Genome Sequence Assembly: Algorithms and Issues

Algorithms that can assemble millions of small DNA fragments into gene sequences underlie the current revolution in biotechnology, helping researchers build the growing database of complete genomes.

Mihai Pop
Steven L. Salzberg
Martin Shumway
The Institute for Genomic Research

Each cell of a living organism contains chromosomes composed of a sequence of DNA base pairs. This sequence, the *genome*, represents a set of instructions that controls the replication and function of each organism. The automated DNA sequencer gave birth to genomics, the analytic and comparative study of genomes, by allowing scientists to decode entire genomes.

Although genomes vary in size from millions of nucleotides in bacteria to billions of nucleotides in humans and most animals and plants, the chemical reactions researchers use to decode the DNA base pairs are accurate for only about 600 to 700 nucleotides at a time.

The process of sequencing begins by physically breaking the DNA into millions of random fragments, which are then "read" by a DNA sequencing machine. Next, a computer program called an *assembler* pieces together the many overlapping reads and reconstructs the original sequence. This general technique, called *shotgun sequencing*, was introduced by Fred Sanger in 1982.[1] The technique took a quantum leap forward in 1995, when a team led by Craig Venter and Robert Fleischmann of The Institute for Genomic Research (TIGR) and Hamilton Smith of Johns Hopkins University used it on a large scale to sequence the 1.83 million base pair (Mbp) genome of the bacterium *Haemophilus influenzae*.[2]

Much like a large jigsaw puzzle, the DNA reads that shotgun sequencing produces must be assembled into a complete picture of the genome. This seemingly simple process is not without technical challenges. For one thing, the data contains errors—some from limitations in sequencing technology and others from human mistakes during laboratory work. Even in the absence of errors, DNA sequences have features that complicate the assembly process—most notably, repetitive sections called *repeats*. The human genome, for example, includes some repeats that occur in more than 100,000 copies each. Similar to pieces of sky in jigsaw puzzles, reads belonging to repeats are difficult to position correctly. Further complicating assembly, some DNA fragments from each genome are impossible to sequence, resulting in gaps in coverage.

The resolution of these problems entails an additional finishing phase involving a large amount of human intervention. Finishing is very costly, as it requires specialized laboratory techniques and highly trained personnel. Assembly programs can dramatically reduce this cost by taking into account additional information obtained during finishing, yet most current assemblers disregard this information and generate the best possible assembly solely from the initial shotgun reads. Advances in assembly algorithms must include features that help finishing efforts.

WHOLE GENOME SHOTGUN SEQUENCING

While shotgun sequencing remains the basic strategy for all genome sequencing projects, its applicability to large genomes has been controversial. Until recently it was applied only at the end of a hierarchical process. This process first breaks the

0018-9162/02/$17.00 © 2002 IEEE

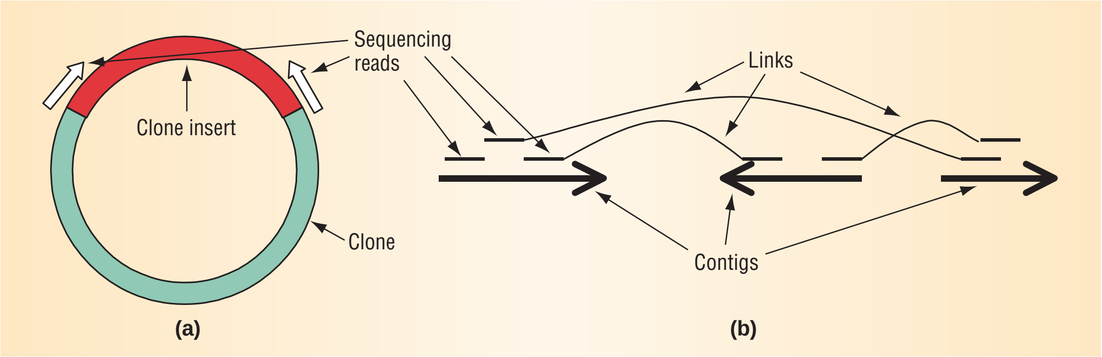

Figure 1. Clone and scaffold. (a) Clone inserts are sequenced from both ends, yielding mated sequence reads. (b) A scaffold uses linking information provided by the clone-pairing data to order and orient contiguous sequences, or contigs, in the genome under assembly.

