The Theory and Practice of Genome Sequence Assembly

Jared T. Simpson[1] and Mihai Pop[2]

[1]Ontario Institute for Cancer Research, Toronto, Ontario M5G ON3, Canada; email: jared.simpson@oicr.on.ca

[2]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, Maryland 20742; email: mpop@umiacs.umd.edu

Annu. Rev. Genomics Hum. Genet. 2015. 16:2.1–2.20

The *Annual Review of Genomics and Human Genetics* is online at genom.annualreviews.org

This article's doi: 10.1146/annurev-genom-090314-050032

Keywords

genome sequencing, sequence assembly, algorithm, bioinformatics, shotgun sequencing

Abstract

The current genomic revolution was made possible by joint advances in genome sequencing technologies and computational approaches for analyzing sequence data. The close interaction between biologists and computational scientists is perhaps most apparent in the development of approaches for sequencing entire genomes, a feat that would not be possible without sophisticated computational tools called genome assemblers (short for genome sequence assemblers). Here, we survey the key developments in algorithms for assembling genome sequences since the development of the first DNA sequencing methods more than 35 years ago.

INTRODUCTION AND BRIEF HISTORY

Base pair: a pair of complementary nucleotides that make up the atomic units of a DNA sequence

Sequence read: a single segment of contiguous genomic sequence that has been experimentally determined; this is the basic input into a genome sequence assembler

The development of methods for elucidating the DNA sequence within the genomes of organisms has revolutionized biomedical research and has led to the current genomic revolution. The genome sequences of tens of thousands of bacteria and viruses, thousands of individual humans, and hundreds to thousands of other organisms have been reconstructed to date. Because genomic technologies and services are increasingly becoming a commodity, we often overlook the close integration of biology, chemistry, engineering, and computer science that have made these advances possible. In this review, we focus on the algorithms and software tools that reconstruct genome sequences from the many small segments of DNA that can be read by modern DNA sequencing machines.

Despite tremendous advances in DNA sequencing technologies, modern instruments can read only small segments of the genomes of most organisms, ranging from approximately 100 base pairs (bp) (e.g., Illumina technology) to approximately 10–20 kb (e.g., Pacific Biosciences technology). By contrast, the human genome comprises more than 3 Gb, and even bacteria have genomes spanning more than 1 Mb. Reconstructing an entire organism's genome sequence thus requires gluing together many small sequence fragments.

The early attempts at reconstructing genome sequences relied on molecular techniques to iteratively sequence overlapping genomic segments (see **Figure 1**). Because the order and position of the corresponding sequence reads were defined by the experimental setup, gluing together the individual pieces into a complete genome sequence was easily done, even without the help of computers (97). This simple approach, however, was time consuming and labor intensive; the next DNA segment could not be sequenced until the previous one had been completed. An alternative approach, called shotgun sequencing, was proposed by Staden in 1979 (104). This approach was

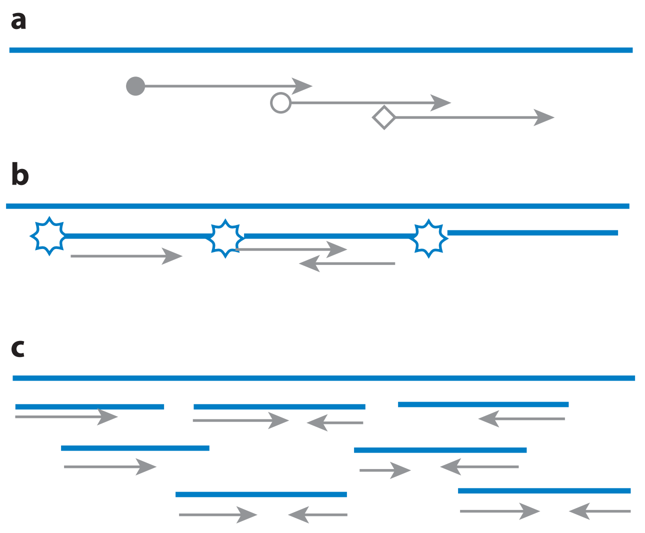

Figure 1

Different strategies for sequencing DNA. The thick blue lines represent the (unknown) DNA being sequenced, and the thin gray arrows represent the sequence reads and their respective orientations (from which strand of DNA they originate). (*a*) Primer walking. The sequencing is primed at specific locations, using specific known sequence primers (*symbols*). Knowing the sequence at the end of one read allows the design of the next sequence primer, thereby expanding the total known sequence. (*b*) Restriction digestion. Sequencing starts at the ends of the fragments obtained by digesting with one or more restriction enzymes (restriction sites indicated by *stars*). (*c*) Shotgun sequencing. Sequencing starts from the ends of fragments obtained by randomly shearing the original DNA.

inherently scalable because the original DNA was fragmented at random into many pieces that were then sequenced in parallel. Because of the randomness of the fragmentation process (usually achieved through mechanical shearing of the DNA), the individual fragments could be expected to overlap one another, and these overlaps could be recognized by comparing the DNA sequences of the corresponding fragments. It quickly became clear that reconstructing genome sequences would require computer programs, leading to the birth of the first genome sequence assemblers.

The simplicity of the shotgun sequencing process captured the interest of mathematicians and computer scientists and led to the rapid development of the theoretical foundations for genome sequence assembly. Scientists were initially interested in determining whether and under which conditions assembly is possible. Lander & Waterman (52) derived the equations that provide an upper bound on the key characteristics of the data that can be assembled. Specifically, they estimated the size and number of contiguous genomic segments that can be reconstructed from a set of sequence reads as a function of just three parameters: the length of the reads, the coverage of the genome (number of total bases in the set of reads divided by the length of the genome), and the minimum overlap between two reads that can be effectively detected by an assembler. Their work showed that genome sequences could be effectively reconstructed with as low as tenfold coverage of sequence reads. Ukkonen and others studied the computational complexity of the assembly problem, formalized as an instance of the shortest common superstring problem (61), the problem of finding the shortest string that encompasses all the reads as substrings. This Occam's razor formulation assumes that the genome being reconstructed is the most parsimonious explanation of the set of reads. Their work showed that genome sequence assembly is computationally intractable—finding the correct solution may require exploring an exponential number of possible solutions (92). Despite this negative result, attempts to find approximate solutions (e.g., a superstring that is provably close to the shortest one) led to the first practical genome sequence assemblers (107).

The shortest common superstring problem ignores an important feature of complex (e.g., vertebrate) genomes: repeats. Most genomes contain DNA segments that are repeated in nearly identical form, thus straying from parsimony. In other words, the correct reconstruction of a genome sequence may not be the shortest. The need to explicitly account for repeats has led to new, graph-based models of sequence assembly that form the basis of virtually all modern genome sequence assemblers. Note, however, that even with such formulations, genome assembly can be shown to be computationally intractable (45, 63). The issue of repeats, and the complexity they introduce in the assembly process, was key to a vigorous debate over the feasibility of assembling the entire human genome from shotgun sequence data (35, 71).

As we end this brief historical introduction, we would like to highlight an apparent disconnect between theoretical results, which show genome sequence assembly to be computationally intractable, and the obvious practical success in sequence assembly witnessed over the past few decades. An explanation for this all-too-familiar gap between theory and practice is that theoretical intractability results are based on worst-case scenarios, which rarely occur in practice. The practical complexity of sequence assembly depends on the ratio between the size of the sequence reads and that of the repeats (73); when reads are longer than repeats, the assembly problem can be solved, whereas when reads are shorter than repeats, the assembly problem is ill defined because one can reconstruct an exponential number of genomes that are compatible with the input set of reads (47). We revisit this trade-off throughout our review.

ASSEMBLY PARADIGMS

As described above, efforts to theoretically formalize and understand genome sequence assembly have led to increasingly complex explanations of the problem. These approaches fall into several broad paradigms, outlined in more detail below.

