Genomics 95 (2010) 315–327

Contents lists available at ScienceDirect

Genomics

journal homepage: www.elsevier.com/locate/ygeno

Review

Assembly algorithms for next-generation sequencing data

Jason R. Miller *, Sergey Koren, Granger Sutton

J. Craig Venter Institute, 9704 Medical Center Drive, Rockville MD 20850-3343, USA

ARTICLE INFO

Article history:
Received 4 August 2009
Accepted 2 March 2010
Available online 6 March 2010

Keywords:
Genome assembly algorithms
Next-generation sequencing

ABSTRACT

The emergence of next-generation sequencing platforms led to resurgence of research in whole-genome shotgun assembly algorithms and software. DNA sequencing data from the Roche 454, Illumina/Solexa, and ABI SOLiD platforms typically present shorter read lengths, higher coverage, and different error profiles compared with Sanger sequencing data. Since 2005, several assembly software packages have been created or revised specifically for de novo assembly of next-generation sequencing data. This review summarizes and compares the published descriptions of packages named SSAKE, SHARCGS, VCAKE, Newbler, Celera Assembler, Euler, Velvet, ABySS, AllPaths, and SOAPdenovo. More generally, it compares the two standard methods known as the de Bruijn graph approach and the overlap/layout/consensus approach to assembly.

Contents

Introduction

The advent of short-read sequencing machines gave rise to a new generation of assembly algorithms and software. This survey reviews algorithms for *de novo* whole-genome shotgun assembly from next-generation sequencing data. It describes and compares algorithms that have been presented in the scientific literature and implemented in software. We use a narrow definition of *de novo* whole-genome shotgun assembly. The shotgun process takes reads from random positions along a target molecule [1]. Whole-genome shotgun (WGS) sequencing samples the chromosomes that make up one genome. WGS assembly is the reconstruction of sequence up to chromosome length. The assembly task is relegated to computer software [2]. Assembly is possible when the target is over-sampled by the shotgun reads, such that reads overlap. *De novo* WGS assembly refers to reconstruction in its pure form, without consultation to previously resolved sequence including from genomes, transcripts, and proteins. *De novo* WGS assembly of next-generation sequencing (NGS) data is a

* Corresponding author.
E-mail address: jmiller@jcvi.org (J.R. Miller).

0888-7543/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ygeno.2010.03.001

specialized problem due to the short read lengths and large volumes of NGS data. Benchmarking the implementations is beyond the scope of this review. Broader introductions can be found elsewhere, *e.g.* [3].

Next-generation Sequence Data

Today's commercial DNA sequencing platforms include the Genome Sequencer from Roche 454 Life Sciences (www.454.com), the Solexa Genome Analyzer from Illumina (www.illumina.com), the SOLiD System from Applied Biosystems (www.appliedbiosystems.com), the Heliscope from Helicos (www.helicos.com), and the commercialized Polonator (www.polonator.org). These platforms have been well reviewed, e.g. [4–7]. A distinguishing characteristic of these platforms is that they do not rely on Sanger chemistry [8] as did first-generation machines including the Applied Biosystems Prism 3730 and the Molecular Dynamics MegaBACE. The second-generation machines are characterized by highly parallel operation, higher yield, simpler operation, much lower cost per read, and (unfortunately) shorter reads. Today's machines are commonly referred to as short-read sequencers or next-generation sequencers (NGS) though their successors may be on the horizon. Pacific Biosciences machines [9] might produce reads longer than first-generation machines. First-generation reads were commonly 500 bp to 1000 bp long. Today's NGS reads are in the 400 bp range (from 454 machines), the 100 bp range (from the Solexa and SOLID machines), or less. Shorter reads deliver less information per read, confounding the computational problem of assembling chromosome-size sequences. Assembly of shorter reads requires higher coverage, in part to satisfy minimum detectable overlap criteria. High coverage increases complexity and intensifies computational issues related to large data sets.

All sequencers produce observations of the target DNA molecule in the form of reads: sequences of single-letter base calls plus a numeric quality value (QV) for each base call [10]. Although QVs offer extra information, their use generally increases a program's CPU and RAM requirements. Only some of the NGS assembly software exploits QVs.

The NGS platforms have characteristic error profiles that change as the technologies improve. Error profiles can include enrichment of base call error toward the 3′ (terminal) ends of reads, compositional bias for or against high-GC sequence, and inaccurate determination of simple sequence repeats. There are published error profiles for the 454 GS 20 [11], the Illumina 1G Analyzer [12], and comparisons of three platforms [13]. Some NGS software is tuned for platform-specific error profiles. Others may have unintentional bias where development targeted one data type.

Sanger platforms could deliver paired-end reads, that is, pairs of reads with a constraint on their relative orientation and separation in the target. Paired ends were essential to assembly of cellular genomes small [14] and large [15] due to their ability to span repeats longer than individual reads. Paired ends, also called mate pairs, have a separation estimate that is usually provided to software as the fragment size distribution measured on a so-called library of reads. A sufficient variety of paired end separations should help resolve large chromosomes to single scaffolds [16]. Early NGS sequencers offered unpaired reads but late models support paired-end protocols. Early NGS assembly software targeted unpaired reads but later programs exploit paired ends as read placement constraints.

What is an Assembly?

An assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target. It groups reads into contigs and contigs into scaffolds. Contigs provide a multiple sequence alignment of reads plus the consensus sequence. The scaffolds, sometimes called supercontigs or metacontigs, define the contig order and orientation and the sizes of the gaps between contigs. Scaffold topology may be a simple path or a network. Most assemblers output, in addition, a set of unassembled or partially assembled reads. The most widely accepted data file format for an assembly is FASTA, wherein contig consensus sequence can be represented by strings of the characters A, C, G, T, plus possibly other characters with special meaning. Dashes, for instance, can represent extra bases omitted from the consensus but present in a minority of the underlying reads. Scaffold consensus sequence may have N's in the gaps between contigs. The number of consecutive N's may indicate the gap length estimate based on spanning paired ends.

Assemblies are measured by the size and accuracy of their contigs and scaffolds. Assembly size is usually given by statistics including maximum length, average length, combined total length, and N50. The contig N50 is the length of the smallest contig in the set that contains the fewest (largest) contigs whose combined length represents at least 50% of the assembly. The N50 statistics for different assemblies are not comparable unless each is calculated using the same combined length value. Assembly accuracy is difficult to measure. Some inherent measure of accuracy is provided by the degrees of mate-constraint satisfaction and violation [17]. Alignment to reference sequences is useful whenever trusted references exist.

The Challenge of Assembly

DNA sequencing technologies share the fundamental limitation that read lengths are much shorter than even the smallest genomes. WGS overcomes this limitation by over-sampling the target genome with short reads from random positions. Assembly software reconstructs the target sequence.

Assembly software is challenged by repeat sequences in the target. Genomic regions that share perfect repeats can be indistinguishable, especially if the repeats are longer than the reads. For repeats that are inexact, high-stringency alignment can separate the repeat copies. Careful repeat separation involves correlating reads by patterns in the different base calls they may have [18]. Repeat separation is assisted by high coverage but confounded by high sequencing error. For repeats whose fidelity exceeds that of the reads, repeat resolution depends on “spanners,” that is, single reads that span a repeat instance with sufficient unique sequence on either side of the repeat. Repeats longer than the reads can be resolved by spanning paired ends, but the analysis is more complicated. Complete resolution usually requires two resources: pairs that straddle the repeat with each end in unique sequence, and pairs with exactly one end in the repeat. The limit of repeat resolution can be explored for finished genomes under some strict assumptions. For instance, it was shown that the theoretical best assembly of the *E. coli* genome from 20 bp unpaired reads would put 10% of bases in contigs of 10 Kbp or longer given infinite coverage and error-free reads [19]. The limit calculation is not straightforward for reads with sequencing error, paired-end reads, or unfinished genomes. Careful estimates of repeat resolution involve the ratio of read length (or paired-end separation) to repeat length, repeat fidelity, read accuracy, and read coverage. In regard to NGS data, shorter reads have less power to resolve genomic repeats but higher coverage increases the chance of spanning short repeats.

Repeat resolution is made more difficult by sequencing error. Software must tolerate imperfect sequence alignments to avoid missing true joins. Error tolerance leads to false positive joins. This is a problem especially with reads from inexact (polymorphic) repeats. False-positive joins can induce chimeric assemblies. In practice, tolerance for sequencing error makes it difficult to resolve a wide range of genomic phenomena: polymorphic repeats, polymorphic differences between non-clonal asexual individuals, polymorphic differences between non-inbred sexual individuals, and polymorphic haplotypes from one non-inbred individual. If the sequencing platforms ever generate error-free reads at high coverage, assembly software might be able to operate at 100% stringency.

WGS assembly is confounded by non-uniform coverage of the target. Coverage variation is introduced by chance, by variation in cellular copy number between source DNA molecules, and by compositional bias of sequencing technologies. Very low coverage induces gaps in assemblies. Coverage variability invalidates coverage-based statistical tests, and undermines coverage-based diagnostics designed to detect over-collapsed repeats.

WGS assembly is complicated by the computational complexity of processing larger volumes of data. For efficiency, all assembly software relies to some extent on the notion of a K-mer. This is a sequence of K base calls, where K is any positive integer. In most implementations, only consecutive bases are used. Intuitively, reads with high sequence similarity must share K-mers in their overlapping regions, and shared K-mers are generally easier to find than overlaps. Fast detection of shared K-mer content vastly reduces the computational cost of assembly, especially compared to all-against-all pair-wise sequence alignment. A tradeoff of K-mer based algorithms is lower sensitivity, thus missing some true overlaps. The probability that a true overlap spans shared K-mers depends on the value of K, the length of the overlap, and the rate of error in the reads. An appropriate value of K should be large enough that most false overlaps don't share K-mers by chance, and small enough that most true overlaps do share K-mers. The choice should be robust to variation in read coverage and accuracy.

WGS assembly algorithms, and their implementations, are typically complex. Assembly operation can require high-performance computing platforms for large genomes. Algorithmic success can depend on pragmatic engineering and heuristics, that is, empirically derived rules of thumb. Heuristics help overcome convoluted repeat patterns in real genomes, random and systematic error in real data, and the physical limitations of real computers.

Metagenomics

Metagenomics is the sequencing of DNA in an environmental sample. Whereas WGS targets one genome, metagenomics usually targets several. The metagenomics assembly problem is confounded by genomic diversity and variable abundance within populations. Assembly reconstructs the most abundant sequences [20]. Simulations indicate high rates of chimera, especially in short contigs assembled from complex mixtures [21]. Studies that rely on characterization of individual reads prefer long reads [22]. The role for *de novo* genomic assembly from NGS metagenomics data should grow as NGS read lengths and NGS paired-end options increase.

Graph Algorithms for Assembly

We organize the NGS assemblers into three categories, all based on graphs. The Overlap/Layout/Consensus (OLC) methods rely on an overlap graph. The de Bruijn Graph (DBG) methods use some form of K-mer graph. The greedy graph algorithms may use OLC or DBG.

