Bioinformatics, 31(6), 2015, 825–833
doi: 10.1093/bioinformatics/btu762
Advance Access Publication Date: 17 November 2014
Original Paper

OXFORD

Sequence analysis

EPGA: *de novo* assembly using the distributions of reads and insert size

Junwei Luo[1,2], Jianxin Wang[1,*], Zhen Zhang[1], Fang-Xiang Wu[3], Min Li[1] and Yi Pan[4]

[1]School of Information Science and Engineering, Central South University, ChangSha 410083, China, [2]College of Computer Science and Technology, Henan Polytechnic University, JiaoZuo, 454000, China, [3]Division of Biomedical Engineering, University of Saskatchewan, Saskatchewan S7N 5A9, Canada and [4]Department of Computer Science, Georgia State University, Atlanta, GA 30302, USA

*To whom correspondence should be addressed.
Associate Editor: John Hancock

Received on September 15, 2014; revised on October 23, 2014; accepted on November 11, 2014

Abstract

Motivation: In genome assembly, the primary issue is how to determine upstream and downstream sequence regions of sequence seeds for constructing long contigs or scaffolds. When extending one sequence seed, repetitive regions in the genome always cause multiple feasible extension candidates which increase the difficulty of genome assembly. The universally accepted solution is choosing one based on read overlaps and paired-end (mate-pair) reads. However, this solution faces difficulties with regard to some complex repetitive regions. In addition, sequencing errors may produce false repetitive regions and uneven sequencing depth leads some sequence regions to have too few or too many reads. All the aforementioned problems prohibit existing assemblers from getting satisfactory assembly results.

Results: In this article, we develop an algorithm, called extract paths for genome assembly (EPGA), which extracts paths from De Bruijn graph for genome assembly. EPGA uses a new score function to evaluate extension candidates based on the distributions of reads and insert size. The distribution of reads can solve problems caused by sequencing errors and short repetitive regions. Through assessing the variation of the distribution of insert size, EPGA can solve problems introduced by some complex repetitive regions. For solving uneven sequencing depth, EPGA uses relative mapping to evaluate extension candidates. On real datasets, we compare the performance of EPGA and other popular assemblers. The experimental results demonstrate that EPGA can effectively obtain longer and more accurate contigs and scaffolds.

Availability and implementation: EPGA is publicly available for download at https://github.com/bioinfomaticsCSU/EPGA.

Contact: jxwang@csu.edu.cn

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

1 Introduction

Knowledge of genome sequences has become indispensable in numerous applied fields such as diagnostics, biotechnology, forensic biology and systems biology. The next generation sequencing (NGS) technologies, also known as high-throughput sequencing, including Illumina, 454 Life Sciences and SOLiD, can provide short sequence fragments named reads, much more quickly and cheaply than previously used Sanger sequencing, but at the sacrifice of read length (Alkan *et al.*, 2011). A number of genome assemblers have been proposed using a great deal of short reads produced by NGS

technologies for reconstructing complete genome sequence (He *et al.*, 2013). The primary issue of these assemblers is how to determine upstream and downstream sequence regions of sequence seeds for constructing contigs or scaffolds. As the length of reads decreases, the number of repetitive regions in one genome will dramatically increase. When extending one seed sequence, repetitive regions can cause multiple feasible candidate regions which increase the difficulty of genome assembly. So, the repetitive regions especially longer than the read length in one genome become one of the most challenges in genome assembly (Chaisson *et al.*, 2009).

The earlier assemblers generally employ read overlaps to extend one sequence seed. When there are multiple feasible extension candidates caused by repetitive regions, the assemblers usually select one with the maximum overlap or the highest consensus to extend (Dohm *et al.*, 2007; Jeck *et al.*, 2007; Warren *et al.*, 2007). The length of read overlap is important to identify extension candidates and choose the correct one among them. Because sequencing errors often result in erroneous read overlaps and low-depth regions miss some read overlaps, so read overlaps usually compromise between continuity and error rate (Ariyaratne and Sung, 2011). De Bruijn graph is a useful data structure which can store read overlaps (Pevzner *et al.*, 2001). There are some improved De Bruijn graphs used for assembly (Medvedev *et al.*, 2011; Iqbal *et al.*, 2012; Pham *et al.*, 2013). However, the information about read overlaps has no effect on repetitive regions longer than the read length. Most recent assemblers make use of paired-end (mate-pair) reads which can be produced by NGS technologies for resolving repetitive region problems.

One paired-end read is referred to a pair of short reads sequenced from two ends of one long sequence fragment and the sequence fragment length (the distance between paired-end reads) is usually called *insert size* which is a random variable with mean μ_{is} and SD δ_{is}. The *insert size* is frequently assumed to have a normal distribution $N(\mu_{is}, \delta_{is})$. For every paired-end read, its two reads are called the mates of each other.

In recent years, there have been numerous assemblers presented for biologists (Bankevich *et al.*, 2012; Chitsaz *et al.*, 2011; Ribeiro *et al.*, 2012). Velvet (Zerbino *et al.*, 2008, 2009) uses paired-end reads to mark nodes, and finds a correct path through marked nodes to connect two long nodes based on De Bruijn graph. However, some nodes shorter than the read length cannot be marked, so it tends to contain more errors at short repetitive regions. Abyss (Simpson *et al.*, 2009) adopts a method similar to Velvet's. When extending one sequence seed or filling one gap region, PE-Assembly (Ariyaratne and Sung, 2011) tries to identify feasible extensions from local read sets produced from paired-end reads. IDBA (Peng *et al.*, 2010) and IDBA-UD (Peng *et al.*, 2012) iterately change *k-mer* (*k* consecutive bases in one read) length and uses paired-end reads to eliminate branches in De Bruijn graph. Telescoper (Maayan *et al.*, 2012) first constructs read-overlap graph based on reads whose mate reads can map to sequence seeds, and it develops a statistical framework using penalty function to choose paths for constructing contigs. SOAPDenovo (Li *et al.*, 2010) and SOAPDenovo2 (Luo *et al.*, 2012) directly use nodes in De Bruijn graph as contigs which are usually short. ALLPATHS-LG (Gnerre *et al.*, 2011) has specific requirements in read length and *insert size*, which differ from common read libraries.