DNA of large genomes into a set of large pieces called *bacterial artificial chromosomes*. The BACs are then mapped to the genome to obtain a tiling path, after which the shotgun method is used to sequence each BAC in the tiling path separately.

In contrast, whole-genome shotgun sequencing (WGSS) assembles the genome from the initial fragments without using a BAC map. This requires enormous computational resources. The sheer size of the data argued against WGSS for large projects. So did the presence of repeats. If positioning the long repeat stretches correctly is difficult, automating the process is even harder.

In 2000, however, Eugene Myers and colleagues put most doubts to rest when they published a whole-genome assembly of the fruit fly *Drosophila melanogaster*.[3] Using a new assembler built specifically for very large genomes, the Myers team successfully sequenced and assembled the 135-Mbp genome. The project was 25 times larger than any previous WGSS project, and the team went on to apply the WGSS strategy to sequence and assemble the draft human genome in 2001.

Clones and coverage

A WGSS project begins in the laboratory, where ultrasound or a high-pressure air stream randomly shatters the DNA into pieces that researchers then insert into cloning vectors, or *clones*, as illustrated in Figure 1a. The clone in this case is a circular piece of DNA called a plasmid. It has a known sequence of base pairs and can accept a *clone insert* of foreign DNA. The bacterium *Escherichia coli* is then used to multiply the plasmid, thus amplifying the clone insert.

In most projects, researchers sequence both ends of each clone insert, yielding a set of *sequencing reads* that defines the clone-pairing data for that insert. This process links each read from a clone insert to its clone mate from the opposite end of the insert. The resulting clone-pairing data is extremely valuable not only in guiding the assembly process but also in correctly ordering the contiguous sequences, or *contigs*, resulting from assembly. Figure 1b shows the process for three contigs.

The ultimate goal of sequencing is to determine all the base pairs contained in the DNA. In practice, however, we try to achieve the goal of having more than 99 percent of the genome covered by reads after the initial shotgun phase. To achieve this goal we need to sequence clones until the reads (averaging 600 to 700 base pairs) provide an eightfold (8X) oversampling of the genome. For example, a 2-Mbp bacterial genome sequenced to 8X coverage requires 16 Mbp, or approximately 27,000 reads.

Researchers choose the inserts from among several "libraries" of clone collections generated in the laboratory. The insert size specifies the average distance separating each pair of clone mates, and sizes vary from one library to the next. Typical projects contain at least two insert libraries of sizes 2 to 3 kbp and 8 to 10 kbp, respectively, and may include others, such as BAC libraries of 100 to 150 kbp. The sequenced portion of each insert averages 1,200 bp out of 3,000 bp total, so the clone inserts of a 3-kbp library sequenced to 8X cover 2.5 times as much distance as the sequences themselves.

These libraries provide a "clone coverage" of more than 20-fold, meaning that, on average, 20 clones span each of the genome's bases, thus offering the theoretical guarantee that each base is contained in at least one of the clones. This guarantee assumes uniformly random-sampled clones from the genome. In practice, this requirement is seldom perfectly satisfied. Cloning biases lead to a nonrandom clone distribution, causing areas of the genome to remain unsequenced regardless of the amount of sequencing performed.

Assembly

A WGSS assembler's task is to combine all the reads into contigs based on sequence similarity between the individual reads. The basic principle is that two overlapping reads—that is, reads where a suffix of one is a prefix of another—presumably originate from the same region of the genome and can be assembled together. This assumption is invalid, however, for repetitive sequences, where it is impossible to distinguish reads from two or more distinct places in the genome.

