# 3

# Next-Generation Sequencing Technologies and the Assembly of Short Reads into Reference Genome Sequences

*Ning Li, Xiaozhu Wang and Zhanjiang Liu*

## Introduction

In this chapter, the various sequencing technologies are first introduced, and then the methods for the assembly of sequences generated using various sequencing platforms are presented.

## Understanding of DNA Sequencing Technologies

The start of the Human Genome Project (HGP) 30 years ago marks the start of a new era in genomics. For the most part, the genomics revolution has been driven by advances in DNA sequencing technologies, including the development of automated DNA sequencers and powerful data-processing algorithms. Sanger sequencing (Sanger *et al*., 1977), considered the "first-generation" DNA sequencing technology, dominated the field for almost 20 years till the mid-1990s. Since then, in the last 15–20 years, rapid progress was made in sequencing technologies. The new set of sequencing technologies since has been collectively known as the "next-generation" sequencing (NGS) technologies. Three platforms of NGS were initially available in the market: the Roche/454 FLX, the Illumina sequencers, and the Applied Biosystems SOLiD System. With years of competition in the sequencing market, the Illumina platform is now among the most popular platforms. In addition, several additional sequencing platforms were introduced to the market, such as Helicos Heliscope, Pacific Biosciences SMRT, Ion Torrent Personal Genome Machine (PGM), and Oxford Nanopore MinION sequencers.

Compared to Sanger sequencing, next-generation sequencing produces massive amounts of data rapidly and cheaply (Mardis, 2013). A typical cloning step in Sanger method is replaced in next-generation sequencing by adding universal adapters to the DNA fragments (Mardis, 2011). There is also no need to perform sequencing reactions in microtiter plate wells. Instead, the library fragments are amplified *in situ* on a solid surface, which is covalently derivatized with adapter sequences. Another difference is that next-generation sequencing conducts sequencing and detection simultaneously (Dolník, 1999). These steps allow hundreds of millions to billions of reaction loci to

*Bioinformatics in Aquaculture: Principles and Methods,* First Edition. Edited by Zhanjiang (John) Liu.

be sequenced per instrument run. Due to the signal-to-noise ratio, the read length of next-generation sequencing is generally short. However, as detailed in the following text, long reads can now be achieved with the third generation of sequencers such as PacBio. The shorter read length is a major shortcoming of next-generation sequencing for large and complex genome sequencing (Alkan, Sajjadian & Eichler, 2011). To overcome this disadvantage, various tactics such as paired-end and mate-pair sequencing can be applied, which help the assembly of short sequences into contigs and scaffolds, as detailed in the following text.

Based on the chemistry, next-generation sequencing technologies can be classified into five groups (Shendure *et al*., 2004): pyrosequencing (454 Life Sciences), sequencing by synthesis (Illumina), sequencing by ligation (SOLiD), semiconductor sequencing (Ion Torrent), and single-molecule real-time sequencing (PacBio, Helicos and Oxford Nanopore). Here, we provide some introduction of these platforms because their characteristics are related to the design of a genome-sequencing project, and to the sequence assembly strategies.

#### 454 Life Sciences

Roche 454 sequencing system was the first next-generation sequencing platform available in the market (Margulies *et al*., 2005). It was founded by Jonathan Rothberg in 2005, and bought by Roche Diagnostics in 2007. In this platform, libraries are prepared either paired-end or mate-pair to obtain a mixture of short, adapter-flanked fragments. The major characteristic of the 454 system is emulsion PCR (Dressman *et al*., 2003), which is used to generate clonal sequencing features with amplicons captured to the surface of 28 μm beads. After amplification, the emulsion shell is broken, and successfully amplified beads are enriched through binding with streptavidin magnetic beads. A sequencing primer is hybridized to the universal adapter at the appropriate position and orientation, that is, immediately adjacent to the start of unknown sequences. The Pico Titer Plate is loaded with one fragment carrying bead per well and smaller beads with the enzymes necessary for sequencing and for packing. Sequencing is performed by the pyrosequencing method (Ronaghi *et al*., 1996). When ATP drives the luciferase-mediated conversion of luciferin to oxyluciferin, a burst of light is generated in amounts that are proportional to the amounts of ATP. Light is captured by CCD camera and recorded as a peak "proportional" to the number of nucleotides incorporated.

Relative to other next-generation platforms, the key advantage of the 454 platform is its long read length (Metzker, 2010). For example, the read length generated from a 454 instrument can reach up to 700 bp (base pairs). A major limitation of the 454 technology relates to homopolymer repeats (multiple bases of the same identity). Because there is no terminating moiety preventing multiple consecutive incorporations at a given cycle, the length of all homopolymer repeats must be inferred from the signal intensity. As a consequence, the dominant error type for the 454 system is insertion/deletion, rather than base substitution (Fox *et al*., 2014). Other disadvantages include relatively low sequence coverage and relatively high price. In October 2013, 454 Life Sciences announced the decision to phase out their pyrosequencing-based instruments (the GS Junior and GS FLX+) in mid-2016 (http://www.biocompare.com/Editorial-Articles/155411-Next-Gen-Sequencing-2014-Update/).

**Illumina Genome Analyzer**

Previously known as "the Solexa", the Illumina platform was originally introduced by Turcatti and colleagues (Fedurco *et al*., 2006; Turcatti *et al*., 2008). This platform has been the most popular next-generation sequencing platform in the market since it took over the market from 454 Life Science in 2012. Illumina Genome Analyzer is a sequencing-by-synthesis-based platform (Mardis, 2013). Libraries can be constructed by any method as long as a mixture of adapter-flanked fragments up to 100–300 bp in length can be obtained. Sequences are amplified through cluster generation, which is performed on the Illumina Cluster Station (or Cbot) (Fedurco *et al*., 2006). In brief, DNA fragments are individually attached to the flow cell by hybridizing to oligos on its surface complementary to the ligated adapters. DNA molecules are amplified by bridge PCR, and then reverse strands are cleaved and washed away in the end. After cluster generation, the amplicons are single-stranded (linearization), and a sequencing primer is hybridized to a universal sequence flanking the region of interest. Clusters are sequenced simultaneously, and each cycle of sequence interrogation consists of single-base extension with a modified DNA polymerase and a mixture of four nucleotides. After each synthesis step, the clusters are excited by a laser, which causes fluorescence of the last incorporated base fluorescence label. The fluorescence signal is captured by a built-in camera, producing images of the flow cell. Then the blocking groups are removed, allowing addition of the next base. This process continues for every base to be sequenced. The read length has increased from the original 25 bp (single-end reads) to the current 150 bp (paired-end reads) using the Illumina HiSeq 2500 instrument. The dominant error type is base substitution, rather than insertion/deletion as those in 454 Life Science (Fox *et al*., 2014). The average raw error rate of most Illumina reads is approximately 0.5% at best, but higher accuracy bases with error rates of 0.1% or less can be identified through quality metrics associated with each base calling (Mardis, 2013). Low expense, high throughput, and high coverage have made Illumina secure roughly 60% of the next-generation sequencing market. However, its major disadvantage is the relatively short read length, making it less effective for complex *de novo* assembly projects (Metzker, 2010). HiSeq 3000/4000 instruments were introduced to the market in early 2015. The dual flow-cell HiSeq 4000 can generate up to 1.5 TB of 150 bp pair-end reads (750 GB for HiSeq 3000). The major difference between the HiSeq 4000/3000 and earlier models is their flow-cell design. Instead of using non-patterned flow cell in which sequencing clusters could form anywhere on the surface, the HiSeq 4000/3000 instruments use a patterned flow cell in which sequencing clusters are restricted to 400 nm wells spaced 700 nm apart (http://www.biocompare.com/Editorial-Articles/171872-Next-Gen-DNA-Sequencing-2015-Update/).

**SOLiD**

The SOLiD platform is based on sequencing by oligo ligation. It was invented by Shendure and colleagues at Church's lab and later acquired by Applied Biosystems in 2006 (now owned by Thermo Fisher Scientific). Same as the 454 platform, clonal sequencing features are generated by emulsion PCR, with amplicons captured to the surface of 1 μM paramagnetic beads (Shendure & Ji, 2008). After emulsion PCR, beads bearing templates are selectively recovered, and then immobilized to a solid surface to generate a dense, disordered array. Sequencing is driven by a DNA ligase: first, a

universal primer is hybridized and ligated to the array of amplicon-bearing beads, and then a set of four fluorescently labeled di-base probes compete for ligation to the sequencing primer (Housby & Southern, 1998; Macevicz, 1998; McKernan *et al*., 2012; Shendure *et al*., 2005). Each cycle of sequencing involves the ligation of a degenerate population of fluorescently labeled octamers. After the first ligation cycle, the extended primer is denatured to reset the system with a primer complementary to the $n - 1$ position for a second round of ligation cycles. Multiple cycles of ligation, detection, and cleavage are performed, and the number of cycles determines the eventual read length.

The efficiency and accuracy of SOLiD is enhanced by the use of two-base encoding, which is an error-correction scheme in which two adjacent bases, rather than a single base, are associated with one color (McKernan *et al*., 2012). Each base position is then queried twice, so that miscalls can be more readily identified. High accuracy up to 99.99% can be achieved through this step. Similar to Illumina, the dominant error type is base substitution (Fox *et al*., 2014). With its high levels of accuracy, SOLiD is the best for genome resequencing or SNP capture. Short read length is the major disadvantage for this platform. An additional disadvantage, such as that with the Roche 454 system, is that emulsion PCR can be cumbersome and technically challenging.

#### Helicos

The Helicos sequencer (Harris *et al*., 2008) is based on the work of Quake and colleagues (Braslavsky *et al*., 2003). It is the first commercial machine to sequence a single DNA molecule rather than one copied many times. Libraries are prepared by random fragmentation and poly-A tailing, then captured by hybridization to surface-tethered poly-T oligomers to yield a disordered array of primed single-molecule sequencing templates. DNA polymerase and a single species of fluorescently labeled nucleotide are added in each cycle, resulting in template-dependent extension of the surface-immobilized primer–template duplexes. A highly sensitive fluorescence detection system is used to directly interrogate individual DNA molecules via sequencing by synthesis. After collection of images tiling the full array, chemical cleavage and release of the fluorescent label allow the subsequent cycle of extension and imaging (Shendure & Ji, 2008).

Homopolymer repeat runs are the major problem of Helicos, almost like the situation of the Roche 454 system. This is because, with Helicos, there is no terminating moiety present on the labeled nucleotides. However, the problem is less serious as compared to Roche 454 because individual molecules are being sequenced, and the problem can be mitigated by limiting the rate of incorporation events. The dominant error type is insertion/deletion due to the incorporation of contaminated, unlabeled, or non-emitting bases (Fox *et al*., 2014).