In the following part, let r denote the read length and s denote a sequence. $s[i]$ is the i-th base in s. $s[i, j]$ is the sub-region of s from i- to j-th base. $L(s)$ is the length of s and $R(s)$ is the set of reads in s. RE(s) is the set of reads in $R(s)$ which exist in the read library. RML(s) is the set of reads in RE(s) which have left mate reads and MRL(s) is one set which includes the corresponding left mate reads. RMLM(s_i, s_j) is the set of reads in RML(s_j) whose one left mate read can be mapped to s_i and the distance between the mapped paired-end reads must be in the interval $[\mu_{is} - 3 * \delta_{is}, \mu_{is} + 3 * \delta_{is}]$, $DL(s_i, s_j)$ is the set including the corresponding distances. RMR(s), MRR(s), RMRM(s_i, s_j) and DR(s_i, s_j) all can be obtained from right mate reads of the sequences. $|T|$ is cardinality of one set T.

The information contained in paired-end reads can facilitate genome assembly (Wetzel *et al.*, 2011). Because paired-end reads can span repetitive regions shorter than *insert size*, for one sequence seed s_s and one downstream extension candidate s, we can estimate the correctness of s through its MRL(s). When the reads in MRL(s) can be mapped to s_s, we consider s as the correct one, as illustrated in Figure 1.

Although paired-end reads are widely applied to resolve problems caused by repetitive regions in genome assembly, the performance of most assemblers is not satisfactory. There are three major problems which prevent most assemblers from identifying correct extension candidates by using paired-end reads:

1. Adjacent repetitive regions and paired repetitive regions: For two same regions B, if the separation distance between the two regions in genome is small, we say that it is adjacent repetitive region, shown in Figure 2a. For two regions (A, C), if the length between A and C is closed to *insert size* and the two regions appear in genome repeatedly, we say that it is paired repetitive region, shown in Figure 2b. The upstream or downstream regions of adjacent and paired repetitive regions are difficult to determine, because the most reads in *MRL* or *MRR* of extension candidates can be mapped to the sequence seed.
2. Sequencing errors: sequencing errors always bring about incorrect reads which probably lead to false repetitive regions and false extension candidates.
3. Uneven sequencing depths: sequencing depth of one sequence region depends on the average number of reads in the read library which can be mapped to the region. Because sequencing depths of different regions in a genome are universally highly uneven (Peng *et al.*, 2012), there are no or fewer reads which can be mapped to low-depth regions, but high-depth regions have too many mapped reads. Uneven sequencing depths aggravate the problems caused by (1) and (2).

In Figure 2a, for the sequence seed AB, which is merged by A and B, and the extension candidate C, the distances in $DL(AB, C)$ should follow the distribution $N(\mu_{is}, \delta_{is})$. D is the correct extension candidate of $ABCB$ and the difference between the length of AB and the length of $ABCB$ is $L(C) + L(B)$. When D is regarded as the extension candidate of AB, the distances in $DL(AB, D)$ will follow $N(\mu_{is} - L(B) - L(C), \delta_{is})$. In Figure 2b, the distances in $DL(ABC, D)$ will follow $N(\mu_{is}, \delta_{is})$. Because H is the correct extension

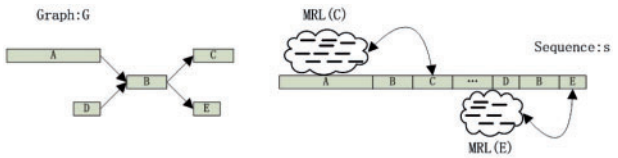

Fig. 1. Paired-end reads using in extending sequence seed. The graph G is constructed based on sequence s ($ABC \ldots DBE$). When extending downstream region of AB, it has two extension candidates C and E. If no sequencing errors, reads in MRL(C) can be mapped to AB, but reads in MRL(E) can not be mapped to AB. So, C is the correct extension candidate

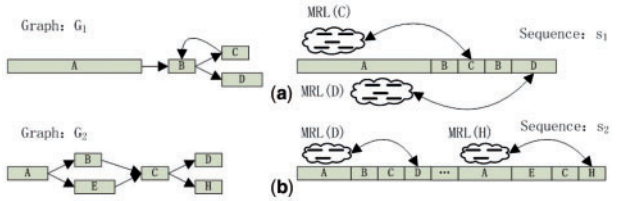

Fig. 2. Adjacent repetitive regions and paired repetitive regions. **(a)** The graph G_1 is constructed based on sequence s_1 (*ABCBD*). *B* is one adjacent repetitive region and the length of *C* is short. If extending downstream region of *AB*, there are two extension candidates *C* and *D*. Most reads in MRL(*C*) and MRL(*D*) can be mapped to *AB*. In this case, it is difficult to choose the correct candidate. **(b)** The graph G_2 is constructed based on sequence s_2 (*ABCD*...*ABCH*). (*A, C*) is one paired repetitive region. When extending downstream of *ABC*, there are two extension candidates *D* and *H*. Most reads in MRL(*D*) and MRL(*H*) can be mapped to *ABC*. In this case, it is also a problem to determine the correct candidate

candidate of AEC and the difference between the length of ABC and the length of AEC is $L(E) - L(B)$, the distances in $DL(ABC, H)$ will follow $N(\mu_{is} - L(E) + L(B), \delta_{is})$. It is clear that we can identify which extension candidate is correct by assessing whether the distances in DL follow $N(\mu_{\text{is}}, \delta_{\text{is}})$. To our knowledge, PE-Assembly is the only tool which considers the variation of the distribution of *insert size* in filling gap step.

For one fixed length sequence s, $R(s)$ includes $L(s) - r + 1$ reads. Due to sequencing depth, some reads in $R(s)$ probably do not appear in the read library. $|\text{RE}(s)|$ is usually smaller than $|R(s)|$. Every read is sequenced randomly, $|RE|$ is a random variable whose distribution is called the distribution of reads. $|\text{RE}(s)|$ is a particular value of the random variable which should be within a reasonable range. The probability P is one read can be sequenced, if P is the same for every read in genome reference, the distribution of reads approximately follows binomial distribution. The reasonable range can be determined by binomial distribution. If $|\text{RE}(s)|$ goes beyond this range, we consider s including false bases.

