 doi:10.1093/bfgp/elr035

Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph

Zhenyu Li, Yanxiang Chen*, Desheng Mu*, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, Bicheng Yang and Wei Fan*

Advance Access publication date 19 December 2011

Abstract

Since the completion of the cucumber and panda genome projects using Illumina sequencing in 2009, the global scientific community has had to pay much more attention to this new cost-effective approach to generate the draft sequence of large genomes. To allow new users to more easily understand the assembly algorithms and the optimum software packages for their projects, we make a detailed comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph, from how they match the Lander–Waterman model, to the required sequencing depth and reads length. We also discuss the computational efficiency of each class of algorithm, the influence of repeats and heterozygosity and points of note in the subsequent scaffold linkage and gap closure steps. We hope this review can help further promote the application of second-generation *de novo* sequencing, as well as aid the future development of assembly algorithms.

Keywords: *OLC; DBG;* de novo *assembly; second-generation*

INTRODUCTION

One of the most important tasks in genome biology is to obtain a complete genome sequence, which is finished by a combination of sequencing technology and assembly software [1–3]. The high cost of Sanger sequencing technology has long been a limiting factor for genome projects, as we can see from the limited number of large genomes published before 2010. Fortunately, the second-generation sequencing technologies Roche/454 (www.454.com), Illumina/solexa (www.illumina.com) and AB/Solid (www.appliedbiosystems.com), which arrived in the market in 2005 and rapidly developing since then, have dramatically lowered the cost per sequenced nucleotide and increased throughput by orders of magnitude. However, although the second-generation technologies are comparatively very cheap, their application was mainly restricted to resequencing projects [4, 5] where a good reference sequence existed, due to the much shorter read length (30–400 bp) in comparison with Sanger sequencing (500–1000 bp). In light of this, a major question that confronted us was, can we *de novo* sequence and assemble a large genome (>100 Mbp) using short reads? If so, sequencing cost no longer becomes a limiting factor for most *de novo* large genome projects, and sequence assembly becomes the major challenge.

The evolution of assembly algorithms has accompanied the development of sequencing technologies. Currently, there are two widely used classes of

Corresponding authors. Bicheng Yang. E-mail: yangbicheng@genomics.org.cn; Wei Fan. Tel/Fax: +86 0755 22358672; E-mail: fanw@genomics.org.cn

*These authors contributed equally to this work.

Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu and **Wei Fan** are from the DNA sequence assembly team at the Science and Technology department of Beijing Genomics Institute at Shenzhen (BGI-SZ).

Bicheng Yang is from the Scientific Cooperation Department, BGI-SZ. BGI-SZ is a leading genomics research center in China.

algorithms: overlap–layout–consensus (OLC) and de-bruijn-graph (DBG) [1–3]. OLC generally works in three steps: first overlaps (O) among all the reads are found, then it carries out a layout (L) of all the reads and overlaps information on a graph and finally the consensus (C) sequence is inferred. It is an intuitionistic assembly algorithm, initially developed by Staden (1980) and subsequently extended and elaborated upon by many scientists. OLC became successful with the wide application of Sanger sequencing technology. Many widely used assembly programs adopted OLC, such as Arachne [6], Celera Assembler [7], CAP3 [8], PCAP [9], Phrap [10], Phusion [11] and Newbler [12].

DBG is an anti-intuition algorithm, working by first chopping reads into much shorter *k*-mers and then using all the *k*-mers to form a DBG and finally inferring the genome sequence on the DBG. This algorithm was originally introduced in 1995 by Ramana M. Idury and Michael S. Waterman [13], and the first DBG assembler EULER was published in 2001 by Pavel Pevzner and Michael Waterman [14]. DBG was initially little known in the assembly area for a long time and few people predicted its potential importance. However, this situation dramatically changed upon Illumina/solexa sequencing technology entering the market, and several short-read assembly software have since been developed based on DBG, such as Euler-USR [15], Velvet [16], ABySS [17], AllPath-LG [18] and SOAPdenovo [19]. The DBG assemblers were initially successful on small genomes such as bacteria, and were then extended to large genomes. Since the completion of the cucumber [20] and panda [21] genome projects using Illumina sequencing, researchers around the world have seen a new cost-effective approach to generate the draft sequence of large genomes.

However, this idea was not unanimously accepted immediately. Some studies discussed the shortages of short-read assembly algorithms, and showed concern about the quality of draft assemblies [22, 23], whereas other studies produced results to support the application of short-read assembly in large genomes [18, 19]. This debate is still far from being resolved. As the reads length of second-generation technologies has increased with time, and DBG-based assembly algorithms have also continued to improve, we believe that *de novo* assembly with second-generation sequencing will generate better results than ever, and this method will be adopted by more and more genome projects. To allow new users to more easily understand the assembly algorithms and choose the correct software for their projects, in this perspective, we make detailed comparisons of the two major classes of assembly algorithms: OLC and DBG. We hope this article can help promote the application of second-generation *de novo* sequencing, as well as shed light on the future development of assembly algorithms.

IDEAL SEQUENCING DATA AND MATHEMATICAL ASSEMBLY MODEL

As an easy-to-understand illustrative example, we will first discuss the simplest assembly model using hypothetical 'ideal' genome sequencing data. The simplest genome can be viewed as a long random sequence comprising four types of bases (A, C, G and T), and ignoring repeats and all other complex structures. Here, we start with the most basic sequencing strategy, single-end whole-genome-shotgun (WGS) [24], which can be thought of as a process of sampling equal-length fragments with the starting points distributed randomly along the genome. Sequencing errors and all other biases are ignored so that the sequencing data can be thought as ideal. Note that the read length is far shorter than the genome size. In computer theory, ideal assembly would involve just finding the common string (genome) of all substrings (reads). In the following examples, we will discuss the concepts of base and *k*-mer coverage, Lander–Waterman model and basic OLC and DBG assembly models by using this ideal sequencing data.