Figure 2 shows how an assembler can incorrectly combine the reads from two copies (rpt1A, rpt1B) of a repeat, producing a misassembled contig and throwing out the unique region between the two

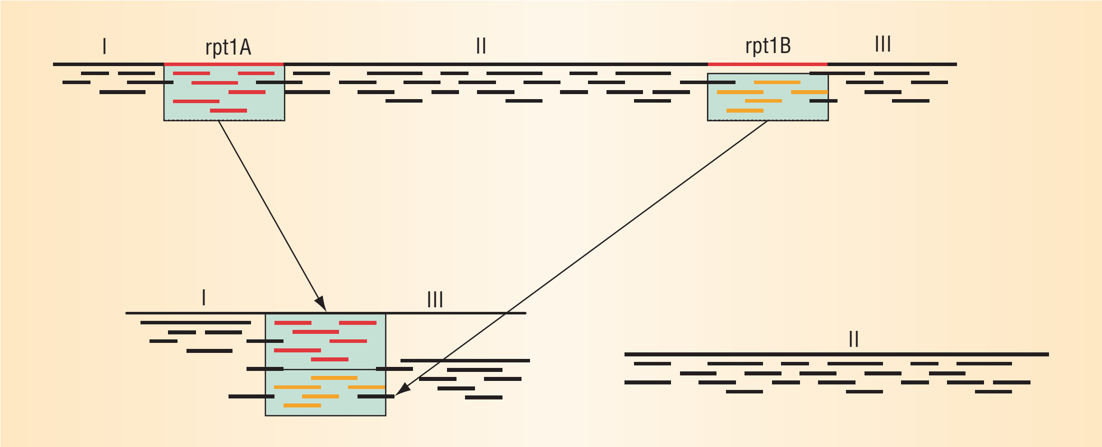

Figure 2. Repeat sequence. The top represents the correct layout of three DNA sequences. The bottom shows a repeat collapsed in a misassembly.

repeat copies as a separate contig. Repeats represent a major challenge to assembly software. An assembler's utility depends in large part on detecting and correctly resolving repeat regions. Resolving misassemblies in the finishing phases can be costly.

Information about clone mates, combined with knowledge about the distribution of clone sizes, may help assembly programs to put some classes of repeats together correctly. If a repeat is shorter than the length of a clone insert, mate-pair information is enough to separate the individual repeat copies because each read within the repeat has an anchoring clone mate in the nearby nonrepetitive region.

Finishing

In practice, imperfect coverage, repeats, and sequencing errors cause the assembler to produce not one but hundreds or even thousands of contigs. The task of closing the gaps between contigs and obtaining a complete molecule is called *finishing*. First, a program called a *scaffolder* uses clone-mate information to order and orient the contigs with respect to each other into larger structures called scaffolds. Within a scaffold, pairs of reads spanning the gaps between contigs determine the order and orientation of contigs, as Figure 1b shows. Note that the physical DNA molecule has an easily determined direction, even though the textual representation of DNA as a string of A, C, T, or G characters appears to be directionless.

The gaps between contigs belonging to the same scaffold are called *sequence gaps*. Although they represent genuine gaps in the sequence, researchers can retrieve the original clone inserts spanning the gap and use a straightforward "walking" technique to fill in the sequence.

Determining the order and orientation of the scaffolds with respect to each other is more difficult. The gaps between scaffolds are called *physical gaps* because the physical DNA that would span them is either not present in the clone inserts or indeterminable due to misassemblies. Filling these gaps involves a large amount of manual labor and complex laboratory techniques.

ASSEMBLY ALGORITHMS

Researchers first approximated the shotgun sequence assembly problem as one of finding the shortest common superstring of a set of sequences: Given a set of input strings $\{s_1, s_2, ...\}$, find the shortest string T such that every s_i is a substring of T.

While this problem has been shown to be NP-hard, there is an efficient approximation algorithm. This greedy algorithm starts by computing all possible overlaps between the strings and assigning a score to each potential overlap. The algorithm then merges strings in an iterative fashion by combining those strings whose overlap has the highest score. This procedure continues until no more strings can be merged.

While it can be argued that the shortest superstring problem does not correctly model the assembly problem, the first successful assembly algorithms applied the greedy merging heuristic in their design. For example, TIGR Assembler,[4] Phrap,[5] and CAP3[6] followed this paradigm.