A graph is an abstraction used widely in computer science. It is a set of nodes plus a set of edges between the nodes. Nodes and edges may also be called vertices and arcs, respectively. If the edges may only be traversed in one direction, the graph is known as a directed graph. The graph can be conceptualized as balls in space with arrows connecting them. Importantly, each directed edge represents a connection from one source node to one sink node. Collections of edges form paths that visit nodes in some order, such that the sink node of one edge forms the source node for any subsequent nodes. A special kind of path, called a simple path, is one that contains only distinct nodes (each node is visited at most once). A simple path may not intersect itself, by definition, and one may additionally require that no other paths intersect it. The nodes and edges may be assigned a variety of attributes and semantics.

An overlap graph represents the sequencing reads and their overlaps [23]. The overlaps must be pre-computed by a series of (computationally expensive) pair-wise sequence alignments. Conceptually, the graph has nodes to represent the reads and edges to represent overlaps. In practice, the graph might have distinct elements or attributes to distinguish the 5′ and 3′ ends of reads, the forward and reverse complement sequences of reads, the lengths of reads, the lengths of overlaps, and the type of overlap (suffix-to-prefix or containment). Paths through the graph are the potential contigs, and paths can be converted to sequence. Paths may have mirror images representing the reverse complement sequence. There are two ways to force paths to obey the semantics of double-stranded DNA. If the graph has separate nodes for read ends, then paths must exit the opposite end of the read they enter. If the graph has separate edges for the forward and reverse strands, then paths must exit a node on the same strand they enter.

The de Bruijn graph was developed outside the realm of DNA sequencing to represent strings from a finite alphabet. The nodes represent all possible fixed-length strings. The edges represent suffix-to-prefix perfect overlaps.

A K-mer graph is a form of de Bruijn graph. Its nodes represent all the fixed-length subsequences drawn from a larger sequence. Its edges represent all the fixed-length overlaps between subsequences that were consecutive in the larger sequence. In one formulation [24], there is one edge for the K-mer that starts at each base (excluding the last K-1 bases). The nodes represent overlaps of K-1 bases. Alternately [25], there is one node representing the K-mer that starts at each base. The edges represent overlaps of K-1 bases. By construction, the graph contains a path corresponding to the original sequence (Fig. 1). The path converges on itself at graph elements representing K-mers in the sequence whose multiplicity is greater than one.

A repeat graph is an application of the K-mer graph [26]. It provides a succinct graph representation of the repetitiveness of a genome. Nodes and edges are drawn from an assembled reference sequence. Whereas non-repetitive genomic sequence would induce a single path through the graph, repeats induce convergence and divergence of paths, as well as cycles. Repeat graphs can be used to identify and catalog repeats [27].

A K-mer graph may represent many sequences instead of one. In its application to WGS assembly, the K-mer graph represents the input reads. Each read induces a path. Reads with perfect overlaps induce a common path. Thus, perfect overlaps are detected implicitly without any pair-wise sequence alignment calculation (Fig. 2). Compared to overlap graphs, K-mer graphs are more sensitive to repeats and sequencing errors. Paths in overlap graphs converge at repeats longer than a read, but paths in K-mer graphs converge at perfect repeats of length K or more, and K must be less than the read

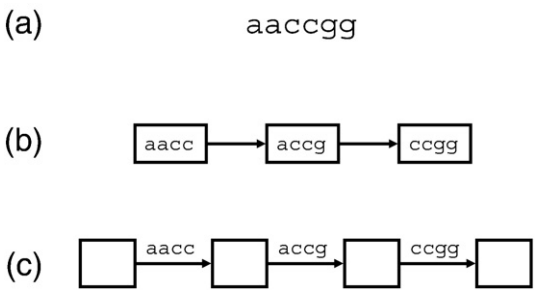

Fig. 1. A read represented by K-mer graphs. (a) The read is represented by two types of K-mer graph with K = 4. Larger values of K are used for real data. (b) The graph has a node for every K-mer in the read plus a directed edge for every pair of K-mers that overlap by K-1 bases in the read. (c) An equivalent graph has an edge for every K-mer in the read and the nodes implicitly represent overlaps of K-1 bases. In these examples, the paths are simple because the value K = 4 is larger than the 2 bp repeats in the read. The read sequence is easily reconstructed from the path in either graph.

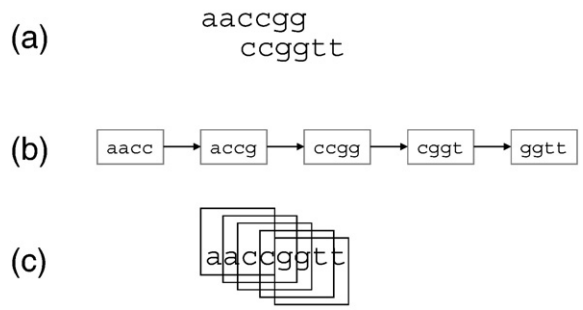

Fig. 2. A pair-wise overlap represented by a K-mer graph. (a) Two reads have an error-free overlap of 4 bases. (b) One K-mer graph, with K = 4, represents both reads. The pair-wise alignment is a by-product of the graph construction. (c) The simple path through the graph implies a contig whose consensus sequence is easily reconstructed from the path.

length. Each single-base sequencing error induces up to K false nodes in the K-mer graph. Each false node has a chance of matching some other node and thereby inducing a false convergence of paths.

Real-world WGS data induces problems in overlap graphs and K-mer graphs.

- Spurs are short, dead-end divergences from the main path (Fig. 3a). They are induced by sequencing error toward one end of a read. They can be induced by coverage dropping to zero.
- Bubbles are paths that diverge then converge (Fig. 3b). They are induced by sequencing error toward the middle of a read, and by polymorphism in the target. Efficient bubble detection is non-trivial [28].
- Paths that converge then diverge form the frayed rope pattern (Fig. 3c). They are induced by repeats in the target genome.
- Cycles are paths that converge on themselves. They are induced by repeats in the target. For instance, short tandem repeats induce small cycles.

In general, branching and convergence increases graph complexity, leading to tangles that are difficult to resolve. Much of the complexity is due to repeats in the target and sequencing error in the reads.

In the graph context, assembly is a graph reduction problem. Most optimal graph reductions belong to a class of problems, called NP-hard, for which no efficient solution is known [29]. Therefore,

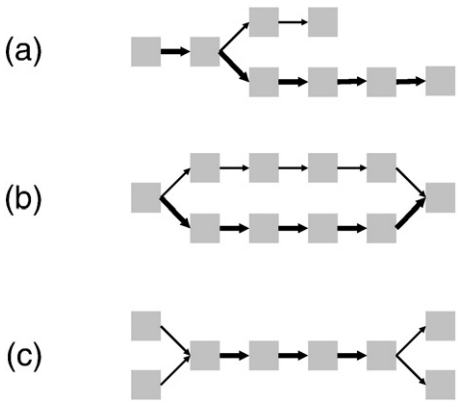

Fig. 3. Complexity in K-mer graphs can be diagnosed with read multiplicity information. In these graphs, edges represented in more reads are drawn with thicker arrows. (a) An errant base call toward the end of a read causes a "spur" or short dead-end branch. The same pattern could be induced by coincidence of zero coverage after polymorphism near a repeat. (b) An errant base call near a read middle causes a "bubble" or alternate path. Polymorphisms between donor chromosomes would be expected to induce a bubble with parity of read multiplicity on the divergent paths. (c) Repeat sequences lead to the "frayed rope" pattern of convergent and divergent paths.

assemblers rely on heuristic algorithms and approximation algorithms to remove redundancy, repair errors, reduce complexity, enlarge simple paths and otherwise simplify the graph.

Greedy Graph-based Assemblers

The first NGS assembly packages used greedy algorithms. These have been reviewed well elsewhere [3,30].

The greedy algorithms apply one basic operation: given any read or contig, add one more read or contig. The basic operation is repeated until no more operations are possible. Each operation uses the next highest-scoring overlap to make the next join. The scoring function measures, for instance, the number of matching bases in the overlap. Thus the contigs grow by greedy extension, always taking on the read that is found by following the highest-scoring overlap. The greedy algorithms can get stuck at local maxima if the contig at hand takes on reads that would have helped other contigs grow even larger.

The greedy algorithms are implicit graph algorithms. They drastically simplify the graph by considering only the high-scoring edges. As an optimization, they may actually instantiate just one overlap for each read end they examine. They may also discard each overlap immediately after contig extension.

Like all assemblers, the greedy algorithms need mechanisms to avoid incorporating false-positive overlaps into contigs. Overlaps induced by repetitive sequence may score higher than overlaps induced by common position of origin. An assembler that builds on false-positive overlaps will join unrelated sequences to either side of a repeat to produce chimera.

SSAKE [31] was the first short-read assembler. It was designed for unpaired short reads of uniform length. It was based on the notion that high coverage would provide a tiling in error-free reads if the erroneous reads could be avoided. SSAKE does not use a graph explicitly. It does use a lookup table of reads indexed by their prefixes. SSAKE iteratively searches for reads that overlap one contig end. Its candidate reads must have a prefix-to-suffix identical overlap whose length is above a threshold. SSAKE chooses carefully among multiple reads with equally long overlaps. First, it prefers reads with end-to-end confirmation in other reads. This favors error-free reads. Second, the software detects when the set of candidates presents multiple extensions. In particular, it detects when the candidate read suffixes exhibit differences that are each confirmed in other reads. This is equivalent to finding a branch in a graph. At this point, the software terminates the contig extension. Users can elect to override the "stringent" behavior, in which case SSAKE takes the higher-scoring extension. When no reads satisfy the initial minimum threshold, the program decrements the threshold until a second minimum is reached. Thus, user settings determine how aggressively SSAKE extends through possible repeat boundaries and low-coverage regions. SSAKE has been extended to exploit paired-end reads and imperfectly matching reads [32].

SHARCGS [33] also operates on uniform-length, high-coverage, unpaired short reads. It adds pre- and post-processor functionality to the basic SSAKE algorithm. The pre-processor filters erroneous reads by requiring a minimum number of full-length exact matches in other reads. An even higher-stringency filter is optional, requiring that the combined QVs of matching reads exceed a minimum threshold. SHARCGS filters the raw read set three times, each at a different stringency setting, to generate three filtered sets. It assembles each set separately by iterative contig extension. Then, in a post-process, it merges the three contig sets using sequence alignment. The merger aims to extend contigs from highly confirmed reads by integrating longer contigs from lower-stringency filters.

VCAKE [34] is another iterative extension algorithm. Unlike its predecessors, it could incorporate imperfect matches during contig extension. VCAKE was later combined with Newbler in a pipeline for Solexa+454 hybrid data [35]. Another pipeline had combined

Newbler and Celera Assembler for 454+Sanger hybrid data [36]. Both pipelines "shred" contigs from the first assembler to produce pseudo-reads suitable for the second assembler. The latter pipeline adjusts the read coverage and base call quality values in the pseudo-reads it generates. This helps the secondary assembler give appropriate weight to high-coverage contigs from the primary assembly, for instance during consensus base calling.

Overlap/Layout/Consensus Assemblers

The OLC approach was typical of the Sanger-data assemblers. It was optimized for large genomes in software including Celera Assembler [37], Arachne [38,39], and CAP and PCAP [40]. The OLC approach has been reviewed elsewhere [41–43].