#### Ion Torrent

Ion Torrent, founded by Rothberg and collogues, was commercialized in 2010 and later purchased by Life Technologies$^{TM}$ Corp, which in turn was acquired by Thermo Fisher Scientific in February 2014. Thus, Thermo Fisher Scientific now owns both SOLiD and Ion Torrent. The Ion Torrent platform is based on the detection of the release of hydrogen ions, a by-product of nucleotide incorporation, as quantitated by changes in pH through a novel coupled silicon detector (Rothberg *et al*., 2011; Mardis, 2013). Genomic DNA is fragmented, ligated to adapters, and then clonally amplified onto beads using

emulsion PCR. After emulsion breaking, template-bearing beads are enriched through a magnetic-bead-based process and primed for sequencing. This mixture is then deposited into the wells of an Ion Chip, a specialized silicon chip designed to detect pH changes within individual wells of the sequencer as the reaction progresses stepwise. The upper surface of the Ion Chip is designed as a microfluidic conduit to deliver the reactants needed for the sequencing reaction. The lower surface of the Ion Chip interfaces directly with a hydrogen ion detector to translate the released hydrogen ions from each well into a quantitative readout of nucleotide bases in each reaction step.

Same as 454, Helicos, and PacBio, the major error type of Ion Torrent sequencing is caused by insertion/deletion due to homopolymer repeats, although base substitution does occur, but at a low frequency (Fox *et al*., 2014). Overall, the error rate of Ion Torrent on a per-read basis averages approximately 1% (Mardis, 2013). The average read length obtained by the Ion Torrent has increased from originally 50 bp to 400 bp, produced as single-end reads. Throughput has increased from 10 MB per run to 1 GB per run, on average. Reaction volume miniaturization and mass production of the Ion Chips make this platform relatively fast and inexpensive, and hence ideal for small-scale applications or small laboratories that do not require large datasets.

**PacBio**

PacBio was commercialized by Pacific Biosciences in 2010. It is based on single-molecule sequencing, and generally regarded as third-generation technology. It is a successful combination of nanotechnology with molecular biology and highly sensitive fluorescence detection to achieve single-molecule DNA sequencing (Eid *et al*., 2009). On this platform, by applying a special nanotechnology called *zero-mode waveguide* (ZMW), tens to thousands of ZMWs can be fitted in a light-focusing structure as a regular array (Mardis, 2013). With fragmented genomic DNA, after polishing their ends, the hairpin adapters are ligated onto the ends of these fragments. Once ligated and denatured, these molecules can form DNA circles, and at the same time reaction by-products should be removed before binding library fragments with DNA polymerase molecules through a primer complementary to the adapter. Then, these mixed molecules are deposited onto the surface of the ZMW chip (SMRT Cell). The instrument excitation/detection optics can be trained on the bottom of each ZMW, where the polymerase attaches. Once fluorescent nucleotides are added to the chip surface, the instrument optics collect the sequencing data by actively monitoring each ZMW to record the incorporating fluorescent molecules dwelling in the active site as identified by their emission wavelength. Here, the highly sensitive fluorescence detection is aimed at single-molecule detection of fluorescently labeled nucleotide incorporation events in real time, as the DNA polymerase is copying the template.

Depending on the size of the library fragments and the time of data collection, this platform produces some of the longest average read lengths available in the market. The average read length is currently up to 10 KB, while some reads can reach 85 KB using a new sequencing chemistry called P6-C4. However, an overall high error rate is the major disadvantage of PacBio. On a per-read basis, the error rate is approximately 10–15% (Mardis, 2013). The dominant error type is insertion/deletion, although a small proportion of base substitution does occur (Fox *et al*., 2014). This platform is mainly used in assembling bacterial genomes and gap filling for sequencing projects with large genomes.

### Oxford Nanopore

Oxford Nanopore platforms (the GridION and MinION), first announced by a UK startup Oxford Nanopore Technologies in 2012 (Eisenstein, 2012), are label-free and light-free exonuclease sequencing platforms. Library preparation is similar to that for other next-generation sequencing applications, including DNA shearing, end repair, adapter ligation, and size selection. The library must be conditioned by addition of a motor protein, and then mixed with buffer and a "fuel mix" and loaded directly into the sequencer. As the sequencer runs, DNA is chewed up base by base by exonuclease tethered to a nanopore, and each base generates a different electric current pattern. Base calling takes place in real time using Oxford Nanopore's Metrichor cloud service. Data can be analyzed using either 1D or 2D workflows. The ingenious 2D workflow uses a hairpin adapter, which links the top and bottom strands of double-stranded DNA into one strand. The base caller recognizes the hairpin sequence and aligns both strands of the template molecule, with the goal of improving sequencing accuracy. The first device, MinION, a USB device, was available through the MinION Access Programme (MAP) (Loman & Watson, 2015). PromethION and GridION are still under development (http://www.biocompare.com/Editorial-Articles/171872-Next-Gen-DNA-Sequencing-2015-Update/). For MinION, with continuing updates of this instrument, especially the so-called "2D" reads, more and more positive publications have emerged to give credit to the accuracy and satisfying read length of this platform (Ammar *et al*., 2015; Jain *et al*., 2015). Its success in the future awaits validation of large sequencing projects.

## Preprocessing of Sequences

During the library preparation and sequencing process, a variety of sequence artifacts, including adapter/primer contamination, base calling errors, and low-quality sequences, will negatively affect the quality of raw data for downstream analyses. Therefore, these quality issues necessitate better programs for quality control and preprocessing of all the raw data (Patel & Jain, 2012; Schmieder & Edwards, 2011; Trivedi *et al*., 2014).

### Data Types

*Single-end data:* Single-end sequencing only sequences DNA fragments from one direction, so all the fragments in the library are only sequenced one end, which means there is only one data output file from each sequencing template.

*Paired end data:* Paired-end sequencing sequences the same DNA fragment from both directions, which produces a pair of reads for each fragment. The genomic distance between these two reads was chosen during the size selection process and is used to constrain assembly solutions (Nagarajan & Pop, 2013). As a result, there are two output files for each of the sequencing templates in the form of paired-end data.

*Mate pair:* Mate pair sequencing requires long-insert-size paired-end libraries that are useful for a number of sequencing applications, such as *de novo* assembly and genome finishing. With large inserts, library preparation is more complicated than with short-insert libraries. Genomic DNA is fragmented to an approximate

size range, usually 3–5 KB, 6–8 KB, or 7–10 KB. Fragments are then end-repaired with biotinylated nucleotides and go through size selection more specifically. After circularization, non-circularized DNA is removed by digestion, while those circularized fragments are then subjected to a second round of fragmentation, and the labeled fragments (corresponding to the ends of the original DNA ligated together) are purified by biotin–streptavidin cleanup. Purified fragments are end-repaired and ligated to sequencing adapters (http://www.illumina.com/technology/next-generation-sequencing/mate-pair-sequencing_assay.html).

The length of the initial size selection determines the mate pair gap size expected during the scaffolding of contiguous sequences (contigs). Combining mate pair data with data from short-insert-size paired-end reads provides a powerful tool for good assembly and high sequencing coverage across the genome. However, the ligation process may create chimeric molecules, which is a major source for artificial structural variations.

#### Quality Control

The quality control process of sequencing data typically involves assessing the quality metrics of the raw reads. One of the most popular programs for performing this is `FastQC` (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/). It is a useful tool with a graphical user interface (GUI), and is widely used as an initial checkpoint to ensure the overall quality of the sequence datasets. Use:

```
fastqc PATH_TO_SEQUENCE_FILE
```

The evaluating result is named "FILENAME_fastqc.html", which can be opened in a browser. There are 12 tabs displaying 12 quality metrics, with the exclamatory mark representing "warning" and the cross representing "failure".

Some important quality metrics are: the *basic statistics*, which helps in the selection of assembly programs; the *per-base sequence quality*, which shows the overall quality distribution and average quality from the first base to the last base, helping to set the quality threshold for quality trimming; and the *per-base sequence content*, which may imply the existence of primers or adapters based on the non-random-sequence content distribution at the beginning. Furthermore, the last three tabs also function closely to display the primers or adapters.

#### Trimming

Based on the results from `FastQC`, adapter/primer trimming and quality trimming should be carried out before downstream analyses. Take mapping programs as an example: most popular mapping tools use global matching algorithms, trying to map the whole length of reads to reference genomes if the number of mismatches is below the user-defined threshold. If the adapter/primer sequences are not removed completely, the reads containing the adapter/primer sequences may not be mapped to reference genomes. Similarly, quality trimming is also important (Kong, 2011; Lindgreen, 2012). Besides these two trimming types, there are also ambiguity trimming (to remove stretches of Ns), length trimming (to remove reads shorter or longer than a specified length), and base trimming (to remove a specified number of bases at either 5' or 3' end of the reads). The choices of trimming types and trimming threshold depend on the needs of further analyses.

A number of trimming programs have been developed, such as Skewer (Jiang *et al*., 2014), Trimmomatic (Bolger, Lohse & Usadel, 2014), Btrim (Kong, 2011), AdapterRemoval (Lindgreen, 2012), and cutadapt (Martin, 2011). Here, we will describe the detailed usage of cutadapt (https://pypi.python.org/pypi/cutadapt).

***Base Trimming*** The parameter `-u` is used to remove a specified number of bases from the beginning or end of each read. If the value is positive, the bases are trimmed from the beginning; and if the value is negative, the bases are trimmed from the end. Base trimming is always performed before adapter trimming. For example, the following command removes five bases from the end of each read. The trimmed output is stored in the "output.fastq" file.

```
cutadapt -u -5 -o output.fastq input.fastq
```

***Quality Trimming*** The parameter `-q` is used to trim low-quality ends from each read before adapter trimming. This step achieves the same goal with base trimming, so they usually do not appear in the same command.

```
cutadapt -q 10,15 -o output.fastq input.fastq
```

In this command, the value before the comma is the cutoff for the 5' end of each read, and the one after the comma is for the 3' end. If you only want to trim the 3' end, then use `-q` *15*. If you only want to trim the 5' end, then use 0 for the 3' end; for example, `-q` *10,0*.

***Adapter Trimming*** Adapter trimming is complex due to the location of the adapters. The parameter `-a` is set for trimming the 3' adapter, while `-g` is used for trimming the 5' adapter. If it is possible for adapters to appear in both ends of the reads, then `-b` will fix this kind of problem. No matter what parameter is used for trimming, it is followed by the adapter sequences.

