PERGA: A Paired-End Read Guided De Novo Assembler for Extending Contigs Using SVM Approach

Xiao Zhu[†]
School of Computer Science and Technology
Harbin Institute of Technology
Harbin 150001, China
zhuxiao.hit@gmail.com

Henry C.M. Leung[†]
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
cmleung2@cs.hku.hk

Francis Y.L. Chin
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
chin@cs.hku.hk

Siu Ming Yiu
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
smyiu@cs.hku.hk

Guangri Quan
National Pilot School of Software
Harbin Institute of Technology
WeiHai 264209, China
grquan@hit.edu.cn

Bo Liu
School of Computer Science and Technology
Harbin Institute of Technology
Harbin 150001, China
bo.liu@hit.edu.cn

Yadong Wang*
School of Computer Science and Technology
Harbin Institute of Technology
Harbin 150001, China
ydwang@hit.edu.cn

ABSTRACT

Since the read lengths of high throughput sequencing (HTS) technologies are short, *de novo* assembly which plays significant roles in many applications remains a great challenge. Most of the state-of-the-art approaches base on de Bruijn graph strategy and overlap-layout strategy. However, these approaches which depend on *k*-mers or read overlaps do not fully utilize information of single-end and paired-end reads when resolving branches, e.g. the number and positions of reads supporting each possible extension are not taken into account when resolving branches.

We present PERGA (Paired-End Reads Guided Assembler), a novel sequence-reads-guided *de novo* assembly approach, which adopts greedy-like prediction strategy for assembling reads to contigs and scaffolds. Instead of using single-end reads to construct contig, PERGA uses paired-end reads and different read overlap size thresholds ranging from $O_{\max}$ to $O_{\min}$ to resolve the gaps and branches. Moreover, by constructing a decision model using machine learning approach based on branch features, PERGA can determine the correct extension in 99.7% of cases. When the correct extension cannot be determined, PERGA will try to extend the contigs by all feasible extensions and determine the correct extension by using look ahead technology.

We evaluated PERGA on both simulated Illumina data sets and real data sets, and it constructed longer and more correct contigs and scaffolds than other state-of-the-art assemblers IDBA-UD, Velvet, ABySS, SGA and CABOG.

Availability: https://github.com/hitbio/PERGA

Categories and Subject Descriptors
J.3 [**Computer Applications**]: Life and Medical Sciences – *Biology and genetics.*

General Terms
Algorithms

Keywords
Genome assembly, Greedy-like prediction, Variable overlap sizes, Look ahead technology

1. INTRODUCTION

The high throughput sequencing (HTS) technologies have emerged for several years [1, 2] and are widely used in many biomedical applications, such as large scale DNA sequencing [3],

* To whom correspondence should be addressed:
Yadong Wang, Professor, School of Computer Science and Technology, Harbin Institute of Technology, 92 West Dazhi Street, Nan Gang District, Harbin 150001, China. Email: ydwang@hit.edu.cn
† Contributed equally

re-sequencing [4] and SNP discovery [5, 6], etc. However, since the length of reads generated by HTS technologies (typically 50 - 150 base pairs [7-9]) are much shorter than those of the traditional Sanger sequencing (typically about 800 base pairs [10]), and the per-base sequencing error is high [11], the short read assembly is still a great challenge for genome sequencing.

There are two major approaches for assembly: the overlap-layout strategy and the de Bruijn graph strategy. The overlap-layout-based approaches first compute the overlaps among reads, and then assemble according to the read overlaps. There are also two subcategories for the overlap-layout approaches: the greedy extension strategy and the overlap graph strategy.

The first several *de novo* assemblers for the HTS data, such as SSAKE [12], VCAKE [13], SHARCGS [14], adopt a greedy extension strategy in which reads are stored in a prefix/suffix tree to record overlaps, and assembly is performed based on base-by-base 3' extension according to the simple greedy heuristics of selecting the base with maximum overlap or using the most commonly represented base. Thus, these assembly heuristics usually have to compromise between contiguity and accuracy. If there are more than one feasible extensions due to sequencing errors or similar regions in the genome, the extension will stop (see Figure 1). In fact, many of these situations should consider the extensions and the ambiguity can be resolved later to generate longer contigs, e.g. erroneous extensions due to sequencing error at the end of a read usually cannot be extended in later steps (dead ends [15]) and multiple extensions due to sequencing error in the middle a read should be extended to the same nucleotide in later steps (bubbles [15]). Moreover, the assembly algorithms store the correct and erroneous reads and their reverse complements inefficiently, so their memory consumptions are usually very large, which limits their application for large amount of HTS datasets.

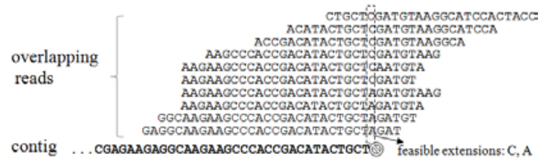

Figure 1. Greedy extension stops caused by feasible extensions that differ not much.

Overlap graph based approaches for short reads, such as Edena [15] and CABOG [16], construct an overlap graph in which a vertex represents a unique read and an edge connects vertices *u* and *v* if and only if *u* and *v* overlap each other sufficiently, and assembly is performed by simplifying the graph based on topologies, such as transitive edges, dead ends and bubbles. Each simple contiguous sequence in the simplified graph represents a contig. These approaches are not suitable for HTS data because they require enormous computations to detect overlaps among a great amount of reads. Recently, new methods based on read overlaps using Burrows-Wheeler Transform [17], such as SGA [18] and fermi [5], could assemble large amount of HTS data. However, they require much more computations to construct a FM-index [19].

The de Bruijn graph strategy, which was first introduced in EULER [20], is particularly suitable for short reads of HTS technologies. This approach breaks up each read into a collection of overlapping *k*-substrings, called *k*-mers, to construct a de Bruijn graph. In the graph, a vertex represents a unique *k*-mer and an edge connects vertices *u* and *v* if and only if *u* and *v* overlapped by *k*–1 nucleotides and appear consecutively in a read, and assembly is performed by removing dead ends and merging bubbles to simplify the graph in which a simple path represents a contig. As the *k*-mers have fixed length and erroneous *k*-mers can be detected from their low sampling rates, the de Bruijn graph consumes much less memories than the overlap graph. Some assemblers, such as Velvet [21], EULER-SR [22], ALLPATHS [23], ABySS [24], IDBA [25], IDBA-UD [26], SOAPdenovo [27], adopt this strategy, and achieve good performances. However, most of them only use a fixed *k*-mer size except IDBA and IDBA-UD. Since small *k* values will lead to better connectivity with much more branches due to repeat segments larger than *k*, whereas large *k* will result in worse connectivity with more gaps due to missing *k*-mers [26]. Most of these assemblers just pick an intermediate *k* to compromise these two problems. IDBA [25] and IDBA-UD [26] give better results by iterating the *k*-mer sizes from $k_{\min}$ to $k_{\max}$ by using small *k* to resolve gaps and large *k* to resolve branches (Figure 2).