Greedy Approaches

Contig: a contiguous sequence of bases generated by an assembler

Misassembly: an incorrectly assembled contig that is not consistent with the structure of the genome

One of the simplest strategies for assembling a genome sequence involves iteratively joining together the reads in decreasing order of the quality of their overlaps. In other words, the process starts by joining the two reads that overlap the best (in terms of either the length of the overlap or a more complex quality measure that accounts for base quality estimates), then repeats this process until a predefined minimum quality threshold is reached. As a result, nascent assembled sequences (i.e., contigs) grow through either the addition of new reads or a joining with previously constructed contigs. Read overlaps that conflict with already constructed contigs are ignored. This strategy is called greedy because it makes the most greedy (locally optimal) choice at each step. Despite its simplicity, this approach and its variants provide a good approximation for the optimal assembly.

Many of the early genome sequence assemblers relied on such a greedy strategy. Among these, we would like to highlight phrap (34), which was the main genome sequence assembler used by the public effort to assemble the human genome sequence during the Human Genome Project; the TIGR Assembler (106), which was used to reconstruct the genome sequence of the first living organism to have its genome sequenced; and the CAP series of tools (37), which have been heavily used in reconstructing the transcriptomes of humans and many other organisms. Greedy approaches were also explored as a possible way to cope with the data generated by the initial set of second-generation DNA sequencing technologies (44, 111).

Despite its tremendous early successes, the greedy strategy has a severe limitation: Because of its local nature, it cannot effectively handle repeated genomic regions, as can be seen in **Figure 2**. As the size and complexity of the genomes being sequenced increased, greedy approaches were replaced by more complex graph-based algorithms that are better able to model and resolve highly repetitive genomic sequences.

Graph-Based Approaches

Theoretical studies on combinatorial solutions to the sequence assembly problem led to the development of practical software using graph-based representations of the sequence data (45, 68). Graph-based sequence assembly models represent sequence reads and their inferred relationships to one another as vertices and edges in the graph. Walks through the graph describe an ordering of the reads that can be assembled together. The sequence assembler tries to find a walk that best reconstructs the underlying genome while avoiding generating misassemblies by taking erroneous paths caused by repeats.

OLC graphs. In the simplest graph-based model, each sequence read is a vertex in the graph, and a pair of vertices are linked with an edge if they overlap (see **Figure 2**). Alternative formulations have a pair of vertices for each read, one representing the start of the read and one representing the end of the read, with an edge linking these vertices that carries the read's sequence (69). In this formulation, overlaps are represented by edges from a terminal vertex of one read to a terminal vertex of another read. Regardless of the representation of the graph, the assembly process typically follows three main stages. First, overlapping pairs of reads are detected. Second, the graph is constructed, and an appropriate ordering and orientation (layout) of the reads are found. Finally, a consensus sequence is computed from the ordered and oriented reads. The set of consensus sequences is then output by the assembler as the sequence contigs. Genome assemblers following this paradigm are called OLC assemblers for the three main stages of the assembly: overlap, layout, and consensus.

The overlap step typically requires significant compute time. Naively, one could simply compare all pairs of reads using dynamic programming to check whether each pair has a significant

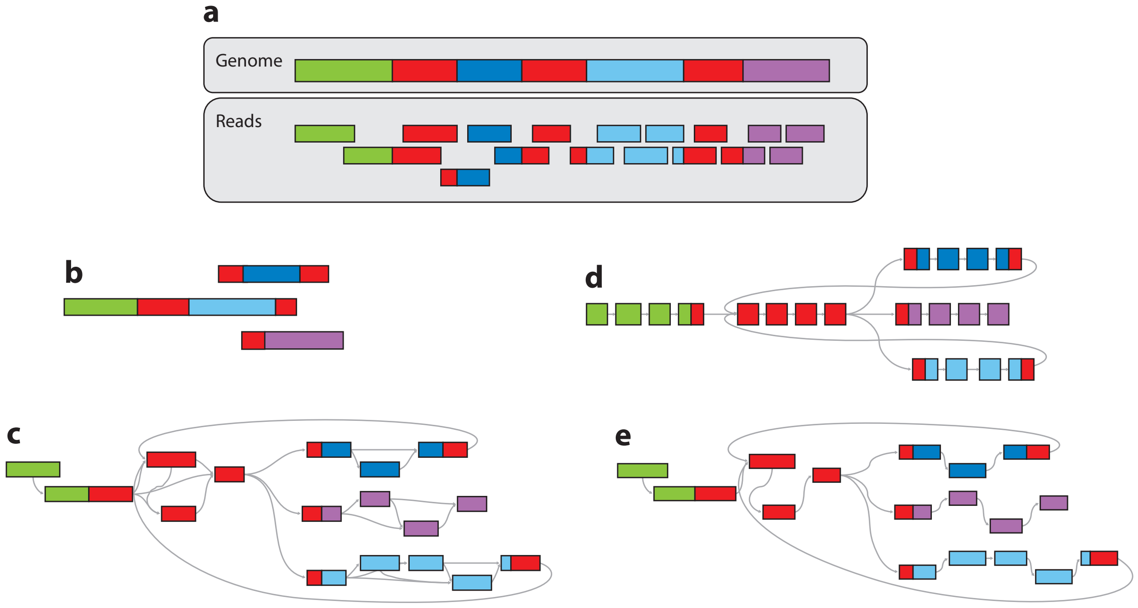

Figure 2

Genome assembly paradigms. (*a*) The layout of a genome containing several repeats (*segments with the same color*), along with a set of shotgun sequence reads. (*b*) The greedy approach, which may misassemble the genome by collapsing the repeats (*red blocks*). (*c*) The OLC (overlap, layout, consensus) paradigm, which represents each read as a node and connects the nodes if they overlap. (*d*) The de Bruijn graph, which models the relationship between *k*-mers. The repeated region is apparent in the graph structure. (*e*) The string graph, which simplifies the graph by removing transitive edges. The location of repeats is more apparent in the string graph than in the OLC graph.

overlap (typically determined by the length of the overlap and the similarity within the overlapping region). Such an algorithm requires O(N^2) time, where N is the total number of sequenced bases. This brute-force approach can be used to assemble the sequences of only very small genomes. To accelerate overlap detection, an index can be constructed that maps *k*-mers to the list of reads containing the *k*-mer (typically *k* is in the range 16–24 bases for this application). The index is used to quickly screen for reads that may overlap, and dynamic programming is then performed to verify the overlap. This technique drastically reduces the search space and has been widely used (70, 78, 106). Numerous refinements have been made, particularly in an effort to avoid generating candidate overlaps from repetitive *k*-mers.

Because a globally optimal solution to the layout stage of the assembly is computationally infeasible, the layout step typically tries to generate unitigs, which are collections of reads that can be unambiguously assembled without significant chance of misassembly (68). The assembler typically removes low-quality sequence reads and overlaps that are likely to be sequencing artifacts, and then removes redundant edges in a process called transitive reduction (e.g., if the graph contains the triangle A-B-C, then the edge A-C can be removed because it is redundant with the path A-B-C). Finally, the layout algorithm finds unambiguous regions of the graph. After the reads are ordered and oriented, a multiple alignment is constructed from the chain of overlaps, and a consensus sequence is inferred.

This approach to assembly has been very successful (2). Celera's construction of the human genome sequence remains the exemplar of whole-genome shotgun sequencing using capillary-based DNA sequencing technology (109). Despite the success of OLC methods with capillary-based sequence data, this strategy struggled when presented with the vast amounts of short-read data generated by the next generation of DNA sequencing instruments. The overlap computation step was a particular bottleneck. Naive methods that scaled quadratically were obviously impractical when faced with data sets containing hundreds of gigabases of sequence. Even with clever techniques to reduce the search space by indexing *k*-mers, spurious matches due to the short read length overwhelmed the overlap computation. The size of the resulting overlap graph also proved problematic. The number of edges in an overlap graph grows quadratically with depth of coverage and repeat copy number. When faced with high-depth data and many spurious overlaps, the resulting graph may be impractically large. For these reasons, most work on assembling high-throughput short-read sequence data has relied on the de Bruijn graph approach, described in more detail below.

De Bruijn graphs. The de Bruijn graph method of sequence assembly has its roots in theoretical work from the late 1980s. Pevzner (84) studied the problem of reconstructing a genome sequence when only its set of constituent *k*-mers is known. This theoretical work and subsequent development (40, 86) laid the foundation for de Bruijn graph-based assembly of whole-genome sequencing data.