Greedy algorithms are relatively easy to implement, but they are inherently local in nature and ignore long-range relationships between reads, which could be useful in detecting and resolving repeats. In addition, all current implementations of the greedy method require up to one gigabyte of RAM for each megabase of assembled sequence, assuming the genome was sequenced at 8X coverage. This limits their applicability on currently available hardware to organisms with genomes of 32 Mbp or less. Such organisms include bacteria and a few single-celled eukaryotes, but not plants, mammals, or other multicellular organisms.

These limitations spurred the development of new algorithms. Two approaches exploit techniques developed in the field of graph theory: one that represents the sequence reads as graph nodes and another that represents them as edges.

Overlap-layout-consensus

The first approach, overlap-layout-consensus,[7] constructs a graph in which nodes represent reads, and edges indicate that the corresponding reads overlap. Each contig is represented as a simple path—that is, a path through the graph that contains each node at most once.

An assembler following this paradigm must first build the graph by computing all possible alignments between the reads. A second stage cleans up the graph by removing transitive edges and resolving ambiguities. The output of this stage comprises a set of nonintersecting simple paths in this refined graph, each such path corresponding to a contig. A final step generates a *consensus sequence* for each contig by constructing the multiple alignment of the reads that is consistent with the chosen path.

Detecting misassemblies is more difficult after assembly is completed, and the misassemblies can lead to incorrect genome reconstructions.

Full information about each read in the input is only necessary for the overlap and the consensus stages. The graph-refinement stage stores only a limited amount of information about each overlap, such as its coordinates and length. This allows a memory-efficient implementation.

Not surprisingly, recent WGSS assemblers use this approach.[4,8] The overlap-layout-consensus technique has the additional value of encoding other relationships between reads, such as clonemate information, which an assembler can use in correctly assembling repetitive areas.

Eulerian path

The second graph-theoretical approach to shotgun sequence assembly uses a sequencing-by-hybridization (SBH) technique.[9] The idea is to create a virtual SBH problem by breaking the reads into overlapping n-mers, where an n-mer is a substring of length n from the original sequence. Next, the assembler builds a directed deBruijn graph in which each edge corresponds to an n-mer from one of the original sequence reads. The source and destination nodes correspond respectively to the $n - 1$ prefix and $n - 1$ suffix of the corresponding n-mer. For example, an edge connecting the nodes ACTTA and CTTAG represents the 6-mer ACTTAG. Under this formulation, the problem of reconstructing the original DNA molecule corresponds to finding a path that uses all the edges—that is, an Eulerian path.

In theory, the Eulerian path approach is computationally far more efficient than the overlap-layout-consensus approach because the assembler can find Eulerian paths in linear time while the problems associated with the overlap-layout-consensus paradigm are NP-complete.[7] Despite this dramatic theoretical difference, the actual performance of existing algorithms indicates that overlap-layout-consensus is just as fast as the SBH-based approach.

HANDLING REPEATS

If genomic data included no repeats, an assembler could use any assembly algorithm to put all the pieces together correctly, even in the presence of sequencing errors. The repeats found in real genomes can, however, prohibit correct automated assembly, at least solely from information contained in the original reads.

For example, a large tandem repeat found in the bacterium *Streptococcus pneumoniae* consists of a 24-bp unit that is repeated in identical and nearly identical copies, in tandem, for a stretch covering approximately 14,000 bp. Given that individual reads have an average length of 600 bp and that all reads obtained from this region are identical, no assembler can determine a unique tiling across this repeat. The resulting assembly is likely to contain a reconstruction of a 600-bp section of the repeat in which all the reads have collapsed on top of each other—similar to the situation in Figure 2. In this particular case, clone mates also fail to resolve the problem because the largest clone insert for this project covered only 10 kbp.

This example highlights several issues that assembly programs must address. First, they must identify repeats, preferably during the assembly process, to avoid mistakes caused by overcollapsing repeat copies. Detecting such misassemblies is much more difficult after assembly is completed, and the misassemblies can lead to incorrect genome reconstructions.