For the 3' adapter, `-a` will remove the whole or partial matches, and also any sequence that may follow. For example, the sequence of interest is `SEQUENCE` and the adapter is `ADAPTER`. The reads before trimming and the trimmed reads may look like this:

```
SEQUEN                          -       SEQUEN
SEQUENCEADAP                    -       SEQUENCE
SEQUENCEADAPTER                 -       SEQUENCE
SEQUENCEADAPTERSOMETHING        -       SEQUENCE
ADAPTERSOMETHING                -       (all trimmed)
```

For the 5' adapter, `-g` will remove the whole or partial matches, and also the sequences preceding the adapter. Again, assume the sequence of interest is *SEQUENCE* and the adapter is *ADAPTER*. The reads and the trimmed reads may look like this:

```
ADAPTERSEQUENCE                 -       SEQUENCE
PTERSEQUENCE                    -       SEQUENCE
SOMETHINGADAPTERSEQUENCE        -       SEQUENCE
SOMETHINGADAPTER                -       (all trimmed)
```

For the 5' or 3' adapter, `-b` has a rule for deciding which part of the read to remove: if there is at least one base before the adapter, then the adapter itself and everything that

follows it are removed. Otherwise, only the adapter at the beginning is removed, and everything that follows it remains.

***Paired-end Reads Trimming*** The parameter *-p* is used to trim paired-end reads separately at the same time, avoiding running `cutadapt` twice. But it functions properly only when the number of reads is the same in both files and the read names before the first space match. Moreover, both of them are processed in the meantime to make sure they are synchronized. For example, if one read is discarded from one file, it will also be removed from the other file. Assume paired-end files "read_1.fastq" and "read_2.fastq" have adapters *ADAPTER_1* and *ADAPTER_2* separately. The trimming command may look like this:

```
cutadapt -a ADAPTER_1 -A ADAPTER_2 -o out_1.fastq -p
   out_2.fastq read_1.fastq read_2.fastq
```

The parameter *-A* is used to specify the adapter that `cutadapt` should remove in the second file. There are also the parameters *-G* and *-B*. All of them only work in the paired-end reads trimming.

***Length Trimming*** This process only functions after all the other types of trimming are completed. The trimmed reads that are shorter than a specified length are discarded using *-m*, while the ones that are longer than a specified length are discarded using *-M*.

Different kinds of parameters could be combined in a single command to perform a complete trimming process. Here is an example:

```
cutadapt -q 20,20 -a ADAPTER_1 -A ADAPTER_2 -g ADAPTER_3 -G
   ADAPTER_4 -m 30 -o out_1.fastq -p out_2.fastq
   read_1.fastq read_2.fastq
```

All the reads are first quality trimmed from both ends using 20 as the quality threshold, and then the 3' adapter *ADAPTER_1* and 5' adapter *ADAPTER_3* are trimmed from the first file "read_1.fastq". At the same time, the 3' adapter *ADAPTER_2* and 5' adapter *ADAPTER_4* are trimmed from the second file "read_2.fastq". After quality trimming and adapter trimming, the reads whose lengths are shorter than 30 are discarded in both "read_1.fastq" and "read_2.fastq". The final trimmed reads are kept in the "out_1.fastq" and "out_2.fastq" files.

A trimming report is printed after `cutadapt` has finished processing the reads. And then `FastQC` (described in the previous section) could be used to check the overall quality of the trimmed reads again to make sure all the issues have been addressed.

#### Error-Correction

After quality trimming, all the reads are above the quality threshold, but the good quality value does not mean the read is totally error free. That is the reason why we need an error-correction step prior to sequence assembly. Although many assembly programs have their error-correction modules in the packages, such as ALLPATHS-LG (Gnerre *et al*., 2011), many stand-alone error correction programs have been developed as well, such as `Quake` (Kelley, Schatz & Salzberg, 2010), `BLESS` (Heo *et al*., 2014), and `Lighter` (Song, Florea & Langmead, 2014). As errors are random and infrequent,

the reads containing an error in a specific position can be corrected by the majority of reads containing the correct base (Yang, Chockalingam & Aluru, 2013).

Here, we will introduce one stand-alone error-correction tool, `Lighter` (https://github.com/mourisl/Lighter) (Song *et al*., 2014). It is a fast and memory-efficient sequencing error-correction program without counting *k*-mers. Running this program is simple, without too many parameters. Assume we have a species with a genome size of 1 GB, and paired-end sequencing reads files "read_1.fastq" and "read_2.fastq" yielding a genome coverage of 70. Then the error-correcting command simply looks like:

```
lighter -r read_1.fastq -r read_2.fastq -k 17 1000000000
   0.1 -t 10 -od output
```

Here, `-r` is a required parameter to exhibit the input files. If the data set is a single-end read file, then only one `-r` is used; `-k` is followed by *k*-mer length, genome size, and alpha (alpha = 7/coverage, here 70); and the genome size does not need to be accurate. Instead, `-K` (capital K) can be used, followed by *k*-mer length and genome size. However, for `-K`, the genome size should be more accurate, because `Lighter` will go through the calculation to acquire the value of alpha. Considering running in parallel, `-t` could be used to set the number of threads. For each input file, `Lighter` will generate an error-corrected file in the directory "output", created by the parameter `-od`. In this case, the error-corrected files will be named "read_1.cor.fastq" and "read_2.cor.fastq". As for the determination of *k*-mer length, the performance of `Lighter` is not overly sensitive to it (Song *et al*., 2014).

## Sequence Assembly

The sequence assembly process is more like solving a jigsaw puzzle (Pop, 2009). Assembling small DNA fragments into whole genome sequences is similar to assembling small puzzle pieces into a whole picture. For jigsaw puzzles, complexity increases with the number of pieces, the size of the picture, and especially the ambiguous but similarly colored pieces. Similarly, the difficulty of the assembly process increases with the number of reads, the size of genomes, and the content of repetitive sequences.

#### Reference-Guided Assembly

After error-correction, the reads could be aligned to a known reference genome sequence or assembled *de novo*. Mapping reads to a known reference genome is also called *reference-guided assembly*. Two different aligners, BWA (Li & Durbin, 2009; 2010) and Stampy (Lunter & Goodson, 2011), are often utilized to do the mapping work. BWA consists of three algorithms: BWA-backtrack, BWA-SW, and BWA-MEM. The first one is designed for Illumina sequence reads of up to 100 bp, while the other two are for longer sequences ranging from 70 bp to 1 MB (http://bio-bwa.sourceforge.net/bwa.shtml). Stampy allows variation during mapping (particularly insertion/deletions), and thus can be used if you do not have a closely related reference genome sequence. In any case, reference genome sequences are required for mapping, and therefore it is not applied to numerous species assemblies without references. For most aquaculture species, reference genome sequences are not yet available, and, therefore, we will focus on *de novo* assembly.

### *De Novo* Assembly

"*De novo*" literally means "from the beginning" in Latin. For genome projects, *de novo* assembly refers to assembly entirely based on reads generated from sequencing instruments. In other words, no prior assembled sequence data is utilized during assembly, such as pre-assembled sequences from genomes, transcripts, and proteins (Rodríguez-Ezpeleta, Hackenberg & Aransay, 2012). The software packages designed for sequence assembly are called *assemblers*. All assemblers rely on the simple assumption that highly similar DNA fragments originate from the same position within a genome (Nagarajan & Pop, 2013).

There are mainly three categories of assemblers: greedy graph-based assemblers, overlap-layout-consensus (OLC) assemblers, and de Bruijn graph (DBG) based assemblers.

#### Graph

A *graph* is the representation of a set of objects (also called *vertices* or *nodes*) where some pairs of objects are connected by links (also called *edges* or *arcs*). If the edges are only traversed in one direction between its connected nodes, they are called *directed edges*. Each directed edge connects a source node and a sink node. In a path where edges visit nodes in some order, the sink node of one edge is also the source node for any subsequent node. However, if the edges may be traversed in either direction, then they are called *undirected edges*. A graph can be conceptualized as a set of balls representing its nodes with connecting arrows representing its edges (Miller, Koren & Sutton, 2010).

When building graphs from real data, there are several complicated situations (Miller *et al*., 2010). Spurs are short, dead-end subpaths diverging from the main path, which is shown in Figure 3.1A. They are mainly induced by sequencing error toward one end of a read. In a frayed-rope pattern shown in Figure 3.1B, paths converge and then diverge, which are induced by repeat sequences. Paths that diverge and then converge as shown in Figure 3.1C form bubbles. They are induced by sequencing error in the middle of a read and polymorphism in the genome. Cycles occur when paths converge on themselves, which are also induced by repeat elements in the genome.

#### Greedy Assemblers

Early assemblers were based on the greedy algorithm, such as phrap and TIGR Assembler. The term "greedy" refers to the fact that the decisions made by the algorithm optimize a local objective function (Pop, 2009), which means the assembler always joins the reads that overlap best and end until no more reads or contigs can be joined. The scoring function for "overlap best" commonly measures the number of matches in the overlap and level of identity (percentage of base pairs shared by the two reads) between the reads within the overlapping region (Pop, 2009). As an optimization, the greedy algorithm may utilize just one overlap for each read end and then may discard each overlap immediately after the highest-scoring reads were recruited to extend the contig. Consequently, it can get stuck at local maxima if the current contig is extended by reads that may have helped other contig extensions (Miller *et al*., 2010).

Since overlaps induced by repetitive sequences may score higher than true overlaps, the greedy algorithm needs mechanisms to avoid the incorporation of false-positive

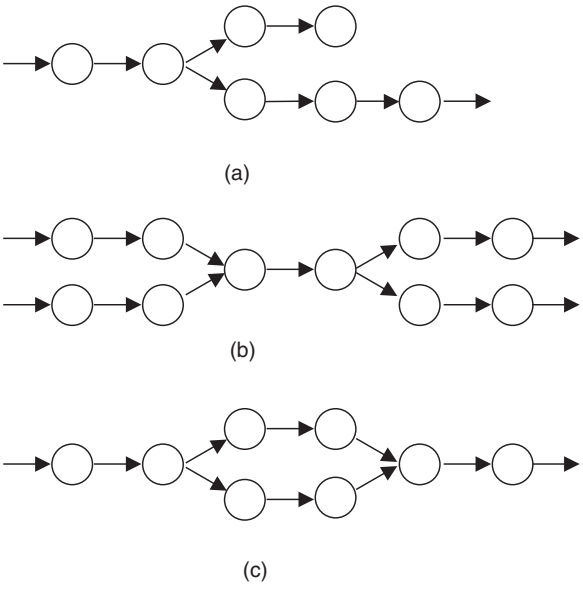


**Figure 3.1** Several different types of graph complications are shown using balls representing nodes with connecting arrows representing edges: (1) spur, (2) frayed-rope, and (3) bubble (adopted from Rodríguez-Ezpeleta *et al*., 2012).

overlaps into contigs (Rodríguez-Ezpeleta *et al*., 2012). Otherwise, the assembler may join unrelated reads to either side of a repeat to create mis-assemblies. The greedy algorithm is not widely used any more (Nagarajan & Pop, 2013) due to the inherently local assembly process that does not use global information (such as mate-pair reads) to resolve repeats.