In this type of assembly, each read is broken into a sequence of overlapping *k*-mers. The distinct *k*-mers are added as vertices to the graph, and *k*-mers that originate from adjacent positions in a read are linked by an edge. The assembly problem can then be formulated as finding a walk through the graph that visits each edge in the graph once—an Eulerian path problem. In practice, sequencing errors and sampling biases obscure the graph, so a complete Eulerian tour through the entire graph is typically not sought (85, 86, 114). Even when an Eulerian path through the entire graph can be found, it is unlikely to reflect an accurate sequence of the genome because of the presence of repeats, as there are a potentially exponential number of Eulerian traversals of the graph, only one of which is correct (47). In most instances, the assembler attempts to construct contigs consisting of the unambiguous, unbranching regions of the graph.

Although de Bruijn graph assembly was developed for capillary-based sequencing data, it did not reach prominence until the emergence of high-throughput short-read sequencing methods. The de Bruijn graph approach has a significant computational advantage when compared with overlap-based assembly strategies: It does not require finding overlaps between pairs of reads and, therefore, does not require expensive dynamic programming procedures to identify such overlaps. Instead, the overlap between reads is implicit in the structure of the graph. The graph can easily be constructed in two passes over the data: First, the set of *k*-mers can be extracted from the reads and added as vertices in the graph, and second, adjacent *k*-mers can be extracted from the reads and added as edges. With suitable choices of data structures to represent the graph, this process can be completed nearly as fast as the data can be read from disk.

Early work by Zerbino & Birney (114) and Chaisson & Pevzner (15) demonstrated that bacterial genome sequences could be efficiently assembled from short reads using de Bruijn graph-based methods. Although the quality of the assemblies was limited by the short read length (47), the profound reduction in the cost of generating a partial genome sequence assembly was a turning point for the field.

The assembly of high-throughput sequencing data for large genomes would require further algorithmic development, however, because the memory costs of de Bruijn assembly are significant. As there is approximately one *k*-mer for every base in a genome, the de Bruijn graph of mammalian-sized genomes has billions of vertices. Sequencing errors compound this problem because each such error corrupts the true genomic sequence into up to *k* erroneous *k*-mers. These erroneous *k*-mers introduce new vertices and edges to the graph, significantly expanding its size. Representing the de Bruijn graph in a minimal amount of memory became a key problem for the field.

The ABySS (Assembly by Short Sequences) assembler (102) introduced a representation of the graph that did not explicitly store edges. Rather, it represented the graph as a hash table of *k*-mers, with each *k*-mer storing a byte representing the presence or absence of its eight possible neighboring *k*-mers. This representation allowed the hash table to be distributed across a cluster of computers. When the de Bruijn graph was traversed, the distributed hash table was queried to recover the neighboring *k*-mers for a given vertex. This distribution strategy allowed the assembly of a human genome sequence from 36-base reads.

The idea of representing a de Bruijn graph as a simple set of *k*-mers was also used by Conway & Bromage (19). Their core idea was to represent each *k*-mer as a number in the range $[0, 4^k)$ and to set the corresponding bit in an array of size 4^k. This bit array can be queried to recover the vertices and edges of the graph. Storing an array of size 4^k is obviously impractical for all but very small values of *k*. To solve this problem, Conway & Bromage used sparse bit vector encoding techniques (79) to represent the bit vector. This shrinks the memory requirements for assembling a human genome sequence using 27-mers to 28.5 bits/*k*-mer, which is a significant improvement over the ABySS encoding approach.

In recent years, the use of Bloom filters (7) to represent a set of *k*-mers has gained popularity. The Bloom filter also uses a bit array to represent a set of objects. Unlike Conway & Bromage's approach, which directly indexes *k*-mers into the bit vector, the Bloom filter uses multiple hash functions to compute the indices of the bits that should be set for a particular *k*-mer. At query time, the same hash functions are used, and the filter checks the corresponding bits. If they are set, the filter claims that the *k*-mer is present in the collection. This algorithm guarantees that if a *k*-mer is added to the collection, it will be reported as present at query time. However, the opposite is not true—the Bloom filter does not guarantee that *k*-mers reported as present were actually added to the collection. The rate that these false positives are generated is controlled by the size of the underlying bit vector and the number of hash functions used. To decrease memory usage, a higher false-positive rate must be tolerated.

The Bloom filter was first applied to *k*-mer counting by Melsted & Pritchard (65). Pell et al. (80) used a Bloom filter to partition a metagenome sequence assembly graph into components that could be individually assembled. This paper demonstrated that graph partitioning and assembly could tolerate a false-positive rate of 15%, for a memory cost of only 4 bits/*k*-mer. Subsequent work by Chikhi & Rizk (16) built a full assembler based on extensions of these principles.

The FM-index data structure developed by Ferragina & Manzini (28), which is extensively used for mapping reads to a reference genome sequence (53, 55, 57, 58), has also been used for sequence assembly. An FM index constructed from a set of sequence reads can be queried for the presence or absence of any *k*-mer (99). This representation is independent of *k*; the FM index can simultaneously represent all de Bruijn graphs of order *k* up to the read length. Two groups recently developed new data structures related to the FM index specifically tailored to representing a de Bruijn graph (9, 94).

The properties of the de Bruijn graph can be exploited to reduce memory consumption. Ye et al. (113) observed that only a fraction of *k*-mers need to be directly stored as vertices. A similar technique was used to improve the memory consumption of the popular SOAPdenovo assembler (60).

String graphs. The de Bruijn graph has the elegant property that repeats get collapsed: All copies of a repeat are represented as a single segment in the graph with multiple entry and exit points. This provides a concise representation of the structure of the genome. In 2005, Myers (69) observed that a similar property could be obtained for overlap-based assembly methods by performing two transformations of an overlap graph. First, contained reads—reads that are substrings of some other read—are removed. Second, transitive edges are removed from the graph. The resulting graph, called a string graph, shares many properties with the de Bruijn graph without the need to break the reads into *k*-mers. The Edena assembler applied the string graph approach for use with early short-read sequencing data (36). Subsequent theoretical work on efficiently constructing the string graph using the FM index (100) led to memory-efficient assemblers for large genomes (54, 101). Several other fast string graph construction algorithms were also developed around this time (5, 23, 33).

Mate pair: a pair of sequence reads from a single fragment of DNA; often the relationship between the reads is only approximately known

Paired ends: a pair of sequence reads derived from each end of a short DNA fragment

Jumping library: a set of mate-pair sequence reads derived from long fragments of DNA; it often requires circularization and/or cloning of DNA prior to sequencing

Scaffold: a series of contigs that has been assembled together using longer-range information, such as mate pairs; also known as a supercontig

Modeling Mate Pairs

Our description so far has focused on the assembly of individual sequence reads—contiguous segments of DNA read by a DNA sequencing machine. Experimental techniques can, however, recover additional information that can constrain the placement of the reads within a reconstruction of the genome. By far the most commonly available information is that associated with mate pairs—pairs of sequence reads whose separation and relative orientations within the genome can be estimated experimentally. (Other types of data are discussed briefly further below.) Before we proceed, we would like to point out that multiple technologies exist for generating mate-pair information, and depending on the specific experimental characteristics, these data are called mate pairs, paired ends, fragment ends, or jumping libraries. For simplicity, here we refer to mate pairs irrespective of the underlying technology used.

Mate-pair information provides valuable constraints on the relative placement of sequence reads in an assembly. These constraints have been used to check the correctness of the assembly (29), and we discuss this specific application in more detail below. Mate-pair information can also be used to link independent contigs into scaffolds—groups of contigs whose relative order and orientation are known and that are separated by gaps of approximately known size. The gaps between contigs represent sections of the genome that could not be reconstructed by the assembler owing to either missing data (e.g., systematic bias of the DNA sequencing technology) or repeats. Mate-pair information can also be used to resolve certain repeats, potentially leading to longer and more accurate contigs.

When using mate-pair information, it is important to realize that the experimental estimates of the spacing between the paired reads (also called library size) are often incorrect and involve a large level of variability. Software using mate-pair information must therefore reestimate the library size. A common strategy for this involves using the placement of reads within long contigs to estimate the distribution of fragment sizes prior to analysis. For long-range mate pairs (e.g., those derived from fragments larger than 20 kb), an accurate estimate may not be easily recovered owing to the small number of mate pairs that map entirely within a contig.