Base coverage and *K*-mer coverage

Sequencing of this 'ideal' sequence can be thought of as a process of sampling bases from all the genomic positions randomly. Given the probability that one base in a specific position of the genome to be sampled is very low in a single sampling process and the number of times of sampling is comparatively a quite large number, the problem of base coverage of the genome follows a Poisson distribution [25]. A *k*-mer is a string extracted from reads with specified length K. Similarly, the problem of *k*-mer coverage of the genome also follows a Poisson distribution [26] (Figure 1). To further clarify this, we can illustrate the coverage problem using two concepts: coverage depth (the average number

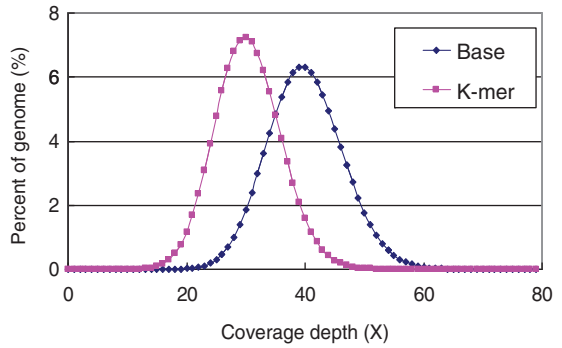

Figure 1: Distribution of base (*k*-mer) coverage, using 40× error-free sequencing data of any genome size. As the base coverage depth (d_b) is 40, assuming that read length (L) is 100 bp and *k*-mer size (K) is 25 bp, the *k*-mer coverage depth (d_k) is 30.4, which can be calculated by $d_k = d_b*(L-K+1)/L$.

of times each base/*k*-mer is being sequenced) and coverage extent (the ratio of genome covered by at least one base/*k*-mer). Given a genome size (G), read length (L), read number (N), and *k*-mer size (K), the total number of bases (n_b) and *k*-mers (n_k) can be easily determined by ($n_b = N \star L$) and [$n_k = N \star (L\text{-}K+1)$], with the ratio between them being [$n_b/n_k = L / (L-K+1)$]. We can also calculate the coverage depth for bases (d_b) and *k*-mers (d_k) by ($d_b = n_b/G$) and ($d_k = n_k/G$), with the ratio between them being ($d_b / d_k = L / (L-K+1)$). For the *de novo* projects, these formulas can be used to estimate the unknown genome size (G) and coverage depth for bases (d_b) from read data before assembly [26], using the formulas ($G = n_k / d_k$) and ($d_b = L / (L-K+1) \star d_k$). The number of *k*-mers (n_k) can be obtained by directly counting *k*-mers from reads [27], whereas the coverage depth of *k*-mers (d_k) can be observed from the peak depth value on the *k*-mer coverage depth distribution curve (Figure 1). In practice, these formulas need some correction because of the effect of sequencing errors.

Base coverage depth (d_b), which reflects the total amount of sequencing data, is one of the most important parameters for a *de novo* sequencing project. Shortened, it is often called and marked as sequencing depth (c). Base coverage extent, is another very useful parameter that can help us decide the required sequencing depth for a *de novo* project. Because the base coverage follows a Poisson distribution, the probability of non-coverage is equal to $P(X=0) = e^{-c}$, so the coverage extent of bases is equal to $P(X>0) = 1-e^{-c}$ (Table 1). To cover >99% of a genome, the sequencing depth should be >4.6. Taking into account sequencing biases, traditional genome projects using Sanger sequencing often use a slightly larger sequencing depth to achieve the 99% coverage extent [28, 29]. To ensure the whole genome is covered, the number of uncovered bases $G \star e^{-c}$ should be <1. Taking the human genome (3 Gb) as an example, the sequencing depth c needs to be at least 22. The larger the genome size, the higher sequencing depth is needed. For genome projects sequenced by second-generation technologies [20, 21], which often generate more than 30× depth of data, there seems to be less need to worry about the coverage extent for most genomes because of ($e^{-30} = 1e-13$), which is a significant improvement over the Sanger sequencing projects [28, 29]. In practice though, because of sequencing bias, not every region of the genome is sampled randomly, so the coverage extent is often worse than expectation.

Looking back to the Lander–Waterman model

In 1988, Lander and Waterman published the first mathematical model for sequence assembly [30], which was established on the ideal sequencing data (discussed in the previous section). In that model, if two reads overlapped and the overlap length was larger than a cutoff (T), then the two reads should be merged into a contig (continuous sequence), and this process is iterated until no reads or contigs can be merged. One of the most useful conclusions to come from this is that the resulting contig number can be calculated. Assuming each contig contains a rightmost read, then the contig number is equal to the number of rightmost reads, which can be calculated as $(G \star c/L) \star e^{-c[(L-T)/L]}$, where $G \star c/L$ is read number and $e^{-c[(L-T)/L]}$ is the probability that a read is rightmost. For further details on these, refer to the Lander–Waterman paper [30]. In practice, an equal or larger overlap length than the cutoff T is often required to determine overlap and the formula to calculate contig number should be changed to $(G \star c/L) \star e^{-c[(L-T+1)/L]}$.

The Lander–Waterman model shows that the resulting contig number is related to four parameters: read length (L), overlap length cutoff (T), sequencing depth (c) and genome size (G). As G is a fixed number in a given genome project, there are therefore only three parameters (L, T and c) that can be adjusted. L is often determined by the sequencing platforms and T determines the reliability of overlap

Table 1: Base coverage extent of the genome versus sequencing depth

Sequencing depth	Uncovered ratio (%)	Uncovered bases	Sequencing depth	Uncovered ratio	Uncovered bases	Sequencing depth	Uncovered ratio	Uncovered bases
1	36.79	1 103 638 324	11	1.67E-05	50 105	21	7.58E-10	2
2	13.53	406 005 850	12	6.14E-06	18 433	**22**	**2.79E-10**	**0.84**
3	4.98	149 361 205	13	2.26E-06	6781	23	1.03E-10	0.31
4	1.83	54 946 917	14	8.32E-07	2495	24	3.78E-11	0.11
5	**0.67**	**20 213 841**	15	3.06E-07	918	25	1.39E-11	0.04
6	0.25	7 436 257	16	1.13E-07	338	26	5.11E-12	0.02
7	0.09	2 735 646	17	4.14E-08	124	27	1.88E-12	0.01
8	0.03	1 006 388	18	1.52E-08	46	28	6.91E-13	0.00
9	0.01	370 229	19	5.60E-09	17	29	2.54E-13	0.00
10	4.54E-05	136 200	20	2.06E-09	6	30	9.36E-14	0.00

Note: We use c to represent sequencing depth. The uncovered ratio of genome is calculated by e^{-c}, whereas the uncovered bases of genome is calculated by $G*e^{-c}$. We use genome size (3 Gb) of human, e.g. to calculate number of uncovered bases. Values from two sequencing depths (5 and 22) are highlighted with bold style, which means 1% genome uncovered and 1 base of genome uncovered, separately.