***SSAKE*** SSAKE (Warren *et al*., 2007) was the first short-read assembler, which was designed for unpaired short reads of uniform length (Miller *et al*., 2010). It is written in PERL and runs on Linux. The algorithm assembles 25–300 bp reads from viral, bacterial, and fungal genomes (http://www.bcgsc.ca/platform/bioinfo/software/ssake). It is also the core of many other assemblers (such as VCAKE, QSRA, and SHARCGS).

SSAKE stands for short sequence assembly by progressive $K$-mer search and 3' read extension program, populating a hash table that is keyed by unique reads with values representing the number of occurrences in a single multi-fasta file (Warren *et al*., 2007). The sequence reads are sorted by decreasing values to display coverage and identify reads that may contain errors. The first 11 of the 5' end bases are memorized in a prefix tree, and each unassembled read (r) is utilized to start an assembly. SSAKE uses the prefix tree to progressively perform perfect matches of each possible $k$-mer with another read ($r'$). Once perfect match is determined, $r$ is extended by the 3' end bases of $r'$, and $r'$ is discarded from the hash table and the prefix tree. The output files are one log file with run information, one fasta file containing contig sequences, and one fasta file containing the unassembled reads.

One way to avoid mis-assemblies is to terminate the extension when a $k$-mer matches the 5' end bases of more than one read (`-s 1`), which could cause shorter contigs. The other way to control the stringency is to terminate the extension when a $k$-mer length is shorter than a user-set minimum word length ($m$).

***VCAKE*** VCAKE (Jeck *et al*., 2007) (verified consensus assembly by $K$-mer extension) is another iterative extension algorithm. Its assembly process is nearly the same as SSAKE, where sequence reads are searched from a hash table. However, compared to SSAKE, the main improvement of VCAKE is its ability to process imperfect matches during contig extension. Specifically, when extending contigs, each read offers a "vote" for the first overhanging base called by the contig (Jeck *et al*., 2007). The votes are summed, and the base that exceeds a threshold ($-c$, which is the ratio of the most represented base required to extend assembly) is used to extend the contig. In this way, error-containing reads could be used for extension by adding one base at a time until no matches are found or the contig is stopped because of assumed repeat sequences.

***QSRA*** QSRA (Bryant, Wong & Mockler, 2009) (quality-value-guided short read assembler) builds directly upon the VCAKE algorithm, where voting method is utilized to extend contigs one base at a time. To lengthen contigs past low coverage regions, one of the main improvements of QSRA is the usage of quality values in the contig extension. As long as the bases match the contig suffix or reach (or exceed) a minimum user-defined q-value ($-m$) at the overhanging region, the current assembly process continues. Another improvement of QSRA is the availability of suspect repeat sequences in a separate fasta file for repeat-related analysis.

#### Overlap-Layout-Consensus (OLC) Assemblers

Unlike the inherently local assembly in greedy algorithm, OLC assemblers use three stages to enable a global analysis of the relationships between the reads (Pop, 2009). The first two stages are to compute all-against-all comparison of reads and then construct an overlap graph. The overlap graph contains each read as a node, and an edge connects two reads that overlap each other. Overlaps are only taken into consideration between reads when they share a segment of sequences of length $k$, which is referred to as a $k$-mer. Therefore, the choice of $k$-mer minimum overlap length and percentage of identity required for an overlap are key factors that would affect discovery of overlaps and construction of graphs (Miller *et al*., 2010). Larger parameter values increase accuracy of contigs but decrease the length. After these two steps, the optimized overlap graph produces an approximate read layout.

The last step is to determine the precise layout and then the consensus sequence by computing multiple sequence alignment (MSA) among the reads, which correspond to a single path traversing each node in the graph at most once. Full information of each read needs to be loaded into memory for base calls, but it can run in parallel, and be partitioned by contigs (Miller *et al*., 2010).

***Celera Assembler*** The Celera Assembler (Myers *et al*., 2000) was originally a Sanger OLC assembler, and then was modified for combinations of Sanger and 454 data in a pipeline called CABOG (Celera Assembler with the Best Overlap Graph) (Miller *et al*., 2008), which is robust to homopolymer run length uncertainty, heterogeneous read lengths, and high read coverage.

In order to exploit the increased expected read coverage, CABOG performs an overlap-based trimming step first. After that, it uses exact-match seeds ($k$-mer) to detect candidate overlapping reads quickly. To avoid highly repetitive $k$-mers, it precomputes a threshold $M$ ($k$-mer with more than $M$ occurrences accounts for at

most 1% of all *k*-mer occurrences) for determining overlap seeds range. After this "anchors and overlaps" stage, directed overlaps with each overlap region spanning at least 40 bases at two-read ends and 94% or higher sequence identity are output for the next stage. Based on overlapping information, the best overlapping graph (BOG) is built by loading the best edge (i.e., best overlap alignment) connecting each node (read). Furthermore, any cycle is eliminated by random deletion of one edge. Then CABOG sorts the reads by their scores, which are counted by the number of other reads reachable from them in BOG paths. Starting from higher-scoring reads, unitigs (uniquely assemblable contigs) are constructed by following paths until reaching a path end or a read already being used for building some unitigs. On paths with a single intersection, CABOG always chooses the longer one first, while multiple intersections would lead to shorter unitigs. By incorporating paired-end constraints, contigs and scaffolds are created from unitigs, and, finally, CABOG derives consensus sequences through computing multiple sequence alignments from scaffold layouts and read sequences. One strength of Celera Assembler is its ability to handle various types of data, including those from sequencing by synthesis platforms (i.e., Illumina HiSeq) and single-module sequencing platforms (i.e., Pacific Biosciences) (Berlin *et al*., 2015).

***Edena*** Most OLC assemblers focus on assembly on Sanger and 454 data, while Edena (Hernandez *et al*., 2008) (Exact DE Novo Assembler) applies the OLC approach to homogeneous-length short reads from the SOLiD and Illumina platforms. In order to improve assembly of such short reads, Edena employs two features: exact matching, and detection of spurious reads. First, it removes duplicate reads and finds all overlaps of a minimum length to build an overlap graph; and then the graph is cleaned by the removal of redundant overlaps and spurious edges and the resolution of ambiguity. Finally, all unambiguous paths in the graph are output as assembled contigs.

#### De Bruijn Graph (DBG) Approach

The DBG approach is also known as the *k*-mer graph approach or Eulerian path approach (Pevzner, Tang & Waterman, 2001). DBG-based assemblers are widely applied to short reads from the SOLiD and Illumina platforms. A whole set of programs have been developed using this approach, including Velvet (Zerbino & Birney, 2008), ABySS (Simpson *et al*., 2009), SOAPdenovo (Li *et al*., 2010; Luo *et al*., 2012), and ALLPATHS (Butler *et al*., 2008; Gnerre *et al*., 2011; Maccallum *et al*., 2009). Unlike the OLC approach, the DBG approach does not require all-against-all overlap search, and consequently, does not need to store reads or their overlaps, which makes it memory efficient for small genome assembly. Still, for large genomes, a massive amount of memory is still necessary for containing actual sequence and the graph.

During the construction of DBG, the reads are chopped up into a set of *k*-mers. Similar to the OLC approach, *k*-mers are represented as the nodes in the graph, and the edges connect the adjacent *k*-mers whose overlaps are exactly $k - 1$ letters (for instance, the 4-mers ATCG and TCGA share exactly three letters, "TCG"). And then, all the *k*-mers are stored in a hash table for graph construction. Hence, the assembly process is simply changed to finding a path that visits each node in the graph at most once.

In most cases, the DBG assemblers attempt to construct contigs originating from unambiguous and unbranching paths in the graph. Unfortunately, three factors in the real sequencing data complicate this process (Miller *et al*., 2010). The first is that DNA

is double-stranded. The forward sequence of a read may overlap the forward sequence or reverse complement sequence of another read. Currently, different implementations have been developed to handle this forward–reverse overlap. The second problem is sequencing error. However, many errors are easily recognized during graph construction. For example, errors in the middle of a read lead to bubbles in the path, and errors at the end of a read usually result in a single *k*-mer in the graph. Some assemblers preprocess the reads to remove errors by correcting erroneous bases or discard error-containing reads. Some discard those paths that are not supported by a high number of *k*-mers. Others convert paths to sequences for alignment algorithms and then collapse those nearly identical paths. The last factor is the repeats in the genome. Repeat regions always create branches in the paths, which means paths converge in the repeats and then diverge. Successful assembly usually collapses the repeats to avoid mis-assembly. On the other hand, mate-pair information is always used to resolve repeat-induced complications.

***ABySS*** ABySS (Simpson *et al*., 2009) (Assembly By Short Sequences) was originally developed for assembly of large data sets from individual human genome sequencing. In order to achieve that goal, ABySS applies a distributed representation of a de Bruijn graph, which allows parallel computation of the algorithm across a set of computers to utilize their combined large memory. It is implemented in C++, and the MPI (Message Passing Interface) protocol is employed for communication between nodes (an object on a network). The assembly process is mainly performed in two steps. First, paired-end information is applied to contigs extension until no unambiguous bases could be added or the path goes to a blunt end because of low coverage. Paired-end information is then applied to resolving ambiguities and merging contigs. Here, the usage of ABySS on Linux will be briefly discussed, including installation of the program, basic commands for assembly process, and simple analysis of assembly results.

The installation of ABySS requires several dependencies: Boost, Open MPI, and sparsehash. With all these libraries installed, the program can be downloaded (http://www.bcgsc.ca/platform/bioinfo/software/abyss). The next step is to compile this program from source (http://seqanswers.com/wiki/ABySS).

**Compiling ABySS from Source**

The following command line should compile and install ABySS:

```
./configure && make && make install
```

To install ABySS in a specified directory (i.e., ABySS), type:

```
./configure --prefix=PATH_TO_ABySS
```

As described earlier, ABySS uses the MPI protocol, so if you wish to build the parallel assembler with MPI support, type:

```
./configure --with-mpi=PATH_TO_openmpi
```

The Google sparsehash library should be used for internal hash table construction to reduce memory usage:

```
./configure --CPPFLAGS=-IPATH_TO_sparsehash/include
```

The default maximum *k*-mer size is 64 and may be decreased to reduce memory usage or increased during compiling. The value must be a multiple of 32 (i.e., 32, 64, 96, etc.):

```
./configure --enable-maxk=96
```

If so desired by the users, the preceding command lines can be combined into one command for compiling. To run ABySS, its executables should be found in the PATH:

```
PATH=$PATH:PATH_TO_ABySS/bin
```

**Assembling Reads from Different Libraries**

After compiling, ABySS can be utilized to assemble reads from different libraries. For example, we are going to assemble six data sets with three different fragment libraries, including two mate-pair reads, two paired-end reads, and two single-end reads.