All the above assemblers have not fully utilized the information of single-end and paired-end reads when resolving branches. Assemblers based on overlap-layout strategy usually stop when there are more than one choices for extension without considering the number of reads supporting each extension. Assemblers based on de Bruijn graph strategy resolve branches by the topology of de Bruijn graph. Similarly, the number and positions of reads supporting each outgoing edge are not taken into consideration when resolving branches. In fact, the number and positions of reads should be used in resolving branches. Given a branch with two possible extensions (or outgoing edges), even though reads are not sampled from the genome uniformly (usually not much different), if there are much more reads supporting one extension than the other, assembler should extend the contig to the one with more supporting reads and treat the other as incorrect. Even when the numbers of reads supporting both extensions are the same, if the set of reads supporting one extension have much longer region aligned to a contig than the set of reads supporting another extension, assembler should extend the contig to the former extension because short overlapped reads may due to sequencing errors or short repeats.

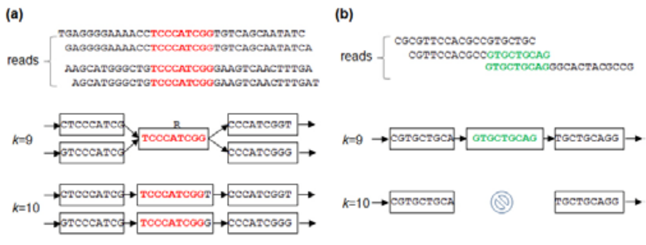

Figure 2. Small *k* will lead to better connectivity and much more branches, large *k* can resolve many repeats smaller than *k* while causing much more gaps. (a) Larger *k* can resolve repeats smaller than *k*. (b) Small *k* will lead to better connectivity.

In order to utilize the information in reads for assembling, we introduce PERGA (Paired-End Reads Guided Assembler), a novel *de novo* sequence reads assembler which adopts greedy-like prediction strategy for assembling reads to form contigs and scaffolds. Instead of using single-end reads to construct contigs, PERGA uses paired-end reads and different read overlap size thresholds ranging from $O_{\max}$ to $O_{\min}$ to resolve the gaps and branches. In PERGA, contigs are extended based on base-by-base extension. Paired-end reads are aligned to contigs for determining possible extensions. When there are not many paired-end reads in some genome regions, single-end reads with variable overlap sizes from larger threshold $O_{\max}$ to smaller threshold $O_{\min}$ are applied to handle branches and gaps. Large overlap size $O \geq O_{\max}$ is used in priority to extend contigs to resolve branches; and if there are missing overlaps for larger O, then a degressive smaller O will be used to obtain better connectivity to resolve gaps until the read overlap are found before $O = O_{\min}$. Moreover, even when there are multiple possible extensions, PERGA will determine which extension is more likely based on several features, e.g. the number of reads supporting each extensions, coverage ratio (ratio of the average number of reads near the branch to the average number of reads in the contig), and the gap span (the distance of the aligned positions of the reads considered in the current extensions and the reads considered in previous extensions but not in the current extension). By constructing decision models using machine learning approach based on these features, PERGA can determine the correct extension in 99.7% of cases. Note that PERGA will also determine the case and stop extending the contig when both extensions are likely to be correct.

As there are still some mis-predictions (about 0.1~0.3%), PERGA handles them by look ahead technology (Figure 3). When there are multiple possible extensions, PERGA extends the contigs by all feasible nucleotides and determine if these extensions are due to sequencing errors or repeats. If the multiple extensions are due to sequencing errors, PERGA will merge the extensions together to form a single contig, otherwise, it will retract and stop extending the contigs at the branch. PERGA is very effective and can greatly improve the contig sizes (e.g., N50 size can be improved from 107 kbp to 130 kbp).

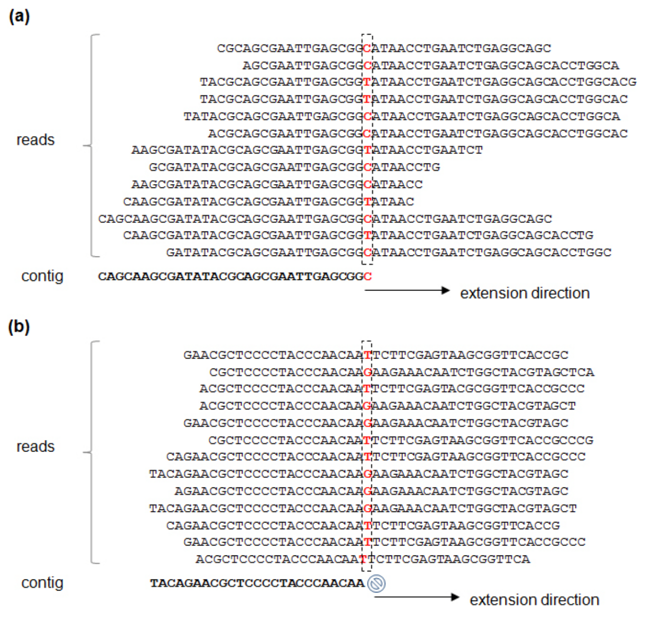
(a)
reads
CGCAGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGC
AGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGCACCTGGCA
TACGCAGCGAATTGAGCGGTATAACCTGAATCTGAGGCAGCACCTGGCACG
TACGCAGCGAATTGAGCGGTATAACCTGAATCTGAGGCAGCACCTGGCAC
TATACGCAGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGCACCTGGCA
ACGCAGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGCACCTGGCAC
AAGCGATATACGCAGCGAATTGAGCGGTATAACCTGAATCT
GCGATATACGCAGCGAATTGAGCGGCATAACCTG
AAGCGATATACGCAGCGAATTGAGCGGCATAACC
CAAGCGATATACGCAGCGAATTGAGCGGTATAAC
CAGCAAGCGATATACGCAGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGC
CAAGCGATATACGCAGCGAATTGAGCGGTATAACCTGAATCTGAGGCAGCACCTG
GATATACGCAGCGAATTGAGCGGCATAACCTGAATCTGAGGCAGCACCTGGC
contig
CAGCAAGCGATATACGCAGCGAATTGAGCGGC
extension direction
(b)
reads
GAACGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCACCGC
CGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGCTCA
ACGCTCCCCTACCCAACAATTCTTCGAGTACGCGGTTCACCGCCC
ACGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGCT
GAACGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGC
CGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCACCGCCCG
CAGAACGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCACCGCCC
TACAGAACGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGC
AGAACGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGC
TACAGAACGCTCCCCTACCCAACAAGAAGAAACAATCTGGCTACGTAGCT
CAGAACGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCACCG
GAACGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCACCGCCC
ACGCTCCCCTACCCAACAATTCTTCGAGTAAGCGGTTCA
contig
TACAGAACGCTCCCCTACCCAACAA
extension direction

Figure 3. The extension situation and the stop situation by look ahead technology. (a) The extension case that only one base mismatch and there is one single path to be extended. (b) The stop case that there are two distinct paths TTCTTCGAGTAAGCG… and GAAGAAACAATCTGG…, and the consensus sequence derived from the paths is not highly agreed at each position, thus the extension should be stopped to avoid mis-assemblies.