If sequencing depth is even and sequencing coverage is large enough without sequencing errors, genome sequence is corresponding to one path contained in De Bruijn graph. However, due to the uneven sequencing depth and sequencing errors, the path is inevitably segmented into many noncontiguous sub paths. So, the target of de novo assembly becomes to analyze De Bruijn graph for seeking out accurate sub paths corresponding to sub regions of genome. We develop a novel assembler, called EPGA (extract paths for genome assembly), to improve genome assembly (see the flowchart of EPGA in Supplementary Figure S1). EPGA selects some nodes in De Bruijn graph as sequence seeds. The precursor nodes and successor nodes of sequence seeds are treated as its upstream and downstream extension candidates. EPGA processes each extension candidate to form an evaluating region which is evaluated based on a new score function. Using the new score function, EPGA iteratively extends sequence seeds on both sides to extract long paths which are contigs. For one sequence seed s_s, its downstream extension candidate s and evaluating region s_e, EPGA uses the following three new strategies which make novel use of paired-end reads:

To resolve problem (1), EPGA adopts the coefficient of determination (CD) to evaluate whether the distances in $DL(s_s, s_e)$ follow $N(\mu_{\text{is}}, \delta_{\text{is}})$. The CD provides a measure of how well the distances are replicated by $N(\mu_{\text{is}}, \delta_{\text{is}})$, as the proportion of total variation of the distances is explained by $N(\mu_{\text{is}}, \delta_{\text{is}})$.

To resolve problem (2), based on the distribution of reads, EPGA determines a reasonable range of $|\text{RE}(s_e)|$ to identify whether s is a false extension candidate. The probability that $|RE(s_e)|$ is within the range should be large enough.

To resolve problem (3), EPGA designs a new index named relative mapping (RM) to evaluate extension candidates. The relative mapping is the ratio of $\text{RMLM}(s_s, s_e)$ to $\text{RML}(s_e)$. So, relative mapping can guarantee that correct extension candidates are given high score no matter sequencing depth is.

In EPGA, we mainly incorporate two new ideas for genome assembly: (i) we consider the distribution of reads to identify whether one extension candidate includes sequencing errors, rather than only using *k-mer* frequency; (ii) based on the distribution of *insert size*, we develop a new score function to overcome complex repetitive regions.

The performance of EPGA and other popular assemblers are compared on real datasets. The results demonstrate that EPGA can get more continuous and correct genome sequences.

2 Methods

2.1 De Bruijn graph

In original De Bruijn graph, each node represents a *k-mer* (k consecutive bases in one read), there will be a directed arc between two nodes if they have overlap with $k-1$ bases and continuously emerge in one read. A read with the length of r can be divided into $r - k + 1$ overlapping *k-mers*. The value of k is important for constructing De Bruijn graph. Larger k will remove some short repetitive regions while reducing the number of nodes in De Bruijn graph, but will give rise to more unconnected sub-graphs which means that the number of gap regions increases. The small value of k will reduce some gap regions while increase the connectivity of De Bruijn graph, but will add more nodes and increase short repetitive regions. Therefore, the value of k cannot be too large or too small. For balancing the conflict, it is generally $\cong r * 2/3$. To ensure that each *k-mer* cannot be its own reverse complement, k must be an odd number.

Many existing assemblers perform error correction/detection step prior to the assembly. Although it is generally effective in detecting and fixing random sequencing errors, it may result in overcorrecting the reads coming from low-depth regions. EPGA takes a simple strategy for avoiding overcorrecting. If the frequency of one *k-mer* is over one, the *k-mer* will be considered in constructing De Bruijn graph; otherwise, the *k-mer* will be thought of including erroneous bases and will be removed.

After obtaining the original De Bruijn graph, it is usually possible to optimize the graph based on its topological structure. First, if nodes in one path have only one outgoing arc except end node and only one ingoing arc except start node, the path will be named simple path and can be merged into one node. After merging simple paths, some tips (nodes whose out-degree plus in-degree is one) shorter than $2 * k$ can be removed. Tips are usually produced by erroneous bases in reads and gap regions. Third, there are some simple cycles (two nodes direct each other) which can be simplified to one path (see Supplementary Figure S2). Some assemblers usually merge one bubble to one path which can simplify De Bruijn graph. However, some bubbles may be caused by single-nucleotide polymorphism or similar sequences, so we do not remove bubbles. After these optimization, one final De Bruijn graph has been constructed and will be used for following steps.

2.2 Sequence seeds and extension candidates

Sequence seeds are used as the start points to extend their upstream and downstream regions for constructing longer contigs.

EPGA selects nodes longer than *insert size* as sequence seeds in De Bruijn graph. The downstream extension candidates are nodes that are directed by the last node of sequence seed. The nodes that direct to the first node of sequence seed are upstream extension candidates. The length of extension candidates should be longer than the minimum length (len_{min}) which is $2 * k$ as default, so that an extension candidate includes enough reads and mate reads to evaluate themselves. When the length of one extension candidate is shorter than len_{min}, EPGA will take the extension candidate as origin node and adopt deep first search (DFS) to get paths longer than len_{min} based on De Bruijn graph. When one path is longer than len_{min} or search deep is greater than the max deep threshold, this search will stop. The DFS usually produces many paths referred to secondary extension candidates. We will evaluate each secondary extension candidates and select the top score as the final score for the extension candidate.

2.3 Evaluating extension candidates

In this step, we develop one new score function based on the distributions of reads and insert size for identifying correct extension candidate. By using the new score function, the ambiguities caused by adjacent and paired repetitive regions can be resolved in the process of extending sequence seed.

For a given sequence seed (s_s)and its one downstream extension candidate (s), they have overlap with $k-1$ bases. The last $r-1$ bases of s_s are merged to the extension candidate. Then the first $r-k+len_{min}$ bases of the extension candidate is considered as its evaluating region s_e. An extension candidate corresponds to an evaluating region. Then, EPGA can get $RE(s_e)$, $RML(s_e)$, RMLM and $DL(s_s, s_e)$. There is an example in Figure 3.