Library pe180 has reads in two files, "pe180_1.fa" and "pe180_2.fa"
Library pe250 has reads in two files, "pe250_1.fa" and "pe250_2.fa"
Library mp3k has reads in two files, "mp3k_1.fa" and "mp3k_2.fa"
Library mp5k has reads in two files, "mp5k_1.fa" and "mp5k_2.fa"
Single-end reads are stored in two files, "se1.fa" and "se2.fa"

To assemble this dataset into contigs in a file named "human-contigs.fa" and scaffolds in a file named "human-scaffolds.fa", run the command:

```
abyss-pe -C PATH_TO_RESULTS k=64 name=human lib='pe180
   pe250' mp='mp3k mp5k' \
pe180='pe180_1.fa pe180_2.fa' pe250='pe250_1.fa
   pe250_2.fa' \
mp3k='mp3k_1.fa mp3k_2.fa' mp5k='mp5k_1.fa mp5k_2.fa'
   se='se1.fa se2.fa'
```

Where *–C* defines the specified directory for storing results; *k* is the *k*-mer length; *name* is the prefix for all the filenames in the results directory; *lib* is for fragment library; and *mp* is for mate-pair library. However, mate-pair libraries are used only for scaffolding (join disconnected contigs together), and they do not contribute to building contigs, *per se*.

**Optimizing *k*-mer Length**

In order to compute the best value for the parameter *k*, run multiple assemblies and then check the assembly statistics to determine the best *k*-mer length. The following command example would assemble every *k*-mer length from 40 to 70:

```
export k
for k in {40..70}; do
mkdir k$k
abyss-pe -C k$k name=human lib='pe180' pe180='pe180_1.fa
   pe180_2.fa'
done
abyss-fac k*/human-contigs.fa
```

*mkdir* is used to create new directory for storing results, corresponding to parameter *–C*. "abyss-fac" is used for analysis of assembly results, which can help determine the best *k*-mer value. The results of "abyss-fac" will be shown in the file named "human-stats.csv".

***ALLPATHS*** ALLPATHS is a DBG assembler that can generate high-quality assemblies from short reads. It was originally published as an assembler based on simulated data (Butler *et al*., 2008), and then revised for real data (Maccallum *et al*., 2009).

First, ALLPATHS removes all read pairs containing 90% or more "A" bases, which are nearly always artifacts of the Illumina sequencing process. It then begins an error-correcting preprocessor by identifying putatively correct (or "trusted") *k*-mers in the reads based on high frequency and high quality (where each base must be confirmed by a minimum number of base calls with a quality score above a threshold). With lower-quality reads, they could be kept under two circumstances—(1) where up to two base substitutions to low-quality base calls make its *k*-mers trusted, or (2) if the read is essential for building a path. Another preprocessor step is called *unipath creation*. This step starts with the calculation of perfect read overlaps seeded by *k*-mers. An index is assigned to each *k*-mer, and a compact searchable database is built for constructing unipaths.

After the initial unipath creation, those with branching paths shorter than 20 *k*-mers, which are always caused by read errors, are removed to provide longer alternative branches. This step is also called "unipath graph shaving". Now, with the unipaths and read pairs in hand, those unipaths around which genome assembly will be constructed are picked as "seeds" (the perfect seeds being long and of low copy number). The genomic region containing the seed and extending 10 KB on both sides is called the "neighborhood" of the seed. The goal is to assemble the neighborhood with help of unipaths and paired-end reads.

In order to resolve repeat regions, ALLPATHS assembles the locally non-repetitive sequences first. This step runs in parallel. Once it ends, the gluing process is applied to iteratively joining together the output graphs that have long end-to-end overlaps. Next, a series of editing steps are carried out to remove spurs and ambiguous regions to yield the final assembly. Those unipaths that are not represented in the assembly are extended unambiguously where possible and then added to the final assembly results.

ALLPATHS-LG (Gnerre *et al*., 2011) is an updated version of ALLPATHS, which is designed for large genomes from massively parallel sequence data. It requires construction of two different paired reads: a fragment library and a jumping library. The fragment library has a short insert size that is less than twice the read length, so that the two paired reads may overlap to produce a single longer "read" (i.e., read length is 100 bp and insert size is 180 bp, then these two reads have an overlap of 20 bp). The jumping library generates long-range mate-pair reads. This new version also makes extensive improvements in handling repetitive sequences and low coverage region, error-correction, and memory usage. Here, we will describe the detailed usage of the ALLPATHS-LG program.

**Requirements for ALLPATHS-LG**

ALLPATHS has a number of requirements, including:

1) A minimum of two paired-end libraries as described earlier: one fragment library whose insert size is less than twice the read length, and one jumping library.
2) Memory peak usage is around 1.7 bytes per read base. To compile and run ALLPATHS-LG on a Linux/UNIX system, a minimum of 32 GB memory for small genomes and 512 GB memory for large genomes are suggested.
3) The g++ compiler from GCC (https://gcc.gnu.org/).

4) The GMP library (https://gmplib.org/) compiled with C++ interface, which may already be included in GCC.
5) The Picard command-line utilities for SAM file manipulation (http://broadinstitute.github.io/picard/).

**ALLPATHS Installation**

The ALLPATHS-LG source code is available for download (http://www.broadinstitute.org/software/allpaths-lg/blog/?page_id=12). After the latest version of the program is downloaded, it can be decompressed using *tar*. Then, it can simply be compiled with *./configure* and *make*, and, at the same time, the *--prefix* parameter can be used to define the installation directory. All the source code is in the unpacked directory called "allpathslg- < revision>".

```
tar xzf allpathslg-<revision>.tar
cd allpathslg-<revision>
./configure --prefix=PATH_TO_INSTALLATION_DIRECTORY
make
make install
```

To run ALLPATHS-LG, its executables should be added to the *PATH*.

**ALLPATHS-LG Pipeline Directory Structure**

The assembly pipeline uses the following directory structure to store its inputs and outputs. If the directory does not exist, it can be automatically created by the pipeline. The names shown here are only for descriptions of the structure, and they are determined by the actual command line.

"PRE / REFERENCE / DATA / RUN / ASSEMBLIES / SUBDIR"

The "PRE" directory is the root for the assemblies, and there could be a number of "REFERENCE" directories in it. The "REFERENCE" directory is also called *organism directory*; in other words, it is used to separate assembly projects by organism. The "DATA" directory consists of the data whose formats are changed by the ALLPATHS-LG pipeline. The "RUN" directory contains intermediate files generated from the read data in preparation for the final assembly process and intermediate files used for evaluation. The "ASSEMBLIES" directory is the actual assembly directory, and its name is fixed. The "SUBDIR" directory (its name is "test", and it is also fixed) is where the localized assembly is generated.

**Prepare the Data**

Before assembly, original data (BAM, FASTQ, FASTA, or FASTB) needs to be prepared and imported into the ALLPATHS-LG pipeline. This step is completed by adding two comma-separated-values (csv) metadata files to describe the locations and library information.

"in_groups.csv"
"in_libs.csv"

Each line in "in_groups.csv" provides the following information:

*group_name*: a unique name for the data set
*library_name*: the library to which the data set belongs

*file_name*: the absolute path to the data set. Wildcards "*" and "?" are accepted for specifying multiple files (i.e., two paired files)

Here is one example for the "in_groups.csv" file:

```
group_name,        library_name,     file_name
1,                 paired-end,       /illumina/reads1*.fastq
2,                 mate-pair,        /illumina/reads2*.fastq
```

Each line in "in_libs.csv" provides the following information:

*library_name*: matches the same field in "in_groups.csv"
*project_name*: a string naming the project
*organism_name*: the name of the organism
*type*: fragment, jumping, etc.; field not necessary
*paired*: "0" for unpaired reads and "1" for paired reads
*frag_size*: estimated average fragment size
*frag_stddev*: estimated fragment size standard deviation
*insert_size*: estimated jumping average insert size
*insert_stddev*: estimated jumping insert size standard deviation
*read_orientation*: *inward* for fragment reads and fosmid jumping reads, *outward* for jumping reads
*genomic_start*: index of the first genomic base in the reads; if non-zero, trim all the bases before *genomic_start*
*genomic_end*: index of the last genomic base in the reads; if non-zero, trim all bases after *genomic_end*; if both *genomic_start* and *genomic_end* are blank or zero, keep all bases

Here is an example for the "in_libs.csv" file (Note: There should be three lines for this example, in which "…" means "to be continued".):

```
library_name,project_name,organism_name,type,paired,
  frag_size,...
paired-end,genome,human,fragment,1,180,...
mate-pair,genome,human,jumping,1, ,...
frag_stddev,insert_size,insert_stddev,read_orientation,...
20, , ,inward,...
  ,2000,200,outward,...
genomic_start,genomic_end
  ,
  ,
```

With these two description files, a Perl script is required to convert the original reads. "DATA_DIR" is the location of the DATA directory where the converted reads are stored. "PICARD_TOOLS_DIR" is the path to the Picard tools for reads conversion, if the data is in BAM format. "PLOIDY" indicates the genome, with "1" for haploid genomes and "2" for diploid genomes. Here is an example:

```
PrepareAllpathsInput.pl
   DATA_DIR=PATH_TO_DATA_DIRECTORY
   IN_GROUPS_CSV=PATH_TO_<in_groups.csv>
```

```
IN_LIBS_CSV=PATH_TO_<in_libs.csv>
PLOIDY=<1 or 2>
PICARD_TOOLS_DIR=PATH_TO_<Picard_tools>
HOSTS=<list of hosts to be used in parallel>
```

After a successful run of "PrepareAllpathsInput.pl", the necessary input files are ready for the assembly stage.

**Assembling**

There are a series of modules in ALLPATHS-LG. Each module performs one step of the assembly process. However, a single module called *RunAllpathsLG* controls the entire pipeline, assigning different modules to run efficiently at the proper time. This step specifically names the directories that were described in the directory structure part. Here is a basic example:

```
RunAllpathsLG
    PRE=allpaths
    REFERENCE_NAME=reference
    DATA_SUBDIR=data
    RUN=run
```

These command lines will create (if it does not exist) the following directory structure: "allpaths / reference / data / run / ASSEMBLIES / test".