According to our experiments, PERGA gives better performances than other assemblers with longer and more accurate contigs (scaffolds) using less running time with moderate memory because of its greedy-like prediction models, the look ahead technology and the variable overlap sizes approach for contig extensions.

2. METHODOLOGY

Figure 4 shows the overview of the proposed approach of PERGA, which consists of two phases: assembly of reads, and assembly of contigs (i.e., scaffolding). In phase (a), a *k-mer hash table*, which is used to represent read overlaps by the consecutive *k*-mers, is constructed from the set of input reads *R*, and then PERGA uses paired-ends and variable read overlap sizes thresholds from $O_{\max}$ to $O_{\min}$ to extend contigs. A greedy-like prediction strategy in which a *k*-mer is chosen as a start of contig extension is performed iteratively in the 3’ direction one base at a time until there are either no overlapping reads or a repeat is found, then the contig will be extended on the 5’ end in the same way. For each base extension, PERGA prefers the reads having more overlaps with contigs and uses the reads having the most represented base for extension. When extending a base, PERGA firstly uses paired-end reads to navigate contig extension with the highest priority, as it can resolve the branches caused by repeats smaller than the insert size with much more confidence than those using single-end reads. However, as there may be genome regions with low sequencing depth and insufficient paired-end reads, PERGA uses single-end reads to extend contigs in such regions by applying the variable overlap size thresholds ranging from larger $O_{\max}$ to smaller $O_{\min}$ to resolve repeats of sizes smaller than $O_{\max}$ and to resolve gaps due to the missing large overlaps.

When extending contigs, there are branches which have more than one feasible extension with various read occurrences, which are mainly due to repeats in genome or sequencing errors in reads. Instead of stopping the extension, PERGA records the branch information to generate hyperplanes for the paired-ends and single-ends respectively by Support Vector Machine method. Finally, these two SVM models are used to determine when to extend or stop for branches while assembling, and in most cases, branches can be correctly resolved.

However, there are also a few exceptions when using the SVM models to decide the navigation: there are a few branches that are incorrectly stopped or incorrectly extended. These situations can be resolved by looking ahead to find the feasible paths and PERGA can resolve the incorrect stops and incorrect extensions, and make the contig much longer with fewer mis-assemblies.

PERGA handles erroneous bases in reads using topological structures while extending the contig, which is similar to the removals of dead ends and bubbles for de Bruijn graph based approaches. During extension, errors at ends of reads will lead to dead ends, and the other errors in the inner part of reads will cause bubbles, PERGA deals with dead ends with lengths smaller than read length and tolerates bubbles with sizes no more than $O_{\min}$. In PERGA, the dead ends containing erroneous *k*-mers will be excluded from assembly by other correct reads during extension; and the bubbles in reads are deemed as valid substitution.

In phase 2, paired-end reads are aligned onto contigs and used to order and orient contigs to form scaffolds (i.e. ordered sets of

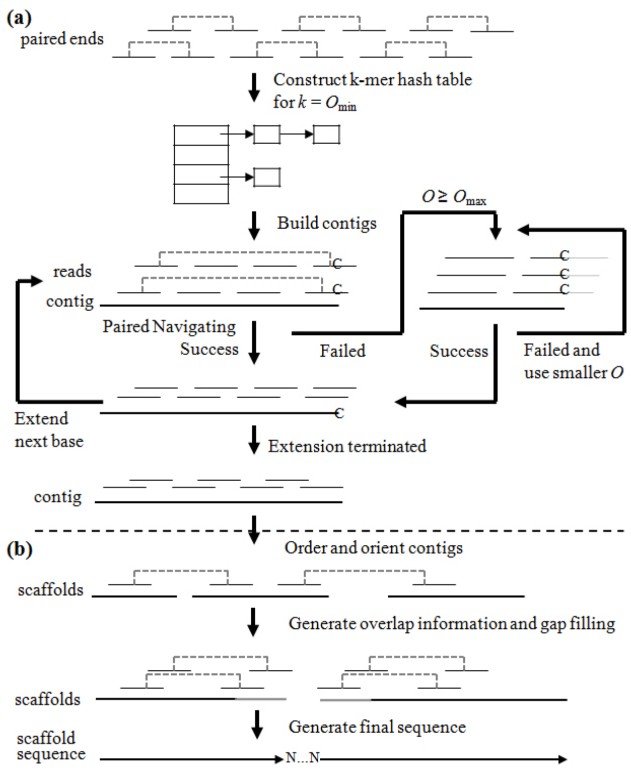

Figure 4. Workflow of PERGA. There are two phases for PERGA: assembly of reads and assembly of contigs. (a) In phase 1, k-mer hash table is constructed using paired-end reads for $k = O_{min}$, then contigs are extended iteratively one base at a time (left feedback loop) at 3' end by using paired-end reads in high priority, and variable overlap size thresholds ranging from O_{max} to O_{min} (right feedback loop) if there are no paired-ends. (b) In phase 2, paired-end reads are used to order and orient contigs, fill intra-scaffold gaps to generate larger scaffolds.

contigs with gaps in between). Then, the overlap sizes and the gap sizes for the linked neighboring contigs are computed, and the overlapped neighboring contigs are merged to form longer contiguous sequences, and the gapped neighboring contigs are processed using a local assembly approach to close their intra-scaffold gaps to generate longer contiguous sequences. Unlike SOAPdenovo [27] which trims *k* bases to exclude erroneous bases at contig ends when scaffolding, PERGA corrects such erroneous bases by pair-wise alignment of the overlapped ends of the neighboring contigs. Finally, the scaffold sequences are generated to form the resultant assembly according to the overlaps and the gap sizes of the contigs in scaffolds.

2.1 Assembly of reads to contigs

The first phase of PERGA is to assemble reads into contigs using a greedy-like prediction method based on paired-end reads information (if possible) and then single-end reads. The algorithm starts with a *k*-mer at the end of an unused read and treats it as contig. PERGA iteratively aligns paired-end reads to contigs and tries to extend it at both ends. In order to determine the possible extension, either A, C, G or T, a SVM model is used to determine whether PERGA should extend the contig using the nucleotide with maximum supports from aligned paired-end reads (instead of extending the contig only when all aligned reads support the same extension as other greedy algorithms) based on the properties of aligned reads. Besides, even when the SVM model cannot determine whether extending the contig or not, PERGA will try to extend the contigs with all possible nucleotides and determine which nucleotide should be used to extend the contig by the later steps (look ahead technology). After extension, errors in aligned reads can be identified and be corrected for later extension. Details of the assembling step are described as follows.