Second, assemblers must attempt to correctly assemble as many repeats as possible to reduce the amount of human labor involved in completing the genome. For short repeats, this step can be as simple as using *anchored reads*, meaning those having mates in the unique areas surrounding the repeat. In more complex repeats, the assembler must be able to use additional information obtained through laboratory experiments.

Detecting repeats

A simple solution to the repeat-detection problem identifies the pileup caused by a misassembly. Because the reads come from a random sampling of the genomic DNA, typically with 8X coverage of the genome, areas covered by a significantly large number of reads indicate an over-collapsed repeat. Most assemblers use variations on this simple idea.

Although the idea is useful, it assumes that the reads are sampled uniformly at random from the genome. In reality, certain areas tend to be poorly represented or absent from the sample—for example, if the insert is toxic to the laboratory organism into which it was cloned—while other areas are overrepresented. In addition, low-copy repeats, which appear only two to three times in a genome, may escape detection because they do not appear to be statistically oversampled.

While statistical methods can provide a rough filter, assembly programs must use other techniques to accurately separate out the repeats. As an example, the recently developed assembly program Euler[9] detects repeats by finding complex areas, or *tangles*, in the graph constructed during assembly. Researchers can use the information contained in the tangle to guide experiments to resolve the repeat. Assemblers that simply mask out repeats—another common strategy—lose this information and must obtain it by other means.

Because the cloning process generates reads in pairs from opposite ends of clone inserts, assemblers can use information about clone mates to help detect areas that have been incorrectly assembled due to repeats. Such areas usually contain many instances of clone mates that were assembled either too close or too far from each other, or whose relative orientation is incorrect. This information must be used with care, however, since clone length estimates are usually imprecise, especially for larger clones.

The difficult problem here is finding outliers in a data set whose distribution is unknown. The most reliable information comes from the relative orientation of the sequencing reads, which can nearly always be tracked correctly. When repeats are widely separated in the genome, clone-pairing data can resolve them effectively for reads whose mates are anchored in the neighboring nonrepetitive areas.

Although some repeats are identical, it is more common to find some differences in them. These differences sometimes provide enough information for the assembler to distinguish the copies from one another. In the absence of sequencing errors, a single nucleotide difference between two copies of a repeat is enough to distinguish them.

Researchers have developed several techniques to correct sequencing errors during repeat resolution—for example, see the work of John Kececioglu and Jun Yu.[10] All current techniques are based on finding statistically significant clusters of reads, where the clusters are based on shared differences in the reads. This approach assumes that sequencing errors are independent and, therefore, that an identical position difference in multiple reads is likely to be a real difference typical of that copy of the repeat. For example, if four reads contain an A in position 200 and four other reads contain a G in that position, then the assembler can infer with high confidence that the first four reads come from one copy of the repeat, while the second four represent a different copy.

One drawback of this approach is the need for relatively deep coverage to detect true differences between repeat copies. If a repeat region is difficult to clone—a common phenomenon—the coverage of that repeat will be low. Moreover, true polymorphisms, such as those between different copies of nearly identical chromosomes—for example, each human chromosome occurs in two copies—or from nonclonal source DNA further complicate this problem.

The difficult problem in using clone mates is finding outliers in a data set whose distribution is unknown.

Unresolved repeats

Even using all these information sources, an assembler cannot resolve every repeat. Humans must intervene to finish some complex areas. The basic technique for this task is to separate out the reads coming from distinct repeat copies. In *directed sequencing experiments*, researchers amplify stretches of DNA anchored in unique areas around the repeat. If we consider each copy of the repeat in isolation from the others, an assembly program can put the genome together by holding these repeat contigs together. Assembling a mixture of contigs and reads, while guaranteeing that the contigs will not break up in the process, is known as a *jumpstart assembly*. Only TIGR Assembler currently supports this capability.