Several strategies have been proposed for generating scaffolds, but they all follow the same broad outline. First, mate-pair information is used to infer links (bundles of consistent mate pairs) between contigs. During this process, incorrect mate pairs are eliminated from further consideration. Second, the relative orientations of the contigs are determined based on the constraints imposed by the mate-pair information. In the presence of errors (e.g., incorrect mate-pair links), the orientation task is NP-hard (nondeterministic polynomial-time hard, implying that finding the correct answer requires sifting through an exponentially large set of possible answers); however, a simple greedy heuristic appears to perform well in practice (45). Third, a linear arrangement of the contigs is determined that minimizes the inconsistencies with the mate-pair data. This latter step is also NP-hard (39), and several heuristics have been proposed in the literature. These include

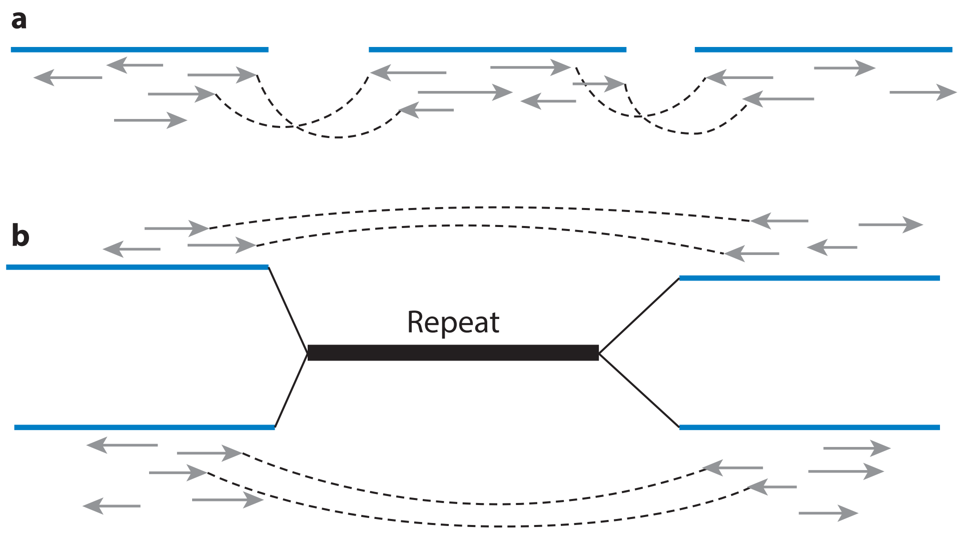

Figure 3
Use of mate pairs for sequence assembly. The thick blue lines represent the (unknown) DNA being sequenced, and the thin gray arrows represent the sequence reads and their respective orientations (from which strand of DNA they originate). Mate pairs (connected by *dotted lines*) constrain the relative placement of reads in the assembled sequence. (*a*) Mate-pair information used to order and orient contigs with respect to each other (scaffolding). (*b*) Mate-pair information used to resolve repeat regions by defining the physical relationships of flanking single-copy regions.

handling different mate-pair libraries in a hierarchical fashion, as done in Bambus (89); treating the mate pairs as flexible springs and minimizing the overall stretch, as done in Celera Assembler (70) and SOPRA (Statistical Optimization of Paired Read Assembly) (21); and performing an exhaustive search, as done in Opera (30), through linear programming (24, 95), or through the greedy extension of scaffolds (8).

Mate pairs can also be used during the assembly process (rather than after contigs have been created, as is the case in scaffolding) to resolve repeats. Specifically, the constraints imposed by mate-pair information can restrict the possible traversals of the assembly graph, thereby reducing or eliminating the ambiguity introduced by repeats. Several algorithms have been developed that explicitly take mate-pair information into account during assembly by identifying traversals of the assembly graph that are consistent with the mate-pair constraints (see **Figure 3**). In ALLPATHS (13), the assembler attempts to enumerate all the paths connecting the endpoints of a mate pair. The paired de Bruijn graph (64) and pathset graph (87) approaches modify the de Bruijn graph structure to implicitly encode the mate-pair information, thereby resolving segments of the graph that are consistent with the mate-pair information.

Our simple description of using mate pairs for repeat resolution belies the actual complexity of the problem. The endpoints of a mate pair may be connected by an exponential number of paths, thereby making it infeasible to find the one or few paths consistent with the length of DNA separating the mate pair. The usefulness of mate pairs for repeat resolution, therefore, decreases as their length increases. Furthermore, the interval size for which mate pairs are most useful for resolving repeats is intimately tied to the size of the repeats themselves, suggesting the need to tune the sequencing approach to the actual structure of the genome being sequenced, or to use a mixture of mate pairs having a varied range of sizes. The latter approach is, for example, advocated by the developers of ALLPATHS-LG (32), an assembler that can be used only if the data are constructed according to a suggested recipe. In brief, this recipe requires a very short mate-pair library with overlapping end reads (effectively an approach for creating longer sequence reads), a short-range mate-pair library (effective for resolving short repeats, which are ubiquitous in many genomes), and a mixture of jumping libraries that span thousands to tens of thousands of base pairs. The jumping libraries are useful for spanning (and possibly resolving) increasingly large repeats as well as for joining contigs across unresolvable repeats (i.e., during scaffolding).

Experimental Design and Long-Read Assemblies

Many groups have focused on strategies for generating longer contigs and scaffolds. The developers of SOAPdenovo used multiple mate-pair libraries to assemble the panda genome sequence into megabase-sized scaffolds (56). As described above, the ALLPATHS-LG group introduced a well-defined experimental protocol consisting of a range of short- and long-insert mate-pair libraries for assembling the sequences of large genomes (32). SPAdes (St. Petersburg Genome Assembler) focuses on assembly of bacterial genome sequences (4). MaSuRCA (Maryland Super-Read Celera Assembler) uses an efficient de Bruijn graph-based preassembly module for the Celera assembler (116), which allowed it to assemble the very large loblolly pine genome sequence (118).

Despite the tremendous improvements to short-read assembly algorithms in the last eight years, read length remains a fundamental limitation (48). For this reason, some projects have opted to use lower-throughput technologies that generate longer reads. One of the first next-generation sequencing instruments, the 454 platform, generates reads of similar length to capillary-based technology and is coupled with an OLC assembler, Newbler. This technology has been used for sequencing many genomes, including the Atlantic cod (105), bread wheat (12), and tomato (108).

The Pacific Biosciences (PacBio) instrument uses single-molecule imaging to generate 10–20-kb reads (26). The relatively high error rate of PacBio data initially limited its use to hybrid assemblies, where Illumina reads are used to error-correct the PacBio reads before assembly using an OLC approach (49). The HGAP (Hierarchical Genome Assembly Process) pipeline was developed to assemble PacBio-generated data without requiring correction using short-read data (17). HGAP is a hierarchical pipeline; it selects the longest PacBio reads to form the basis of the assembly and error-corrects this subset of reads using the remaining data. The corrected long reads are then assembled, and a consensus sequence is generated using the complete data set. This method often generates single-contig assemblies of bacterial genomes. The increasing interest in PacBio-based assemblies has spurred recent algorithmic development and applications for sequencing larger genomes (6, 72).

Assembling Genomes from Mixtures of Organisms and Variant Detection

Our description so far has implicitly assumed that one is trying to reconstruct the sequence of a single genome. The sequencing of genomes from mixtures of organisms (metagenomics) or of diploid or polyploid organisms poses additional challenges. The detection of repeats, for example, can no longer rely on simple depth of coverage statistics and requires a more careful analysis of the assembly or scaffolding graph (51). Furthermore, in this context, it becomes important to identify and characterize the differences between closely related coassembled genome sequences or haplotypes, both to increase the contiguity of the assembly (75, 81, 82) and to inform the biological interpretation of the data (27, 67). In a metagenomic context, approaches for detecting genomic variants include tools that enable the manual inspection of assemblies, such as Strainer (27);

heuristic methods, as implemented in Bambus 2 (51); and the use of graph theoretic algorithms, as done in Marygold (77).