2.1.1 *Constructing k-mer hash table*

PERGA applies a *k*-mer based, cost effective approach to perform read alignments. Overlaps of two reads can be represented by their consecutive common *k*-mers, for example, two reads overlap with *w* nucleotides should share $w - k + 1$ consecutive *k*-mers. Thus, PERGA uses a hash table to store occurrences of *k*-mers in reads. We refer *occurrence* of a *k*-mer as the positions on reads it appears. Note that a *k*-mer may occur in multiple reads and a *k*-mer and its reverse complement are stored at the same entry.

2.1.2 *Aligning reads to contig*

Paired-end reads information is used to extend a contig before single-end reads information because it can resolve longer repeats, i.e. up to the insert size of paired-end reads. PERGA automatically infers the mean insert size as well as the standard deviation of the paired-end reads that have been assembled onto contigs. Reads with similarity >90% are aligned onto contigs. Only those two ends which are aligned in correct directions, i.e. pointed to each other on different strands, are used to infer the mean insert size and the standard deviation.

As shown in Figure 5, PERGA aligns paired-end read onto a single contig. Reads with one end totally aligned to the contigs and the other end partially aligned to the contig are used to determine the extension of contig.

When starts the extension of a contig, the *k*-mer at end of a read is selected as the start contig, reads sharing the same *k*-mer can be effectively aligned to the contig end by using *k*-mer hash table, and the contig is extended iteratively using SVM model to eliminate sequencing errors and avoid the impacts of short repeats by applying the variable overlap size approach based on single reads. When the contig is long enough to use the paired-end reads, the extension will be applied using the paired-end reads in highest priority to avoid repeats shorter than the insert size. As single reads are not suitable for resolving repeats, so that the look ahead technology is just used when paired-end reads are possible. Moreover, when the start *k*-mer contains sequencing errors, it typically has low frequency in *k*-mer hash table, and such *k*-mers are excluded from the start construction of a contig.

When PERGA cannot determine the extension from the aligned paired-end reads, single-end reads information, including paired-end read with one end aligned to the end of a contig and the other end unaligned or unsequenced, is used to determine the extension of contig (Figure 6).

Since the alignment of single-end reads are less confident than the paired-end reads especially when the length of aligned region *O* is short, single-end read information is used carefully from reads with large *O* to reads with small *O*. PERGA determines the possible extension using reads with *O* larger than a larger threshold O_{max} then to smaller threshold O_{min} iteratively. Thus, if PERGA can determine the extension using reads with large *O* confidently, it will not consider those reads with small *O*. In Figure 7, the contig is extended by reads 1 and 2 ($O \geq 6$) and

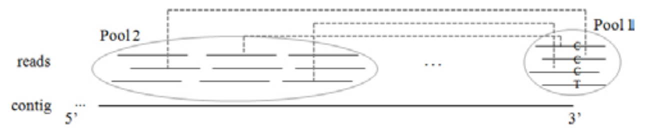

Figure 5. Extension using paired-end reads. Contig is extended at the 3' end according to the reads in Pool 1 and whose mates in Pool 2. There are two candidate bases 'C' and 'T', and 'C' is well supported by the mates in the two pools, whereas 'T' has no paired-end reads support, thus 'C' will be chosen to append onto the contig.

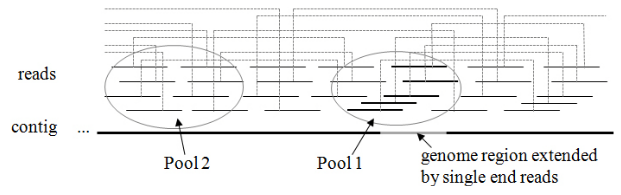

Figure 6. Using single-end reads for extension. When assembling the grey color region which cannot be assembled by paired-end reads and the reads in Pool 1 have no mates in Pool 2, the reads in Pool 1 are used as single-end reads to extend the contig.

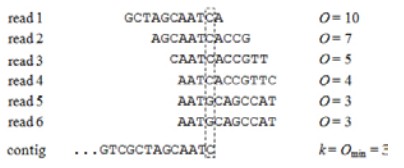

Figure 7. Example of the variable overlap size approach in contig extension. Suppose *k* = *O*$_{\text{min}}$ = 3, *O*$_{\text{max}}$ = 6. There are 6 assembling reads being taking part in assembly, reads 1-4 are the reads that can be assembled onto the contig, while reads 5-6 are the reads that should be assembled onto other regions. The contig is extended using *O* ≥ 6 by reads 1-2, and if there are no reads having *O* ≥ 6, then smaller *O* ≥ 5 will be used in the same way until the contig can be extended successful. AAT is a repeat that can be resolved by *O* ≥ 4 and GCA is another repeat that has been resolved by reads as such reads do not overlap each other.

resolves the repeat AAT in reads 5 and 6 from other genome regions, and if there are no reads having $O \geq 6$, then smaller $O \geq 5$ will be applied again in the same way. Moreover, the read overlap approach can resolve repeats in the reads without overlaps among each other, e.g., GCA from reads 1, 2, 5 and 6.

2.1.3 *SVM navigation models*

When extending contigs, there may be more than one feasible extensions with various supporting reads that are mainly due to repeats or sequencing errors, i.e. there is a branch. When determining correct extension at branch, PERGA records the branch information as features (*maxOcc*, *secOcc*, *covRatio*, *gapLen*), where *maxOcc* is the number of reads supporting the majority nucleotide, *secOcc* is the number of reads supporting the second majority nucleotide, *covRatio* is the ratio of the average number of aligned reads (per nucleotide) at the extended ends (within two read lengths) to the average number of aligned reads for the contig, *gapLen* is the maximal base span of start aligned positions of two reads on contig. The idea is that for a branch, if its maxOcc and secOcc differ a lot (e.g., secOcc/maxOcc < 0.7), the feasible extension corresponding to the maxOcc is usually a correct extension; otherwise, the extension corresponding to maxOcc might be incorrect and should be stopped for further checking. A branch with low gapLen suggested that the number reads aligned to the end of contig is high and the maxOcc should be a correct extension. A branch with covRatio larger than one suggests that there is a repeat nearby and PERGA should extend more carefully.

For training the SVM prediction models, we recorded the branches of the four features while assembling, and treated each branch as a point in a four-dimensional space in which these points can be used to draw a hyperplane by machine learning approach to separate the branches that should be continued or stopped. By comparing them to the reference while assembling, these branches can be classified into four categories: correct extension, wrong extension, correct stop, wrong stop. We labeled each branch as CONTINUE if the branch is a correct extension or a wrong stop that should be continued; otherwise it is a STOP branch that should be stopped.

Based on training dataset on branches, PERGA can determine the cases when we should extend a contig using the nucleotide with maximum support or not. A Support Vector Machine method using polynomial kernel function $K(x, y) = <x, y> \times (1+<x, y>)^2$, where x, y are vectors containing branch information, $<x, y>$ is the dot product of x, y being constructed based on the four features and is used to determine if a contig should be extended. Note that PERGA will first determine the correct extension using features calculated based on aligned paired-end reads. If PERGA fails to decide whether to extend the contig, it will recalculate the features using aligned single-end reads from $O \geq O_{\text{max}}$ to O_{min} and will determine when to extend the contig.