SCAFFOLDING SOFTWARE

The scaffolding process groups contigs together into subsets with a known order and orientation. Researchers generally infer relationships between contigs from clone-mate information. Most recent assemblers include a scaffolding step.[3,8,11] Moreover, the Human Genome Project BAC collections were ordered and oriented through scaffolding.[12,13]

To reformulate the scaffolding problem in graph-theoretic terms, we can construct a graph in which the nodes correspond to contigs, and a directed edge links two nodes when mate pairs bridge the gap between them. In this case, each pair of reads implies a particular orientation and spacing of the contigs to form a correct pair (see Figure 1b).

The scaffolding program must now solve three problems:

- Find all connected components in the defined graph.
- Find a consistent orientation for all nodes in the graph, where nodes are connected by two

Ambiguities in a graph can highlight possible misassemblies that finishing teams can investigate.

edge types: those requiring the two nodes to have the same orientation and those forcing the two nodes to have different orientations. We call the latter *reversal edges*. A consistent orientation of all the nodes is possible only if all undirected cycles contain an even number of reversal edges. Because errors in the pairing data or misassemblies can invalidate this condition, we must solve an optimization problem: Find the smallest number of edges that must be removed so that no cycle has an odd number of reversal edges. This optimization problem is NP-complete.

- Given the length estimates of the edges, embed the graph on a line or—for some bacterial and archaeal genomes—on a circle, such that the least number of constraints is invalidated. This problem is a special case of the optimal linear arrangement problem, which is also NP-complete.

While the last two problems are difficult from a theoretical standpoint, simple heuristics can easily handle the instances encountered in practice. Moreover, in practice we can relax the optimality criteria. During the finishing phase, for example, a linear embedding of the contigs is not necessary. In fact, ambiguities in the graph can highlight possible misassemblies, and finishing teams can use this information in designing experiments to confirm a particular embedding of the graph.

The complexity of scaffolding stems specifically from the presence of errors in the data. Again, simple heuristics can reduce the effect of such errors. For example, we can reduce errors caused by data tracking problems or misassemblies by requiring at least two sources of linking information between contigs or by ignoring links anchored in repeat areas.

For any but the smallest genomes, it is unlikely that a single scaffold will hold all contigs. Thus we will need additional information to order and orient the scaffolds themselves. Two common sources of such information are physical maps and comparisons to related organisms.

Physical mapping encompasses a variety of laboratory techniques for characterizing a set of markers along a DNA strand. Markers include known genes and short, unique sequences of a few hundred nucleotides, called tags, that researchers have fluorescently tagged and mapped to an approximate point on a chromosome. Determining the layout of these markers before sequencing provides an independent information source for scaffolding software. Using contigs created by an assembler, researchers can simulate the mapping experiment computationally by searching for the tag locations in the contig sequence. The comparison between the electronic map and the physical map also provides ordering information that the scaffolding program can use.

The sequence of a closely related organism is another source of scaffolding information. For example, by aligning the scaffolds from a preliminary assembly of the mouse genome to the human genome, we can obtain the likely order of the mouse contigs. Of course, this information will be incorrect where major genome rearrangements have occurred in the evolutionary divergence of the two species. This technique therefore works best with a very closely related genome that has been sequenced to completion.

The sources of linking information used to construct scaffolds vary in quality. In particular, the error in determining the length of inserts, and thus the distance between clone mates, increases with the insert size. Physical map data is inherently error prone. Finally, large-scale genome rearrangements can affect the homology data. TIGR has developed Bambus, a scaffolder that factors our confidence in the linking information into hierarchically constructed scaffolds. The algorithm first builds a set of scaffolds based on the highest confidence links, then incorporates the lower confidence information to combine each scaffold into a larger structure. This hierarchical method reduces the effect of incorrect linking data, while still using all the information sources.

ASSESSING ASSEMBLY QUALITY

Correcting misassemblies is expensive, especially if they go undetected until the late stages of a sequencing project. Assemblers highlight problematic areas by outputting the confidence level in each base of the consensus. Because this simple quality-control method is an inherently local measure, it fails to capture larger scale phenomena, such as whole DNA sections that are incorrectly spliced together. The assembly pipeline must therefore contain a validation module that uses additional information to determine the contig quality.