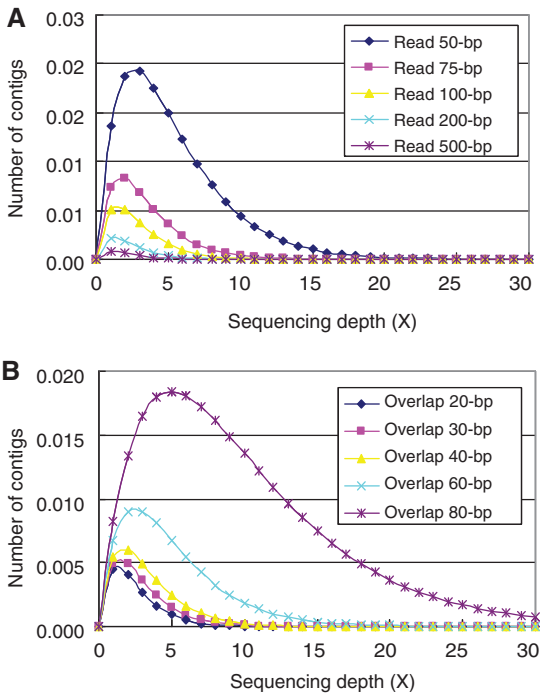

Figure 2: Calculation of contig number for various combination of c, L and T, by the normalized Lander–Waterman formula $(c/L)*e^{-c[(L-T)/L]}$, which is unrelated with genome size. (**A**) Fix the overlap length cutoff (T) to 31 bp, use different curves to represent result of different read lengths (L). (**B**) Fix the read length (L) to 100 bp, use different curves to represent result of different overlap length cutoffs (T).

between reads, with a larger T usually resulting in a more reliable overlap. In most cases, we would want to know that given a pair of L and T, what should c be to achieve an expected assembly result? To answer that question, two figures need to be plotted, one with T fixed, and the other L. The number of contigs represents the fragmentation level of the assembly. The fewer the remaining contigs, the better the assembly result is. In Figure 2A, T is fixed and L changed to compare assembly results under different read lengths. This shows that using 30×50 bp reads can generate an assembly result similar to that of using 10×500 bp reads, which means that a high sequencing depth can compensate for the disadvantage of short read length, and given a specified sequencing depth, longer reads can result in a better assembly result. In Figure 2B, L is fixed and T changed to compare assembly results under different overlap lengths. This shows that using $10\times$ data with T (20 bp) can achieve similar assembly results as that obtained using $20\times$ data with T (61 bp), which means that a larger overlap length requires higher sequencing depth and given a specified sequencing depth, a smaller T generates longer contigs but sacrifices some of the overlap detection accuracy. Choosing the correct L and T value is important for a *de novo* project and when the L and T are determined, the required sequencing depth c can be inferred according to the expected assembly result.

Basic OLC and DBG assembly algorithms

In this section, we discuss the basic OLC and DBG algorithms using the ideal sequencing data. Both OLC- and DBG-based algorithms assemble the genome by utilizing the overlap information among the input set of reads that conform to the Lander–Waterman model [30]. The OLC algorithm is very consistent with Lander–Waterman model,

sharing the same definition of parameter T. In the DBG algorithm, to ensure the k-mers can be linked, the minimum overlap between two reads should be no less than the k-mer size K that is equivalent to the T parameter in the Lander–Waterman model. In the formula used to calculate the contig number in the Lander–Waterman model, $c[(L-K+1)/L]$ is equal to d_k, and c/L is equal to $d_k/(L-K+1)$, allowing the formula to be converted to $[d_k/(L-K+1)]^{\star}(G^{\star}e^{-dk})$, indicating that in the DBG algorithm, the result is related directly to the k-mer coverage depth rather than the base coverage depth (or sequencing depth). The $G^{\star}e^{-dk}$ also relates to the number of uncovered genomic positions by k-mers, which means that if the genome is fully covered by k-mers, it can be completely assembled.

The methods used to exploit the overlap information are different in OLC and DBG algorithms [1–3]. In the OLC algorithm, the identification of overlap between each pair of reads is explicit, typically by doing all-against-all pair-wise reads aligning. In the DBG algorithm, the overlap relationship between neighboring k-mers is established implicitly. This process is completed by chopping all the reads into k-mers and simultaneously recording their neighboring relations. As a result, the OLC algorithm constructs a reads graph, which places reads as nodes and assigns a link between two nodes when these two reads overlap larger than a cutoff length (Figure 3A). The nodes number is equal to the reads number, increasing linearly with sequencing depth and the links number will increase by a logarithmic scale. The DBG algorithm constructs a k-mer graph that places k-mer as nodes and assigns a link between two nodes when these two k-mers are neighbors on the genome (Figure 3B). The node number is equal to the genome size (i.e. $G-K+1$), and the links number is also equal to the genome size (i.e. $G-K$) and unrelated to the sequencing depth. Here we assume all k-mers are unique on the whole genome sequence. In practice, the DBG nodes number will be much higher than $G-K+1$ because of the introduction of many false k-mers caused by sequencing errors.

Although OLC and DBG algorithms have essentially equivalent roles, they differ in the algorithm complexity and computational efficiency [1–3]. In OLC assembly using the reads graph, the layout step is a Hamiltonian path problem, which is known to be NP hard; however, in DBG assembly using the k-mer graph, infering the contig sequence is an Euler path problem that is easier to resolve [14]. Therefore, the main advantage of DBG is that it transforms assembly problems to an easier problem in algorithm theory. After the layout step, OLC needs to call the consensus sequence from the multiple sequence alignments; whereas after the construction of DBG, the k-mers already include the consensus information. In addition, computational feasibility is very important for genome assembly. The DBG algorithm does not contain a CPU-intensive reads aligning step and as mentioned, the nodes (k-mers) and links numbers are approximately equal to the genome size, which makes it achieve both higher CPU and Memory efficiency than the OLC algorithm does when the sequencing

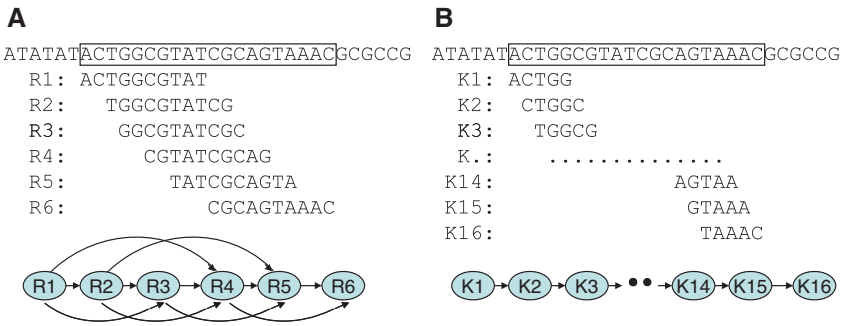

Figure 3: Construction of OLC and DBG graph using example data from 20-bp length genomic region (top). (**A**) We generated six reads (R1–R6) on this region. The read length (L) is 10 bp, and the cutoff of overlap length (T) is 5 bp. The reads were layout-orderly along the genome according to their starting position and the corresponding OLC graph illustrated below, with most nodes having more than one ingoing or outgoing arcs. (**B**) The reads were chopped into k-mers (K = 5 bp), there are in total 16 different k-mers, most of which occur in more than one reads. The k-mers were layout-orderly along the genome according to their starting position, and the structure of DBG graph illustrated below, with most nodes having only one in-arc and one out-arc.

depth becomes very high. This makes it especially attractive for the second-generation sequencing projectsthat usually use high sequencing depth (>30×) to compensate for the short read length (30–100 bp). With this in mind, the DBG algorithm seems to be a better choice for assembly of large genomes using second-generation short reads.