EPGA uses RM in Equation (1) to evaluate the correctness of extension candidates. RM is always in [0, 1]. When a correct extension candidate is in high-depth region or low-depth region, its $|RML(s_e)|$ and $|RMLM(s_s, s_e)|$ will simultaneously increase or decrease. So, RM can guarantee that correct extension candidates are given high score no matter sequencing depth is. Therefore, as long as RM of one extension candidate is high, EPGA considers it as correct.

$$RM(s_s, s_e) = \frac{|RMLM(s_s, s_e)|}{|RML(s_e)|} \tag{1}$$

Moreover, the distance set $DL(s_s, s_e)$ is employed to overcome problems introduced by adjacent and paired repetitive regions. EPGA uses CD in Equation (2) to measure whether distances in $DL(s_s, s_e)$ fits $N(\mu_{is}, \delta_{is})$. For computing CD, EPGA partitions some intervals in $(-\infty, +\infty)$. If EPGA sets too many intervals, $DL(s_s, s_e)$ should include more distances and s_e must be one high-depth region. If EPGA sets too few intervals, CD cannot correctly evaluate fitness between two distributions. For balancing sequencing depth and accuracy, EPGA sets six intervals: $(-\infty, \mu_{is} - 2 * \delta_{is})$, $[\mu_{is} - 2 * \delta_{is}, \mu_{is} - \delta_{is})$, $[\mu_{is} - \delta_{is}, \mu_{is})$, $[\mu_{is}, \mu_{is} + \delta_{is})$, $[\mu_{is} + \delta_{is}, \mu_{is} + 2 * \delta_{is})$ and $[\mu_{is} + 2 * \delta_{is}, +\infty)$. In Equation (2), x is the size of the distance set. n is six and x_i is the number of distances within i-th interval. P_i is the probability of i-th interval of $N(\mu_{is}, \delta_{is})$. y_i is $x * P_i$.

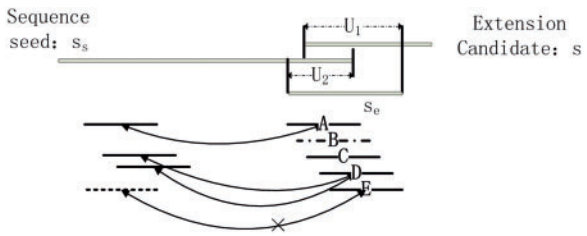

Fig. 3. Illustration of evaluating an extension candidate. s_e is evaluating region of s and merged by U_1 and U_2. U_1 is the first l_{min} bases of s. U_2 is the last $r-1$ base of s_s. U_1 and U_2 have overlap with $k-1$ bases. *A, B, C, D, E* are reads coming from s_e. *A, C, D, E* exist in read library, *B* does not exist in read library, so $RE(s_e)$ is $\{A, C, D, E\}$. *A, D, E* have left mate reads. The left mate reads of *A* and *D* can be mapped to s_s. The left mate read of *E* cannot be mapped to s_s. So, $RML(s_e)$ is {*A, D, E*}, $RMLM(s_s, s_e)$ is {*A, D*}

$$CD(s_s, s_e) = \frac{\left(\sum_{i=1}^{n} x_i y_i\right)^2}{\left(\sum_{i=1}^{n} x_i\right)^2 \left(\sum_{i=1}^{n} y_i\right)^2} \tag{2}$$

In addition, EPGA uses $|RE(s_e)|$ to identify false extension candidate. For each evaluating region, $|RE(s_e)|$ is a particular value of a random variable X which approximately follows the binomial distribution $B(len_{min} - k + 1, P)$. P is the probability that one read can be sequenced, which is estimated as the ratio of $RE(s_n)$ to $R(s_n)$, s_n is the longest sequence seed. For one evaluating region, there are $len_{min} - k + 1$ reads. The expectation value of X is $(len_{min} - k + 1) * P$, denoted by μ_r, the SD of X is $\sqrt{(len_{min} - k + 1) * P * (1 - P)}$, denoted by δ_r. If $|RE|$ of one evaluating region is beyond interval $[\mu_r - \beta * \delta_r, \mu_r + \beta * \delta_r]$, denoted by D_r, EPGA gives it a penalty score (PS) computed in Equation (3). In default, the probability that $|RE(s_e)|$ is beyond D_r should be smaller than 5%, so $\beta = 4$.

$$PS(s_s, s_e) = (1 - CD) * c * d \tag{3}$$

$$c = \frac{AKF_s}{AKF} \tag{4}$$

$$d = \frac{|\mu_r - |RE(s_e)||}{\delta_r} \tag{5}$$

$$S(s_s, s_e) = \begin{cases} RM(s_s, s_e) * \sqrt{CD(s_s, s_e)} & \text{if } |RE| \in D_r \\ RM(s_s, s_e) * \sqrt{CD(s_s, s_e)} - PS(s_s, s_e) & \text{else} \end{cases} \tag{6}$$

In Equation (4), AKF_s is the average frequency of *k-mers* in an extension candidate (U_1 in Fig. 3). AKF is the average frequency of *k-mers* in the entire read library. In Equation (5), d weights the deviation from normal span of |RE|. The final score S of each extension candidate is calculated by Equation (6).

When there are multiple read libraries, EPGA will use all of them to score extension candidates, rather than using long *insert size* libraries only during scaffolding or filling gap regions. Once the sequence seed achieves a length longer than the *insert size* of one read library, the library will be considered in the evaluating extension candidates. For one sequence seed, if there is only one extension candidate, the extension candidate is directly appended to the sequence seed. If there are multiple extension candidates, EPGA scores each extension candidate by Equation (6) and chooses extension candidates s_1 and s_2 with top two scores(s_1 is with the maximum score). If scores of s_1 and s_2 are similar or high, the extending process is terminated. Otherwise, s_1 is appended to sequence seed and the process continues. Upstream extension can be carried out using the same strategy. The reasons why the extension process stops include: (i) Sequence errors not only lead to false nodes or false edges in De Bruijn graph but also causes missing some mate reads of evaluating regions. (ii) In some low-depth regions, there are probably no extension candidates.

After extending all sequence seeds from both upstream and downstream, EPGA gets one contig set consisting of some long contigs. The contig set is kept to be used in the following steps.

2.4 Merging contigs

This step is to delete redundant contig and reduce contig number which can enhance the efficiency of the following steps. Rather than using the ends of the two contigs as a whole to judge whether they can be merged, we divide the ends of the two contigs to some short sub-regions. This new way can extract un-repetitive regions from the ends of the two contigs, and eliminate ambiguities caused by repetitive regions.