**Assembly Results**

In the preceding example, the assembly results are in the "test" directory. Both files ("final.assembly.fasta" and "final.assembly.efasta") are scaffold results, where the *N*s represent the gaps between contigs. In the fasta file, an ambiguous base is represented by an *N* within each contig. For example, a C/G SNP is shown as "ATCNTGC". The efasta is "enhanced" fasta, which is generated and used by ALLPATHS-LG. In the efasta file, ambiguity is represented by listing all the possible bases within a pair of braces. For example, a C/G SNP is shown as "ATC{C,G}TGC". The options in the braces are ordered by decreasing likelihood. Hence, the fasta file can be easily generated from the efasta file by picking the first option for every ambiguity. Moreover, statistics for assembly results could be found in the "assembly.report" file.

### PacBio Long Reads and Their Applications

Despite many advances in genome assembly programs, generation of complete and accurate assembly of genomes using only short reads data is still difficult, largely due to repetitive sequences (Alkan *et al*., 2011; Kingsford, Schatz & Pop, 2010). Repetitive sequences can be properly assembled only when the read length is long enough to span the whole repetitive sequences or the read pair is uniquely anchored on both sides of the repeats. In 2011, Pacific Biosciences released their commercial "third-generation" sequencing instrument PacBio RS (Eid *et al*., 2009). According to the information on the PacBio website (http://www.pacb.com/smrt-science/smrt-sequencing/read-lengths/), the average read length could reach 10 KB. Unfortunately, the long reads also come with relatively low nucleotide accuracy (~85%), compared with the 98% accuracy from short reads. However, short reads can be utilized to correct the long reads, allowing potential use of the long reads for the assembly.

***Hybrid Assembly*** Short reads can be used in combination with long reads to improve the completeness and correctness of an assembly, making it a hybrid assembly strategy. Several programs have been developed to take advantage of both short and long reads. A PBcR (PacBio corrected Reads) algorithm (Koren *et al*., 2012), implemented in the Celera Assembler, trims and corrects individual long reads by simultaneously mapping high-identity short reads to them and computing a highly accurate consensus sequence for each of them. This "hybrid" correction could achieve >99.9% base-call accuracy, and then the corrected reads may be *de novo* assembled or co-assembled with other data types using OLC assembly techniques. This method allows for the assembly of large genomes, but requires a large amount of running time for the correction process.

Cerulean (Deshpande *et al*., 2013), unlike other hybrid assembly programs, does not use the short reads directly. Instead, it starts with a graph from ABySS (Simpson *et al*., 2009) and extends contigs by resolving bubbles in the graph using PacBio long reads. The advantages of this method are that it requires minimal computational resources, and results in a highly accurate assembly. But the efficiency of cerulean when applied to large genomes remains to be verified.

ALLPATHS-LG (Gnerre *et al*., 2011; Ribeiro *et al*., 2012) is another program that can deal with PacBio long reads in combination with short reads from Illumina sequencing platforms. As described in the preceding section, it requires one fragment library and also a jumping library. These three different data types, whose accuracy, coverage, and range of repeat resolution are compensatory, thus enable satisfying assembly results. However, good performance requires much more efforts and money to construct the libraries and larger amounts of time to compute the data, which limits its application to large genomes. The assembly algorithm is implemented as parts of ALLPATHS-LG and is automatically invoked when PacBio data are provided as input to the algorithm.

Here is an example of descriptions for "in_groups.csv" and "in_libs.csv" when PacBio data is utilized for assembly:

For "in_groups.csv":

```
group_name, library_name, file_name
1,          paired-end,   /illumina/reads1*.fastq
2,          mate-pair,    /illumina/reads2*.fastq
3,          pacbio,       /pacbio/reads.fastq
```

For "in_libs.csv" (Note: There should be four lines for this example, in which "…" means "to be continued".):

```
library_name,project_name,organism_name,type,paired,
  frag_size,...
paired-end,genome,human,fragment,1,180,...
mate-pair,genome,human,jumping,1, ,...
pacbio,genome,human,long,0, ,...
frag_stddev,insert_size,insert_stddev,read_orientation,...
20, , ,inward,...
  ,2000,200,outward,...
  , , , ,...
genomic_start,genomic_end,
   ,
   ,
   ,
```

***PacBio-only Assembly*** In order to avoid using different sequencing libraries and multiple sequencing platforms, a homogeneous workflow requiring only PacBio reads is favored to increase efficiency. The Hierarchical Genome Assembly Process (HGAP) (Chin *et al*., 2013) was developed for high-quality *de novo* microbial genome assemblies using only PacBio long reads. This method uses the longest reads as seeds to recruit all the other reads to map to them, resulting in long and highly accurate preassembled reads. Next, quality trimming of preassembled reads is performed to improve the ability to detect sequence overlaps. OLC assemblers are then used for *de novo* assembly of multi-KB trimmed reads. To significantly reduce the remaining errors in the draft assembly, a quality-aware consensus algorithm implemented in HGAP is computed to derive a highly accurate consensus for the final assembly (99.999% accuracy). Undoubtedly, this method is successful with such high completeness and correctness. However, overlapping steps still consume large amounts of time, making it less computationally efficient for large genomes.

## Scaffolding

The output of most assemblers is a large collection of fragmented contigs. However, downstream analyses such as analyzing gene synteny and carrying out comparative or functional genomics rely heavily on an assembly with good continuity (Hunt *et al*., 2014). Hence, it is necessary to join disconnected contigs into continuous genome sequences in the correct orientation and order. This process is also called *scaffolding*, and it includes contig orientation, contig ordering, and contig distancing (Barton & Barton, 2012).

***Contig Orientation*** Each strand of DNA could be sequenced to generate reads, and therefore the constructed contigs from assemblers could represent either strand. When scaffolding, contig orientation needs to be performed to keep contigs in the same direction, which requires reverse complementing sequences where necessary (Barton & Barton, 2012).

***Contig Ordering*** Contigs are ordered within each scaffold, reflecting their spatial placement in the true genome sequences (Barton & Barton, 2012).

***Contig Distancing*** Given the correct orientation and order, the distance is determined by putting certain numbers of "*N*s", which represent unknown nucleotides, between contigs. The size of each inter-contig gap corresponds to the number of *N*s between them. The total length of known and unknown regions is an estimate of the complete genome size (Barton & Barton, 2012).

Most commonly, the process of scaffolding is achieved from mate-pair information. Two contigs may be scaffolded together if one end of a mate-pair is assembled within the first contig, and the second contig contains the other end of the mate-pair. And the insert size of the mate-pair library provides an estimate for how far apart each read should appear in the final assembly (Paszkiewicz and Studholme, 2010). Furthermore, scaffolding information can also be obtained from whole-genome mapping data (Pop, 2009), which can be achieved from the program SOMA (Nagarajan, Read & Pop, 2008).

Several assemblers also contain a scaffolding module, such as Celera Assembler (Myers *et al*., 2000), ABySS (Simpson *et al*., 2009), SOAPdenovo (Li *et al*., 2010; Luo

*et al*., 2012), and ALLPATHS-LG (Gnerre *et al*., 2011). Stand-alone scaffolders are also available, such as GRASS (Gritsenko *et al*., 2012), Scaffolder (Barton & Barton, 2012), SSPACE (Boetzer *et al*., 2011), and SOPRA (Dayarian, Michael & Sengupta, 2010). Due to genomic complexity and missing data, scaffolding may finally produce a collection of scaffolds instead of a completed sequence, but a wise choice of scaffolder still can yield a decent scaffolding result. A comprehensive evaluation of scaffolding tools was performed to test their overall performance (Hunt *et al*., 2014). Scaffolding tools vary in their ease of use, speed, and the number of correct and missed joins between contigs. The quality of the results depends highly on the choice of mapping tools and genome complexity. In general, SGA's scaffolding module (Simpson & Durbin, 2012), SOPRA (Dayarian *et al*., 2010), and SSPACE (Boetzer *et al*., 2011) perform better on the tested data sets (Hunt *et al*., 2014).

Here, we introduce SSPACE and its detailed usage for scaffolding. SSPACE starts with the longest contig to build the first scaffold and continues to make joins with the majority of read pairs supporting it (Hunt *et al*., 2014). It uses Bowtie (Langmead *et al*., 2009) to map all the reads to the contigs, and, if desired, unmapped reads are used for contig extension. It requires limited computation resources and is able to handle large genomes in a reasonable amount of time (Boetzer *et al*., 2011).

To apply for the basic version of SSPACE (free for academics), one needs to register on its website (http://www.baseclear.com/genomics/bioinformatics/basetools/SSPACE). Then, a library file containing information about each library needs to be established. All the columns are separated by spaces. Here is an example:

```
lib1 bowtie file1_1.fasta file1_2.fasta 200 0.25 FR
lib2 bowtie file2_1.fastq file2_2.fastq 2000 0.25 RF
unpaired bwa unpaired_reads.fasta
```

***Column 1*** Name of the library. Libraries of the same name are considered to possess the same insert size and deviation (columns 5 and 6). Moreover, they are used for the same scaffolding iteration. Libraries should be sorted by insert size (column 5). The first library is always scaffolded first, followed by the next libraries. For unpaired reads, the library name is "unpaired", as shown in the example.

***Column 2*** Name of the read aligner (bowtie, bwa, or bwasw).

***Columns 3 and 4*** Fasta or fastq files for datasets. For paired reads, the first read file is in column 3, and the second read file in column 4. They are always on the same line. For unpaired reads, only column 3 is required.

***Columns 5 and 6*** Column 5 represents the mean distance between paired reads, which is 200 for *lib1* in the example. Column 6 represents the minimum allowed deviation of the mean distance, which is 0.25 for *lib1*. This means the distance for paired reads in *lib1* is between 150 and 250.

***Column 7*** Column 7 indicates the orientation of paired reads. *F* means forward, and *R* means reverse. Usually for the case of mate-pair data, it is *RF*.

Before running SSPACE, its executables should be added to the `PATH`. There is one contig extension option (`-x`) whose values indicate whether to do extension using unmapped reads. Based on this option, different parameters are used in the commands.

Here is one example for scaffolding contigs without extending them:

```
perl SSPACE_Standard_v3.0.pl -l library_file -s contig_file
   -k 5 -a 0.7 -x 0 -b scaffolds_no_extension
```

`-l` is followed by the library file, and `-s` is followed by contig sequences file. To run SSPACE without extension, `-x` should be set to 0. `-k` and `-a` are parameters for controlling the scaffolding process. `-k` is the minimum number of read pairs (links) to compute scaffolds, and `-a` is the maximum link ratio between the two best contig pairs. Their default values are 5 and 0.7, which do not have to be listed if not reset. The final results are generated in the folder named "scaffolds_no_extension". The file "scaffolds_no_extension.summaryfile.txt" contains the statistics for the final scaffolds. The file "scaffolds_no_extension.final.scaffolds.fasta" contains the resulting scaffolds.