2.1.4 *Look ahead technology*

Since SVM is not perfect, there are a few exceptions when using the SVM models to decide whether to extend a contig or not. For these exceptional cases, PERGA looks ahead to find all feasible extensions. Starting from the branches, PERGA finds the feasible extensions and continues extending the contig. PERGA will search 30 nucleotides for all these possible extensions from branches, and then compare the sequences of different extensions. Based on the assumption that if the multiple extensions are due to repeat, it will be hard to get a highly agreed consensus sequence than the case that the multiple extensions are due to sequencing errors, PERGA calculates the ratio of support for the majority nucleotide at each position and assumes the majority nucleotide as incorrect if its ratio is less than 0.9. If there are more than 3 incorrect nucleotides, the extension will be stopped, otherwise, the extension will be continued using the majority nucleotide.

2.1.5 *Handling erroneous bases*

Erroneous bases in reads for HTS data can make the assembly problem much more complex and error-prone, and cannot be easily solved by paired-end reads and variable overlap size approach. To resolve ambiguities arising due to sequencing errors, PERGA applies a method similar to the approach based on topological structures [15, 21, 24]. Errors at the ends of reads usually lead to short dead ends which are likely to be terminated prematurely, and errors in the inner part of reads will cause small bubbles in which the two paths have similar bases with the same starting and ending reads. Note that such dead ends and bubbles are checked in reads rather than in contigs, and PERGA checks such errors rather than corrects them, as it only needs to determine whether the reads can be assembled onto contigs. It checks the similarity between a read and the contig according to topological structures, and if similarity is high, say ≥ 90%, then the read is assembled onto contigs regardless the errors; otherwise, the read will be not assembled onto contigs, instead, it might assembles into other genome regions with higher similarity.

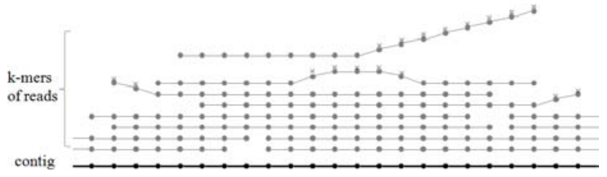

Figure 8. Scheme for removing erroneous bases. Erroneous bases in reads will cause dead ends and bubbles that can be implicitly resolved as these errors can be masked by these correct reads. Reads with low similarities probably can be assembled onto other contigs with higher similarity.

As the extension is carried out based on greedy approach, the dead ends and bubbles in reads are removed implicitly (Figure 8). There are many overlapping reads when assembling a contig region, thus consensus bases of contigs are derived from the corrected reads, and erroneous bases are masked by these correct reads to avoid adverse effects to contigs.

2.2 Assembly of contigs

PERGA assembles generated contigs into larger scaffolds using paired-end information. In this procedure, reads are aligned onto contig ends to order and orient contigs to generate scaffolds. After constructing scaffolds, PERGA merges the overlapped neighboring contigs, fills intra-scaffold gaps, and generates consensus sequences to give final assembly (Figure 4b). Detailed scaffolding method is described in the following subsections.

2.2.1 *Reads alignment*

If one end of a paired-end read uniquely aligned to one contig and the other end uniquely aligned to another contig, these two contigs should appear adjacently in the genome. Note that reads aligned to multiple contigs should not be considered. As reads with both ends aligned to the same contigs does not provide extra information for constructing scaffold, PERGA aligns reads to the end of contig, called *linking regions*, within default 1Kbp only.

2.2.2 *Linking contigs to scaffolds*

Since reads may be sampled from positive strand or negative strand and whether a contig sequence represents the positive strand or negative strand is unknown, there are four valid *placements* $\{P_1, P_2, P_3, P_4\}$ for two adjacent contigs (A, B) as shown in Figure 9. The relative positions and directions of the contigs can be determined from the aligned paired-end reads. However, because of sequencing errors and misalignment, the relative direction and position of two contigs can be different using different paired-end reads. In order to determine the correct relative direction and position of contigs, a *scaffold graph* $G = (V, E)$ is constructed over the set of linking regions V to capture all placements of adjacent contigs by the set of edges E. In the graph, *placement weight* is defined as the number of paired-end reads support each placement of two linking regions v_i and v_j in distinct contigs, and each edge $e_{ij} = (v_i, v_j)$ is associated with a quaternion (N_1, N_2, N_3, N_4), where N_i is the weight for placement P_i. Only

placements	contig End (A)	read orientation (A)	contig End (B)	read orientation (B)
P_1: (A+, B+)	3'	+	5'	-
P_2: (A+, B-)	3'	+	3'	+
P_3: (A-, B-)	5'	-	3'	+
P_4: (A-, B+)	5'	-	5'	-

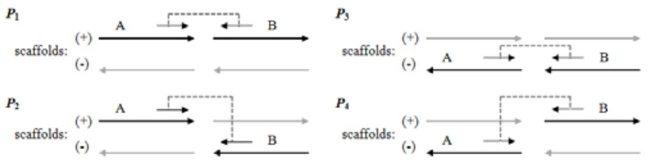

Figure 9. Four placements for two adjacent contigs. Placements depicted at bottom correspond to the ones in top table. Adjacent contigs (bold arrows) are placed based on their aligned read pairs. Grey arrows indicate reverse complements of contigs. Contig orientation ('+'/'-') in top table is the contig orientation in scaffolds.

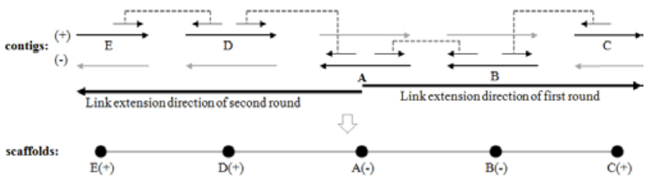
contigs:
(+)
(-)
E
D
A
B
C
Link extension direction of second round
Link extension direction of first round
scaffolds:
E(+)
D(+)
A(-)
B(-)
C(+)

Figure 10. Scheme for contigs linking. The first link round (right) extends contigs by paired-end reads from the starting contig A to the right until no extension are possible, then the second link round (left) is carried out from A to the left in the same way. Scaffold is a linear structure of a set of linked contigs (bottom) that have been ordered and oriented.

uniquely aligned reads are used to construct graph to prevent introducing errors by repeats.

Contigs are linked based on a greedy approach. A contig longer than the linking region size is randomly selected as the initial scaffold to be extended. The extension is performed iteratively by including the neighboring contigs to the right, and once a contig is included in a scaffold, its orientation is assigned according to the placement. Extension is performed iteratively and is terminated on the following conditions: no neighboring contigs, or multiple candidates undifferentiated which one is correct. When the extension is terminated from the 3' end, the 5' end will be extended in a similar fashion (Figure 10). After contigs are linked, their orders and orientations in scaffolds will be determined.