SEQUENCING DATA AND ASSEMBLY ALGORITHMS IN PRACTICE

In this section, we now turn to discussing the assembly algorithms using real sequencing data. In practice, there are several factors interfering with the detection of real overlap among reads, all of which affect the assembly. Some factors originate from the genome and others originate from the sequencing technology. The real genomes of plants and animals often have large sizes ranging from ∼100 Mb to ≥10 Gb [31], often containing a huge amount of repetitive sequences, which are distributed across the whole genome and composed of transposable elements, short tandem repeats and large segmental duplications [32, 33]. According to previous reports, genomes containing >30% repeats include silkworm [28], panda [21], rice [29], cucumber [20], amongst many others. In addition, DNA is not always extracted from a haploid genome (or homozygous diploid genome), but extracted from heterozygous diploid genomes in most cases. As polyploid genomes will make assembly even more difficult, they are seldomly chosen for *de novo* sequencing, especially using short reads. The available sequencing technologies are far from perfect: the read length for different sequencing platforms ranges from 50 bp to 1000 bp, with single-base error rate of raw reads ranging from 0.1% to 3% [34, 35]. Repeats, heterozygosity, limited read length and sequencing errors, together create ambiguity in the overlap detection between reads and make it difficult to determine read order by the observed overlaps. Under specified read length and single-base error rate, longer repeat units, higher similarity among copies, larger amount of repeats and higher heterozygous rates will result in more fragmental assembly. Fortunately, most current sequencing technologies provide pair-end sequencing technology that provides further long-range linkage information and is useful to cross repeats.

The usual purpose of assembly algorithms is to produce a haploid genome sequence from a set of pair-end WGS reads, which are derived from a slightly heterozygous (<0.1%) diploid genome. For *de novo* genome sequencing, it is better to extract DNA from the haploid individual or the individual with lowest heterozygous rate. Sequencing errors are generated by the sequencing platforms, and a lower rate of sequencing errors is beneficial for assembly. Sequencing-error bases can be reduced by prefiltering the raw reads with extremely low quality values and also by performing error correction by utilizing the high coverage information. The OLC algorithm can tolerate some small heterozygous difference in overlap detection by allowing a few mismatches and producing a single path consisting of read nodes. In contrast, with the DBG algorithm, the heterozygous difference always leads to two paths consisting of *k*-mer nodes coming from heterozygous regions, respectively. However, these two paths can be merged into one by some additional work. Usually, the sequencing error rate (<1%) and heterozygous rate (<0.1%) are low so they do not seriously affect the assembly. In that case, most effort is to deal with repeats. An idea shared between both OLC and DBG algorithms is to identify the repeats boundary and break the path at these boundaries, which prevent it from creating artificial paths that do not exist in the genome. For both OLC and DBG algorithms, the whole assembly pipeline can be generally divided into four parts: data pre-processing, contig construction, scaffold linkage and gap closure. Each part will be discussed in the following sections. Note that not all parts are necessary in assembly software.

Data preprocessing

It is well known that raw reads from any current sequencing platforms contain many sequencing errors that affect sequence assembly. These can be dealt with by filtering and correcting them. Most sequencing technologies generate a quality value for each base in the reads [36, 37]. Most sequencing errors are flagged by a low quality value and can be easily filtered by checking this value. However, some sequencing errors may still demonstrate a high quality value preventing them to be filtered in this way. For high-coverage sequencing, a base will be sequenced many times and the correct form is likely to appear with much higher frequency than the errors. Thus, the data accuracy can be further improved by correcting the errors in raw reads based on the frequency information [14, 38], a

process often referred to as pre-assembly error correction. In OLC assembly software, the low-quality filtering process is often performed, although the pre-assembly error correction is often omitted. The small amount of sequencing errors remaining after filtering do not usually cause serious problem because these sequencing errors can be tolerated in the pair-wise alignment (O) by allowing some mismatches, which will not increase the computational cost much. Furthermore, OLC identifies and excludes sequencing errors in the inferring consensus (C) step based on the multiple sequencing alignments [6, 7]. In DBG software, both the low-quality filtering and pre-assembly error correction are usually necessary, because reads containing lots of sequencing errors will create huge amount of false *k*-mers that are not contained in the genome sequence and usually appear only once in all reads data set. These false *k*-mers will consume several times more computer memory in building the *k*-mer graph. Moreover, sequencing errors will create many branch paths with low depth in the graph, which will add complexity to the graph and make it difficult to infer the contig correctly [18, 19]. As the sequencing cost decreases with the development of sequencing technologies, the sequencing coverage for *de novo* projects generally increases. Higher sequencing coverage will benefit the pre-assembly error correction, as well as the final consensus sequence.

There are generally two ways to do pre-assembly error correction, both of which can be used with either OLC or DBG software. The first method is based on the reads alignment. This initially finds all the overlapped reads by doing multiple alignments, and then distinguish sequencing errors from correct bases through a probability model. Note that the overlap detection step is CPU-intensive. Many software applications, including Allpath-LG [18], SHREC [39], HiTEC [40] and ECHO [41] currently adopt this method. The second method is based on the *k*-mer frequency spectrum. First, this counts the frequencies of all *k*-mers in the reads data set, and then divides them into two types: trusted *k*-mers and untrusted *k*-mers. Several of the initial programs developed only used the *k*-mer frequency and an arbitrarily made cutoff as the judgment call (Figure 4A) [14, 19]. More recently developed packages find the optimal cutoff by minimizing the total false positive and false negative errors [42] and some further take the quality value into consideration [38] to prevent mistaking low-frequency *k*-mers as untrusted *k*-mers caused by sequencing coverage bias problems. The error correction tools can identify genomic positions with sequencing error by using the distribution pattern of *k*-mers (Figure 4B), and then try to find a path with minimal change that will transform all the untrusted *k*-mers into trusted *k*-mers. Software such as Euler [14], SOAPdenovo [19], Reptile [43] and Quake [38] all adopt this method. No sequence alignment is needed in this method so it saves substantial computation time.