To run SSPACE with contig extension, `-x` should be set to 1. `-k` and `-a` use default values. Here, the minimal overlap of the reads with the contig during overhang consensus build-up is at least 32 bases (`-m`). The minimum number of reads required to extend a contig is 20 (`-o`). These two numbers are also default values for SSPACE version 3.0. The file "scaffolds_extension.summary.txt" in the folder "scaffolds_extension" contains the information on extended contigs. The sequences of the extended contigs are kept in the "intermediate_results" folder.

```
perl SSPACE_Standard_v3.0.pl -l library_file -s contig_file
   -x 1 -m 32 -o 20 -b scaffolds_extension
```

Considering that PacBio long reads are playing an increasingly important role in scaffolding genome assemblies in a cost-effective manner, SSPACE-LongRead (Boetzer and Pirovano, 2014) was developed to scaffold pre-assembled contigs using long reads as a backbone. This may enhance the quality of incomplete and inaccurate genomes constructed from next-generation sequencing data.

## Gap Filling (Gap Closing)

In scaffolds construction, contigs with certain distance relationships are connected with *N*s. The majority of these gaps are composed of repetitive and low-coverage regions, which may contain essential sequences for the downstream analyses and need to be filled. The standard strategy to address this problem usually involves the design of specific primers to undergo target Sanger sequencing at contig ends (Tsai, Otto & Berriman, 2010). Without doubt, this method is expensive, labor intensive, and time consuming. At present, the gap-filling process can be made easier by using the original read-pair information (Ekblom and Wolf, 2014) with programs such as IMAGE (Tsai *et al*., 2010), GapCloser (a module for SOAPdenovo2) (Luo *et al*., 2012), FGAP (Piro *et al*., 2014), GapFiller (Boetzer and Pirovano, 2012), and PBJelly (English *et al*., 2012).

IMAGE (Iterative Mapping and Assembly for Gap Elimination) uses Illumina sequence data to improve draft genome assemblies by aligning paired-reads against contig ends and performing local assemblies to produce gap-spanning contigs (Tsai

*et al*., 2010). Similarly, this method also functions in GapCloser and GapFiller. But the improvement in GapFiller is that it takes into account the estimated gap size prior to closure and closes the gap only if the size of the sequence insertion corresponds closely to the estimated size. This will ensure that the local assembly results may reflect the true genomic situation. It also takes into account the contig edges, which is often a source of mis-assemblies, and trims them prior to gap filling (Boetzer and Pirovano, 2012). PBJelly was designed for genome finishing using long reads from the Pacific Biosciences RS platform (English *et al*., 2012).

Here, we will introduce the detailed usage of GapFiller. It was developed by the same group as SSPACE. You need to register for downloading the program (free for academics) (http://www.baseclear.com/genomics/bioinformatics/basetools/gapfiller). The library information file from SSPACE is also applied to GapFiller. Before running, its executables should be added to the `PATH`. Here is an example of the gap-filling command:

```
perl GapFiller.pl -l library.txt -s scaffolds.fa -m 29 -o 2
   -r 0.7 -d 50 -n 10 -t 10 -T 1 -i 10 -b gapfiller
```

This command uses the paired reads in the "library.txt" to fill the scaffolds of "scaffolds.fa". The `-s` parameter should be followed by a fasta format file. `-m` is the minimum number of overlapping bases of the reads with the edge of the gap. It is suggested to take a value close to the read length. For example, it is suggested to use 30–35 as the `-m` value for a library with 36 bp reads. `-o` is the minimum number of reads required to extend the edge during gap filling. `-r` is the minimum base ratio needed to accept an overhang consensus base. Higher values of these three values lead to more accurate gap filling. `-d` is the maximum difference between the estimated gap size and the number of gap-closed sequences. Extension is stopped if the difference exceeds 50 bases for this case. `-n` is the minimum overlap for merging two sequences. `-t` is the number of nucleotides to be trimmed from the sequence edges of the gap. This parameter may discard low-quality and misassembled sequences to ensure correct overlap and extension. `-T` indicates the number of threads for reading in the files and mapping of the reads. `-i` is the number of iterations used to fill the gaps. GapFiller keeps using original reads and mapping them against the scaffold gaps until no more reads are used for gap filling or until the specified number of iterations is reached. `-b` gives the name for the output folder. In this case, all the gap-filling results are stored in the "gapfiller" folder, and the gapclosed sequences are in "gapfiller.gapfilled.final.fa", with the statistics information in "gapfiller.summaryfile.final.txt". The assembly process *in silico* is now considered to be finished.

## Evaluation of Assembly Quality

Evaluation of assembly quality is of great importance. Considering the large number of operations and uncertain factors in the assembly processes, this evaluation can be complex and challenging. It is less difficult for the evaluation of reference assembly sequences. Since an available reference genome provides a comparable ground truth, we can assess the quality of a new assembly by simply comparing it to the reference genome. However, the lack of a reference genome makes the evaluation of *de novo* assembly a lot more complicated, and there are still no well-accepted measures to date. Before we get into the specific evaluation protocols, it should be very helpful to know the potential

errors that may exist in the assembled sequences. Potential mis-assemblies (Phillippy, Schatz & Pop, 2008) may include: (1) repeat collapse regions where an assembler incorrectly joins distinct repeat copies into a single unit, resulting in increased coverage of the read density and sometimes generating "orphan" regions; (2) repeat expansion regions where the addition of copies occurs and the read density drops below normal coverage; and (3) rearrangement regions where the order of multiple repeat copies shuffles leading to the misplacement of the unique sequences, in a special case, inversions may occur when two repeat copies are oppositely oriented. Clearly, most of these mis-assemblies have always been caused by repeats, and the longer the reads, the fewer the repeats that cause errors in the assembly process.

The quality of assembled sequences is basically assessed through two indexes: size statistics and correctness scores. In the case of size statistics, commonly considered statistical values include maximum contig/scaffold length, minimum contig/scaffold length, mean contig/scaffold length, genomic coverage, and N50 size. In particular, the N50 size is the most widely used statistic, which is the minimum size of the contig/scaffold making the sum of the lengths equal to or greater than 50% of the entire genome size after sorting all contigs/scaffolds from longest to shortest. The N50 size can be used to compare different programs only if it is measured with the same combined length value.

The correctness of an assembly is very difficult to measure. Scientists tend to focus on contiguity and ignore the accuracy. To detect mis-assemblies, one measurement relies on comparing assemblies with other independently generated data such as finished BAC sequences (Istrail *et al*., 2004), mapping data (Nagarajan *et al*., 2008), transcriptome data (Zimin *et al*., 2009), or the genomes of closely related organisms (Gnerre *et al*., 2009). Another measurement uses intrinsic consistency to identify mis-assemblies. One case is the detection of regions with unusual depth of coverage (too high or too low); as we mentioned earlier in error types, repeat collapse or expansion may cause changes in the read density.

In addition, some computational technologies provide tools to evaluate assembly results, including: (1) AMOSvalidat (Phillippy *et al*., 2008), a tool collected with several detecting mis-assembly mechanisms; (2) GAV (Choi *et al*., 2008), an approach combined with multiple accuracy measures; (3) CGAL (Rahman & Pachter, 2013), a measurement considering both genome coverage and assembly accuracy with no need of reference sequences; (4) GAGE (Salzberg *et al*., 2012), an evaluation of the very latest large-scale genome assembly algorithms; (5) REAPR (Hunt *et al*., 2013), an evaluation tool providing a positional error call metric and identifying potential collapsed repeats; (6) SuRankCo (Kuhring *et al*., 2015), a machine learning approach to predict quality scores and ranking for contigs; and (7) the validation scripts used in Assemblathon (Bradnam *et al*., 2013; Earl *et al*., 2011), which is a competition aiming to improve methods of genome assembly.

Even though there are so many methods out there evaluating assembly results, they only aim to display the issues existing in the sequences. Fixing them by sequencing technologies and varieties of programs is key to acquiring a complete and high-quality genome sequence as the ultimate resource for further genomic approaches. However, in our experience, validation using an independent method such as high-density linkage mapping is quite effective for the assessment of the sequence assembly. The order of the

SNP markers along the linkage map and along the whole-genome reference sequence should be the same, if the whole genome is assembled correctly.

## References


Alkan, C., Sajjadian, S. and Eichler, E.E. (2011) Limitations of next-generation genome sequence assembly. *Nature Methods*, **8**, 61–65.

Ammar, R., Paton, T.A., Torti, D., Shlien, A. and Bader, G.D. (2015). Long read nanopore sequencing for detection of HLA and CYP2D6 variants and haplotypes. *F1000Research*, **4**, 17.

Barton, M.D. and Barton, H.A. (2012) Scaffolder – software for manual genome scaffolding. *Source Code for Biology and Medicine*, **7**, 4.

Berlin, K., Koren, S., Chin, C.-S. *et al*. (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology*, **33**, 623–630.

Boetzer, M., Henkel, C.V., Jansen, H.J. *et al*. (2011) Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**, 578–579.

Boetzer, M. and Pirovano, W. (2012) Toward almost closed genomes with GapFiller. *Genome Biology*, **13**, R56.

Boetzer, M. and Pirovano, W. (2014) SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information. *BMC Bioinformatics*, **15**, 211.

Bolger, A.M., Lohse, M. and Usadel, B. (2014) Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, **30**, 2114–2120.

Bradnam, K.R., Fass, J.N., Alexandrov, A. *et al*. (2013) Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species. *GigaScience*, **2**, 10.

Braslavsky, I., Hebert, B., Kartalov, E. and Quake, S.R. (2003) Sequence information can be obtained from single DNA molecules. *Proceedings of the National Academy of Sciences of the United States of America*, **100**, 3960–3964.

Bryant, D.W. Jr.,, Wong, W.K. and Mockler, T.C. (2009) QSRA: a quality-value guided *de novo* short read assembler. *BMC Bioinformatics*, **10**, 69.

Butler, J., MacCallum, I., Kleber, M. *et al*. (2008) ALLPATHS: *De novo* assembly of whole-genome shotgun microreads. *Genome Research*, **18**, 810–820.

Chin, C.-S., Alexander, D.H., Marks, P. *et al*. (2013) Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature Methods*, **10**, 563–569.

Choi, J.-H., Kim, S., Tang, H. *et al*. (2008) A machine-learning approach to combined evidence validation of genome assemblies. *Bioinformatics*, **24**, 744–750.

Dayarian, A., Michael, T. and Sengupta, A. (2010) SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, **11**, 345.

Deshpande, V., Fung, E.K., Pham, S. and Bafna, V. (2013) Cerulean: a hybrid assembly using high throughput short and long reads, in *Algorithms in Bioinformatics* (eds A. Darling and J. Stoye), Springer Berlin, Heidelberg, pp. 349–363.

Dolnık, V. (1999) DNA sequencing by capillary electrophoresis (review). *Journal of Biochemical and Biophysical Methods*, **41**, 103–119.