OLC assemblers use an overlap graph. Their operation has three phases:

1. Overlap discovery involves all-against-all, pair-wise read comparison. The seed & extend heuristic algorithm is used for efficiency. The software pre-computes K-mer content across all reads, selects overlap candidates that share K-mers, and computes alignments using the K-mers as alignment seeds. Overlap discovery is sensitive to settings of K-mer size, minimum overlap length, and minimum percent identity required for an overlap. These three parameters affect robustness in the face of base calling error and low-coverage sequencing. Larger parameter values lead to more accurate but shorter contigs. Overlap discovery can run in parallel with a matrix partition.
2. Construction and manipulation of an overlap graph leads to an approximate read layout. The overlap graph need not include the sequence base calls, so large-genome graphs may fit into practical amounts of computer memory.
3. Multiple sequence alignment (MSA) determines the precise layout and then the consensus sequence. There is no known efficient method to compute the optimal MSA [44]. Therefore, the consensus phase uses progressive pair-wise alignments guided by, for instance, the approximate read layout. The consensus phase must load the sequence base calls into memory. It can run in parallel, partitioned by contig.

Newbler [45] is widely used software distributed by 454 Life Sciences. The first release, described in the published supplement, targeted unpaired reads of approximately 100 bp as generated by the GS 20 machine. Newbler has since been revised, in particular to build scaffolds from paired-end constraints. As described in 2005, Newbler implements OLC twice. The first-phase OLC generates unitigs from reads. Unitigs are mini-assemblies that are, ideally, uncontested by overlaps to reads in other unitigs [37]. The unitigs serve as preliminary, high-confidence, conservative contigs that seed the rest of the assembly pipeline. The second-phase OLC generates larger contigs from the unitigs. This phase joins unitigs into a contig layout based on pair-wise overlaps between unitigs. It may split unitigs whose prefix and suffix align to different contigs. Unitig splitting may split individual reads, leading to reads placed in multiple contigs. Such reads may have been chimera or derived from a repeat boundary.

Newbler exploits coverage, if possible, to overcome base calling error. In particular, it uses instrument metrics to overcome inaccurate calls of the number of bases in homopolymer repeats. Newbler calculates unitig and contig consensus in "flow space" using the platform-supplied signal strength associated with each flow of a particular nucleotide. The normalized signal is proportionally correlated to the number of direct repeats of that nucleotide at that position in the read. Consensus calculation in "base space," equivalent to rounding the signals before averaging, would sacrifice precision. For each column in the MSA, Newbler calculates the average signal and rounds to an integer to form the consensus.

The Newbler package offers functionality beyond *de novo* assembly, and it includes a comprehensive user guide. The Newbler software is distributed with the 454 sequencing machines. Customers receive frequent updates. Release descriptions indicate that recent versions differ from the published algorithm. The source code is not generally available.

The Celera Assembler [37] is a Sanger-era OLC assembler revised for 454 data [46]. The revised pipeline, called CABOG, discovers overlaps using compressed seeds. CABOG reduces homopolymer runs, that is, repeats of single letters, to single bases to overcome homopolymer run length uncertainty in data. CABOG builds initial unitigs excluding reads that are substrings of other reads. Substrings account for a large portion of the data in high-coverage 454 data due to highly variable read lengths. CABOG avoids the substring-reads initially because they are more susceptible to repeat-induced false overlaps.

CABOG applies a base call correction scheme first described for Arachne [38]. It compares each read to its set of overlapping reads. It infers sequencing error at any base contradicted by a preponderance of overlaps. It does not fix the read. Rather, it adjusts the tabulated error rates in overlaps spanning the inferred error. Next, it applies a user-supplied threshold for error rates. From the overlaps that survive the error filter and a filter for minimum alignment length, CABOG selects one "best" overlap per read end. Best is defined as aligning the most bases. The best-overlap filter is presumed to eliminate many of the same overlaps removed by the more time-costly transitive edge removal algorithm [23], which was employed by the original Celera Assembler. CABOG constructs an overlap graph from the reads and "best" overlaps. Within the graph, it builds unitigs from maximal simple paths that are free of branches and intersections. CABOG next constructs a graph of unitigs plus paired-end constraints. Within that graph, it joins unitigs into contigs and connects contigs into scaffolds. It applies a series of graph reductions including removal of transitively inferable edges. Finally, CABOG derives consensus sequences by computing multiple sequence alignments from the scaffold layouts plus the read sequences.

Two assemblers apply the OLC approach to the short reads from the Solexa and SOLiD platforms. The Edena software [47] discards duplicate reads and finds all perfect, error-free, overlaps. It removes individual overlaps that are redundant with pairs of other overlaps, an application of the transitive overlap reduction algorithm [23]. Edena prunes spurs and bubbles. Edena was designed for unpaired reads of uniform length. The Shorty software [48] attacks the special case where a few long reads are available to act as seeds to recruit short reads and their mate pairs. Proceeding in iterations, Shorty uses contigs to seed larger contigs.

The de Bruijn Graph Approach

The third approach to assembly is most widely applied to the short reads from the Solexa and SOLiD platforms. It relies on K-mer graphs, whose attributes make it attractive for vast quantities of short reads. The K-mer graph does not require all-against-all overlap discovery, it does not (necessarily) store individual reads or their overlaps, and it compresses redundant sequence. Conversely, the K-mer graph does contain actual sequence and the graph can exhaust available memory on large genomes. Distributed memory approaches may alleviate this constraint.

The K-mer graph approach dates to an algorithm for Sanger read assembly [24] based on a proposal [49] for an even older sequencing technology; see [3] for review. The approach is commonly called a de Bruijn graph (DBG) approach or an Eulerian approach [50] based on an ideal scenario. Given perfect data – error-free K-mers providing full coverage and spanning every repeat – the K-mer graph would be a de Bruijn graph and it would contain an Eulerian path, that is, a path that traverses each edge exactly once. The path would be trivial to find

making the assembly problem trivial by extension. Of course, K-mer graphs built from real sequencing data are more complicated.

To the extent that the data is ideal, assembly is a by-product of the graph construction. The graph construction phase proceeds quickly using a constant-time hash table lookup for the existence of each K-mer in the data stream. Although the hash table consumes extra memory, the K-mer graph itself stores each K-mer at most once, no matter how many times the K-mer occurs in the reads. In terms of computer memory, the graph is smaller than the input reads to the extent that the reads share K-mers.

Pevzner [49] explored problems that genomic repeats introduce. Repeats induce cycles in the K-mer graph. These would allow more than one possible reconstruction of the target. Idury and Waterman [24] also explored problems of real data. They added two extra types of information to the K-mer graph and named the result a sequence graph. Each edge was labeled with the reads, and positions within each read, of the sequences that induced it. Where nodes had one inbound and one outbound edge, the three elements could be compressed into one edge. This was called the elimination of singletons. Further research led to the Euler software implementation [50] for Sanger data. Impractical for large-scale Sanger sequencing projects, Euler and the DBG approach were well positioned when the Illumina platform started to produce data composed of very short unpaired reads of uniform size.

Three factors complicate the application of K-mer graphs to DNA sequence assembly.

1. DNA is double stranded. The forward sequence of any given read may overlap the forward or reverse complement sequence of other reads. One K-mer graph implementation contains nodes and edges for both strands, taking care to avoid output of the entire assembly twice [24]. Another implementation stores forward and reverse sequence together as cognate half-nodes with the constraint that paths enter and exit the same half [25]. Yet another implementation represents alternate strands in a single node with two sides, constraining paths to enter and exit opposite sides [51].
2. Real genomes present complex repeat structures including tandem repeats, inverted repeats, imperfect repeats, and repeats inserted within repeats. Repeats longer than K lead to tangled K-mer graphs that complicate the assembly problem. Perfect repeats of length K or greater collapse inside the graph, leaving a local graph structure that resembles a rope with frayed ends (Fig. 3c); paths converge for the length of the repeat and then they diverge. Successful assembly requires separation of the converged path, which represents a collapsed repeat. The graph contains insufficient information to disambiguate the repeat. Assemblers typically consult the reads, and possibly the mate pairs, in attempts to resolve these regions.
3. A palindrome is a DNA sequence that is its own reverse complement. Palindromes induce paths that fold back on themselves. At least one assembler avoids these elegantly; Velvet [25] requires K, the length of a K-mer, to be odd. An odd-size K-mer cannot match its reverse complement.
4. Real data includes sequencing error. DBG assemblers use several techniques to reduce sensitivity to this problem. First, they preprocess the reads to remove error. Second, they weight the graph edges by the number of reads that support them, and then "erode" the lightly supported paths. Third, they convert paths to sequences and use sequence alignment algorithms to collapse nearly identical paths. Many of these techniques derive from the Euler family of assemblers.

The de Bruijn Graph in Euler

The Euler software was developed for Sanger reads [26,50,52]. It was subsequently modified for short 454 GS20 reads [53], even shorter unpaired Illumina/Solexa reads [54], and paired-end Solexa reads [55].

Euler applies a filter to the reads before it builds its graph. The filter detects erroneous base calls by noting low-frequency K-mers. The filter relies on redundancy of reads: most true K-mers should be repeated in several reads. The filter also relies on randomness of sequencing error: for any K where 4^K exceeds twice the genome size, most erroneous K-mers should be unique. The Euler filter is implemented with a list of K-mers and their observed frequencies in the reads. The filter excludes or corrects low-frequency K-mers. Correction is especially important for short reads with high error and high coverage. Correction reduces the total number of K-mers and thus the node count in the graph. Correction risks masking true polymorphism [50]. Correction can corrupt valid K-mers that had low coverage by chance. Correction could leave a read incorrect by settling on K-mers that each occur in several reads but never occur together in any read. OLC assemblers have an analogous base call correction step that uses overlaps rather than K-mers.

The Euler filter process is called spectral alignment [50]. The software identifies sequencing error by comparing K-mer content between individual reads and all reads. It distrusts individual-read K-mers whose frequency in all reads is below a threshold. The threshold is chosen after calculating the distribution of K-mer frequencies present in the reads. The distribution is usually bi-modal. The first peak represents the many K-mers that occur once or twice, due to sequencing error (or low coverage). The second peak represents the redundant K-mers induced by the read coverage (or genomic repeats). Euler selects the threshold between the peaks and effectively labels all K-mers as bad or good. Euler then examines each read. For each read with bad K-mers, it executes a greedy exploration for base call substitutions that reduce the bad K-mer count [55]. Finally, it either accepts a fully corrected read or rejects the read. (Rejected reads can be re-introduced to patch low-coverage regions after assembly.) Note Euler corrects substitution errors but not insertions or deletions, i.e. indels. Substitutions are the most common base call error type in Solexa data [12].

Euler builds a K-mer graph from the filtered and corrected reads. Then it applies a series of graph manipulations designed to overcome the effects of sequencing errors and genomic repeats.