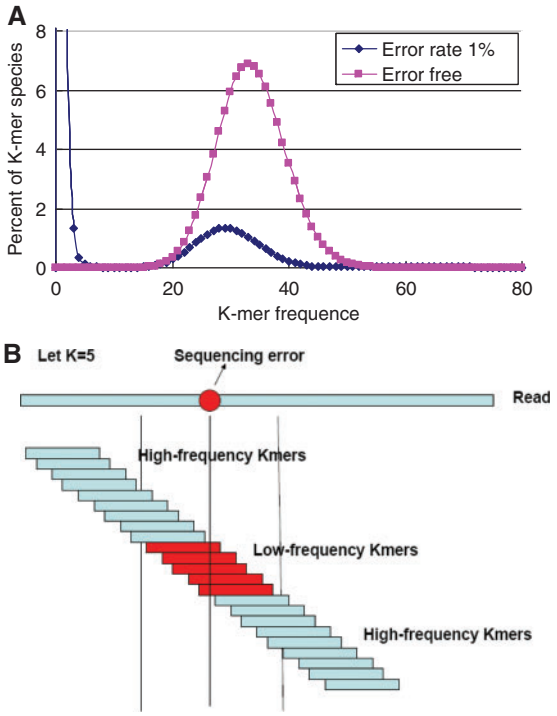

Figure 4: Error correction by the *k*-mer spectrum method utilizing coverage information. (**A**) Distribution of *k*-mer (K = 17) frequency for two sets of 40× simulated *Arabidopsis* WGS data with read length (L) 100 bp. One data set is error free, while the other data set has 1% sequencing error. The average *k*-mer frequency for error-free and 1% error reads are 34 and 31, respectively. In the 1% error curve, about 80% *k*-mer species have frequency below five, most of which are caused by sequencing errors. (**B**) The simplest pattern of *k*-mers (K = 5 bp) on a read where a sequencing error happens. The 5 *k*-mers which crossed the error base appear in low frequency, whereas the surrounding *k*-mers appear in high frequency. In practice, the situations are often more complex than this, some of the false *k*-mers may appear in high frequency, some of the correct *k*-mers may appear in low frequency and more than one sequencing errors nearby each other may create a longer set of low-frequency *k*-mers.

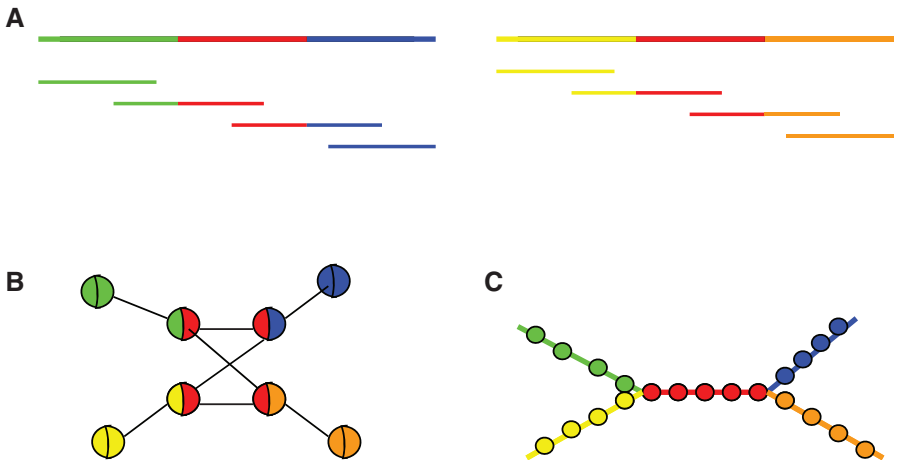

Figure 5: The difference to represent repeats in OLC and DBG graphs. (**A**) Two separate genomic regions share a repeat fragment (in the middle) and the flanking regions are unique sequences. Top is the genomic sequence and bottom are the sequenced reads. (**B**) The OLC reads graph. The nodes represents reads and the links show overlap relations. All the repeat reads are placed on the graph as nodes. (**C**) The DBG *k*-mer graph. The reads are chopped into shorter *k*-mers. The nodes represents *k*-mers and the links show neighboring relations. The *k*-mers from repeat regions are collapsed together.

Contig construction

Contig construction is building a continuous sequence using reads overlap information, which is the core step in any assembly software. Essentially, OLC and DBG algorithms only corresponding to contig construction; however, people often use them to refer to the whole set of assembly pipelines. Assuming that the usual sequencing error rate and heterozygous rate are low, the major effort expended in this step is to deal with repeats. Here a model OLC and DBG graph represent repeats in different ways (Figure 5). Repeat reads are all placed as nodes in the OLC graph, whereas repeat *k*-mers are collapsed into single nodes in the DBG graph. Repeats will increase the computational time needed for pair-wise reads alignment in the OLC algorithm because reads coming from repeat regions have many areas of overlap with other reads. Besides, it is very memory intensive to store these overlap relationships. A solution to this problem is to mask the repeat patterns (partial or whole reads) first (pre-masking) before or during finding the overlap and recover the masked repeats after contig construction or by gap closure with pair-end information [7, 44]. In this case, no repeats exist in the repeat-masked OLC graph that also makes it much easier to infer contigs. In contrast, in the DBG algorithm, the repeats will not increase the computational consumption because there is no pair-wise reads alignment step and *k*-mers nodes from repeats are collapsed together during the construction of DBG [14, 16]. Despite their different strategies, OLC and DBG algorithms have the same goal in contig construction, that is to find continuous paths without branching and stopping at repeat boundaries. These found paths forms initial contigs, which serve as the input to scaffold linkage. Besides repeats, the sequencing error and heterozygosity also affect contig construction and OLC tolerates them in finding overlaps by allowing some mismatches, whereas DBG excludes them on the *k*-mer graph by removing tips and low-coverage edges and by merging bubble edges.

One of the most important issues to consider are repeat sequences, and the first question to ask is: what is a repeat? The answer is that what is defined as a repeat is relative to the sequence length. In genome assembly, the repeats that we are concerned about are those with lengths longer than the read length, meaning that no single read can cross-span these repeat regions. Important to note is that by extending read length to exceed the longest of all of the repeats, from an assembly point of view there will no longer be any repeats. With regard to OLC and DBG algorithms, the overlap length between reads, i.e. the cutoff of overlap length (T) in OLC and the *k*-mer size (K) in DBG determines the ability to overcome repeats. When T or K is larger than the size of any repeats, then repeats will disappear from

the assembly view. A rise in read length (L) is the precondition for the rise of overlap length (T and K) because, L is the upper end of T and K. In the implementation, when reads length gets longer, it is easy to increase T in OLC, however, it is hard to increase K in DBG for several reasons including computational limitations. Therefore, OLC works better with longer reads to overcome repeats. When reads length and overlap length are long enough, the premasking and recovering of repeats steps can then be omitted in the OLC algorithm. Whereas longer read lengths help less with DBG, most of the current DBG software can only accept a *k*-mer size of up to 31 bp, with some of the latest versions going up to 127 bp [16, 18, 19]. The limited *k*-mer size in DBG has therefore limited it's potential to use long reads to overcome repeats. Although DBG has intrinsic high computational efficiency in dealing with repeats, it also has this major weakness of a low-efficiency in utilizing longer reads.