2.2.3 *Overlap between contigs*

To generate final scaffolds, it is necessary to compute the distance between each two adjacent contigs in scaffolds, which may be overlapped or have gaps in between. For overlapped contigs, the overlapped region will be detected and the two contigs should be merged into a single contig. For contigs with gaps in between, the gap size will be computed based on the paired-end reads that link the two contigs.

PERGA first estimates the gap size between adjacent contigs in scaffolds. Given a paired-end read with two ends a and b aligned to different contigs A, B, the gap size g can be estimated by $g = \bar{m} - l_1 - l_2$, where $\bar{m}$ is the mean insert size, l_1 is the distance from 5' end of read a to the gap margin of contig A, l_2 is the distance from 5' end of read b to gap margin of contig B. In practice, there can be multiple read pairs falling into two adjacent contigs, thus the final gap size $d(A, B)$ can be inferred by

$$d(A,B) = \frac{1}{n}\sum_{i=1}^{n} g_i$$

where n is number of read pairs between A and B.

PERGA further checks the inferred gap size. If the inferred gap size is a large positive number, there is probably a gap between contigs with the estimated gap size; and if the gap size is a large negative number, there is probably an overlap. If the gap size is not significant, further check is needed by comparing the prefix and suffix of the two contigs.

When the gap size is a large negative number or insignificant, the two contigs may overlap with certain proportion. Because of sequencing error and mistakes in assembling, the overlapping sequence may not be exactly the same. PERGA performs the pair-wise sequence alignment to capture the overlaps. If an overlap is larger than 3 and is agreed with the estimated gap size, this pair of contigs will be recorded as overlapping contigs and merged into a single contig; otherwise, there will be a gap between them.

2.2.4 *Gap filling*

After estimating gap size between adjacent contigs, it is necessary to fill the gap regions for better continuity by local assembly using paired-end reads with one end aligned onto contigs and the other end aligned in gap regions. Most of the sequences in the gap regions are repetitive sequences, thus gap filling can be used to resolve such repeats. As the sequences adjacent to gap regions have been recognized, repetitive sequences in gap regions can be easily reconstructed by local assembly which is based on the algorithm of assembly of reads for PERGA using paired-ends.

Consensus sequences are generated from contigs in scaffolds considering their overlaps and gaps. If adjacent contigs are overlapped, then they will be merged; and if contigs are gapped, the gap region between these contigs will be filled with ambiguous bases ('N').

3. RESULTS

3.1 Datasets

We evaluated the performance of PERGA on both simulated *E.coli* datasets and the real dataset (details are shown in Table 1). The simulated Illumina paired-ends datasets were generated using GemSIM [28] with various coverages 50x, 60x, 100x (can be downloaded from https://github.com/hitbio/PERGA), and the real Illumina paired-end reads data were downloaded from http://bix.ucsd.edu/projects/singlecell/nbt_data.html, with standard genomic DNA prepared from culture, with coverage around 600x. We evaluated the performance of PERGA on resolving branches using SVM prediction model and look ahead technology. We also compared the performance of PERGA in assembling with other leading state-of-the-art assemblers, including IDBA-UD [26] (v1.0.9), ABySS [24] (v1.3.2), Velvet [21] (v1.2.01), and also including overlap-based assemblers SGA [18] (v0.9.20) and CABOG [16] (v7.0).

To evaluate the performances of each assembler, we used the number of correctly assembled contigs, length of N50, length of the longest contig to evaluate their length metrics, and we used BLASTN [29] to align the contigs and scaffolds to reference to evaluate their accuracy by using reference covered ratio, number and lengths of mis-assemblies. If a contig (or scaffold) entirely matches with the reference with similarity <95%, it is considered as a mis-assembled contigs. As Velvet does not produce contigs for paired-ends data, we split the scaffolds at the positions of poly-N to get the contigs for comparison. Note that repeats from different genomic regions will be collapsed into a single copy which can be aligned to more than one location or in disjoint locations when using BLAST, and we also deem that all those genomic locations are covered by these repeats.

The experiments for the simulated reads data were carried out on a 64-bit Linux machine with an Intel(R) Core-2 CPU 2.53-GHz supplied with 3 GB memory except the experiments for CABOG. The experiments for CABOG and the real reads data were carried out on an Intel(R) Xeon(R) Core-8 CPU 2.00-GHz server supplied with 24 GB memory.

Table 1. Datasets of *E.coli* for assemblies

Datasets	**D1**	**D2**	**D3**	**D4**
Data type	simulated	simulated	simulated	real
Read length	100 bp	100 bp	100 bp	100 bp
# Reads (million)	2× 1.16	2× 1.4	2× 2.3	2× 14.2
Cov. depth	50×	60×	100×	600×
Insert size (bp)	370 ± 56	370 ± 58	370 ± 59	215 ± 11

3.2 Greedy-like prediction model

The performance of our greedy-like SVM prediction model was assessed by counting the numbers of correctly and incorrectly predicted extensions and stops for all branches during the assembling step. The statistical results on all datasets are shown in Table 2. By constructing the decision models using machine learning approach, PERGA can determine the correct extension in 99.7% of cases for the simulated raw reads data D1~D3. And PERGA also determines the stop cases that both extensions are likely to be correct. PERGA can produce only a few incorrect extensions and incorrect stops (less than 0.1%).

Table 2. Statistical results for greedy-like prediction model

Datasets	Corr. exts.	Incorr. exts.	Corr. stops	Incorr. stops
D1	70299 (99.70%)	60 (0.09%)	123 (0.17%)	26 (0.04%)
D2	84829 (99.74%)	46 (0.05%)	148 (0.18%)	25 (0.03%)
D3	136309 (99.82%)	48 (0.04%)	169 (0.12%)	27 (0.02%)

3.3 Look ahead technology

Although the performance of SVM prediction model is good, PERGA still has some incorrect extensions and stops. These incorrect predictions can be resolved by the look ahead technology. Table 3 shows the number of correct and incorrect navigations for branches when applying this technology to generate contigs based on the datasets D1~D3. Most of the branches can be correctly resolved in very high probability (>99.77%) with very low error rate (<0.23%), which is the primary reason that PERGA can generate long and accurate contigs. According to Table 2 and Table 3, about only 10% of the branches (D1~D3) are adjusted by this technology. As look ahead technology is very effective, the mis-prediction branches can be easily handled by this technology.

