to 4500 bp (Table 4). The single reads were not used.

As the reference genome is provided for every dataset, the evaluation criteria remained the same as Section 3.1. However, since the reference genome and the sequenced genome are not expected to be identical, some minor errors are expected and allowed when we map the assembled contigs onto the reference genome. The results are summarized in Table 5. It shows that PE-Assembler is equally adept in handling experimental data. It records the highest contiguity in the form of N50 sizes across all four datasets.

For the two smaller genomes, the coverage statistics are nearly identical for all four approaches. Assemblies produced by Velvet and ABySS shows several large misassemblies whereas those of PE-Assembler and Allpaths2 are void of such errors. Performance-wise, PE-Assembler is more efficient in memory consumption compared with all other programs. Especially noteworthy is the large amount of memory consumed by Allpaths2 to assemble even the smallest of genomes.

Repeated attempts to assemble the two larger datasets using Allpaths2 failed in our system. We suspect this is due to high memory usage of Allpaths2. Therefore, the comparison is based on the output provided at Allpaths website. The timing quoted here is that reported on the Allpaths2 publication.

For the highly repetitive *S.pombe* genome, PE-Assembler results in an assembly with N50 and N90 sizes far greater than that of Allpaths2, Velvet and ABySS. PE-Assembler also shows better coverage than Allpaths2. The high number of large misassemblies in Velvet and ABySS assemblies demonstrates the susceptibility of de Bruijn graph approach to misassemble genomes in the presence of short repeat regions. In contrast, PE-Assembler and Allpaths2 results in only three and two large misassemblies, respectively. Of the three 'misassembled' contigs in PE-Assembler output, two of them can

Table 5. Comparison of experimental data results

	Staphylococcus aureus				*Escherichia coli*			
	PA	Velvet	Allpaths2	ABySS	PA	Velvet	Allpaths2	ABySS
Parameters	−minol = 25	−*k* = 23 −cov = auto −exp = auto	−*k* = 21	−*k* = 25 −*j* = 2 −*n* = 10 −np = 8	−minol = 25	−*k* = 27 −cov = 12 −exp = auto	−*k* = 21	−*k* = 25 −*j* = 2 −*n* = 10 −np = 8
Contig statistics								
No. of contigs (>200 bp)	24	60	14	187	21	121	25	277
Average length (kb)	119.8	48.0	205.0	18.3	176.8	37.5	184.1	21.4
Maximum length (kb)	949.9	475.6	1122.8	175.1	895.9	356.6	1015.3	160.4
Contig N50 size (kb)	685.8	314.9	477.2	63.8	428.8	105.6	337.1	55.2
Contig N90 size (kb)	107.5	37.79	84.0	31.9	143.1	25.4	81.7	31.8
Coverage (%)	99.45	98.99	99.24	98.28	99.56	99.19	99.63	98.96
Evaluation								
Large misassemblies	0	5	0	1	0	4	0	1
Segment maps (%)	98.48	96.66	98.55	94.56	98.73	95.60	99.18	94.55
Performance[a]								
Total execution time (min)	17	8	95	13	34	25	222	29
Peak memory usage (gb)	1.9	2.8	20	2.6	3.3	6.9	37.6	5.3
	Schizosaccharomyces pombe				*Neurospora crassa*			
	PA	Velvet	Allpaths2	ABySS	PA	Velvet	Allpaths2	ABySS
Parameters	−minol = 25	−*k* = 25 −cov = 3 −exp = auto		−*k* = 25 −*j* = 2 −*n* = 10 −np = 8	−minol = 25	−*k* = 25 −cov = auto −exp = auto		−*k* = 25 −*j* = 2 −*n* = 10 −np = 16
Contig statistics								
No. of contigs (>200 bp)	169	362	353	1028	2708	5079	1687	9916
Average length (kb)	72.1	33.7	33.8	13.0	12.8	6.8	18.3	3.8
Maximum length (kb)	571.1	443.0	257.2	136.8	156.2	71.0	161.2	56.0
Contig N50 size (kb)	147.7	110.6	50.0	36.0	20.7	11.6	17.6	8.1
Contig N90 size (kb)	40.0	33.2	12.2	12.3	–	–	–	1.0
Coverage (%)	96.97	97.82	95.20	97.93	87.40	87.70	78.38	88.70
Evaluation								
Large misassemblies	3	26	2	27	16	273	18	395
Segment maps (%)	95.51	94.26	92.60	91.08	82.06	77.44	74.66	71.28
Performance[a]								
Total execution time (min)	364	125	4830[b]	72	1416	266	5196[b]	331
Peak memory usage (gb)	6.6	15	N/A	6.6	21	45	N/A	25.6

[a]All experiments were run in a 8-core machine except for *N.crassa* dataset, which was run using 16-cores.
[b]Reported as in Allpaths2 publication, where experiments were carried out in a 16-core machine.

be properly aligned against other strains of *S.pombe* and therefore they are likely due to differences between assembled strain and the reference. PE-Assembler's assembly for *S.pombe* also results in the highest number of segments maps, testament to both its coverage and accuracy.