In this step, EPGA discovers overlaps between contigs and identifies whether contigs can be merged together. When one short contig is included in another long contig, then the short contig is removed from the contig set. Once two contigs C_l and C_r have overlap C_{ol} of len_{ol} bases, EPGA temporarily merges them together to form a new contig C_{new}. C_{new} consists of three regions C_{ll}, C_{ol} and C_{rr}. After C_l deletes C_{ol}, the rest region is C_{ll}. After C_r cuts off C_{ol}, the rest region is C_{rr}. Then, EPGA extracts the first μ_{is}-len_{ol} bases of C_{rr} and the last μ_{is}-len_{ol} bases of C_{ll} as two test regions. The test regions are divided into some short sub-regions. And, each sub-region is regarded as extension candidates, EPGA computes their RM by Equation (1). If all RMs are bigger than the minimum value α, C_{new} retains, C_l and C_r are removed, else EPGA deletes C_{new} and leaves C_l and C_r alone. An example is shown in Figure 4. When the len_{ol} is longer than all *insert sizes* of multiple read libraries, the two contigs will be merged directly. The reason why contigs can be merged in this step but not linked in extending sequence seed step is that, it maybe harder to choose a correct extension candidate from one extending direction; however, it is probably easy to determine in another extending direction.

2.5 Scaffolding

In this step, according to the contig set constructed from previous steps, EPGA will build a scaffold graph in which each node is one contig and each edge represents that two connected nodes are the closest neighbors in actual genome sequence.

In the process of scaffolding step, the first step is to determine the left neighbor and the right neighbor of every contig. EPGA assumes every contig has only one left neighbor and only one right neighbor. For two contigs u and v, the gap distance len_{gap} is estimated by paired-end reads whose mate reads are separately mapped to u and v. EPGA extracts last μ_{is}-len_{gap} bases from u and first μ_{is}-len_{gap} bases from v as test regions. For the sake of eliminating ambiguities caused by repetitive regions, the test regions are divided into some short sub-regions which will be used to score the linkage between u and v. Each sub-region of u is treated as a candidate sequence, and v as a sequence seed, if any RM of sub-regions is lower than the minimum value α, the process will be terminated and there is no linkage between u and v. Otherwise, EPGA computes the average RM of all sub-regions. After getting all average scores between u and other contigs, EPGA chooses the contig with the top score as its right neighbor and an edge will be linked between u and its right neighbor. Iteratively, the scaffold graph will be built. The second step is to remove erroneous edges in the scaffold graph. If the right neighbor of u is v; however, the left neighbor of v is not u, then the edge (u, v) is one erroneous edge and will be deleted. Finally, this step gets the explicit ordering relation between contigs based on the constructed scaffold graph.

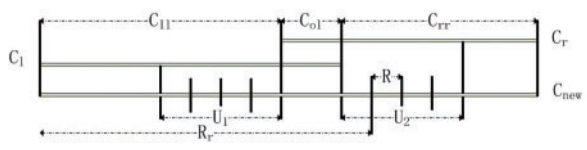

Fig. 4. Merge contigs. C_{ol} is the overlap region between contig C_l and contig C_r. Contig C_{new} is merged by C_l and C_r and is consist of C_{ll}, C_{ol} and C_{rr}. The region U_1 and U_2 are test regions which are all divided into sub-regions. R is one sub-region as extension candidate, R_r is its sequence seed. Then RM of R and R_r can be computed by Equation (1)

2.6 Filling gap

The gap regions between connected nodes in scaffold graph are usually low-depth regions or more complicated regions, which are the reasons why nodes are separated. If the gap region is in low-depth region, the contigs cannot be connected through one path in De Bruijn graph. If the gap region is more complicated region, it is difficult to decide the accurate extension candidates to connect two nodes. For filling one gap, EPGA will iteratively change *k-mer* length to construct sub De Bruijn graphs. When the value of k becomes larger, there will be less branches in sub De Bruijn graph and eliminate some repetitive regions. When k becomes smaller, the connectivity of sub De Bruijn graph will get larger and low depth regions will be connected.

For two nodes C_l and C_r in a scaffold graph, EPGA will first collect a local read set whose mate reads can be mapped to the end of C_l and C_r. Then, EPGA iteratively changes *k-mer* from *k*-max to *k*-min based on the local read set to create sub De Bruijn graph. In default, *k*-max equals to $k+4$, *k*-min equals to $k-4$. In sub De Bruijn graph, EPGA firstly finds two nodes $node_1$ and $node_2$. $node_1$ is referred to the right end of C_l, $node_2$ is referred to the left end of C_r. Starting from $node_1$, EPGA takes the DFS method to find all possible paths which can connect $node_1$ and $node_2$. In the process of search, C_l is regarded as original sequence seed. When extending the sequence seed, EPGA not only computes S between extension candidates and the sequence seed, but also computes RM between extension candidates and C_r. If the score of one extension candidate is smaller than the minimum value α, or the extending length is bigger than the gap distance, or the extension candidate is v, then this path search will terminate. Finally, if there is only one path connecting C_l and C_r, then the gap will be filled by the path. If there are multiple paths, the gap will be filled by 'N'. When there are no paths found, then EPGA constructs new sub De Bruijn graph based on smaller *k-mer*. The process is illustrated in Figure 5.

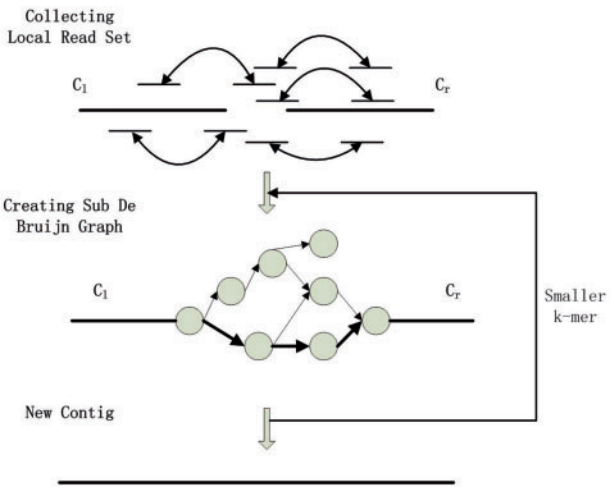

Fig. 5. Filling gap. It includes three steps: collecting local read set of two contigs C_l and C_r, creating sub De Bruijn graph, and discovering path connecting two contigs

3 Experiment

For evaluating the performance of EPGA, our experiments are carried out on real datasets with different properties. Real sequencing datasets include two bacteria (*Staphylococcus aureus* and *Escherichia coli*) and two fungi (*Schizosaccharomyces pombe* and *Neurospora crassa*) using the Illumina platform provided by AllPath2 (MacCallum *et al.*, 2009). All genome references are also provided by AllPath2 and used for validation. Details about real sequencing data are showed Table 1. Each genome includes one short *insert size* library and one long *insert size* library. The read lengths are short and from 26 to 36. We benchmark five assemblers on four real datasets. The assemblers include Velvet, PE-Assembly, SOAPDenovo2, Abyss and EPGA. To provide unbiased benchmarking, we use the assembly evaluation tool GAGE (Salzberg *et al.*, 2012). All benchmarkings were done on a computer with 24 cores (Intel Xeon E5-2620 2.00 GHz).