Finding errors in assemblies is easy when the complete sequence is already known, and we can use known benchmark data sets to fine-tune assembly software. These data sets, either artificially generated or representing real sequencing reads from completed projects, provide both the correct consensus sequence and the exact location of all reads in the true DNA sequence. Such information lets us detect both local errors in the consensus base calls and large-scale rearrangements, such as reversals and insertions, in the genome assembly.

We can apply some of these ideas to the challenge in real practice: finding assembly errors when the true layout is unknown. For example, physical maps provide markers that we can use to validate large contigs. Similarly, we can use the sequence of a closely related organism to confirm areas that we do not expect to have significantly diverged. In the absence of any other types of information, clone mates have been used to detect assembly errors.[14] Areas of the genome that violate the orientation and distance constraints imposed by the clone mates indicate potential misassemblies.

Most reported measures of assembly quality are aggregate measures, such as the number and sizes of contigs. They assume that an assembly consisting of a few large contigs is better than one composed of many small contigs. This assumption is partly true, in that the number of contigs indicates the number of gaps, which in turn correlates with the amount of work needed to finish the genome. Aggregate size measures do not, however, account for the possibility of misassemblies, and they are therefore only marginally useful. If anything, an assembler can generate large contig sizes at the expense of misassemblies.

To demonstrate these concepts, we performed a series of tests on the genome of *Wolbachia*, an endosymbiotic bacterium found in the *Drosophila* fruit fly and other insects. We recently completed this genome at TIGR, so we had a "true" DNA sequence to which we could compare assembly results. We assembled this genome from the original shotgun reads using Phrap, TIGR Assembler, and Celera Assembler. We ran all assemblers with their default settings. We verified the assemblies by aligning the resulting contigs to the finished sequence. Table 1 summarizes the results.

According to the number and length of contigs, Phrap appears to produce the best output, followed closely by TIGR Assembler. The Phrap assembly contains about one-fourth as many contigs as Celera Assembler's, and its contigs are about four times larger on average. In addition, the total size of these contigs (1.26 Mbp) matches the actual size of the *Wolbachia* genome (1.26 Mbp).

In contrast, if we look at the proportion of the sequence covered by correct assemblies, the Celera Assembler's output spans more than 99 percent of all bases, while the TIGR Assembler contigs cover just over 93 percent, and Phrap covers barely 36 percent.

Table 1. Comparison of shotgun sequence data from the *Wolbachia* genome project.

Assembler	Number of contigs	Average contig length	Total size	Percent genome covered	Number of misassemblies
Phrap	56	22.4 kbp	1.26 Mbp	36.0	14
TIGR Assembler	76	16.8 kbp	1.28 Mbp	93.1	2
Celera Assembler	220	6.3 kbp	1.39 Mbp	99.1	1
Celera Assembler trimmed data	101	12.5 kbp	1.26 Mbp	98.4	0

These results lead to a couple of conclusions. First, Phrap and TIGR Assembler appear to have misassembled some repeats, which explains the lack of coverage. At the same time, the small length of the Celera Assembler's contigs, combined with their large total size, lead us to believe that it failed to combine many contigs that should have been assembled together. Closer examination—and similar experience with many other genomes—indicates that this usually results from poor quality data at the ends of sequences.

To get Celera Assembler to combine more contigs, we performed an additional step of more aggressively trimming poor quality data from the ends of the input sequences. The fourth row in Table 1 indicates that this technique closed a number of gaps, yielding larger contigs overall. At the same time, the coverage of the genome decreased, indicating a potential drawback to the technique.

These observations correlate with our understanding of the assembly algorithms used by the three programs. TIGR Assembler and Phrap are more tolerant of incorrect data at the sequence ends, which allows them to create bigger contigs. At the same time, their handling of clone-mate information is less sophisticated than Celera Assembler's. In particular, TIGR Assembler uses a greedy approach that lets it walk through a repeat, occasionally violating clone-link constraints.

The Phrap program simply does not take these constraints into consideration, leading to the over-collapse of repeat regions. Closer analysis verified the hypothesis that all the misassemblies presented in Table 1 correlated with repeats in the *Wolbachia* genome.