Table 3. Statistical results for look ahead technology

Datasets	Correct navi.	Incorrect navi.	Total
D1	6039 (99.82%)	11 (0.18%)	6050
D2	7074 (99.77%)	16 (0.23%)	7090
D3	8408 (99.77%)	19 (0.23%)	8427

Table 4. Assemblies for *E.coli* simulated short reads data (D1, 50×)

	k/*Overlap*	Contigs					Scaffolds					Time	Mem.
		#contigs	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	#scaffolds	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	(min)	(GB)
PERGA	*O*≥25	97	132.7	327.0	100.0	0 / 0	82	176.0	327.0	100.0	0 / 0	3	1.0
IDBA-UD	*k*=20-100	153	112.6	327.1	99.98	2 / 559	119	148.5	327.1	99.98	1 / 321	11	0.6
ABySS	*k*=45	110	119.2	270.3	99.90	0 / 0	103	119.2	270.3	99.42	1 / 3617	9	1.0
Velvet	*k*=45	177	108.1	326.9	99.76	7 / 6658	153	148.3	326.9	99.89	1 / 1596	3	0.9
SGA	*O*≥31	3143	22.6	138.3	99.06	1 / 105	2778	88.4	269.7	99.06	2 / 4225	43	0.6
CABOG	default	134	83.1	201.6	99.03	1 / 2638	98	88.5	204.0	99.03	1 / 2638	77	2.6

Table 5. Assemblies for *E.coli* simulated short reads data (D2, 60×)

	k/*Overlap*	Contigs					Scaffolds					Time	Mem.
		#contigs	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	#scaffolds	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	(min)	(GB)
PERGA	*O*≥25	96	133.3	327.8	100.0	0 / 0	84	174.1	327.8	100.0	0 / 0	3	1.1
IDBA-UD	*k*=20-100	153	124.6	327.1	99.99	0 / 0	120	173.9	327.1	99.97	0 / 0	13	0.6
ABySS	*k*=45	104	119.2	328.1	99.92	0 / 0	93	135.0	328.1	99.56	1 / 25k	10	1.1
Velvet	*k*=45	176	125.2	326.8	99.79	6 / 4451	155	148.5	326.8	99.87	0 / 0	5	1.0
SGA	*O*≥31	3564	21.6	138.2	98.89	1 / 105	3163	87.3	270.0	98.91	2 / 597	50	0.6
CABOG	default	155	68.4	180.3	98.72	0 / 0	112	77.1	180.3	98.64	1 / 4996	98	2.6

Table 6. Assemblies for *E.coli* simulated short reads data (D3, 100×)

	k/*Overlap*	Contigs					Scaffolds					Time	Mem.
		#contigs	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	#scaffolds	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	(min)	(GB)
PERGA	*O*≥25	111	125.1	327.2	100.0	0 / 0	89	149.7	327.2	100.0	0 / 0	5	1.4
IDBA-UD	*k*=20-100	153	124.6	327.1	99.99	0 / 0	113	148.6	327.1	99.96	2 / 1723	21	0.7
ABySS	*k*=45	106	126.2	328.1	99.90	1 / 524	95	135.0	328.1	90.89	4 / 206k	16	1.7
Velvet	*k*=45	178	117.5	327.0	99.75	9 / 7347	159	148.5	327.0	100.0	0 / 0	7	1.4
SGA	*O*≥31	4387	18.2	121.0	98.73	3 / 376	3657	95.0	269.7	98.81	8 / 9151	103	0.6
CABOG	default	196	37.3	180.3	93.63	1 / 61k	135	56.7	189.0	93.56	1 / 65k	209	2.6

Table 7. Assemblies for *E. coli* real short reads data (D4, 600×)

	k/*Overlap*	Contigs					Scaffolds					Time	Mem.
		#contigs	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	#scaffolds	N50 (Kbp)	Max. (Kbp)	Cov. (%)	Misass. (#/sumLen)	(min)	(GB)
PERGA	*O*≥25	114	133.4	266.0	100.0	2 / 383	98	154.6	284.7	100.0	2 / 383	20	4.9
IDBA-UD	*k*=20-100	144	106.8	236.6	99.93	1 / 2105	107	148.5	284.4	99.98	0 / 0	31	2.0
ABySS	*k*=45	156	96.0	210.8	93.61	4 / 293k	141	113.4	236.4	91.45	5 / 372k	64	0.3
Velvet	*k*=45	215	82.8	177.8	95.25	11 / 212k	182	95.5	209.4	86.21	5 / 633k	33	5.1
SGA	*O*≥31	5497	16.2	73.1	98.69	42 / 4962	5299	17.2	73.1	99.11	44 / 5943	357	5.9

3.4 Simulated data results

Table 4 shows the performances of PERGA as well as other assemblers on the 50x simulated paired-end reads data D1. PERGA generated the longest contigs in N50 measures, highest reference coverage and the most accurate result with no mis-assemblies. PERGA and Velvet were the fastest assemblers among four assemblers with moderate memory usage and were about 3 times faster than IDBA-UD and ABySS, since SGA uses the FM-index to compute the read overlap, and CABOG computes the read overlaps between each other, thus they cost more time while assembling (43 minutes and 77 minutes). PERGA generated the largest N50 (132 kbp and 176 kbp). This is because that PERGA handles branches for extension much more carefully, it utilizes a greedy-like prediction SVM models which contains much branch information to give much better extensions, and PERGA distinguishes sequencing errors and repeats for branches using the look ahead technology to decide the correct extensions. Thus, PERGA can provide fewer but longer contigs than the others without producing erroneous contigs.

When the number of reads in the dataset increases, the running times of all the other assemblers increase (Table 5 for data D2). However, the running time of PERGA does not increase significantly and is still the fastest assembler. The contigs and scaffolds produced by PERGA have the largest N50 and coverage with no mis-assemblies. Because of the increase in sequencing depth, IDBA-UD and Velvet performed better than on D2 with contig N50 increased from 112.6kbp and 108.1 kbp to 124.6kbp and 125.2 kbp respectively with less assembly errors. PERGA and IDBA-UD had the largest scaffold sizes (174.1 kbp and 173.9 kbp respectively), while the results of others assemblers were much shorter (only around 140kbp). Compared with other assemblers, SGA and CABOG produced shorter contigs and scaffolds with longer running time. The coverages of all assemblers on D1 and D2 are much the same and they do not differ much between contigs construction and scaffolds production.

For the simulated 100x data set D3, since the sequencing depth is high, all assemblers generated similar numbers of contigs with similar coverages in both contigs and scaffolds except ABySS which dropped from 99.90% to 90.89% because of the mis-assembled 4 scaffolds (206kbp). CABOG generated 1 mis-assembled contig (61kbp) and 1 mis-assembled scaffold (65kbp), thus its genome coverage dropped to 93.5%. From the experiments on D1~D3, it can be observed that CABOG may be not suitable for high coverage data since its contigs (scaffolds) sizes decreased with the increasing coverage depth, and it can also seen that the overlap-based assemblers (SGA and CABOG) is not suitable for high coverage data. PERGA also was the fastest assembler and produced the most accurate results while others all produced several mis-assembled contigs (scaffolds). In all experiments on simulated data, PERGA did not produce any mis-assembled contigs and scaffolds while the other assemblers mis-assembled some reads in some datasets.

3.5 Real data result

We further used the downloaded *E.coli* dataset D4 with coverage ~600x to highlight the performance of PERGA on high coverage data, and compared its performances with other assemblers. CABOG could not be run on D4 may be due to the high coverage depth, thus it was excluded from the comparison. Before assembling, paired-end reads data were corrected using Quake [30], and the results are shown in Table 7.