By processing K-mers not reads, the K-mer graph construction discards long-range continuity information in the reads. Euler repairs this defect by threading the reads through the graph. Mapping a read onto the graph is easy. At least initially, the K-mers in reads map to unique nodes and reads are consistent with some path. Exploitation of the mapping is more complex. Reads ending inside a repeat are consistent with any path exiting the repeat, but reads spanning a repeat are consistent with fewer paths. For the latter, read threading pulls out one piece of string from a frayed rope pattern, thus resolving one copy of the collapsed repeat (Fig. 4a). Thus read threading constrains the set of valid paths through the graph. This allows resolution of repeats whose length is between K and the read length.

A paired-end read is effectively a long read that is missing base calls in the middle. Euler treats paired ends this way to resolve repeats longer than individual reads. The technique could be called mate threading. Paired ends that span a repeat provide the evidence to join one path that enters a repeat to one path that exits the repeat (Fig. 4b). Paired ends can also resolve some complex tangles induced by repeats. A complex graph may have multiple paths between two nodes corresponding to opposite ends of a mate pair. Each path implies a putative DNA sequence. In many cases, only one of the paths implies a sequence whose length satisfies the paired-end constraint (Fig. 4c). Between any mate pair, there could be too many paths for exhaustive search to be feasible. Euler seems to restrict the search space using the mate constraint as a bound on path length. (The number of paths in a general graph scales with N^E for N nodes and E out-going edges per node. K-mer graphs of DNA sequence can be constrained to $E \leq 4$ to represent the 4 possible one-letter extensions to a sequence. The constraint is violated by some graph simplifications, and N^4 is still not tractable.)

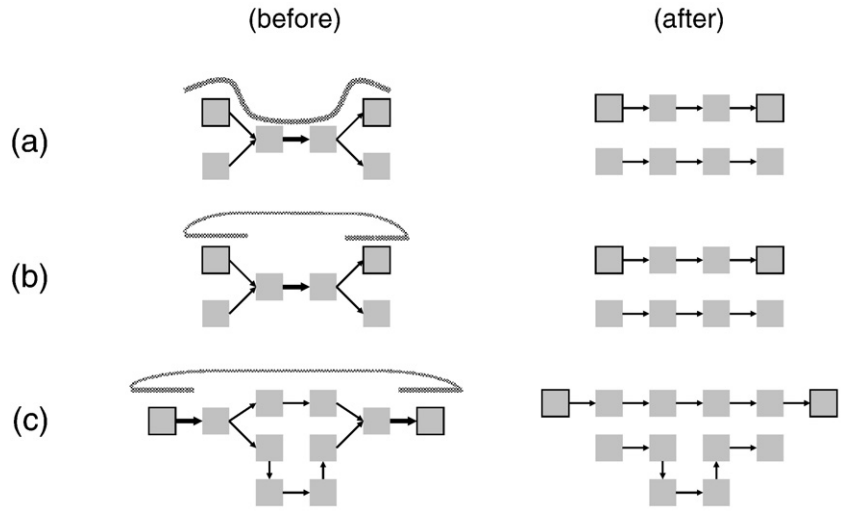

Fig. 4. Three methods to resolve graph complexity. (a) Read threading joins paths across collapsed repeats that are shorter than the read lengths. (b) Mate threading joins paths across collapsed repeats that are shorter than the paired-end distances. (c) Path following chooses one path if its length fits the paired-end constraint. Reads and mates are shown as patterned lines. Not all tangles can be resolved by reads and mates. The non-branching paths are illustrative; they could be simplified to single edges or nodes.

After threading, Euler implements graph simplifications at regions with low and high depth of coverage in reads. Euler's spur erosion reduces branching in graph paths and thereby lengthens simple paths. The spurs are presumed due to sequencing error that survived the spectral alignment filter. Euler identifies remaining edges that appear repetitive and removes them from the set of paths. This is equivalent to breaking contigs at repeat boundaries in OLC assembly.

Reads from many platforms contain lower quality base calls at their 3′ ends, and the problem can be exacerbated by long-read protocols on short-read platforms. Euler addresses this problem by trusting read prefixes more than their suffixes. It chooses trustable prefixes during the error correction step. Prefix length varies per read. During read threading, prefixes and suffixes can map to multiple paths. By a heuristic, Euler trusts mappings that are significantly better than their second-best choice. Just as the suffixes would add coverage to a multiple sequence alignment, they add connectivity to the graph. The extra sequence leads to greater contig size. Euler chooses not to alter the assembly consensus sequence based on the suffixes, so the mapped suffixes contribute connectivity only.

Overlap graphs are sensitive to the minimum overlap length threshold, and K-mer graphs are sensitive to the parameter K. Larger values of K resolve longer repeats but they also fracture assemblies in regions of low read coverage. Euler addresses this with a heuristic. Euler constructs and simplifies two K-mer graphs with different values of K. It identifies edges present in the smaller-K graph that are missing in the larger-K graph. It adds corresponding pseudo-edges to the second graph. The borrowed edges extend paths in the second graph and thus enlarge contigs in the assembly. This technique effectively uses large K-mers to build reliable initial contigs, and then fills gaps with more prolific small K-mers. This is analogous to gap filling approaches in OLC assemblers [37].

Some of the Euler software incorporates another structure called the A-Bruijn graph. It gets its name from being a combination of a de Bruijn graph and an adjacency matrix or A-matrix. Nodes of the graph represent consecutive columns in multiple sequence alignments. Compared to nodes representing K-mers in individual reads, the adjacency nodes can be less sensitive to sequencing error. The A-Bruijn graph was deployed for converting a genome sequence to a repeat graph and classifying repeats. It was proposed as a basis for assembly [26].

In summary, Euler compares de Bruijn graphs built from different K-mer sizes. Euler applies heuristics to mitigate graph complexity induced by sequencing error. It exploits low-quality read ends and paired-end constraints to tease apart graph tangles induced by genomic repeats. The software targets de novo assembly from short reads, including paired-ends, from the Solexa platform.

The de Bruijn Graph in Velvet

Velvet [25,56] is a reliable and easy to use DBG assembler. Velvet makes extensive use of graph simplification to reduce simple non-intersecting paths to single nodes. Simplification compresses the graph without loss of information. Velvet invokes the simplification step during graph construction and again several times during the assembly process. The technique, introduced as elimination of singletons for K-mer graphs [24], is analogous to unitig formation in overlap graphs [23] and OLC assemblers [37].

Velvet prunes the K-mer graph by removing spurs iteratively. Its tip removal algorithm is similar to Euler's erosion procedure. The spur removal drastically reduced the graph size on real data [25], possibly because it was the pipeline's first attempt at filtering out base call errors. Velvet does not implement Euler's spectral alignment filter. Velvet has a parameter for the minimum number of occurrences in the reads for a K-mer to qualify as a graph node. The Velvet publication seems to discourage use of this naïve filter.

Velvet reduces graph complexity with a bounded search for bubbles in the graph. Velvet's tour bus algorithm uses breadth-first-search, fanning out as much as possible, starting at nodes with multiple out-going edges. Since graphs of real data can have bubbles within bubbles, an exhaustive search for all bubbles would be impractical. The search is bounded to make it tractable; the candidate paths are traversed in step, moving ahead one node on all paths per iteration, until the path lengths exceed a threshold. Velvet narrows the bubble candidates to those with a sequence similarity requirement on the alternate paths. Having found a bubble, Velvet removes the path representing fewer reads and, working outside the graph, re-aligns reads from the removed path to the remaining path. Because higher read multiplicity determines the target path, the re-aligner effectively calls the consensus bases by a column-wise voting algorithm. The operation risks “papering over” genuine sequence differences due to polymorphism in the donor DNA or over-collapse of near-identical repeats. Velvet's algorithm is similar to bulge removal

in Euler [26] and analogous to bubble detection and bubble smoothing in OLC assemblers [28].

Velvet further reduces graph complexity by read threading. It removes paths that represent fewer reads than a threshold. This operation risks removing low-coverage sequence but it is thought to remove mostly spurious connections induced by convergent sequencing errors. Velvet exploits long reads, if any were provided, by an algorithm it calls Rock Band. This forms nodes out of paths that are confirmed by two or more long reads provided no other two long reads provide a consistent contradiction.

Velvet's final graph reduction involves mate pairs. Early versions used an algorithm called breadcrumb [25] that is similar to mate pair threading in DBG algorithms [52] and gap filling in OLC algorithms [37]. It operated on pairs of long contigs (simple paths) connected by paired-ends. Using the long contigs as anchors, it tried to fill the gap between them with short contigs. It gathered short contigs linked to the long contigs, and applied a breadth-first search through the DBG for a single path linking the long contigs and by traversing the short contigs. Later versions of Velvet use an algorithm called pebble [56]. In this algorithm, unique and repeat contigs substitute for breadcrumb's long and short contigs, respectively. The unique/repeat classifier is based on read coverage per contig. Velvet uses a statistical test similar to Celera Assembler's A-stat [37] and a given coverage expectation. It exploits the given (per-library) insert length distribution to build an approximate contig layout. It searches the DBG for a path that is consistent with the layout.

Velvet may be run several times per data set to optimize selection of three critical parameters. The length of the K-mers is constrained to be odd to preclude nodes representing palindromic repeats. The minimum frequency expected of K-mers in the reads determines which K-mers are pruned *a-priori*. The expected coverage of the genome in reads controls spurious connection breaking.

In summary, Velvet offers a full implementation of DBG assembly. It does not use an error-correction pre-processor, though it does have an error-avoidance read filter. It applies a series of heuristics that reduce graph complexity. The heuristics exploit local graph topology, read coverage, sequence identity, and paired-end constraints. The software targets de novo assembly from short reads with paired ends from the Solexa platform. An extension allows it to assemble data sets composed solely of SOLiD reads (www.solidsoftwaretools.com). Memory requirements currently preclude Velvet from assembling very large genomes.

The de Bruijn Graph in ABySS

ABySS is a distributed implementation [51] designed to addresses memory limitations of mammalian-size genome assembly by DBG. ABySS distributes the K-mer graph, and the graph computations, across a compute grid whose combined memory is quite large. This scheme allowed it to assemble 3.5 billion Solexa reads from a human donor.

Several problems confront any single-CPU algorithm that is ported to a grid. It must be possible to partition the problem for parallel computation, to distribute the partitions evenly across the grid, and to marshal the results. ABySS partitions the assembly at the granularity of the individual graph node. (Each graph node is processed separately. For efficiency, many graph nodes are assigned to each CPU.) The assignment of a graph node to CPU is accomplished by converting the K-mer to an integer. The formula is strand-neutral such that a K-mer and its reverse complement map to the same integer. The formula would be helpful if it somehow mapped neighbor K-mers to the same CPU. It is not clear to what extent this was accomplished.

ABySS introduces parallel implementations of graph simplifications present in Euler and Velvet. ABySS iteratively removes spurs shorter than a threshold. ABySS performs bubble smoothing by bounded search and prefers the path supported by more reads. It also transforms simple non-intersecting paths into contigs and performs mate threading. ABySS uses a compact representation of the K-mer graph. Each graph node represents a K-mer and its reverse complement. Each graph node keeps 8 bits of extra information: the existence or non-existence of each of the four possible one-letter extensions at each end. The graph edges are implicit in this extra information. ABySS follows paths in parallel starting at arbitrary graph nodes per CPU. From any node, ABySS finds its successor elegantly: the node's last K-1 bases, plus a one-base extension indicated by an edge, is converted numerically to the address (including CPU assignment) of the successor node. When a path traverses a node on a different CPU, the process emits a request for information. Since inter-CPU communication is typically slow, the process works on other graph nodes while waiting for the response.