The assembly result varies significantly on genomes with various repeat contents. The genomic regions with continuous unique *k*-mers form the DBG initial contigs, and this process can be simulated on well-sequenced reference genomes (Table 2). We see that for relatively repeat-less genomes such as *Arabidopsis*, DBG algorithms can produce a good assembly result, however, for the relatively repeat-rich genomes such as maize, DBG algorithms produce very poor results. As outlined, increasing the *k*-mer size will be beneficial in resolving more repeats and resulting in longer initial contigs, however, this will further increase the consumption of computer resources as that is often already very significant when assembling large genomes. Taking the human genome for example, it often requires >100 G of memory and several days of running time [19]. A larger *k*-mer size also decreases the sensitivity for solving heterozygotes and sequencing errors, thus making it more difficult to assemble.

Scaffold linkage

To further resolve repeats and obtain a longer assembled sequence, the scaffold linkage step orders and orients the contigs into scaffolds using pair-end reads [6, 19, 45–48], which can be generated by most sequencing technologies and is often utilized by *de novo* sequencing techniques. If one read of pair-end reads is aligned to one contig and the other read is aligned to another contig, we assign a link between these two contigs. All the contigs along with their related links form the contig graph. Since the repeat contigs can have many links with other contigs, making it difficult to infer the relationship between contigs, one can first mask the repeat contigs and use unique contigs only to construct scafflolds and recover the repeat contigs in the end [7, 18, 19]. In OLC algorithms (without premasking repeats), repeat contigs can be identified by the number of reads they contain, because repeat contigs usually contains many more reads than a unique contig. In contrast, in the DBG algorithm, repeat contigs can be identified by the *k*-mer coverage depth of contig, which is usually higher than that of unique contigs. In both OLC and DBG algorithms, repeat contigs can also be inferred from the topology structure of

Table 2: Simulation of contig construction on reference genomes of 10 species

Species	Assembly version	Genome size (bp)	Total contig size (bp)	% Contig coverage	Contig N50 size (bp)	Contig N90 size (bp)
Escherichia coli	NC_000913	460 278	447 452	97.21	25 814	5802
Yeast	SGD1	12 070 899	11 264 319	93.32	29 274	8388
Human	NCBI36	2 832 359 855	2 117 618 241	74.77	2231	431
Chicken	WASHUC2	917 492 994	830 489 291	90.52	6735	1666
Fruitfly	BDGP5	120 290 946	112 001 873	93.11	29 729	5799
Arabidopsis	TIGR Release 5	**118 998 160**	**101 983 194**	**85.7**	**9540**	**1145**
Rice	IRGSP build 3	370 733 456	231 393 840	62.4	2185	366
Maize	prepublished	**2 033 474 566**	**325 408 406**	**16.0**	**1013**	**172**
Poplar	release v1.0	284 264 963	227 183 403	79.9	2580	420
Grape	assembly v1	290 237 009	195 259 295	67.3	2952	329

Note: We download the reference genome sequences from the official websites for each species genomes and excluded the inside gap N-sequences from the reference genome sequences first before doing analysis. The contig simulation is equivalent to find the continuous regions with unique *k*-mers. We use *k*-mer size 31 bp to construct contigs for all the species. Values in rows of *Arabidopsis* and Maize are highlighted with bold style, which represents relatively repeat-less and repeat-rich genome, separately.

the contig graph before scaffolding [19]. The unique contigs from either OLC or DBG algorithms form a non-redundant sequence blocks of the genome, and in theory there should be no overlap between any of these contigs. Links between neighboring contigs are determined by pairs of reads mapping to these contigs. As OLC contigs contain reads information, alignment from reads to contigs is unnecessary. In contrast, reads information are usually lost in the DBG contigs, so it is necessary to remap the paired reads onto the contigs. The DBG contigs are often much shorter than the OLC contigs, making the DBG scaffold linkage and gap closure more important and also more difficult [49].

Besides repetitive contigs, there are two other problems for scaffold construction. The first is the false mapping links that can be distinguished from real links by the supported pair number. Links with a pair number less than that of a threshold (often set to three), are often considered as unconfident links and excluded from the scaffold construction [19, 45]. The other problem is interleaving that is caused by short contigs. In theory, scaffold linkage with interleaving problems is classified as a NP-hard problem [46]. Simple interleaving structures can be identified on the contig graph and resolved by heuristic approaches (Figure 6). However, if the interleaving problem is complex, it will be difficult for heuristic approaches to resolve. For complex interleaving problems restricted to small local regions, we can select the best topology by the enumeration method utilizing pair-end supporting evidence, because the correct contig order will be supported best by all the read pairs. As *de novo* projects often generate pair-end reads with gradient insert sizes, to make scaffold construction easier and reduce interleaving, we suggest to construct scaffolds starting with short paired-end reads and then iterating the scaffolding process, step by step, using longer insert size paired-end reads [19]. The gap size can then be estimated by all the reads pairs mapping to the neighboring contigs.

Gap closure

To make the assembled sequence more complete, we need to close the leaving gaps [19, 50, 51]. After the scaffold linkage step, a set of non-redundant scaffold sequences is obtained which distribute separately along the genome. The gaps between scaffolds are called out-gaps that are often large and cannot be crossed by any pair-end reads; whereas the gaps inside scaffolds are called in-gaps that are often small and therefore can be crossed by available pair-end reads. Both the in-gap and out-gap can be formed because of either repeats (repeat gap) or uncoverage of sequencing (Lander–Waterman gap). In theory, the repeat gaps can be closed by retrieving the repeat reads and contigs which were not assembled in scaffolds and utilizing the pair-end relations [7, 19]. However, for Lander–Waterman gaps, the missing reads need to be generated by additional sequencing of the fragments localized in the gap regions, which are often created by PCR amplification [52]. After obtaining the necessary reads and contigs for a gap, the closure process is the same as local scale contig assembly, which can use either OLC or DBG algorithms [50, 51]. For most small and simple gaps, the local assembly can be relatively easily completed; however, for large and complex gaps, it is often difficult to resolve the local assembly. This means that most effort in gap closure has been mainly focussed on closing the small in-gaps. Note though, that because the sequences inside gaps are often repeats that tend to cause aligning problems, the accuracy of filled sequences are often relatively low and of questionable quality [22, 23].