For the relatively larger *Neurospora crassa* genome, PE-Assembler's result leads in terms of contiguity and coverage. Note that Allpaths2's assembly is of significantly low coverage in comparison with other assemblies. Also note that *N.crassa* reference genome is unfinished and it consists of many contigs. The 'large misassemblies' reported is likely to be inflated.

Current version of SOAPdenovo ignores reads of length <35 bp for the scaffolding process. Therefore, we did not test SOAPdenovo against the experimental datasets as it would not be a fair comparison.

3.3 Parallelization and running time

One of the most important aspects of our method is its ability to carry out the entire assembly process in parallel. We carried out a series of experiments to determine how parallelization affects the execution time of the assembler. The simulated *E.coli* dataset with 200 and 10 000 bp libraries were assembled using 1–8 separate threads in an 8-core-CPU machine. Each thread was executed in a separate CPU core.

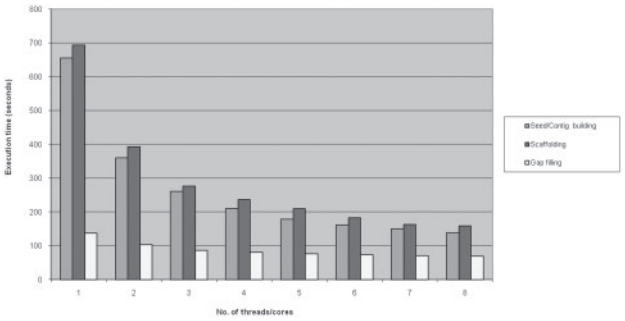

Fig. 7. Execution time with respect to number of threads/cores utilized. Utilizing multiple cores dramatically reduces execution time. Theoretically, the improvement should be linear with number of parallel threads; however, this is masked by the fact that each step has constant IO overhead which cannot be parallelized.

Figure 7 shows that distributing each step across multiple CPU cores in parallel decreases the execution time proportionally to the number of CPUs utilized. However, unlike the implementation in Allpaths2, the parallel implementation does not come at an extra memory overhead as the data structures are shared by each thread. In each of the experiments, the maximum memory utilization was constant at 1.3 GB.

4 DISCUSSION

PE-Assembler has demonstrated that it is possible to obtain complete and highly accurate *de novo* genome assemblies using high-throughput sequencing data within reasonable time and memory constraints. The highlight of PE-Assembler is that it eschews the traditional graph-based approach in favor of a simple extension approach.

The advantages of this approach are numerous. Memory requirements of graph-based approaches seem to increase exponentially as genome and data size increase. This was highlighted by the inability of Velvet and Allpaths2 to cope with simulated HG18 Chr10 dataset. In contrast, PE-Assembler produced a very usable assembly within a realistic memory limit.

Our approach is fundamentally similar to other 3′ extension approaches such as SSAKE, SHARCGS and VCAKE, but distinguishes itself due to its extensive use of paired-end reads. Not only does it make such approach scalable to larger genomes' datasets by localizing data, it also contributes to its high accuracy. As evident from both simulated and experimental data results, PE-Assembler is the least prone of all algorithms to misassemble different regions of the genome in a continuous segment.

Perhaps the most important aspect of PE-Assembler is its ability to seamlessly parallelize the assembly process. Multiple threads can simultaneously assemble the genome at various positions across the genome, while a simple detection mechanism will ensure that multiple assemblies of the same region are highly unlikely. Also noteworthy is that parallel assembly in PE-Assembler does not come at an extra cost in memory as in other methods such as Allpaths2 or ABySS. Being able to massively parallelize the assembly process at no extra overhead, it will prove valuable in assembling mammalian genomes as well as in larger metagenomics projects. With minor modifications, this approach can be extended to be run in a computer cluster across multiple nodes to further decrease the running time.

ACKNOWLEDGEMENTS

The authors would like to extend their gratitude to Pauline Chen of Research Computing Group, GIS, for her help in evaluation and testing process. We further like to thank Daniel Zerbino for his help in running Velvet and the reviewers for their useful feedback and insight.

Funding: This research was supported by MOE AcRF Tier 2 funding R-252-000-444-112 and Agency for Science, Technology and Research (A*STAR).

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.

Chaisson,M.J.P. *et al*. (2009) De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res.*, **19**, 336–346.

Dohm,J.C. *et al*. (2007) SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res*., **17**, 1697–1706.

Jeck,W.R. *et al*. (2007) Extending assembly of short DNA sequences to handle error. *Bioinformatics*, **23**, 2942–2944.

Kent,J.W. (2002) BLAT–the BLAST-like alignment tool. *Genome Res.,***12**, 656–664.

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

MacCallum,I. *et al*. (2009) ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biol.*, **10**, R103.

Pevzner,P.A. *et al.* (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci. USA*, **98**, 9748–9753.

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

Warren,R.L. *et al*. (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**, 500–501.

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