In a post-process, ABySS exploits paired-end reads to merge contigs. ABySS does not build scaffolds. In summary, ABYSS is scalable assembly software for Solexa short reads and paired end reads.

The de Bruijn Graph in AllPaths

AllPaths is a DBG assembler intended for application to large genomes. It was published with results on simulated data [57] and revised for real data [58].

AllPaths uses a read-correcting pre-processor related to spectral alignment in Euler. It trusts K-mers that occur at high frequency in reads and at high quality (where each base must be confirmed by a minimum number of base calls with QV above a threshold). It retains reads whose K-mers are trusted. The filter operates on K-mers for three values of K. The filter is relaxed in two ways. Reads are restored if up to two substitutions to low-QV base calls make its K-mers trusted. K-mers are restored later if they are essential for building a path between paired-end reads.

AllPaths invokes a second pre-processor that creates “unipaths.” The process begins with the calculation of perfect read overlaps seeded by K-mers. It assigns numerical identifiers to the K-mers such that many K-mers seen consecutively in reads and overlaps receive consecutive identifiers. It populates a database of identifier intervals and the read links between them. It merges intervals to an extent that is consistent with all the reads. The operation is equivalent to DBG construction followed by elimination of singletons. The database implementation probably reduces the RAM requirement for the graph construction, which comes next.

AllPaths builds a DBG from the database. Its first graph operation is spur erosion, which it calls unitig graph shaving. AllPaths partitions the graph. Its goal is to resolve genomic repeats by assembling regions that are locally non-repetitive. It applies heuristics to choose partitions that form a tiling path across the genome. It seeds partitions with nodes corresponding to long, moderately covered, widely separated contigs. It populates partitions with nodes and reads linked by alignments and mate pairs. It seeks to fill gaps between paired-end reads by searching the K-mer graph for instances where exactly one path satisfies the distance constraint. For tighter variance on constraints, it uses short-range paired-ends first. AllPaths assembles each partition separately and in parallel. Then, AllPaths “glues” the local graphs where they have overlapping structure. This is analogous to joining contigs based on sequence overlaps. In the global graph, AllPaths heuristically removes spurs, small disconnected components, and paths not spanned by paired-ends. It unrolls cycles to match paired-end distance constraints; this decides copy number on tandem repeats. It also uses paired-ends to tease apart collapsed repeats that display the frayed rope pattern.

In summary, AllPaths targets large-genome assembly using paired-end short reads from the Solexa platform. Its read filter uses quality values to fix some substitution errors. It simplifies its K-mer graph initially based on reads and overlaps. Going beyond read and

mate threading, it applies the read and paired-end data outside the graph in a divide-and-conquer approach.

The de Bruijn Graph in SOAPdenovo

The SOAPdenovo program [59] generated high-quality mammalian genome sequences from billions of Solexa reads of length 75 bp and less [60,61]. The program is freely available but without source code.

SOAP filters and corrects reads using pre-set thresholds for K-mer frequencies. It builds a de Bruijn graph and erodes the tips. SOAP threads the reads and splits paths that display a symmetrical frayed rope pattern. SOAP removes bubbles with an algorithm like Velvet's tour bus, with higher read coverage determining the surviving path. Though SOAP's DBG implementation borrows from Euler and Velvet, its graph is more space-efficient. Devoid of read-tracking information, the graph required only 120 GB RAM to store 5 billion nodes from 3.3 billion reads (after filtering). As described [59], the graph was constructed without data from large-insert libraries because those were suspected of generating many chimeric reads.

SOAP builds contigs from reads by the DBG method. Then it discards the de Bruijn graph to build scaffolds. It maps all paired reads to the contig consensus sequences, including reads not used in the DBG. Then it builds a contig graph whose edges represent the inter-contig mate pair constraints. SOAP reduces complexity in the contig graph by removing edges that are transitively inferable from others. It also isolates contigs traversed by multiple, incompatible paths, since these appear to be collapsed repeats. Similar techniques are implemented in CABOG and its predecessor, Celera Assembler. As does AllPaths, SOAP processes the edges in order of insert size, from small to large. SOAP does this to preclude construction of scaffolds that interleave others. SOAP uses mate pairs to assign reads to gaps between neighbor contigs within a scaffold. This is similar to CABOG's "rocks and stones" technique [37,62] and to Velvet's breadcrumbs and pebble techniques. SOAP uses de Bruijn graphs to assemble the reads assigned to each gap. In summary, SOAP is a large-genome implementation that amalgamates OLC and DBG techniques from its NGS assembly predecessors.

Other Software

The PCAP long-read assembler also assembles 454 reads [40]. LOCAS targets low-coverage short-read data (www-ab.informatik.uni-tuebingen.de/software/locas). The MIRA long-read assembler also assembles short reads (chevreux.org/projects_mira.html), as does Forge [63]. Similar to Edena, Taipan [64] implements greedy contig extension and transitive overlap reduction for unpaired short reads. Proprietary, commercial short-read assemblers include CLC Workbench (www.clcbio.com) and SeqMan (www.dnastart.com). Complete Genomics offers human genome sequencing services without distributing its sequencing or assembly technology (www.completegenomics.com). SHRAP is a protocol for sequencing human-scale genomes [65] with short reads by clone-based sequencing and a hierarchical assembly strategy. The protocol was demonstrated with Euler on simulated data. Network flow analysis has been described for resolution of repeat copy number during genome assembly [52,66]. It was implemented with generic flow analysis software [67].

The SOLiD platform from ABI has an unusual characteristic that requires special attention in software. The base calls are represented using the four digits 0, 1, 2, 3. These are referred to as colors. Each color is a di-base encoding of one base in relation to its preceding base. The reads' first base is a constant per run. Software gains robustness to error by forming alignments in color space and testing their validity by conversion to base space. Software support for *de novo* assembly of SOLiD reads is limited and has not been formally described. ABI provided an update to Velvet (www.solidsoftwaretools.com) that assembles homogeneous SOLiD data sets. There is *de novo* assembly support in commercial software such as CLC Workbench (www.clcbio.com). Some packages that assemble SOLiD reads in the presence of Sanger reads, 454 reads, or contigs include Shorty [48], SeqWrite (www.seqwright.com) and NextGENe (www.softgenetics.com).

An alternative to *de novo* assembly is mapping. For some applications, sufficient information can be extracted from the mapping of reads to a reference sequence, such as a finished genome from a related individual. Mapped reads can reveal small-scale population differences such as substitutions and indels. (These are routinely referred to as single-nucleotide substitutions, or SNPs, and deletion-insertion polymorphisms, or DIPs, respectively.) Mapped mate pairs can reveal larger indels and structural re-arrangements. Using a mapping, it is possible to construct contigs and scaffolds, each with their consensus sequences, as in *de novo* assembly.

Short-read mapping software is widely available. Published software includes SOAP [68,69], MAQ [70], Bowtie [71], RMAP [72], CloudBurst [73], SHRiMP [74], RazerS [75], PerM [76], segemehl [77], GenomeMapper [78], and BOAT [79]. Platform-specific mappers are available from platform vendors: Eland from Illumina (www.illumina.com), Corona from ABI [80], and Reference Mapper from 454 (www.454.com). Other commercial or unpublished solutions include ZOOM [81], CLC Workbench (www.clcbio.com), Novoalign (www.novocraft.com), Myrialign (savannah.nongnu.org/projects/myrialign) and Mr. Fast (mrfast.sf.net). Mapping-based variant discovery algorithms include MoDIL [82], VariationHunter [83] and BreakDancer [84]. Somewhere between *de novo* and mapping assembly there is reference-guided assembly. Software implementations include AMOScmp [85] as revised for short reads, MOSAIK as featured in a short-read, cross-species comparison [86], and SeqMan which is commercial software (www.dnastar.com). A related approach, called gene-boosted assembly, finishes an assembly by comparison of related genomes and protein sequences [87]. Our survey is no doubt incomplete and we apologize for omissions.

Summary and Outlook

We have compared a dozen algorithms for the *de novo* assembly of whole-genome shotgun (WGS) data from next-generation sequencing (NGS) platforms. Table 1 offers a summary feature comparison. Our comparison is an interpretation of the primary literature. Few of the published descriptions include measures of the relative contributions by their various algorithmic features, and none include the unit tests and controls that might verify that the implementations follow the algorithms. No algorithm or implementation solves the WGS assembly problem. Each of the various software packages was published with claims about its own superiority. Each package seems to improve in subsequent software releases. In our experience, the success of an assembler depends largely on the sophistication of its heuristics for real reads including error, real genomes including repeats, and the limitations of modern computers.

Large genomes tend to present more complex repeat structures. Most of the NGS assemblers discussed here were initially published with assemblies of bacterial-length genomes (or larger genomes by simulated reads). Four *de novo* assemblies of mammalian-scale genomes from Solexa short reads have been described in the scientific literature. The first assembly of the human genome from short reads [51] represented a software engineering feat. Later assemblies of panda [60] and two humans [61] used slightly larger reads and more variety in pair insert sizes and generated larger contigs. Improved mammalian assemblies are certain to become reality thanks to decreases in sequencing cost, increasing read lengths by every platform, more variety in paired-end protocols, continual improvement to assembly software, and the application of more powerful compute nodes and grids.

In the graph context, optimal path discovery is impractical because there can be an exponential number of paths between any source and

Table 1
Feature comparison between de novo assemblers for whole-genome shotgun data from next-generation sequencing platforms. OLC refers to the overlap/layout/consensus architecture. DBG refers to the de Bruijn graph architecture. The table is based on the literature cited in the text. It may not reflect the current state of each software package.