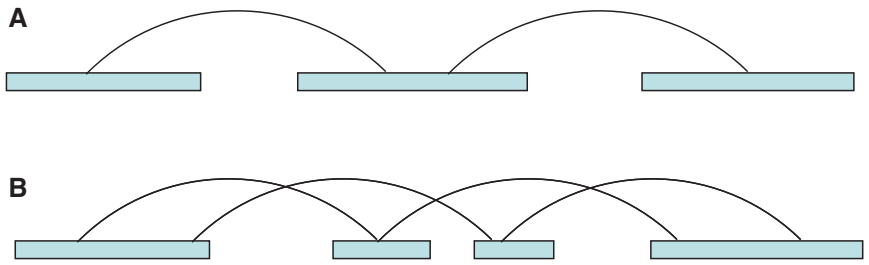

Figure 6: Models of scaffold linkage. (**A**) Example without interleaving. Each contig has only one in-going arc and one out-going arc (except at the border) and this situation is easy to resolve. (**B**) Example with interleaving. One contig can have more than one in-going arcs or out-going arcs that are often caused by small contigs. This situation is more difficult to resolve.

CONCLUSION AND DISCUSSION

Considering the computational consumption of time and memory, the OLC algorithm is more suitable for the low-coverage long reads, whereas the DBG algorithm is more suitable for high-coverage short reads and especially for large genome assembly. It should be noted though, that for both OLC and DBG algorithms, the assembly results may vary significantly among different genomes and sequencing technologies. Given a specified species genome, longer reads and lower sequencing errors will be beneficial for the assembly; whereas for any specific sequencing technology, fewer repeats and lower heterozygosity will be beneficial for the assembly process. Besides OLC and DBG algorithm, the application of another algorithm: string graph in *de novo* assembly, has also been studied in recent years [53].

As sequencing by second-generation technologies has got progressively cheaper and cheaper, more and more genome projects have moved towards short-read *de novo* assembly. Sequencing cost has become less of a limitation for genomics research, but the bioinformatics has conversely grown more important than ever before. In May 2011, as illumina (the most popular second-generation technology) launched the V3 sequencing kit for its HiSeq machine, its throughput (pair-end 100 bp) has been elevated to 600 Gb/run compared to 200 Gb/run in 2010, and the price of its personal genome sequencing service (40× coverage, 120 G data) has been reduced to 5000$ compared with 15 000 in 2010 (www.illumina.com). Due to this unmatched accessibility, the number of researchers using second-generation technologies has rapidly grown, and the debates and competition surrounding short-read *de novo* assembly is likely to carry on for several years in future, accompanied by further improvements of both sequencing technologies and assembly algorithms.

If longer reads are likely to come in the future, where should assembly algorithms go? In June 2011, Roche/454 launched its latest machine with read lengths of up to 800 bp, and a reduced cost of one-third of its original level (www.454.com). Another similar technology is Ion Torrent, aiming to be able to achieve a 400 bp read length and 1 G/run throughput by the year 2012 (www.iontorrent.com). We seem to have arrived at a turning point for *de novo* assembly, because the cheap long reads have significant advantages over short reads in the *de novo* assembly of large genomes. If long reads become as cheap as and accurate as short reads, then long reads will certainly become the only option. However, the current Roche/454 sequencing is still about 100 times more expensive than Illumina/Solexa technologies, which has limited its large-scale application to date. Ion Torrent may be cheaper but until new chips become available, it is unlikely to be able to compete with Illumina in the near future. A further limitation is that both Roche/454 and Ion torrent produce insertion/deletion errors in polymer regions, a deficiency compared with Illumina/Solexa. A practical short term solution is to do hybrid assembly using both Roche/454 and Illumina/Solexa reads, for example, one can use the combination of less than 10× Roche/454 reads and more than 30× Illumina/solexa reads. A growing number of software has began to support the hybrid assembly approach, such as Newbler [12] and CABOG [54]. The key point is to design algorithms that can distinguish data characteristics from various types of sequencing technologies, as well as combine the advantages of different technologies and overcome the deficiencies of each other.

Entering the second decade of 21st century, high-quality genome sequences for many species are still in great demand by the genomics field. Besides the second-generation sequencing technologies, there are many other new technologies helpful for *de novo* sequencing, including the single molecular sequencing PacBio (www.pacificbiosciences.com) and the Optical Mapping physical technology OpGen (www.opgen.com), which has recently entered the market. PacBio produces extreme long reads (1–10 kb) but with a high error rate (15–20%), whereas OpGen can generate 100 kb–1 Mb length physically linked markers, which facilitates the physical map construction. Driven by the development of these technologies, we see a bright future for *de novo* assembly of large genomes, with the standard likely to be raised from draft sequence to chromosome level sequence and from haploid sequence to diploid sequence. The development of assembly algorithms is tied closely to the development of sequencing technologies. The history has turned from OLC for Sanger sequencing, to DBG for second-generation sequencing, and the future will likely lead back to OLC for long reads sequencing. In the next few years, short reads assembly and long reads assembly may co-exist and both the OLC and DBG algorithms will be improved continuously. Small simple

genomes can be assembled well with pure short reads, middle difficulty genomes can use the hybrid assembly method using both long and short reads, whereas large complex genomes will rely more on long reads assembly. As outlined here, it is clear that sequencing technologies and assembly algorithms will change rapidly over the next few years, and assembly will get easier and better as technologies continue improve.

Key Points

- High-quality genome sequences for many species are still strongly desired by the genomics community. With the rapid development of sequencing technologies and assembly algorithms, we have seen practical improvements and a bright future lies ahead.
- There are two major types of assembly algorithms: OLC and DBG; both of them are in accordance with Lander–Waterman model, but suit the assembly of different read lengths and sequencing depths, and have significant differences in computational efficiency.
- How well a genome can be assembled depends not only on sequencing technologies such as read length and sequencing error rate, but also on the characteristics of the genome, including repeat and the heterozygosity rate of the sequenced sample.

Acknowledgements

We are grateful to Zhiwen Wang, Linfeng Yang, Zhen Yue, Yan Chen, Yinlong Xie, Yunjie Liu, Ruibang Luo, and many others at BGI-SZ, for their helpful discussions and suggestions. The authors would also like to thank Scott Edmunds for assistance revising the language in this manuscript.

FUNDING

This work was supported by the Basic Research Program Supported by Shenzhen City (grants JC2010526019), and the Key Laborabory Project Supported by Shenzhen City (grants CXB200903110066A; CXB201108250096A), and Shenzhen Key Laboratory of Gene Bank for National Life Science.