The overall performance of all other assemblers dropped while the performance of PERGA still had similar performance in simulated data. It may suggest that the SVM model used by PERGA can capture the properties in real datasets. PERGA was the fastest assembler and generated the longest contigs (scaffolds) (133.4kbp and 154.6kbp) with the highest coverage (100%), while others assemblers had much lower N50 except scaffolds of IDBA-UD (148.5kbp). Since PERGA generated very long and accurate contigs, the scaffolds produced by PERGA had the largest N50 and highest coverage even though it did not connect many contigs in scaffolding.

After scaffolding, ABySS and Velvet produced longer scaffolds with lower coverage, while PERGA and IDBA-UD did not have coverage difference between contigs and scaffolds as they produced accurate assemblies. ABySS and Velvet both had >200kbp mis-assembled contigs and >300kbp mis-assembled scaffolds, thus their contig coverage dropped dramatically from 96% to 86%, and SGA generated accurate contigs and scaffolds, however, their N50 sizes are very small (16.2 kbp and 17.2 kbp), and it used more time than others. This shows that ABySS, Velvet and SGA might not be suitable for high coverage sequencing data. They can have good performance on low coverage data but might not be good on high coverage data.

4. CONCLUSIONS

In this article, we present PERGA, a novel *de novo* sequence reads assembler, which can generate large and accurate assemblies using the greedy-like prediction strategy to handle branches and errors to give much better extensions. By using look ahead technology to distinguish sequencing errors and repeats more accurately, PERGA makes use of read overlaps represented in the form of overlapping *k*-mers to deal with short repeats. Moreover, instead of using single-end reads to construct contigs, PERGA uses paired-end reads in the first step and gives different priority to different read overlap size thresholds ranging from $O_{\max}$ to $O_{\min}$ to resolve the gap and branch problem. Experiments showed that PERGA could generate very long and accurate contigs and scaffolds with fewer mis-assembly errors both for simulated reads data and real data sets for both low and high coverage datasets than the existing methods.

5. ACKNOWLEDGMENTS

This work was supported in part by National Nature Science Foundation of China (grant 61173085) and HKGRF (HKU 7111/12E).

6. REFERENCES

[1] Shendure, J., et al., *Accurate multiplex polony sequencing of an evolved bacterial genome.* Science, 2005. 309(5741): p. 1728-1732.

[2] Margulies, M., et al., *Genome sequencing in microfabricated high-density picolitre reactors.* Nature, 2005. 437(7057): p. 376-80.

[3] Li, R.Q., et al., *The sequence and de novo assembly of the giant panda genome.* Nature, 2010. 463(7279): p. 311-317.

[4] Bentley, D.R., et al., *Accurate whole human genome sequencing using reversible terminator chemistry.* Nature, 2008. 456(7218): p. 53-59.

[5] Li, H., *Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly.* Bioinformatics, 2012. 28(14): p. 1838-44.

[6] Blanca, J.M., et al., *ngs_backbone: a pipeline for read cleaning, mapping and SNP calling using Next Generation Sequence.* BMC Genomics, 2011. 12.

[7] Schatz, M.C., A.L. Delcher, and S.L. Salzberg, *Assembly of large genomes using second-generation sequencing.* Genome Res., 2010. 20(9): p. 1165-1173.

[8] Surget-Groba, Y. and J.I. Montoya-Burgos, *Optimization of de novo transcriptome assembly from next-generation sequencing data.* Genome Res., 2010. 20(10): p. 1432-1440.

[9] Treangen, T.J. and S.L. Salzberg, *Repetitive DNA and next-generation sequencing: computational challenges and solutions.* Nat Rev Genet, 2012. 13(1): p. 36-46.

[10] Flicek, P. and E. Birney, *Sense from sequence reads: methods for alignment and assembly.* Nature Meth., 2009. 6(11 Suppl): p. S6-S12.

[11] Shendure, J. and H. Ji, *Next-generation DNA sequencing.* Nature Biotech., 2008. 26(10): p. 1135-45.

[12] Warren, R.L., et al., *Assembling millions of short DNA sequences using SSAKE.* Bioinformatics, 2007. 23(4): p. 500-1.

[13] Jeck, W.R., et al., *Extending assembly of short DNA sequences to handle error.* Bioinformatics, 2007. 23(21): p. 2942-4.

[14] Dohm, J.C., et al., *SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing.* Genome Res., 2007. 17(11): p. 1697-706.

[15] Hernandez, D., et al., *De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer.* Genome Res., 2008. 18(5): p. 802-9.

[16] Miller, J.R., et al., *Aggressive assembly of pyrosequencing reads with mates.* Bioinformatics, 2008. 24(24): p. 2818-2824.

[17] Burrows, M. and D.J. Wheeler, *A block-sorting lossless data compression algorithm.* Technical Report, 1994. 124: p. Palo Alto, CA, Digital Equipment Corporation.

[18] Simpson, J.T. and R. Durbin, *Efficient de novo assembly of large genomes using compressed data structures.* Genome Res., 2012. 22(3): p. 549-556.

[19] Ferragina, P. and G. Manzini. *Opportunistic Data Structures with Applications.* in *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000)*. 2000.

[20] Pevzner, P.A., H. Tang, and M.S. Waterman, *An Eulerian path approach to DNA fragment assembly.* Proc. Natl. Acad. Sci., 2001. 98(17): p. 9748-53.

[21] Zerbino, D.R. and E. Birney, *Velvet: algorithms for de novo short read assembly using de Bruijn graphs.* Genome Res., 2008. 18(5): p. 821-9.

[22] Chaisson, M.J. and P.A. Pevzner, *Short read fragment assembly of bacterial genomes.* Genome Res., 2008. 18(2): p. 324-330.

[23] Butler, J., et al., *ALLPATHS: de novo assembly of whole-genome shotgun microreads.* Genome Res., 2008. 18(5): p. 810-20.

[24] Simpson, J.T., et al., *ABySS: A parallel assembler for short read sequence data.* Genome Res., 2009. 19(6): p. 1117-1123.

[25] Peng, Y., et al., *IDBA - A Practical Iterative de Bruijn Graph De Novo Assembler.* Research in Computational Molecular Biology, Proceedings, 2010. 6044: p. 426-440.

[26] Peng, Y., et al., *IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth.* Bioinformatics, 2012. 28(11): p. 1420-8.

[27] Li, R., et al., *De novo assembly of human genomes with massively parallel short read sequencing.* Genome Res., 2009. 20(2): p. 265-272.

[28] McElroy, K.E., F. Luciani, and T. Thomas, *GemSIM: general, error-model based simulator of next-generation sequencing data.* BMC Genomics, 2012. 13: p. 74.

[29] Altschul, S.F., et al., *Basic local alignment search tool.* J. Mol. Biol., 1990. 215(3): p. 403-10.

[30] Kelley, D.R., M.C. Schatz, and S.L. Salzberg, *Quake: quality-aware detection and correction of sequencing errors.* Genome Biol., 2010. 11(11): p. -.