Algorithm Feature	Greedy Assemblers	OLC Assemblers	DBG Assemblers
Modeled features of reads			
Base substitutions			Euler, AllPaths, SOAP
Homopolymer miscount		CABOG	
Concentrated error in 3′ end			Euler
Flow space		Newbler	
Color space		Shorty	Velvet
Removal of erroneous reads			
Based on K-mer frequencies			Euler, Velvet, AllPaths
Based on K-mer freq and QV			AllPaths
For multiple values of K			AllPaths
By alignment to other reads		CABOG	
By alignment and QV	SHARCGS		
Correction of erroneous base calls			
Based on K-mer frequencies			Euler, SOAP
Based on Kmer freq and QV			AllPaths
Based on alignments		CABOG	
Approaches to graph construction			
Implicit	SSAKE, SHARCGS, VCAKE		
Reads as graph nodes		CABOG, Newbler, Edena	
K-mers as graph nodes			Euler, Velvet, ABySS, SOAP
Simple paths as graph nodes			AllPaths
Multiple values of K			Euler
Multiple overlap stringencies	SHARCGS		
Approaches to graph reduction			
Filter overlaps		CABOG	
Greedy contig extension	SSAKE, SHARCGS, VCAKE		
Collapse simple paths		CABOG, Newbler	Euler, Velvet, SOAP
Erosion of spurs		CABOG, Edena	Euler, Velvet, AllPaths, SOAP
Transitive overlap reduction		Edena	
Bubble smoothing		Edena	Euler, Velvet, SOAP
Bubble detection			AllPaths
Reads separate tangled paths			Euler, SOAP
Break at low coverage			Velvet, SOAP
Break at high coverage		CABOG	Euler
High coverage indicates repeat		CABOG	Velvet
Special use of long reads		Shorty	Velvet
Graph partitions			
Partition by K-mers			ABySS
Partition by scaffolds			AllPaths
Uses for mate pairs			
Constrain path searches			Euler, Velvet, AllPaths
Guide path selection			Euler, Allpaths
Detect misassembled contigs		CABOG, Shorty	
Merge contigs or fill gaps		CABOG, Shorty	Velvet, ABySS, SOAP
Transitive link reduction		CABOG	SOAP
Detect, avoid repeat contigs		CABOG	Velvet, SOAP
Create scaffolds		CABOG, Shorty	Euler, Velvet, AllPaths, SOAP

sink node. Each assembler searches the graph but each constrains its searches for reasons of scale. A search in CABOG's overlap graph follows only one outward edge per node. Searches in AllPaths and Velvet examine local regions of the K-mer graph that are anchored by large contigs and populated by direct and transitive paired-end linkage. Euler and ABySS seem to rely on branch-and-bound search algorithms.

Common to most assemblers, a core set of features is apparent:

- Error detection and correction based on sequence composition of the reads.
- Graph construction to represent reads and their shared sequence.
- Reduction of simple non-intersecting paths to single nodes in the graph.
- Removal of error-induced paths. These are recognized as spurs or bubbles.
- Collapse of polymorphism-induced complexity. This is recognized as bubbles.
- Simplification of tangles using information outside the graph. Individual reads or paired-end reads act as constraints on path distance and outcome.
- Conversion of reduced paths to contigs and scaffolds.
- Reduction of alignments to a consensus sequence.

OLC and DBG are two robust approaches to assembly. Both rely to some extent on a set of overlaps between the input reads. Both represent the overlaps in a directed graph. Their different graph representations are similar if not equivalent [66]. The OLC approach

directly incorporates connections (overlaps) of varying length, as expected in long-read, low-coverage data. K-mer graphs are limited initially to short connections (shared K-mers) of uniform size, though read threading mitigates this. The DBG approach is more appropriate for the large volumes of reads associated with short-read sequencing. DBG avoids the computationally expensive all-against-all pair-wise read comparisons. DBG avoids loading all the replicate sequences associated with high-coverage sequencing.

Both techniques must contend with noisy data. Sequencing error induces false positive and false negative overlaps. Noise filtration is inherent in overlap graphs constructed from non-identical alignments. K-mer graphs are more sensitive to sequencing error, as every miscalled base introduces up to K erroneous nodes. The OLC and DBG approaches both employ pre-processing steps to filter or correct unconfirmed portions of reads, as well as post-processes to repair graphs by erosion, smoothing, and threading.

The OLC assemblers grew up on long reads. The DBG assemblers proliferated with the introduction of short reads. The OLC assemblers target variable-length reads in the 100-800 bp range. The DBG assemblers target uniformly sized reads in the 25-100 bp range. Reads of the near future may be intermediate-sized, matching more closely the expectations of OLC assemblers. Further ahead, very long reads may become feasible while short reads may become even more affordable.

Reads of the future will challenge assembly software in many ways. Sequencing platforms may reveal inter-read associations richer than the paired-end model allows. Future platforms may target error rates that are higher or lower than today's standards. Almost certainly, data volume will continue to increase while manufacturing cost declines. The next-generation technology will surely be applied to larger genomes, more repetitive sequences, and less homogeneous samples. Developers of assembly algorithms will continue to be challenged by novel applications and larger data sets. Simultaneously, assembly developers will be challenged to extract more useful information from each sequence to enable lower coverage per genome and thus higher yield. The quest for more powerful and efficient assembly software remains an area of critical research.

Acknowledgments

The authors receive funding for assembly research from the National Institutes of Health via grant 2R01GM077117-04A1 from the National Institute of General Medical Sciences.

References

[1] F. Sanger, A.R. Coulson, B.G. Barrell, A.J. Smith, B.A. Roe, Cloning in single-stranded bacteriophage as an aid to rapid DNA sequencing, J. Mol. Biol. 143 (1980) 161–178.
[2] R. Staden, A strategy of DNA sequencing employing computer programs, Nucleic Acids Res. 6 (1979) 2601–2610.
[3] M. Pop, Genome assembly reborn: recent computational challenges, Brief. Bioinform. 10 (2009) 354–366.
[4] E.R. Mardis, The impact of next-generation sequencing technology on genetics, Trends Genet. 24 (2008) 133–141.
[5] O. Morozova, M.A. Marra, Applications of next-generation sequencing technologies in functional genomics, Genomics 92 (2008) 255–264.
[6] R.L. Strausberg, S. Levy, Y.H. Rogers, Emerging DNA sequencing technologies for human genomic medicine, Drug Discov. Today 13 (2008) 569–577.
[7] E. Pettersson, J. Lundeberg, A. Ahmadian, Generations of sequencing technologies, Genomics 93 (2009) 105–111.
[8] F. Sanger, S. Nicklen, A.R. Coulson, DNA sequencing with chain-terminating inhibitors, Proc. Natl. Acad. Sci. U. S. A. 74 (1977) 5463–5467.
[9] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, A. Bibillo, K. Bjornson, B. Chaudhuri, F. Christians, R. Cicero, S. Clark, R. Dalal, A. Dewinter, J. Dixon, M. Foquet, A. Gaertner, P. Hardenbol, C. Heiner, K. Hester, D. Holden, G. Kearns, X. Kong, R. Kuse, Y. Lacroix, S. Lin, P. Lundquist, C. Ma, P. Marks, M. Maxham, D. Murphy, I. Park, T. Pham, M. Phillips, J. Roy, R. Sebra, G. Shen, J. Sorenson, A. Tomaney, K. Travers, M. Trulson, J. Vieceli, J. Wegener, D. Wu, A. Yang, D. Zaccarin, P. Zhao, F. Zhong, J. Korlach, S. Turner, Real-time DNA sequencing from single polymerase molecules, Science 323 (2009) 133–138.
[10] B. Ewing, P. Green, Base-calling of automated sequencer traces using phred. II. Error probabilities, Genome Res. 8 (1998) 186–194.
[11] S.M. Huse, J.A. Huber, H.G. Morrison, M.L. Sogin, D.M. Welch, Accuracy and quality of massively parallel DNA pyrosequencing, Genome Biol. 8 (2007) R143.
[12] J.C. Dohm, C. Lottaz, T. Borodina, H. Himmelbauer, Substantial biases in ultra-short read data sets from high-throughput DNA sequencing, Nucleic Acids Res. 36 (2008) e105.
[13] O. Harismendy, P.C. Ng, R.L. Strausberg, X. Wang, T.B. Stockwell, K.Y. Beeson, N.J. Schork, S.S. Murray, E.J. Topol, S. Levy, K.A. Frazer, Evaluation of next generation sequencing platforms for population targeted sequencing studies, Genome Biol. 10 (2009) R32.
[14] R.D. Fleischmann, M.D. Adams, O. White, R.A. Clayton, E.F. Kirkness, A.R. Kerlavage, C.J. Bult, J.F. Tomb, B.A. Dougherty, J.M. Merrick, et al., Whole-genome random sequencing and assembly of Haemophilus influenzae Rd, Science 269 (1995) 496–512.
[15] M.D. Adams, S.E. Celniker, R.A. Holt, C.A. Evans, J.D. Gocayne, P.G. Amanatides, S.E. Scherer, P.W. Li, R.A. Hoskins, R.F. Galle, R.A. George, S.E. Lewis, S. Richards, M. Ashburner, S.N. Henderson, G.G. Sutton, J.R. Wortman, M.D. Yandell, Q. Zhang, L.X. Chen, R.C. Brandon, Y.H. Rogers, R.G. Blazej, M. Champe, B.D. Pfeiffer, K.H. Wan, C. Doyle, E.G. Baxter, G. Helt, C.R. Nelson, G.L. Gabor, J.F. Abril, A. Agbayani, H.J. An, C. Andrews-Pfannkoch, D. Baldwin, R.M. Ballew, A. Basu, J. Baxendale, L. Bayraktaroglu, E.M. Beasley, K.Y. Beeson, P.V. Benos, B.P. Berman, D. Bhandari, S. Bolshakov, D. Borkova, M.R. Botchan, J. Bouck, P. Brokstein, P. Brottier, K.C. Burtis, D.A. Busam, H. Butler, E. Cadieu, A. Center, I. Chandra, J.M. Cherry, S. Cawley, C. Dahlke, L.B. Davenport, P. Davies, B. de Pablos, A. Delcher, Z. Deng, A.D. Mays, I. Dew, S.M. Dietz, K. Dodson, L.E. Doup, M. Downes, S. Dugan-Rocha, B.C. Dunkov, P. Dunn, K.J. Durbin, C.C. Evangelista, C. Ferraz, S. Ferriera, W. Fleischmann, C. Fosler, A.E. Gabrielian, N.S. Garg, W.M. Gelbart, K. Glasser, A. Glodek, F. Gong, J.H. Gorrell, Z. Gu, P. Guan, M. Harris, N.L. Harris, D. Harvey, T.J. Heiman, J.R. Hernandez, J. Houck, D. Hostin, K.A. Houston, T.J. Howland, M.H. Wei, C. Ibegwam, et al., The genome sequence of Drosophila melanogaster, Science 287 (2000) 2185–2195.
[16] A.F. Siegel, G. van den Engh, L. Hood, B. Trask, J.C. Roach, Modeling the feasibility of whole genome shotgun sequencing using a pairwise end strategy, Genomics 68 (2000) 237–246.
[17] A.M. Phillippy, M.C. Schatz, M. Pop, Genome assembly forensics: finding the elusive mis-assembly, Genome Biol. 9 (2008) R55.
[18] J. Kececioglu, J. Ju, Separating repeats in DNA sequence assembly, Annual Conference on Research in Computational Molecular Biology, 2001, pp. 176–183.
[19] N. Whiteford, N. Haslam, G. Weber, A. Prugel-Bennett, J.W. Essex, P.L. Roach, M. Bradley, C. Neylon, An analysis of the feasibility of short read sequencing, Nucleic Acids Res. 33 (2005) e171.
[20] D.B. Rusch, A.L. Halpern, G. Sutton, K.B. Heidelberg, S. Williamson, S. Yooseph, D. Wu, J.A. Eisen, J.M. Hoffman, K. Remington, K. Beeson, B. Tran, H. Smith, H. Baden-Tillson, C. Stewart, J. Thorpe, J. Freeman, C. Andrews-Pfannkoch, J.E. Venter, K. Li, S. Kravitz, J.F. Heidelberg, T. Utterback, Y.H. Rogers, L.I. Falcon, V. Souza, G. Bonilla-Rosso, L.E. Eguiarte, D.M. Karl, S. Sathyendranath, T. Platt, E. Bermingham, V. Gallardo, G. Tamayo-Castillo, M.R. Ferrari, R.L. Strausberg, K. Nealson, R. Friedman, M. Frazier, J.C. Venter, The Sorcerer II Global Ocean Sampling expedition: northwest Atlantic through eastern tropical Pacific, PLoS Biol. 5 (2007) e77.
[21] K. Mavromatis, N. Ivanova, K. Barry, H. Shapiro, E. Goltsman, A.C. McHardy, I. Rigoutsos, A. Salamov, F. Korzeniewski, M. Land, A. Lapidus, I. Grigoriev, P. Richardson, P. Hugenholtz, N.C. Kyrpides, Use of simulated data sets to evaluate the fidelity of metagenomic processing methods, Nat. Methods 4 (2007) 495–500.
[22] K.E. Wommack, J. Bhavsar, J. Ravel, Metagenomics: read length matters, Appl. Environ. Microbiol. 74 (2008) 1453–1463.
[23] E.W. Myers, Toward simplifying and accurately formulating fragment assembly, J. Comput. Biol. 2 (1995) 275–290.
[24] R.M. Idury, M.S. Waterman, A new algorithm for DNA sequence assembly, J. Comput. Biol. 2 (1995) 291–306.
[25] D.R. Zerbino, E. Birney, Velvet: algorithms for de novo short read assembly using de Bruijn graphs, Genome Res. 18 (2008) 821–829.
[26] P.A. Pevzner, H. Tang, G. Tesler, De novo repeat classification and fragment assembly, Genome Res. 14 (2004) 1786–1796.
[27] D. Zhi, B.J. Raphael, A.L. Price, H. Tang, P.A. Pevzner, Identifying repeat domains in large genomes, Genome Biol. 7 (2006) R7.
[28] D. Fasulo, A. Halpern, I. Dew, C. Mobarry, Efficiently detecting polymorphisms during the fragment assembly process, Bioinformatics 18 (Suppl 1) (2002) S294–S302.
[29] N. Nagarajan, M. Pop, Parametric complexity of sequence assembly: theory and applications to next generation sequencing, J. Comput. Biol. 16 (2009) 897–908.
[30] M. Pop, S.L. Salzberg, Bioinformatics challenges of new sequencing technology, Trends Genet. 24 (2008) 142–149.
[31] R.L. Warren, G.G. Sutton, S.J. Jones, R.A. Holt, Assembling millions of short DNA sequences using SSAKE, Bioinformatics 23 (2007) 500–501.
[32] R.L. Warren, R.A. Holt, SSAKE 3.0: Improved speed, accuracy and contiguity, Pacific Symposium on Biocomputing, 2008.
[33] J.C. Dohm, C. Lottaz, T. Borodina, H. Himmelbauer, SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing, Genome Res. 17 (2007) 1697–1706.
[34] W.R. Jeck, J.A. Reinhardt, D.A. Baltrus, M.T. Hickenbotham, V. Magrini, E.R. Mardis, J.L. Dangl, C.D. Jones, Extending assembly of short DNA sequences to handle error, Bioinformatics 23 (2007) 2942–2944.
[35] J.A. Reinhardt, D.A. Baltrus, M.T. Nishimura, W.R. Jeck, C.D. Jones, J.L. Dangl, De novo assembly using low-coverage short read sequence data from the rice pathogen Pseudomonas syringae pv. oryzae, Genome Res. 19 (2009) 294–305.
[36] S.M. Goldberg, J. Johnson, D. Busam, T. Feldblyum, S. Ferriera, R. Friedman, A. Halpern, H. Khouri, S.A. Kravitz, F.M. Lauro, K. Li, Y.H. Rogers, R. Strausberg, G. Sutton, L. Tallon, T. Thomas, E. Venter, M. Frazier, J.C. Venter, A Sanger/pyrosequencing hybrid approach for the generation of high-quality draft assemblies of marine microbial genomes, Proc. Natl. Acad. Sci. U. S. A. 103 (2006) 11240–11245.