Techniques for assembling human genome sequences are increasingly being used to discover genomic variants. It was clear from the first human short-read genome sequencing studies that finding insertion and deletion polymorphisms (indels) is more challenging than finding single-nucleotide polymorphisms (SNPs), which is due to the difficulty of mapping reads that have complex differences with respect to the reference genome sequence (1, 54). Assembly-based variant detection can help address this by inferring variants directly from the structure of the assembly graph (83). The Fermi assembler used a string graph assembly algorithm to explore how assembly-based variant detection compares with alignment-based approaches (54). The Cortex algorithm is designed to find variants within populations; it represents multiple samples in its de Bruijn graph, including a reference genome sequence (42). SMUFIN (Somatic Mutation Finder) directly compares reads from tumor/normal pairs in cancer studies to find somatic mutations (66). The SGA (String Graph Assembler) de novo assembler (101) was modified to find variants by assembling population and cancer genome sequence data. An alternative to full de novo assembly is performing local assembly of regions of interest; in this approach, reads are mapped to approximate locations on a reference genome and then reassembled in place to infer new haplotypes carrying variants (14, 76, 93, 112).

VALIDATION OF GENOME SEQUENCE ASSEMBLIES

As should be apparent from the discussion above, genome sequence assembly is a complex computational challenge. No algorithms are guaranteed to accurately and completely reconstruct the sequences of large genomes from the short sequences generated by modern DNA sequencing machines. Furthermore, software tools implementing assembly algorithms cannot be guaranteed to be free of programming bugs. Users of assembly tools must therefore be wary of mistakes in the contigs and scaffolds generated by the assemblers. These mistakes, which range from single-nucleotide errors to large-scale misrepresentations of the genome, must be taken into account when analyzing the resulting data.

Detecting such errors is nontrivial. In general, assembly is used to reconstruct sequences that were previously unknown, and therefore no gold standards are available against which the output can be validated. These issues were apparent from the early applications of genome sequence assembly, leading to the development of several de novo approaches for assessing an assembly's correctness. These approaches broadly attempt to identify internal inconsistencies in the reconstructed data based on knowledge or assumptions of the characteristics of the input data. For example, the placement of mate pairs along an assembly should be consistent with the estimated library size and the known relative orientations of the paired reads. This argument was first used to prove the correctness of the first bacterial genome sequence (29), and several approaches have been developed over the years to detect such inconsistencies: TAMPA (Tool for Analyzing Mate Pairs in Assemblies) (22), AMOSvalidate (88), the CE (compression/expansion) statistic (117), and REAPR (Recognising Errors in Assemblies Using Paired Reads) (38).

Other indicators of errors include regions of unusually deep coverage (70, 88), positions in the assembly that contain too many differences among the assembled reads, and even consistent breakpoints in the alignment of unassembled reads against the reconstructed sequence (88). To exemplify the latter example, which may appear unintuitive, imagine a two-copy tandem repeat that is incorrectly replaced with a single instance within the output of the assembler. The reads overlapping the boundary between the two repeat copies in the correct genome reconstruction cannot fit within the misassembled genome and therefore remain unassembled. Their alignment

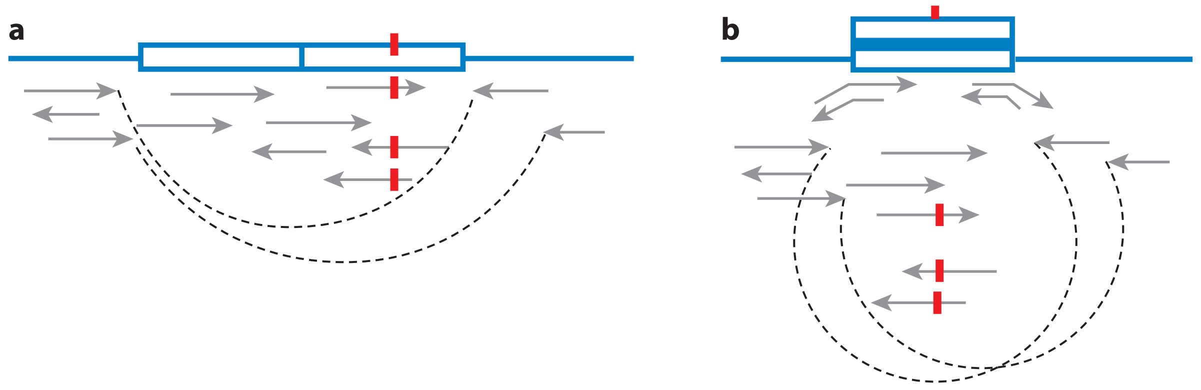

Figure 4

Signatures of misassembly. The thick blue lines represent the (unknown) DNA being sequenced, and the thin gray arrows represent the sequenced reads and their respective orientations (from which strand of DNA they originate). (*a*) A correct assembly of a tandem repeat (*two boxes*) where one copy contains a variant (*vertical red bar*). (*b*) A misassembly of this region, highlighting several signatures: compressed distance between mate pairs (*dotted lines*), disagreements between coassembled reads (*vertical red bars*), and partial read alignments (*bent arrows*) for reads spanning the boundary between repeats.

to the sequence assembly, however, reveals a consistent pattern—all of these reads align partially to the beginning and end of the misassembled repeat, highlighting the presence of a misassembly (see **Figure 4**).

The internal consistency of genome sequence assemblies can also be viewed globally in terms of the fit between the assembled data and the characteristics of the assembly one would expect given the parameters of the sequencing process. Such approaches are effectively statistical hypothesis tests comparing the theoretical distribution of reads along an assembly with the distribution found in the actual assembly. Such a formulation was originally proposed by Myers (68) in the 1990s and has recently reemerged in the literature (18, 31, 62, 91).

Finally, we would like to point out a parallel between algorithms developed for the validation of sequence assemblies and recent work on identifying structural variants in genomes. Structural variants and misassemblies have similar signatures, and thus similar approaches can be applied in both contexts. In fact, a paper targeting the discovery of structural variants (103) rediscovered the geometric strategy used in TAMPA to identify mate-pair inconsistencies (22).

The evaluation of assembler performance has been the focus of multiple benchmarking competitions. The Assemblathon series assessed assembler performance using synthetic data (25) and real data from efforts to sequence the genomes of three species (11). In these competitions, the developers of the assembly software submitted assemblies to the organizers. The organizers scored each entry according to a variety of metrics designed to capture the contiguity, completeness, and accuracy of the assembly. The GAGE (Genome Assembly Gold-Standard Evaluations) competition (96) evaluated assemblies generated by the organizers using four Illumina-derived data sets. In all of these competitions, the same set of broad conclusions was reached: There was highly variable performance among assembly pipelines and data sets. In the GAGE competition, the contiguity and accuracy of the assemblies were not well correlated, and the quality of the input data was a major determining factor of the quality of the resulting assembly. In the Assemblathon, no single pipeline performed best in all scenarios; the organizers suggested that users use multiple assemblers with their data and select the "best" assembly according to multiple metrics instead of optimizing for a single metric, such as N50.

OTHER TECHNOLOGIES

Reducing Complexity by Subcloning

The complexity of genome sequence assembly can, at some level, be reduced by focusing the shotgun sequencing process on smaller sections of the genome. Specifically, one can envisage a hierarchical process wherein the genome is sheared into a collection of large fragments, each of which is then sequenced through the traditional shotgun-sequencing process. Such a strategy was initially employed by the Human Genome Project's effort to sequence the human genome (41). Fragments of DNA of 50–200 kb were cloned into bacterial artificial chromosomes (BACs), and each BAC was then sequenced. A given BAC clone can be expected to contain few or no repeats, thus leading to a simpler assembly problem. Note that in repeat-rich genomes, such as the human genome, assembly remains challenging even at the relatively small scale of an individual clone owing to closely spaced clusters of repeats. The assembled BAC sequences can then be treated as very long reads and assembled using one of the assembly algorithms outlined above. The assembly of these sequences is much simpler because the length of the BAC "reads" exceeds that of the majority of repeats in a typical genome (46).