Dressman, D., Yan, H., Traverso, G. *et al*. (2003) Transforming single DNA molecules into fluorescent magnetic particles for detection and enumeration of genetic variations.

*Proceedings of the National Academy of Sciences of the United States of America*, **100**, 8817–8822.

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

Eid, J., Fehr, A., Gray, J. *et al*. (2009) Real-time DNA sequencing from single polymerase molecules. *Science*, **323**, 133–138.

Eisenstein, M. (2012) Oxford nanopore announcement sets sequencing sector abuzz. *Nature Biotechnology*, **30**, 295–296.

Ekblom, R. and Wolf, J.B. (2014) A field guide to whole-genome sequencing, assembly and annotation. *Evolutionary Applications*, **7**, 1026–1042.

English, A.C., Richards, S., Han, Y., Wang, M., Vee, V., Qu, J. *et al*. (2012). Mind the gap: upgrading genomes with pacific biosciences RS long-read sequencing technology. *PLoS ONE*, **7**, e47768.

Fedurco, M., Romieu, A., Williams, S. *et al*. (2006) BTA, a novel reagent for DNA attachment on glass and efficient generation of solid-phase amplified DNA colonies. *Nucleic Acids Research*, **34**, e22.

Fox, E.J., Reid-Bayliss, K.S., Emond, M.J. and Loeb, L.A. (2014) Accuracy of next generation sequencing platforms. *Journal of Next Generation, Sequencing & Applications*, **1**, 1000106.

Gnerre, S., Lander, E.S., Lindblad-Toh, K. and Jaffe, D.B. (2009) Assisted assembly: how to improve a *de novo* genome assembly by using related species. *Genome Biology*, **10**, R88.

Gnerre, S., MacCallum, I., Przybylski, D. *et al*. (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, **108**, 1513–1518.

Gritsenko, A.A., Nijkamp, J.F., Reinders, M.J.T. and de Ridder, D. (2012) GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, **28**, 1429–1437.

Harris, T.D., Buzby, P.R., Babcock, H. *et al*. (2008) Single-molecule DNA sequencing of a viral genome. *Science*, **320**, 106–109.

Heo, Y., Wu, X.-L., Chen, D. *et al*. (2014) BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, **30**, 1354–1362.

Hernandez, D., François, P., Farinelli, L. *et al*. (2008) *De novo* bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, **18**, 802–809.

Housby, J.N. and Southern, E.M. (1998) Fidelity of DNA ligation: a novel experimental approach based on the polymerisation of libraries of oligonucleotides. *Nucleic Acids Research*, **26**, 4259–4266.

Hunt, M., Kikuchi, T., Sanders, M. *et al*. (2013) REAPR: a universal tool for genome assembly evaluation. *Genome Biology*, **14**, R47.

Hunt, M., Newbold, C., Berriman, M. and Otto, T. (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, **15**, R42.

Istrail, S., Sutton, G.G., Florea, L. *et al*. (2004) Whole-genome shotgun assembly and comparison of human genome assemblies. *Proceedings of the National Academy of Sciences of the United States of America*, **101**, 1916–1921.

Jain, M., Fiddes, I.T., Miga, K.H. *et al*. (2015) Improved data analysis for the MinION nanopore sequencer. *Nature Methods*, **12**, 351–356.

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

Jiang, H., Lei, R., Ding, S.-W. and Zhu, S. (2014) Skewer: a fast and accurate adapter trimmer for next-generation sequencing paired-end reads. *BMC Bioinformatics*, **15**, 182.

Kelley, D., Schatz, M. and Salzberg, S. (2010) Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, **11**, R116.

Kingsford, C., Schatz, M. and Pop, M. (2010) Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinformatics*, **11**, 21.

Kong, Y. (2011) Btrim: a fast, lightweight adapter and quality trimming program for next-generation sequencing technologies. *Genomics*, **98**, 152–153.

Koren, S., Schatz, M.C., Walenz, B.P. *et al*. (2012) Hybrid error correction and *de novo* assembly of single-molecule sequencing reads. *Nature Biotechnology*, **30**, 693–700.

Kuhring, M., Dabrowski, P.W., Piro, V.C. *et al*. (2015) SuRankCo: supervised ranking of contigs in *de novo* assemblies. *BMC Bioinformatics*, **16**, 240.

Langmead, B., Trapnell, C., Pop, M. and Salzberg, S. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, **10**, R25.

Li, H. and Durbin, R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li, H. and Durbin, R. (2010) Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, **26**, 589–595.

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

Lindgreen, S. (2012) AdapterRemoval: easy cleaning of next-generation sequencing reads. *BMC Research Notes*, **5**, 337.

Loman, N.J. and Watson, M. (2015) Successful test launch for nanopore sequencing. *Nature Methods*, **12**, 303–304.

Lunter, G. and Goodson, M. (2011) Stampy: a statistical algorithm for sensitive and fast mapping of Illumina sequence reads. *Genome Research*, **21**, 936–939.

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

Maccallum, I., Przybylski, D., Gnerre, S. *et al*. (2009) ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology*, **10**, R103.

Macevicz, S. (1998) DNA sequencing by parallel oligonucleotide extensions. U.S. patent No. 5,750,341. Washington, DC: U.S. Patent and Trademark Office.

Mardis, E.R. (2011) A decade's perspective on DNA sequencing technology. *Nature*, **470**, 198–203.

Mardis, E.R. (2013) Next-generation sequencing platforms. *Annual Review of Analytical Chemistry*, **6**, 287–303.

Margulies, M., Egholm, M., Altman, W.E. *et al*. (2005) Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**, 376–380.

Martin, M. (2011) Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet Journal*, **17**, 10–12.

McKernan, K., Blanchard, A., Kotler, L. and Costa, G. (2012) Reagents, methods, and libraries for bead-based sequencing. U.S. Patent No. 8,329,404. Washington, DC: U.S. Patent and Trademark Office.

Metzker, M.L. (2010) Sequencing technologies – the next generation. *Nature Reviews Genetics*, **11**, 31–46.
Miller, J.R., Delcher, A.L., Koren, S. *et al*. (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.
Miller, J.R., Koren, S. and Sutton, G. (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.
Myers, E.W., Sutton, G.G., Delcher, A.L. *et al*. (2000) A whole-genome assembly of Drosophila. *Science*, **287**, 2196–2204.
Nagarajan, N. and Pop, M. (2013) Sequence assembly demystified. *Nature Reviews Genetics*, **14**, 157–167.
Nagarajan, N., Read, T.D. and Pop, M. (2008) Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, **24**, 1229–1235.
Paszkiewicz, K. and Studholme, D.J. (2010) *De novo* assembly of short sequence reads. *Briefings in Bioinformatics*, **11**, 457–472.
Patel, R.K. and Jain, M. (2012). NGS QC Toolkit: a toolkit for quality control of next generation sequencing data. *PloS one*, **7**, e30619.
Pevzner, P.A., Tang, H. and Waterman, M.S. (2001) An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, **98**, 9748–9753.
Phillippy, A.M., Schatz, M.C. and Pop, M. (2008) Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology*, **9**, R55.
Piro, V., Faoro, H., Weiss, V. *et al*. (2014) FGAP: an automated gap closing tool. *BMC Research Notes*, **7**, 371.
Pop, M. (2009) Genome assembly reborn: recent computational challenges. *Briefings in Bioinformatics*, **10**, 354–366.
Rahman, A. and Pachter, L. (2013) CGAL: computing genome assembly likelihoods. *Genome Biology*, **14**, R8.
Rodríguez-Ezpeleta, N., Hackenberg, M. and Aransay, A.M. (2012) *Bioinformatics for high throughput sequencing*, Springer, New York.
Ribeiro, F.J., Przybylski, D., Yin, S. *et al*. (2012) Finished bacterial genomes from shotgun sequence data. *Genome Research*, **22**, 2270–2277.
Ronaghi, M., Karamohamed, S., Pettersson, B. *et al*. (1996) Real-time DNA sequencing using detection of pyrophosphate release. *Analytical Biochemistry*, **242**, 84–89.
Rothberg, J.M., Hinz, W., Rearick, T.M. *et al*. (2011) An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, **475**, 348–352.
Salzberg, S.L., Phillippy, A.M., Zimin, A. *et al*. (2012) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, **22**, 557–567.
Sanger, F., Air, G.M., Barrell, B.G. *et al*. (1977) Nucleotide sequence of bacteriophage [phi]X174 DNA. *Nature*, **265**, 687–695.
Schmieder, R. and Edwards, R. (2011) Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, **27**, 863–864.
Shendure, J. and Ji, H. (2008) Next-generation DNA sequencing. *Nature Biotechnology*, **26**, 1135–1145.
Shendure, J., Mitra, R.D., Varma, C. and Church, G.M. (2004) Advanced sequencing technologies: methods and goals. *Nature Reviews Genetics*, **5**, 335–344.
Shendure, J., Porreca, G.J., Reppas, N.B. *et al*. (2005) Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, **309**, 1728–1732.

Simpson, J.T. and Durbin, R. (2012) Efficient *de novo* assembly of large genomes using compressed data structures. *Genome Research*, **22**, 549–556.
Simpson, J.T., Wong, K., Jackman, S.D. *et al*. (2009) ABySS: a parallel assembler for short read sequence data. *Genome Research*, **19**, 1117–1123.
Song, L., Florea, L. and Langmead, B. (2014) Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biology*, **15**, 509.
Sutton, G.G., White, O., Adams, M.D. *et al*. (1995) TIGR assembler: a new tool for assembling large shotgun sequencing projects. *Genome Science & Technology*, **1**, 9–19.
Trivedi, U.H., Cézard, T., Bridgett, S. *et al*. (2014) Quality control of next-generation sequencing data without a reference. *Frontiers in genetics*, **5**, 111.
Tsai, I.J., Otto, T.D. and Berriman, M. (2010) Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. *Genome Biology*, **11**, R41.
Turcatti, G., Romieu, A., Fedurco, M. and Tairi, A.P. (2008) A new class of cleavable fluorescent nucleotides: synthesis and optimization as reversible terminators for DNA sequencing by synthesis. *Nucleic Acids Research*, **36**, e25.
Warren, R.L., Sutton, G.G., Jones, S.J.M. and Holt, R.A. (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**, 500–501.
Yang, X., Chockalingam, S.P. and Aluru, S. (2013) A survey of error-correction methods for next-generation sequencing. *Briefings in Bioinformatics*, **14**, 56–66.
Zerbino, D.R. and Birney, E. (2008) Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Research*, **18**, 821–829.
Zimin, A.V., Delcher, A.L., Florea, L. *et al*. (2009) A whole-genome assembly of the domestic cow. *Bos taurus. Genome Biology*, **10**, R42.