[37] E.W. Myers, G.G. Sutton, A.L. Delcher, I.M. Dew, D.P. Fasulo, M.J. Flanigan, S.A. Kravitz, C.M. Mobarry, K.H. Reinert, K.A. Remington, E.L. Anson, R.A. Bolanos, H.H. Chou, C.M. Jordan, A.L. Halpern, S. Lonardi, E.M. Beasley, R.C. Brandon, L. Chen, P.J. Dunn, Z. Lai, Y. Liang, D.R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G.M. Rubin, M.D. Adams, J.C. Venter, A whole-genome assembly of Drosophila, Science 287 (2000) 2196–2204.
[38] S. Batzoglou, D.B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J.P. Mesirov, E.S. Lander, ARACHNE: a whole-genome shotgun assembler, Genome Res. 12 (2002) 177–189.
[39] D.B. Jaffe, J. Butler, S. Gnerre, E. Mauceli, K. Lindblad-Toh, J.P. Mesirov, M.C. Zody, E.S. Lander, Whole-genome sequence assembly for mammalian genomes: Arachne 2, Genome Res. 13 (2003) 91–96.
[40] X. Huang, S.P. Yang, Generating a genome assembly with PCAP. Curr Protoc Bioinformatics Chapter 11 (2005) Unit11 3.
[41] S. Batzoglou, Algorithmic Challenges in Mammalian Genome Sequence Assembly, in: M. Dunn, L. Jorde, P. Little, S. Subramaniam (Eds.), Encyclopedia of genomics, proteomics and bioinformatics, John Wiley and Sons, Hoboken (New Jersey), 2005.
[42] M. Pop, DNA sequence assembly algorithms, in: McGraw-Hill (Ed.), McGraw-Hill 2006 Yearbook of Science and Technology, McGraw-Hill, New York, 2005.
[43] G. Sutton, I. Dew, Shotgun Fragment Assembly, in: I. Rigoutsos, G. Stephanopoulos (Eds.), Systems Biology: Genomics, Oxford University Press, New York, 2007, pp. 79–117.
[44] L. Wang, T. Jiang, On the complexity of multiple sequence alignment, J. Comput. Biol. 1 (1994) 337–348.
[45] M. Margulies, M. Egholm, W.E. Altman, S. Attiya, J.S. Bader, L.A. Bemben, J. Berka, M.S. Braverman, Y.J. Chen, Z. Chen, S.B. Dewell, L. Du, J.M. Fierro, X.V. Gomes, B.C. Godwin, W. He, S. Helgesen, C.H. Ho, G.P. Irzyk, S.C. Jando, M.L. Alenquer, T.P. Jarvie, K.B. Jirage, J.B. Kim, J.R. Knight, J.R. Lanza, J.H. Leamon, S.M. Lefkowitz, M. Lei, J. Li, K.L. Lohman, H. Lu, V.B. Makhijani, K.E. McDade, M.P. McKenna, E.W. Myers, E. Nickerson, J.R. Nobile, R. Plant, B.P. Puc, M.T. Ronan, G.T. Roth, G.J. Sarkis, J.F. Simons, J.W. Simpson, M. Srinivasan, K.R. Tartaro, A. Tomasz, K.A. Vogt, G.A. Volkmer, S.H. Wang, Y. Wang, M.P. Weiner, P. Yu, R.F. Begley, J.M. Rothberg, Genome sequencing in microfabricated high-density picolitre reactors, Nature 437 (2005) 376–380.
[46] J.R. Miller, A.L. Delcher, S. Koren, E. Venter, B.P. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, G. Sutton, Aggressive assembly of pyrosequencing reads with mates, Bioinformatics 24 (2008) 2818–2824.
[47] D. Hernandez, P. Francois, L. Farinelli, M. Osteras, J. Schrenzel, De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer, Genome Res. 18 (2008) 802–809.
[48] M.S. Hossain, N. Azimi, S. Skiena, Crystallizing short-read assemblies around seeds, BMC Bioinformatics 10 (Suppl 1) (2009) S16.
[49] P.A. Pevzner, 1-Tuple DNA sequencing: computer analysis, J. Biomol. Struct. Dyn. 7 (1989) 63–73.
[50] P.A. Pevzner, H. Tang, M.S. Waterman, An Eulerian path approach to DNA fragment assembly, Proc. Natl. Acad. Sci. U. S. A. 98 (2001) 9748–9753.
[51] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J. Jones, I. Birol, ABySS: A parallel assembler for short read sequence data, Genome Res. 19 (2009) 1117–1123.
[52] P.A. Pevzner, H. Tang, Fragment assembly with double-barreled data, Bioinformatics 17 (Suppl 1) (2001) S225–S233.
[53] M. Chaisson, P. Pevzner, H. Tang, Fragment assembly with short reads, Bioinformatics 20 (2004) 2067–2074.
[54] M.J. Chaisson, P.A. Pevzner, Short read fragment assembly of bacterial genomes, Genome Res. 18 (2008) 324–330.
[55] M.J. Chaisson, D. Brinza, P.A. Pevzner, De novo fragment assembly with short mate-paired reads: Does the read length matter? Genome Res. 19 (2009) 336–346.
[56] D.R. Zerbino, G.K. McEwen, E.H. Margulies, E. Birney, Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler, PLoS One 4 (2009) e8407.
[57] J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, D.B. Jaffe, ALLPATHS: de novo assembly of whole-genome shotgun microreads, Genome Res. 18 (2008) 810–820.
[58] I. Maccallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T.P. Shea, L. Williams, S. Young, C. Nusbaum, D.B. Jaffe, ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads, Genome Biol. 10 (2009) R103.
[59] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, H. Yang, J. Wang, De novo assembly of human genomes with massively parallel short read sequencing, Genome Res. 20 (2009) 265–272.
[60] R. Li, W. Fan, G. Tian, H. Zhu, L. He, J. Cai, Q. Huang, Q. Cai, B. Li, Y. Bai, Z. Zhang, Y. Zhang, W. Wang, J. Li, F. Wei, H. Li, M. Jian, R. Nielsen, D. Li, W. Gu, Z. Yang, Z. Xuan, O.A. Ryder, F.C. Leung, Y. Zhou, J. Cao, X. Sun, Y. Fu, X. Fang, X. Guo, B. Wang, R. Hou, F. Shen, B. Mu, P. Ni, R. Lin, W. Qian, G. Wang, C. Yu, W. Nie, J. Wang, Z. Wu, H. Liang, J. Min, Q. Wu, S. Cheng, J. Ruan, M. Wang, Z. Shi, M. Wen, B. Liu, X. Ren, H. Zheng, D. Dong, K. Cook, G. Shan, H. Zhang, C. Kosiol, X. Xie, Z. Lu, Y. Li, C.C. Steiner, T.T. Lam, S. Lin, Q. Zhang, G. Li, J. Tian, T. Gong, H. Liu, D. Zhang, L. Fang, C. Ye, J. Zhang, W. Hu, A. Xu, Y. Ren, G. Zhang, M.W. Bruford, Q. Li, L. Ma, Y. Guo, N. An, Y. Hu, Y. Zheng, Y. Shi, Z. Li, Q. Liu, Y. Chen, J. Zhao, N. Qu, S. Zhao, F. Tian, X. Wang, H. Wang, L. Xu, X. Liu, T. Vinar, Y. Wang, T.W. Lam, S.M. Yiu, et al., The sequence and de novo assembly of the giant panda genome, Nature 463 (2009) 311–317.
[61] R. Li, Y. Li, H. Zheng, R. Luo, H. Zhu, Q. Li, W. Qian, Y. Ren, G. Tian, J. Li, G. Zhou, X. Zhu, H. Wu, J. Qin, X. Jin, D. Li, H. Cao, X. Hu, H. Blanche, H. Cann, X. Zhang, S. Li, L. Bolund, K. Kristiansen, H. Yang, J. Wang, Building the sequence map of the human pan-genome, Nat. Biotechnol. 28 (2009) 57–63.
[62] J.C. Venter, M.D. Adams, E.W. Myers, P.W. Li, R.J. Mural, G.G. Sutton, H.O. Smith, M. Yandell, C.A. Evans, R.A. Holt, J.D. Gocayne, P. Amanatides, R.M. Ballew, D.H. Huson, J.R. Wortman, Q. Zhang, C.D. Kodira, X.H. Zheng, L. Chen, M. Skupski, G. Subramanian, P.D. Thomas, J. Zhang, G.L. Gabor Miklos, C. Nelson, S. Broder, A.G. Clark, J. Nadeau, V.A. McKusick, N. Zinder, A.J. Levine, R.J. Roberts, M. Simon, C. Slayman, M. Hunkapiller, R. Bolanos, A. Delcher, I. Dew, D. Fasulo, M. Flanigan, L. Florea, A. Halpern, S. Hannenhalli, S. Kravitz, S. Levy, C. Mobarry, K. Reinert, K. Remington, J. Abu-Threideh, E. Beasley, K. Biddick, V. Bonazzi, R. Brandon, M. Cargill, I. Chandramouliswaran, R. Charlab, K. Chaturvedi, Z. Deng, V. Di Francesco, P. Dunn, K. Eilbeck, C. Evangelista, A.E. Gabrielian, W. Gan, W. Ge, F. Gong, Z. Gu, P. Guan, T.J. Heiman, M.E. Higgins, R.R. Ji, Z. Ke, K.A. Ketchum, Z. Lai, Y. Lei, Z. Li, J. Li, Y. Liang, X. Lin, F. Lu, G.V. Merkulov, N. Milshina, H.M. Moore, A.K. Naik, V.A. Narayan, B. Neelam, D. Nusskern, D.B. Rusch, S. Salzberg, W. Shao, B. Shue, J. Sun, Z. Wang, A. Wang, X. Wang, J. Wang, M. Wei, R. Wides, C. Xiao, C. Yan, et al., The sequence of the human genome, Science 291 (2001) 1304–1351.
[63] S. Diguistini, N.Y. Liao, D. Platt, G. Robertson, M. Seidel, S.K. Chan, T.R. Docking, I. Birol, R.A. Holt, M. Hirst, E. Mardis, M.A. Marra, R.C. Hamelin, J. Bohlmann, C. Breuil, S.J. Jones, De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data, Genome Biol. 10 (2009) R94.
[64] B. Schmidt, R. Sinha, B. Beresford-Smith, S.J. Puglisi, A fast hybrid short read fragment assembly algorithm, Bioinformatics 25 (2009) 2279–2280.
[65] A. Sundquist, M. Ronaghi, H. Tang, P. Pevzner, S. Batzoglou, Whole-genome sequencing and assembly with high-throughput, short-read technologies, PLoS ONE 2 (2007) e484.
[66] E.W. Myers, The fragment assembly string graph, Bioinformatics 21 (Suppl 2) (2005) ii79–ii85.
[67] P. Medvedev, M. Brudno, Ab initio Whole Genome Shotgun Assembly with Mated Short Reads Proceedings of the 12th Annual Research in Computational Biology Conference (RECOMB), 2008.
[68] R. Li, Y. Li, K. Kristiansen, J. Wang, SOAP: short oligonucleotide alignment program, Bioinformatics 24 (2008) 713–714.
[69] R. Li, C. Yu, Y. Li, T.W. Lam, S.M. Yiu, K. Kristiansen, J. Wang, SOAP2: an improved ultrafast tool for short read alignment, Bioinformatics 25 (2009) 1966–1967.
[70] H. Li, J. Ruan, R. Durbin, Mapping short DNA sequencing reads and calling variants using mapping quality scores, Genome Res. 18 (2008) 1851–1858.
[71] B. Langmead, C. Trapnell, M. Pop, S.L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biol. 10 (2009) R25.
[72] A.D. Smith, Z. Xuan, M.Q. Zhang, Using quality scores and longer reads improves accuracy of Solexa read mapping, BMC Bioinformatics 9 (2008) 128.
[73] M.C. Schatz, CloudBurst: Highly Sensitive Read Mapping with MapReduce, Bioinformatics 25 (2009) 1363–1369.
[74] S.M. Rumble, P. Lacroute, A.V. Dalca, M. Fiume, A. Sidow, M. Brudno, SHRiMP: accurate mapping of short color-space reads, PLoS Comput. Biol. 5 (2009) e1000386.
[75] D. Weese, A.K. Emde, T. Rausch, A. Doring, K. Reinert, RazerS–fast read mapping with sensitivity control, Genome Res. 19 (2009) 1646–1654.
[76] Y. Chen, T. Souaiaia, T. Chen, PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds, Bioinformatics 25 (2009) 2514–2521.
[77] S. Hoffmann, C. Otto, S. Kurtz, C.M. Sharma, P. Khaitovich, J. Vogel, P.F. Stadler, J. Hackermuller, Fast mapping of short sequences with mismatches, insertions and deletions using index structures, PLoS Comput. Biol. 5 (2009) e1000502.
[78] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, D. Weigel, Simultaneous alignment of short reads against multiple genomes, Genome Biol. 10 (2009) R98.
[79] S.Q. Zhao, J. Wang, L. Zhang, J.T. Li, X. Gu, G. Gao, L. Wei, BOAT: Basic Oligonucleotide Alignment Tool, BMC Genomics 10 (Suppl 3) (2009) S2.
[80] K.J. McKernan, H.E. Peckham, G.L. Costa, S.F. McLaughlin, Y. Fu, E.F. Tsung, C.R. Clouser, C. Duncan, J.K. Ichikawa, C.C. Lee, Z. Zhang, S.S. Ranade, E.T. Dimalanta, F.C. Hyland, T.D. Sokolsky, L. Zhang, A. Sheridan, H. Fu, C.L. Hendrickson, B. Li, L. Kotler, J.R. Stuart, J.A. Malek, J.M. Manning, A.A. Antipova, D.S. Perez, M.P. Moore, K.C. Hayashibara, M.R. Lyons, R.E. Beaudoin, B.E. Coleman, M.W. Laptewicz, A.E. Sannicandro, M.D. Rhodes, R.K. Gottimukkala, S. Yang, V. Bafna, A. Bashir, A. MacBride, C. Alkan, J.M. Kidd, E.E. Eichler, M.G. Reese, F.M. De La Vega, A.P. Blanchard, Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding, Genome Res. 19 (2009) 1527–1541.
[81] H. Lin, Z. Zhang, M.Q. Zhang, B. Ma, M. Li, ZOOM! Zillions of oligos mapped, Bioinformatics 24 (2008) 2431–2437.
[82] S. Lee, F. Hormozdiari, C. Alkan, M. Brudno, MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions, Nat. Methods 6 (2009) 473–474.
[83] F. Hormozdiari, C. Alkan, E.E. Eichler, S.C. Sahinalp, Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes, Genome Res. 19 (2009) 1270–1278.
[84] K. Chen, J.W. Wallis, M.D. McLellan, D.E. Larson, J.M. Kalicki, C.S. Pohl, S.D. McGrath, M.C. Wendl, Q. Zhang, D.P. Locke, X. Shi, R.S. Fulton, T.J. Ley, R.K. Wilson, L. Ding, E.R. Mardis, BreakDancer: an algorithm for high-resolution mapping of genomic structural variation, Nat. Methods 6 (2009) 677–681.
[85] M. Pop, A. Phillippy, A.L. Delcher, S.L. Salzberg, Comparative genome assembly, Brief. Bioinform. 5 (2004) 237–248.
[86] L.W. Hillier, G.T. Marth, A.R. Quinlan, D. Dooling, G. Fewell, D. Barnett, P. Fox, J.I. Glasscock, M. Hickenbotham, W. Huang, V.J. Magrini, R.J. Richt, S.N. Sander, D.A. Stewart, M. Stromberg, E.F. Tsung, T. Wylie, T. Schedl, R.K. Wilson, E.R. Mardis, Whole-genome sequencing and variant discovery in C. elegans, Nat. Methods 5 (2008) 183–188.
[87] S.L. Salzberg, D.D. Sommer, M.C. Schatz, A.M. Phillippy, P.D. Rabinowicz, S. Tsuge, A. Furutani, H. Ochiai, A.L. Delcher, D. Kelley, R. Madupu, D. Puiu, D. Radune, M. Shumway, C. Trapnell, G. Aparna, G. Jha, A. Pandey, P.B. Patil, H. Ishihara, D.F. Meyer, B. Szurek, V. Verdier, R. Koebnik, J.M. Dow, R.P. Ryan, H. Hirata, S. Tsuyumu, S. Won Lee, P.C. Ronald, R.V. Sonti, M.A. Van Sluys, J.E. Leach, F.F. White, A.J. Bogdanove, Genome sequence and rapid evolution of the rice pathogen Xanthomonas oryzae pv. oryzae PXO99A, BMC Genomics 9 (2008) 204.