For one real genome reference gr, SR is one set which consists of all sub-regions of length len in gr, $\text{len} = r + k$. Each sub-region s_{sr} in SR has one corresponding sequence seed s_{srs}, which is its upstream region of length $\mu_{is} + 3 * \delta_{is} - 1$, except for some sub-regions which come from gr[0, $\mu_{is} + 3 * \delta_{is} - 1$]. It is clear that s_{sr} and s_{srs} have overlap with $r - 1$ bases. In the process of assembly, for all extension candidates of one sequence seed, the previous $r - 1$ bases of their evaluating regions are the same and the r-th base is different. For all sub-regions in SR, we randomly change their r-th base to create false sub-regions which are included in one set FSR. For one sub-region in SR, its corresponding sub-region in FSR is treated as one false extension candidate for its corresponding sequence seed.

3.1 Distribution of reads

The distribution of reads is tested based on SR and FSR of genome references. The probability P that one read can be sequenced is the ratio of |RE(gr)| to |R(gr)|. In EPFA, the probability P is estimated by the ratio of $|\text{RE}(s_n)|$ to $|\text{R}(s_n)|$ (denoted by P_n) since we do not have the genome reference, where s_n is the longest node in De Bruijn graph. As shown in Table 2, P_n is very close to P which illuminates the effectiveness of our estimating method. Assuming the distribution of reads follows binomial distribution, the probability that $|\text{RE}(s_{sr})|$ is beyond the interval $[\mu_r - 4 * \delta_r, \mu_r + 4 * \delta_r]$ (D_r) should be <5%, in which μ_r is $\text{len} * P_n$ and δ_r is $\sqrt{(\text{len} * P_n * (1 - P_n))}$. For testing whether |RE| of sub-regions in SR is within D_r, |RE| of every sub-region in SR is calculated, and the percentage of sub-regions whose |RE| are within D_r (denoted by P_c) is listed in Table 2. In addition, |RE| of every sub-region in FSR is also calculated, and the percentage of sub-regions whose |RE| are within D_r (denoted by P_f) is also listed in Table 2. For all genome references, the smallest value of P_c is 0.9567 and the largest value of P_f is 0.0036. So, the correctness of one sub-region s_{sr} can be evaluated by using $|\text{RE}(s_{sr})|$ and D_r.

3.2 Relative mapping and CD

The effectiveness of RM and CD to distinguish correct extension candidates from false extension candidates is tested based on SR and FSR of genome references. For sub-regions in SR and FSR whose

Table 2. Experimental results for the distribution of reads

Genome	P_n	P	len	Sub-region number	P_c	P_f
S.aureus[1]	0.839	0.835	56	2903002	0.9961	0.0002
S.aureus[2]	0.715	0.727	47	2903029	0.9979	0.0003
E.coli[1]	0.852	0.918	56	4638867	0.9655	0.0001
E.coli[2]	0.681	0.729	47	4638876	0.9975	0.0002
S.pombe[1]	0.805	0.818	56	12554178	0.9896	0.0008
S.pombe[2]	0.901	0.883	47	12554214	0.9952	0.0008
N.crassa[1]	0.661	0.662	58	39216548	0.9567	0.0007
N.crassa[2]	0.643	0.609	47	39219309	0.9961	0.0036

[1]read library 1; [2]read library 2.

Table 3. Statistics of RM

Genome	Sub-regions in SR		Sub-regions in FSR	
	μ_{rm}	S^2_{rm}	μ_{rm}	S^2_{rm}
S.aureus[1]	0.9096	0.0093	0.0328	0.0317
S.aureus[2]	0.9537	0.0069	0.01275	0.0125
E.coli[1]	0.8977	0.01397	0.0415	0.0398
E.coli[2]	0.9544	0.0074	0.0097	0.0096
S.pombe[1]	0.8650	0.0168	0.0430	0.0412
S.pombe[2]	0.9510	0.0063	0.0323	0.0313
N.crassa[1]	0.7910	0.0444	0.0323	0.0313
N.crassa[2]	0.9062	0.0212	0.0207	0.0203

[1]read library 1; [2]read library 2.

Table 4. Statistics of CD

Genome	Sub-regions in SR		Sub-regions in FSR	
	μ_{cd}	S^2_{cd}	μ_{cd}	S^2_{cd}
S.aureus[1]	0.5291	0.0080	0.0109	0.0044
S.aureus[2]	0.7593	0.0139	0.0049	0.0020
E.coli[1]	0.7422	0.0153	0.0188	0.0084
E.coli[2]	0.7927	0.0145	0.0034	0.0014
S.pombe[1]	0.7069	0.0291	0.0186	0.0086
S.pombe[2]	0.7654	0.0120	0.0138	0.0061
N.crassa[1]	0.4781	0.0130	0.0140	0.0058
N.crassa[2]	0.7427	0.0321	0.0077	0.0033

[1]read library 1; [2]read library 2.

Table 1. Details of real datasets

	S.aureus		*E.coli*		*S.pombe*		*N.crassa*	
G-length(bp)	2903107		4638902		12554318		39225835	
C-num	3		1		4		251	
L-num	2		2		2		2	
R-len(bp)	35	26	35	26	35	26	36	26
R-num(M)	11.0	7.7	30.0	10.8	55.1	86.5	191.4	123.7
Fold	~130	~70	~230	~60	~160	~180	~170	~80
Insert size(bp)	220	3800	220	3800	210	3800	210	3800

G-length denotes genome length, C-num denotes chromosome number, L-num denotes read library number, R-len denotes read length, R-num denotes read number.