References

1. Flicek P, Birney E. Sense from sequence reads: methods for alignment and assembly. *Nat Methods* 2009;**6**:S6–12.
2. Miller JR, Koren S, Sutton G. Assembly algorithms for next-generation sequencing data. *Genomics* 2010;**95**: 315–27.
3. Schatz MC, Delcher AL, Salzberg SL. Assembly of large genomes using second-generation sequencing. *Genome Res* 2010;**20**:1165–73.
4. Xia Q, Guo Y, Zhang Z, *et al*. Complete resequencing of 40 genomes reveals domestication events and genes in silkworm (Bombyx). *Science* 2009;**326**:433–6.
5. Wang J, Wang W, Li R, *et al*. The diploid genome sequence of an Asian individual. *Nature* 2008;**456**:60–5.
6. Batzoglou S, Jaffe DB, Stanley K, *et al*. ARACHNE: a whole-genome shotgun assembler. *Genome Res* 2002;**12**: 177–89.
7. Myers EW, Sutton GG, Delcher AL, *et al*. A whole-genome assembly of Drosophila. *Science* 2000;**287**:2196–204.
8. Huang X, Madan A. CAP3: A DNA sequence assembly program. *Genome Res* 1999;**9**:868–77.
9. Huang X, Yang SP. Generating a genome assembly with PCAP. *Curr Protoc Bioinformatics* 2005. Chapter 11:Unit11.3.
10. de la Bastide M, McCombie WR. Assembling genomic DNA sequences with PHRAP. *Curr Protoc Bioinformatics* 2007. Chapter 11:Unit11.4.
11. Mullikin JC, Ning Z. The phusion assembler. *Genome Res* 2003;**13**:81–90.
12. Marcel Margulies1 ME, William EA, Said A, *et al*. Genome sequencing in open microfabricated high density picoliter reactors. *Nature* 2005;**437**:376–80.
13. Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. *J Comput Biol* 1995;**2**:291–306.
14. Pevzner PA, Tang H, Waterman MS. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci USA* 2001;**98**:9748–53.
15. Chaisson MJ, Brinza D, Pevzner PA. De novo fragment assembly with short mate-paired reads: does the read length matter? *Genome Res* 2009;**19**:336–46.
16. Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res* 2008;**18**:821–9.
17. Simpson JT, Wong K, Jackman SD, *et al*. ABySS: a parallel assembler for short read sequence data. *Genome Res* 2009;**19**: 1117–23.
18. Gnerre S, Maccallum I, Przybylski D, *et al*. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci USA* 2011;**108**: 1513–18.
19. Li R, Zhu H, Ruan J, *et al*. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*;**20**:265–72.
20. Huang S, Li R, Zhang Z, *et al*. The genome of the cucumber, Cucumis sativus L. *Nat Genet* 2009;**41**: 1275–81.
21. Li R, Fan W, Tian G, *et al*. The sequence and de novo assembly of the giant panda genome. *Nature* 2010;**463**: 311–7.
22. Birney E. Assemblies: the good, the bad, the ugly. *Nat Methods* 2011;**8**:59–60.
23. Alkan C, Sajjadian S, Eichler EE. Limitations of next-generation genome sequence assembly. *Nat Methods* 2011; **8**:61–5.
24. Staden R. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res* 1979;**6**:2601–10.
25. Wendl MC. A general coverage theory for shotgun DNA sequencing. *J Comput Biol* 2006;**13**:1177–96.
26. Li X, Waterman MS. Estimating the repeat structure and length of DNA sequences using L-tuples. *Genome Res* 2003; **13**:1916–22.
27. Marcais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of *k*-mers. *Bioinformatics* 2011;**27**:764–70.

28. Xia Q, Zhou Z, Lu C, *et al*. A draft sequence for the genome of the domesticated silkworm (Bombyx mori). *Science* 2004;**306**:1937–40.
29. Yu J, Wang J, Lin W, *et al*. The genomes of Oryza sativa: a history of duplications. *PLoS Biol* 2005;**3**:e38.
30. Lander ES, Waterman MS. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics* 1988;**2**:231–9.
31. Gregory TR, Nicol JA, Tamm H, *et al*. Eukaryotic genome size databases. *Nucleic Acids Res* 2007;**35**:D332–8.
32. Bergman CM, Quesneville H. Discovering and detecting transposable elements in genome sequences. *Brief Bioinform* 2007;**8**:382–92.
33. She X, Cheng Z, Zollner S, *et al*. Mouse segmental duplication and copy number variation. *Nat Genet* 2008;**40**: 909–14.
34. Metzker ML. Sequencing technologies - the next generation. *Nat Rev Genet* 2010;**11**:31–46.
35. Aird D, Ross MG, Chen WS, *et al*. Analyzing and minimizing PCR amplification bias in Illumina sequencing libraries. *Genome Biol* 2011;**12**:R18.
36. Rothberg JM, Hinz W, Rearick TM, *et al*. An integrated semiconductor device enabling non-optical genome sequencing. *Nature* 2011;**475**:348–52.
37. Bentley DR, Balasubramanian S, Swerdlow HP, *et al*. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* 2008;**456**:53–9.
38. Kelley DR, Schatz MC, Salzberg SL. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol* 2010;**11**:R116.
39. Schroder J, Schroder H, Puglisi SJ, *et al*. SHREC: a short-read error correction method. *Bioinformatics* 2009;**25**: 2157–63.
40. Ilie L, Fazayeli F, Ilie S. HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*;**27**: 295–302.
41. Kao WC, Chan AH, Song YS. ECHO: A reference-free short-read error correction algorithm. *Genome Res* 2011;**21**: 1181–92.
42. Chin FY, Leung HC, Li WL, *et al*. Finding optimal threshold for correction error reads in DNA assembling. *BMC Bioinformatics* 2009;**10**(**Suppl. 1**):S15.
43. Yang X, Dorman KS, Aluru S. Reptile: representative tiling for short read error correction. *Bioinformatics* 2010;**26**: 2526–33.
44. Wang J, Wong GK, Ni P, *et al*. RePS: a sequence assembler that masks exact repeats identified from the shotgun data. *Genome Res* 2002;**12**:824–31.
45. Boetzer M, Henkel CV, Jansen HJ, *et al*. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics* 2011; **27**:578–9.
46. Huson DH. The greedy path-merging algorithm for contig scaffolding. *J ACM* 2002;**49**:603–15.
47. Zerbino DR, McEwen GK, Margulies EH, Birney E. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS One* 2009;**4**:e8407.
48. Dayarian A, Michael TP, Sengupta AM. SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics* 2010;**11**:345.
49. Pop M. Genome assembly reborn: recent computational challenges. *Brief Bioinform* 2009;**10**:354–66.
50. Koren S, Miller JR, Walenz BP, *et al*. An algorithm for automated closure during assembly. *BMC Bioinformatics* 2010;**11**:457.
51. Tsai IJ, Otto TD, Berriman M. Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. *Genome Biol* 2010;**11**:R41.
52. Herve Tettelin DR, Simon K, Hoda K, Steven LS. Optimized multiplex PCR: efficiently closing a whole-genome shotgun sequencing project. *Genomics* 1999;**62**: 500–7.
53. Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics* 2010;**26**: i367–73.
54. Miller JR, Delcher AL, Koren S, *et al*. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics* 2008;**24**: 2818–24.