Jason Rafe Miller manages software research and development on whole-genomeshotgun assembly at the J. Craig Venter Institute (JCVI). He previously contributed to assembly infrastructure development at TIGR, genome comparison and annotation software at Celera Genomics, and genomics visualization software at GlaxoSmithKline. Mr. Miller received a Master's degree from University of Pennsylvania and a Bachelor's degree from New York University.

Sergey Koren a software engineer at the J. Craig Venter Institute, is a Ph.D. student at the University of Maryland, College Park, where he received his Master of Science andBachelor of Science (cum laude, with honors) degrees in Computer Science. His research interests include genome assembly, application of graph analysis to metagenomics, and applications of high-performance computing. He is a contributor to the Celera Assembler, AMOS, and k-mer Tools projects hosted on Source Forge.

Granger Sutton is Senior Director of Informatics at the J. Craig Venter Institute (JCVI). Prior to joining JCVI, Dr. Sutton was a director in the Informatics Research department at Celera Genomics where he developed and managed research programs ingene finding, comparative genomics, and shotgun fragment assembly including the development of the Celera Assembler for assembling the human genome. As Computer Scientist at The Institute for Genomic Research (TIGR), he developed protein homology search, multiple sequence alignment, and shotgun fragment assembly algorithms. Dr.Sutton also worked at AT&T Bell Labs to design and implement office automation software. Dr. Sutton earned his Bachelor's degree in electrical engineering and Doctoratein Computer Science from University of Maryland, College Park, and a Master's degreein Computer Engineering from Stanford University.