Table 5. The assembly results on *S.aureus*

Assembler	Contigs							Scaffolds						
	Num	Max (kb)	N50 (kb)	Errors	Corrected N50 (kb)	Cov-R (%)	Cov-O (%)	Num	Max (kb)	N50 (kb)	Errors	Corrected N50 (kb)	Cov-R (%)	Cov-O (%)
Velvet	90	312.9	169.2	36	59.8	99.61	99.43	49	1133.1	1091.9	49	284.6	99.45	97.48
SOAP2	581	43.0	10.4	**1**	10.3	98.13	94.71	53	924.1	652.1	15	239.6	98.23	96.4
Abyss	52	274.0	144.7	9	104.2	**99.68**	98.73	31	634.3	357.0	**0**	356.9	99.62	98.25
PE	72	191.8	92.1	7	92.1	99.60	**100**	25	600.6	323.5	**0**	323.5	**99.82**	**99.99**
AllPath2	**19**	**1124.7**	**385.8**	**1**	**385.8**	99.52	99.96	12	1377.4	611.6	**0**	**611.2**	99.72	99.97
EPGA	22	956.4	220.3	9	220.3	**99.68**	**100**	**5**	**1480.0**	**1480.0**	1	597.3	99.69	99.80

Bold values represent best results.

Table 6. The assembly results on *E.coli*

Assembler	Contigs							Scaffolds						
	Num	Max (kb)	N50 (kb)	Errors	Corrected N50(kb)	Cov-R (%)	Cov-O (%)	Num	Max (kb)	N50 (kb)	Errors	Corrected N50(kb)	Cov-R (%)	Cov-O (%)
Velvet	154	391.7	76.0	36	53.2	99.67	99.85	71	721.2	502.9	26	251.5	99.57	97.74
SOAP2	2653	13.7	2.4	**1**	2.4	97.71	83.12	549	951.4	194.5	20	169.0	97.85	92.45
Abyss	65	486.8	171.0	6	156.7	99.83	98.38	26	1666.0	661.8	**0**	660.8	99.81	98.00
PE	173	149.7	60.5	3	56.5	99.90	**100**	49	618.5	303.2	**0**	303.2	99.97	**100**
AllPath2	**33**	**1015.1**	**336.9**	**1**	**336.9**	99.77	99.88	**16**	**2212.1**	699.1	**0**	699.0	98.85	99.95
EPGA	38	493.8	198.4	3	198.4	**99.98**	**100**	19	1061.2	**823.1**	**0**	**822.9**	**99.98**	99.81

Bold values represent best results.

Table 7. The assembly results on *S.pombe*

Assembler	Contigs							Scaffolds						
	Num	Max (kb)	N50 (kb)	Errors	Corrected N50(kb)	Cov-R (%)	Cov-O (%)	Num	Max (kb)	N50 (kb)	Errors	Corrected N50(kb)	Cov-R (%)	Cov-O (%)
Velvet	963	90.2	28.6	77	25.7	**99.29**	99.55	350	520.6	154.3	28	139.5	**99.20**	99.24
SOAP2	9160	13.6	1.9	**2**	1.9	96.73	59.85	2592	481.4	146.4	60	112.1	96.61	90.14
Abyss	834	154.7	45.2	205	31.2	98.18	82.36	390	644.1	210.5	3	210.5	98.04	81.56
PE	959	111.7	32.9	10	32.4	98.72	**99.85**	494	203.6	67.0	**0**	62.8	98.89	**99.84**
AllPath	361	184.4	50.9	29	48.5	95.18	99.80	202	1302.3	243.2	5	224.8	98.01	99.64
EPGA	**334**	**255.6**	**80.6**	43	**70.2**	98.44	**99.85**	**103**	**2444.3**	**659.5**	7	**494.9**	98.47	99.30

Bold values represent best results.

Table 8. The assembly results on *N.crassa*

Assembler	Contigs							Scaffolds						
	Num	Max (kb)	N50 (kb)	Errors	Corrected N50(kb)	Cov-R (%)	Cov-O (%)	Num	Max (kb)	N50 (kb)	Errors	Corrected N50 (kb)	Cov-R (%)	Cov-O (%)
Velvet	**5855**	70.7	10.0	873	8.7	88.46	99.25	**1838**	291.9	46.2	529	35.8	88.23	**98.66**
SOAP2	51 486	8.0	0.8	**16**	0.8	95.23	59.19	41111	170.5	12.5	651	4.3	92.86	65.51
Abyss	10 135	103.7	**15.7**	1274	**10.3**	**96.77**	82.89	5773	**421.7**	**77.6**	18	**71.0**	**96.72**	77.44
PE	8569	66.5	8.4	80	8.2	92.51	**99.57**	5212	89.3	14.9	**0**	14.3	92.90	98.52
AllPath2[a]	—	—	—	—	—	—	—	—	—	—	—	—	—	—
EPGA	6206	**106.8**	10.2	348	9.6	90.08	99.14	4651	165.7	22.1	**0**	20.6	90.14	95.54

[a]The assembly result cannot be obtained by AllPath2. Bold values represent best results.

corresponding sequence seeds exist, their RM and CD are computed by Equations (1) and (2). The sub-regions in SR are treated as correct extension candidates and sub-regions in FSR are considered as false extension candidates. The mean value of RM (μ_{rm}), the variance of RM (S^2_{rm}), the mean value of CD (μ_{cd}) and the variance of CD (S^2_{cd}) are calculated and listed in Tables 3 and 4. For sub-regions in SR, we can find that the smallest μ_{rm} is 0.8650 and the smallest μ_{cd} is 0.4781. For sub-regions in FSR, the biggest μ_{rm} is 0.0430 and the biggest μ_{cd} is 0.0188. In addition, S^2_{rm} and S^2_{cd} are small. So, RM and CD can be used as two significant characteristics

to distinguish correct extensions candidates from false extension candidates.

3.3 Assembly results

We adopt several metrics provided by GAGE for measuring the continuity and completeness of assembly results. The continuity is judged by the number of contigs (or scaffods), *N*50 and the length of longest contig (or scaffold). *N*50 is the length of the longest contig (or scaffold) such that all the contigs longer than this contig cover at least half of the genome being assembled (Earl *et al.*, 2011). The completeness is estimated by two coverages (*Cov-R* and *Cov-O*) which are the proportion of the genome reference being covered by output contigs (or scaffolds) and the proportion of the output contigs (or scaffolds) being covered by genome reference. GAGE includes corrected analysis. For corrected analysis, misjoin and indel longer than 5 bases are viewed as errors. GAGE splits contigs at every error position and provides revised results. GAGE tallies three types of misjoins: (i) inversions, where part of a contig or scaffold is reversed with respect to the true genome; (ii) relocations, or rearrangements that move a contig or scaffold within a chromosome; and (iii) translocations, or rearrangements between chromosomes. The statistics of the assembly results of different assemblers are summarized in Tables 5–8 and the best results are in bold.