In addition to BACs, which can be difficult to construct, scientists have used other cloning strategies to derive long DNA fragments. These include fosmids (3), which can accept fragments between 30 and 40 kb in size, and clone-free molecular approaches such as Moleculo technology (90, 110).

An important consideration when using subcloning is the cost of sequencing the individual clones. As sequencing throughput increases and sequencing costs decrease, the cost associated with constructing shotgun libraries from individual clones becomes a significant component of the total cost, and may even exceed the cost of sequencing itself. As a result, several approaches have been developed for reducing the number of shotgun libraries constructed. Broadly, these approaches work by pooling multiple clones and then constructing shotgun libraries from the pools (rather than individual clones). Such an approach was first used by Csűrös & Milosavljevic (20). To reconstruct the identity of the individual clones, each clone was redundantly placed in multiple pools in a matrix design that ensured that pool membership was distinct for each clone. Lonardi et al. (59) recently proposed a similar approach.

Optical Mapping Data

Many sequence assembly challenges arise from the fact that the data generated by sequencing experiments are inherently local: Reads span at most several thousand base pairs, and even mate pairs do not extend this range by much (libraries beyond several tens of thousands of base pairs are expensive and difficult to construct reliably). Optical mapping provides information across longer genomic ranges, commonly spanning several hundred thousand base pairs or more (98). This information is sparse, however, and consists only of the location of specific restriction sites along a genome. Individual restriction sites cannot be distinguished from one another, and the information generated by an optical mapping instrument consists solely of an ordered list of restriction fragment sizes. Nonetheless, this information is complementary to the sequencing data and has been used to validate genome sequence assemblies (11) as well as to guide the genome scaffolding process (74). However, raw optical mapping data have high error rates and require a specialized assembly process to correct errors and extend the range of the maps. Nonetheless, assembled optical maps that span entire chromosomes can be invaluable in the assembly and validation of sequences generated from complex genomes (115).

Integration of Assembly and Validation

As presented above, the validation of genome sequence assemblies is an independent process that takes place after assembly. However, the two processes can and have been intertwined. For example, misassemblies detected by the validation process can be corrected, and the corrected data fed back into the assembler, leading to improved results. Such approaches have been used to some extent in most assemblers (43). The recent development of global likelihood–based measures of assembly quality have also led to new approaches for incorporating quality in the assembly process itself. For example, Medvedev & Brudno (62) explicitly formulated the assembly as the task of traversing an assembly graph in a way that maximizes the assembly likelihood [a similar formulation was also proposed by Myers (68) in the 1990s]. They solved this problem using network flow algorithms. In the same vein, Boža et al. (10) used a simulated annealing approach to find the assembly that maximizes a likelihood score. Finally, Koren et al. (50) used likelihood scores to choose between multiple genome sequence assemblers as well as between multiple possible parameters for each given assembler in order to find the best assembly strategy for a given data set.

DISCUSSION AND CONCLUSIONS

Research into the assembly of genome sequences spans more than 35 years, during which time many algorithms and software packages have been developed. In this review, we have highlighted key developments in this field, with particular attention to current challenges. The development of assembly algorithms has been intimately tied to the rapid evolution of DNA sequencing technologies. Early advances in both fields were the result of close interactions between "wet" and "dry" researchers. Although these interactions became less apparent as the technologies matured, the closer integration between biology and computation has reemerged in recent years as necessary for extracting the most information from the massive data currently being generated.

The latest advances in DNA sequencing technologies may, however, render sequence assembly unnecessary. Most of the developments over the past 35 years have been devoted to mitigating the limitations of DNA sequencing technologies—current sequencing reads are much shorter than genomic repeats. But DNA sequencing technologies that generate reads in the tens of thousands of base pairs largely address this problem, at least for bacterial genomes, making complex assembly algorithms unnecessary. In the coming years, we expect the field to focus on new challenges posed by emerging applications of DNA sequencing technologies. Of particular relevance will be the analysis of genomic variants, whether within populations of eukaryotes (e.g., within the human population) or in metagenomic settings. Long sequence reads alone will not be sufficient to resolve these challenges. Many of the techniques described above, developed in the context of single organisms, will likely find use in addressing these emerging challenges; however, new algorithms also remain to be developed.

SUMMARY POINTS

1. Genome sequence assembly research has a long history, spanning more than 35 years.
2. Longer sequence reads make assembling a genome sequence easier.
3. Experimental design and computation must be tied together for effective genome sequence assembly.

4. Dealing with populations is an emerging and unsolved problem in genome sequencing.
5. The computational challenges of assembly drove much algorithm development for high-throughput genome sequencing technologies.

FUTURE ISSUES

1. High-error reads: Third-generation DNA sequencing technologies (e.g., from Pacific Biosciences and Oxford Nanopore) generate much longer reads than previously possible (tens of thousands of base pairs), but with the cost of much higher error rates.
2. Metagenomics/mixtures of organisms: Increasingly, scientists are sequencing the genomes within mixtures of organisms, whether in the context of metagenomics or within clinical samples (e.g., mixtures of tumor cells). Most of the theoretical framework for sequence assembly was developed for isolated genomes. New methods will need to be developed that can both cope with and characterize heterogeneity within closely related genomes.
3. Dealing with big data: As DNA sequencing costs drop, scientists are increasingly able to focus on larger genomes (such as those of plants) and mixtures (such as soil metagenomes). New approaches will be needed that allow genome sequence assemblers to scale with the amount of data being generated.

DISCLOSURE STATEMENT

J.T.S. has formerly been a consultant for Moleculo and Illumina.

LITERATURE CITED

1. 1000 Genomes Proj. Consort. 2010. A map of human genome variation from population-scale sequencing. *Nature* 467:1061–73
2. Adams MD, Celniker SE, Holt RA, Evans CA, Gocayne JD, et al. 2000. The genome sequence of *Drosophila melanogaster*. *Science* 287:2185–95
3. Ammiraju JS, Yu Y, Luo M, Kudrna D, Kim H, et al. 2005. Random sheared fosmid library as a new genomic tool to accelerate complete finishing of rice (*Oryza sativa* spp. Nipponbare) genome sequence: sequencing of gap-specific fosmid clones uncovers new euchromatic portions of the genome. *Theor. Appl. Genet.* 111:1596–607
4. Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, et al. 2012. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.* 19:455–77
5. Ben-Bassat I, Chor B. 2014. String graph construction using incremental hashing. *Bioinformatics* 30:3515–23
6. Berlin K, Koren S, Chin C-S, Drake J, Landolin JM, Phillippy AM. 2014. Assembling large genomes with single-molecule sequencing and locality sensitive hashing. bioRxiv. doi: 10.1101/008003
7. Bloom BH. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13:422–26
8. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W. 2011. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics* 27:578–79
9. Bowe A, Onodera T, Sadakane K, Shibuya T. 2012. Succinct de Bruijn graphs. In *Algorithms in Bioinformatics*, ed. B Raphael, J Tang, pp. 225–35. Lect. Notes Bioinform. 7534. Berlin: Springer

10. Boža V, Brejová B, Vinař T. 2014. GAML: genome assembly by maximum likelihood. In *Algorithms in Bioinformatics*, ed. D Brown, B Morgenstern, pp. 122–34. Lect. Notes Bioinform. 8701. Berlin: Springer
11. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, et al. 2013. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience* 2:10
12. Brenchley R, Spannagl M, Pfeifer M, Barker GL, D'Amore R, et al. 2012. Analysis of the bread wheat genome using whole-genome shotgun sequencing. *Nature* 491:705–10
13. Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, et al. 2008. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Res.* 18:810–20
14. Carnevali P, Baccash J, Halpern AL, Nazarenko I, Nilsen GB, et al. 2012. Computational techniques for human genome resequencing using mated gapped reads. *J. Comput. Biol.* 19:279–92
15. Chaisson MJ, Pevzner PA. 2008. Short read fragment assembly of bacterial genomes. *Genome Res.* 18:324–30
16. Chikhi R, Rizk G. 2012. Space-efficient and exact de Bruijn graph representation based on a bloom filter. In *Algorithms in Bioinformatics*, ed. B Raphael, J Tang, pp. 236–48. Lect. Notes Bioinform. 7534. Berlin: Springer
17. Chin CS, Alexander DH, Marks P, Klammer AA, Drake J, et al. 2013. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat. Methods* 10:563–69

17. Provided the first demonstration that third-generation sequencing data can be used to assemble entire bacterial genomes despite the high error rates.

18. Clark SC, Egan R, Frazier PI, Wang Z. 2013. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics* 29:435–43
19. Conway TC, Bromage AJ. 2011. Succinct data structures for assembling large genomes. *Bioinformatics* 27:479–86
20. Csűrös M, Milosavljevic A. 2002. Pooled genomic indexing (PGI): mathematical analysis and experiment design. In *Algorithms in Bioinformatics*, ed. R Guigó, D Gusfield, pp. 10–28. Lect. Notes Comput. Sci. 2452. Berlin: Springer
21. Dayarian A, Michael TP, Sengupta AM. 2010. SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinform.* 11:345
22. Dew IM, Walenz B, Sutton G. 2005. A tool for analyzing mate pairs in assemblies (TAMPA). *J. Comput. Biol.* 12:497–513
23. Dinh H, Rajasekaran S. 2011. A memory-efficient data structure representing exact-match overlap graphs with application for next-generation DNA assembly. *Bioinformatics* 27:1901–7
24. Donmez N, Brudno M. 2013. SCARPA: scaffolding reads with practical algorithms. *Bioinformatics* 29:428–34
25. Earl D, Bradnam K, St. John J, Darling A, Lin D, et al. 2011. Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Res.* 21:2224–41
26. Eid J, Fehr A, Gray J, Luong K, Lyle J, et al. 2009. Real-time DNA sequencing from single polymerase molecules. *Science* 323:133–38
27. Eppley JM, Tyson GW, Getz WM, Banfield JF. 2007. Strainer: software for analysis of population variation in community genomic datasets. *BMC Bioinform.* 8:398
28. Ferragina P, Manzini G. 2000. *Opportunistic data structures with applications*. Presented at Annu. Symp. Found. Comput. Sci., 41st, Redondo Beach, CA, Nov. 12–14
29. Fleischmann RD, Adams MD, White O, Clayton RA, Kirkness EF, et al. 1995. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* 269:496–512

29. Provided the first demonstration that shotgun sequencing can be used to reconstruct an entire bacterial genome.

30. Gao S, Sung WK, Nagarajan N. 2011. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J. Comput. Biol.* 18:1681–91
31. Ghodsi M, Hill CM, Astrovskaya I, Lin H, Sommer DD, et al. 2013. De novo likelihood-based measures for comparing genome assemblies. *BMC Res. Notes* 6:334
32. Gnerre S, Maccallum I, Przybylski D, Ribeiro FJ, Burton JN, et al. 2011. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *PNAS* 108:1513–18

32. Described ALLPATHS-LG, among the best assemblers for large genomes, and introduced the idea that the integration of assembly algorithms with experimental design is necessary to achieve good results.

33. Gonnella G, Kurtz S. 2012. Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC Bioinform.* 13:82
34. Green P. 1994. Appendix: algorithms. In *Documentation for Phrap and Cross_Match* (*Version 0.990319*). **http://www.phrap.org/phredphrap/phrap.html**

35. Green P. 1997. Against a whole-genome shotgun. *Genome Res.* 7:410–17
36. Hernandez D, Francois P, Farinelli L, Osteras M, Schrenzel J. 2008. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Res.* 18:802–9
37. Huang X, Madan A. 1999. CAP3: a DNA sequence assembly program. *Genome Res.* 9:868–77
38. Hunt M, Kikuchi T, Sanders M, Newbold C, Berriman M, Otto TD. 2013. REAPR: a universal tool for genome assembly evaluation. *Genome Biol.* 14:R47
39. Huson DH, Reinert K, Myers E. 2001. The greedy path-merging algorithm for sequence assembly. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, pp. 157–63. New York: ACM
40. Idury RM, Waterman MS. 1995. A new algorithm for DNA sequence assembly. *J. Comput. Biol.* 2:291–306
41. **Int. Hum. Genome Seq. Consort. 2001. Initial sequencing and analysis of the human genome. *Nature* 409:860–921**

41. Described the reconstruction of the human genome using a hierarchical sequencing strategy.

42. Iqbal Z, Caccamo M, Turner I, Flicek P, McVean G. 2012. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genet.* 44:226–32
43. Jaffe DB, Butler J, Gnerre S, Mauceli E, Lindblad-Toh K, et al. 2003. Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Res.* 13:91–96
44. Jeck WR, Reinhardt JA, Baltrus DA, Hickenbotham MT, Magrini V, et al. 2007. Extending assembly of short DNA sequences to handle error. *Bioinformatics* 23:2942–44
45. **Kececioglu JD, Myers EW. 1995. Combinatorial algorithms for DNA sequence assembly. *Algorithmica* 13:7–51**

45. Introduced the main concepts and algorithmic underpinnings of the OLC assembly paradigm.

46. Kent WJ, Haussler D. 2001. Assembly of the working draft of the human genome with GigAssembler. *Genome Res.* 11:1541–48
47. Kingsford C, Schatz MC, Pop M. 2010. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinform.* 11:21
48. Koren S, Harhay GP, Smith TP, Bono JL, Harhay DM, et al. 2013. Reducing assembly complexity of microbial genomes with single-molecule sequencing. *Genome Biol.* 14:R101
49. Koren S, Schatz MC, Walenz BP, Martin J, Howard JT, et al. 2012. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol.* 30:693–700
50. Koren S, Treangen TJ, Hill C, Pop M, Phillippy A. 2014. Automated ensemble assembly and validation of microbial genomes. *BMC Bioinform.* 15:126
51. Koren S, Treangen TJ, Pop M. 2011. Bambus 2: scaffolding metagenomes. *Bioinformatics* 27:2964–71
52. **Lander ES, Waterman MS. 1988. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics* 2:231–39**

52. Laid the foundation for the statistical analysis of the shotgun sequencing process.

53. Langmead B, Trapnell C, Pop M, Salzberg SL. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.* 10:R25
54. Li H. 2012. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics* 28:1838–44
55. Li H, Durbin R. 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25:1754–60
56. Li R, Fan W, Tian G, Zhu H, He L, et al. 2010. The sequence and de novo assembly of the giant panda genome. *Nature* 463:311–17
57. Li R, Yu C, Li Y, Lam TW, Yiu SM, et al. 2009. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* 25:1966–67
58. Lippert RA. 2005. Space-efficient whole genome comparisons with Burrows-Wheeler transforms. *J. Comput. Biol.* 12:407–15
59. Lonardi S, Duma D, Alpert M, Cordero F, Beccuti M, et al. 2013. Combinatorial pooling enables selective sequencing of the barley gene space. *PLOS Comput. Biol.* 9:e1003010
60. Luo R, Liu B, Xie Y, Li Z, Huang W, et al. 2012. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience* 1:18
61. Maier D. 1978. The complexity of some problems on subsequences and supersequences. *J. ACM* 25:322–36

62. Medvedev P, Brudno M. 2009. Maximum likelihood genome assembly. *J. Comput. Biol.* 16:1101–16
63. Medvedev P, Georgiou K, Myers G, Brudno M. 2007. Computability of models for sequence assembly. In *Algorithms in Bioinformatics*, ed. R Giancarlo, S Hannenhalli, pp. 289–301. Lect. Notes. Bioinform. 4645. Berlin: Springer
64. Medvedev P, Pham S, Chaisson M, Tesler G, Pevzner P. 2011. Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *J. Comput. Biol.* 18:1625–34
65. Melsted P, Pritchard JK. 2011. Efficient counting of *k*-mers in DNA sequences using a bloom filter. *BMC Bioinform.* 12:333
66. Moncunill V, Gonzalez S, Beà S, Andrieux LO, Salaverria I, et al. 2014. Comprehensive characterization of complex structural variations in cancer by directly comparing genome sequence reads. *Nat. Biotechnol.* 32:1106–12
67. Morowitz MJ, Denef VJ, Costello EK, Thomas BC, Poroyko V, et al. 2011. Strain-resolved community genomic analysis of gut microbial colonization in a premature infant. *PNAS* 108:1128–33
68. Myers EW. 1995. Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.* 2:275–90