3.3.1 *De novo* assembly of *S.aureus*

The genome of *S.aureus* is not too long and it is usually used as a sequencing object. The assembly results about *S.aureus* are shown in Table 5. For contigs and scaffolds, EPGA and AllPath2 produce few contigs and scaffolds. After scaffolding, all assemblers produced longer scaffolds. Velvet and SOAPDenovo create more errors which lead to weak corrected *N*50. Although AllPath2 have the longest corrected *N*50 for contigs and scaffolds, the coverages of AllPath2 is smaller than EPGA. In addition, EPGA produces fewer scaffolds than AllPath2 and the corrected *N*50 of scaffold is much closed to AllPath2.

3.3.2 *De novo* assembly of *E.coli*

For contigs and scaffolds, EPGA and AllPath2 perform comparably and better than other assemblers. According to assembly results in Table 6, EPGA and AllPath2 still have few contigs and scaffolds. Although AllPath2 has the longest scaffold, the corrected *N*50 of EPGA is longer than AllPath2. EPGA has the best coverages for contigs and the best Cov-R for scaffolds. SOAPDenovo2 has the fewest errors for contigs but too many errors for scaffolds.

3.3.3 *De novo* assembly of *S.pombe*

This genome reference is long and relatively more repetitive. In Table 7, we can find that all assemblers are inclined to produce more errors and worse results. EPGA creates the fewest contigs and scaffolds than other assemblers. Corrected N50 for contigs and scaffolds are the longest among all assemblers. Abyss has the most errors for contigs while SOAPDenovo2 has the most errors for scaffolds.

3.3.4 *De novo* assembly of *N.crassa*

For the relatively larger *N.crassa* genome, the assembly result is shown in Table 8. The assembly results of *N.crassa* cannot be obtained by AllPath2, because there are too many reads in libraries which exceed the read number limit of AllPath2. EPGA leads in terms of the number and *N*50 for contigs. For scaffolds, Abyss is of significant *N*50 in comparison with other assemblies. The errors of contigs and scaffolds for Abyss are high. Note that, although the *Cov-R* of Abyss is high, the *Cov-O* is too low, which means Abyss produces too much redundant contigs. Also note that *N.crassa* reference genome is unfinished and it consists of many chromosomes which influence the evaluation of assembly results.

3.3.5 Running time and peak memory

The running time and peak memory of EPGA are ~15 min and 9 G for *S.aureus*, 40 min and 28 G for *E.coli*, 261 min and 97 G for *S.pombe*, 2830 min and 198 G for *N.crassa*. AllPath2 needs much more running time and peak memory than EPGA, 72 min and 40 G for *S.aureus*, 156 min and 74 G for *E.coli*, 982 min and 464 G for *S.pombe*. The efficiency of Velvet, SOAPDenovo2, Abyss and PE-Assembly are better than EPGA, this is because EPGA needs more memory to store mapping information between sequence seeds and extension candidates to score. In addition, EPGA executes strict detection program to determine contigs merging and scaffolding which consume some time. However, EPGA gets more satisfactory results, which is worth of more time and peak memory.

4 Conclusion

EPGA extends sequence seeds by iteratively evaluating extension candidates and choosing correct one between them to construct contig set. Because there exists sequencing errors and uneven sequencing depth in read libraries, we present new ideas to avoid problems produced by them. For adjacent repetitive regions and paired repetitive regions, we use CD to distinguish correct extension candidates from false ones. The performance of EPGA is validated on real datasets from two bacteria and two fungi using the Illumina platform. In our experiments, we included some popular algorithms with available implementation. We considered several standard metrics for comparing assemblies. For all four datasets, EPGA produced more satisfactory results than the other assemblers considered. EPGA has been demonstrated that it is possible to obtain complete and highly accurate de novo genome assemblies. As one direction of future work, we would study how to reduce peak memory and running time for better performance.

Funding: This work was supported in part by the National Natural Science Foundation of China [61232001, 61420106009, 61379108] and the Program for New Century Excellent Talents in University [NCET-12-0547].

Conflict of Interest: none declared.

References

Alkan,C. *et al.* (2011) Limitations of next-generation genome sequence assembly. *Nat. Methods*, **8**, 61–65.

Ariyaratne,P. and Sung,W.K. (2011) PE-assembler: de novo assembly using short paired end reads. *Bioinformatics*, **27**, 167–174.

Bankevich,A. *et al.* (2012) SPAdes: a New Genome Assembly Algorithm and its Applications to Single-Cell Sequencing. *J. Comp. Biol.*, **19**, 455–477.

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

Chitsaz,H. *et al.* (2011) Efficient de novo assembly of single-cell bacterial genomes from short-read datasets. *Nature Biotech.*, **29**, 915–921.

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.

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

Gnerre,S. *et al.* (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl Acad. Sci. USA*, **108**, 1513–1518.

He,Y. *et al.* (2013) De novo assembly methods for next generation sequencing data. *Tsinghua Sci. Technol.*, **5**, 500–514.

Iqbal,Z. *et al.* (2012) De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genet.*, **44**, 226–232.

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

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

Luo,R. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience,* **1**, 18.

Maayan,B. *et al.* (2012) Telescoper: de novo assembly of highly repetitive regions. *Bioinformatics*, **28**, 311–317.

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

Medvedev,P. *et al.* (2011) Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. In: *Proceedings of Research in Computational Molecular Biology*, Vancouver, Lecture Notes in Computer Science, Vol. 6577, pp. 238–251.

Peng,Y. *et al.* (2010) IDBA—a practical iterative de Bruijn graph de novo assembler. In: Proceedings of Research in Computational Molecular Biology, Lecture Notes in Computer Science, Lisbon, Vol. 6044, pp. 426–440.

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

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

Pham,S.K. *et al.* (2013) Pathset graphs: a novel approach for comprehensive utilization of paired reads in genome assembly. *J. Comput. Biol.*, **20**, 359–371.

Ribeiro,F. *et al.* (2012) Finished bacterial genomes from shotgun sequence data. *Genome Res.*, **22**, 2270–2277.

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

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.

Wetzel,J. *et al.* (2011) Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies. *BMC Bioinformatics*, **12**, 95.

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

Zerbino,D.R. *et al.* (2009) Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS One*, **4**, e8407.