69. Introduced the string graph assembly framework.

69. **Myers EW. 2005. The fragment assembly string graph. *Bioinformatics* 21(Suppl. 2):ii79–85**
70. Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, et al. 2000. A whole-genome assembly of *Drosophila*. *Science* 287:2196–204
71. Myers EW, Weber JL. 1997. Is whole human genome sequencing feasible? In *Theoretical and Computational Methods in Genomic Research*, ed. S Suhai, pp. 73–89. New York: Springer
72. Myers G. 2014. Efficient local alignment discovery amongst noisy long reads. In *Algorithms in Bioinformatics*, ed. D Brown, B Morgenstern, pp. 52–67. Lect. Notes Bioinform. 8701. Berlin: Springer
73. Nagarajan N, Pop M. 2009. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J. Comput. Biol.* 16:897–908
74. Nagarajan N, Read TD, Pop M. 2008. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics* 24:1229–35
75. Namiki T, Hachiya T, Tanaka H, Sakakibara Y. 2011. MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. In *Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, pp. 116–24. New York: ACM
76. Narzisi G, O'Rawe JA, Iossifov I, Fang H, Lee YH, et al. 2014. Accurate de novo and transmitted indel detection in exome-capture data using microassembly. *Nat. Methods* 11:1033–36
77. Nijkamp JF, Pop M, Reinders MJ, de Ridder D. 2013. Exploring variation-aware contig graphs for (comparative) metagenomics using MaryGold. *Bioinformatics* 29:2826–34
78. Ning Z, Cox AJ, Mullikin JC. 2001. SSAHA: a fast search method for large DNA databases. *Genome Res.* 11:1725–29
79. Okanohara D, Sadakane K. 2007. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments* (*ALENEX*), ed. D Applegate, GS Brodal, pp. 60–70. Philadelphia: Soc. Ind. Appl. Math.
80. Pell J, Hintze A, Canino-Koning R, Howe A, Tiedje JM, Brown CT. 2012. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *PNAS* 109:13272–77
81. Peng Y, Leung HC, Yiu SM, Chin FY. 2011. Meta-IDBA: a de Novo assembler for metagenomic data. *Bioinformatics* 27:i94–101
82. Peng Y, Leung HC, Yiu SM, Chin FY. 2012. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics* 28:1420–28
83. Peterlongo P, Schnel N, Pisanti N, Sagot M-F, Lacroix V. 2010. Identifying SNPs without a reference genome by comparing raw reads. In *String Processing and Information Retrieval*, ed. E Chavez, S Lonardi, pp. 147–58. Lect. Notes. Comput. Sci. 6393. Berlin: Springer
84. Pevzner PA. 1989. 1-Tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.* 7:63–73
85. Pevzner PA, Tang H. 2001. Fragment assembly with double-barreled data. *Bioinformatics* 17(Suppl. 1):S225–33

86. Introduced the use of the de Bruijn graph approach in the assembly of shotgun sequencing data.

86. **Pevzner PA, Tang H, Waterman MS. 2001. An Eulerian path approach to DNA fragment assembly. *PNAS* 98:9748–53**

87. Pham SK, Antipov D, Sirotkin A, Tesler G, Pevzner PA, Alekseyev MA. 2013. Pathset graphs: a novel approach for comprehensive utilization of paired reads in genome assembly. *J. Comput. Biol.* 20:359–71
88. Phillippy AM, Schatz MC, Pop M. 2008. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biol.* 9:R55
89. Pop M, Kosack DS, Salzberg SL. 2004. Hierarchical scaffolding with Bambus. *Genome Res.* 14:149–59
90. Price J, Ward J, Udall J, Snell Q, Clement M. 2013. *Identification and correction of substitution errors in Moleculo long reads*. Presented at IEEE Int. Conf. Bioinform. Bioeng., 13th, Chania, Greece, Nov. 10–13
91. Rahman A, Pachter L. 2013. CGAL: computing genome assembly likelihoods. *Genome Biol.* 14:R8
92. Räihä K-J, Ukkonen E. 1981. The shortest common supersequence problem over binary alphabet is NP-complete. *Theor. Comput. Sci.* 16:187–98
93. Rimmer A, Phan H, Mathieson I, Iqbal Z, Twigg SR, et al. 2014. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nat. Genet.* 46:912–18
94. Rodland EA. 2013. Compact representation of *k*-mer de Bruijn graphs for genome read assembly. *BMC Bioinform.* 14:313
95. Salmela L, Makinen V, Valimaki N, Ylinen J, Ukkonen E. 2011. Fast scaffolding with small independent mixed integer programs. *Bioinformatics* 27:3259–65
96. Salzberg SL, Phillippy AM, Zimin AV, Puiu D, Magoc T, et al. 2012. GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.* 22:557–67
97. Sanger F. 1975. The Croonian Lecture, 1975: Nucleotide sequences in DNA. *Proc. R. Soc. Lond. B* 191:317–33
98. Schwartz DC, Li X, Hernandez LI, Ramnarain SP, Huff EJ, Wang YK. 1993. Ordered restriction maps of *Saccharomyces cerevisiae* chromosomes constructed by optical mapping. *Science* 262:110–14
99. Simpson JT. 2014. Exploring genome characteristics and sequence quality without a reference. *Bioinformatics* 30:1228–35
100. Simpson JT, Durbin R. 2010. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics* 26:i367–73
101. Simpson JT, Durbin R. 2012. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.* 22:549–56
102. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. 2009. ABySS: a parallel assembler for short read sequence data. *Genome Res.* 19:1117–23
103. Sindi S, Helman E, Bashir A, Raphael BJ. 2009. A geometric approach for classification and comparison of structural variants. *Bioinformatics* 25:i222–30
104. Staden R. 1979. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res.* 6:2601–10
105. Star B, Nederbragt AJ, Jentoft S, Grimholt U, Malmstrom M, et al. 2011. The genome sequence of Atlantic cod reveals a unique immune system. *Nature* 477:207–10
106. Sutton GG, White O, Adams MD, Kerlavage AR. 1995. TIGR Assembler: a new tool for assembling large shotgun sequencing projects. *Genome Sci. Technol.* 1:9–19
107. Tarhio J, Ukkonen E. 1988. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.* 57:131–45
108. Tomato Genome Consort. 2012. The tomato genome sequence provides insights into fleshy fruit evolution. *Nature* 485:635–41
109. Venter JC, Adams MD, Myers EW, Li PW, Mural RJ, et al. 2001. The sequence of the human genome. *Science* 291:1304–51
110. Voskoboynik A, Neff NF, Sahoo D, Newman AM, Pushkarev D, et al. 2013. The genome sequence of the colonial chordate, *Botryllus schlosseri*. *eLife* 2:e00569
111. Warren RL, Sutton GG, Jones SJ, Holt RA. 2007. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics* 23:500–1
112. Weisenfeld NI, Yin S, Sharpe T, Lau B, Hegarty R, et al. 2014. Comprehensive variation discovery in single human genomes. *Nat. Genet.* 46:1350–55
113. Ye C, Ma ZS, Cannon CH, Pop M, Yu DW. 2012. Exploiting sparseness in de novo genome assembly. *BMC Bioinform.* 13(Suppl. 6):S1

109. Demonstrated that the shotgun sequencing approach is feasible for the human genome.

114. Described the first successful practical implementation of the de Bruijn graph paradigm for assembly.

114. Zerbino DR, Birney E. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* 18:821–29

115. Zhou S, Wei F, Nguyen J, Bechner M, Potamousis K, et al. 2009. A single molecule scaffold for the maize genome. *PLOS Genet.* 5:e1000711

116. Zimin AV, Marcais G, Puiu D, Roberts M, Salzberg SL, Yorke JA. 2013. The MaSuRCA genome assembler. *Bioinformatics* 29:2669–77

117. Zimin AV, Smith DR, Sutton G, Yorke JA. 2008. Assembly reconciliation. *Bioinformatics* 24:42–45

118. Zimin AV, Stevens KA, Crepeau MW, Holtz-Morris A, Koriabine M, et al. 2014. Sequencing and assembly of the 22-Gb loblolly pine genome. *Genetics* 196:875–90