The ultimate goal of genome sequencing is the complete DNA sequence of an organism. A good assembler can aid the human effort involved in the finishing phase. An assembler designed for finishing should use multiple sources of information, such as data from finishing experiments in the laboratory; it should also let the human experts put restrictions on the assembly, such as regions that need to be held together or repeats that should be kept separate. Better quality-control tools are essential, and defining quality measures that make it possible to evaluate assembly algorithms is a first step toward their improvement. This issue is particularly critical for incomplete genomes, such as the various and constantly changing versions of the draft human genome sequence. Assemblers that can report "weak" areas in the assembly and highlight potential misassembly sites are essential not only for the subsequent analysis of assembly data but also for guiding the efforts of finishing experts. Moreover, well-defined objective quality measures will provide an additional level of validation even in the case of completely finished genomes. ■

Acknowledgments

The authors are supported in part by grants from the National Science Foundation and the National Institutes of Health.

References

1. F. Sanger et al., "Nucleotide Sequence of Bacteriophage Lambda DNA," *J. Molecular Biology*, vol. 162, no. 4, 1982, pp. 729-773.
2. R.D. Fleischmann et al., "Whole-Genome Random Sequencing and Assembly of *Haemophilus Influenzae* Rd," *Science*, vol. 269, no. 5223, 1995, pp. 296-512.
3. E.W. Myers et al., "A Whole-Genome Assembly of *Drosophila*," *Science*, vol. 287, 2000, pp. 2196-2204.
4. G.G. Sutton et al., "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects," *Genome Science and Technology*, 1995, vol. 1, pp. 9-19.
5. P. Green, "Phrap Documentation: Algorithms," Phred/Phrap/Consed System Home Page; http://www.phrap.org (current June 2002).
6. X. Huang and A. Madan, "CAP3: A DNA Sequence Assembly Program," *Genome Research*, vol. 9, no. 9, 1999, pp. 868-877.
7. J.D. Kececioglu and E.W. Myers, "Combinatorial Algorithms for DNA Sequence Assembly," *Algorithmica*, vol. 13, 1995, pp. 7-51.
8. S. Batzoglou et al., "Arachne: A Whole-Genome Shotgun Assembler," *Genome Research*, vol. 12, no. 1, 2002, pp. 177-189.
9. P.A. Pevzner, H. Tang, and M.S. Waterman, "An Eulerian Path Approach to DNA Fragment Assembly," *Proc. Nat'l Academy of Science USA*, vol. 98, no. 17, 2001, pp. 9748-9753.
10. J. Kececioglu and J. Yu, "Separating Repeats in DNA Sequence Assembly," *Proc. 5th Ann. Int'l Conf. Computational Biology* (RECOMB), ACM Press, New York, 2001, pp. 176-183.
11. P.A. Pevzner and H. Tang, "Fragment Assembly with Double-Barreled Data," *Bioinformatics*, vol. 17, suppl. 1, 2001, pp. S225-S233.
12. W.J. Kent and D. Haussler, "Assembly of the Working Draft of the Human Genome with GigAssembler," *Genome Research*, vol. 11, 2001, pp. 1541-1548.
13. D.H. Huson, K. Reinert, and E. Myers, "The Greedy Path-Merging Algorithm for Sequence Assembly," *Proc. 5th Ann. Int'l Conf. Computational Biology* (RECOMB), ACM Press, New York, 2001, pp. 157-163.
14. D.H. Huson et al., "Comparing Assemblies Using Fragments and Mate Pairs," *Proc. Workshop Algorithms in Bioinformatics*, BRICS, Aarhus, Denmark, 2001, pp. 294-306.

***Mihai Pop** is a bioinformatics scientist at the Institute for Genomics Research in Rockville, Maryland. His research interests include DNA sequencing and closure techniques and sequence assembly algorithms and applications. Pop received a PhD in computer science from the Johns Hopkins University. Contact him at mpop@tigr.org.*

***Steven L. Salzberg** is the senior director of Bioinformatics at the Institute for Genomic Research. He is also a research professor in both the computer science and biology departments at Johns Hopkins University. His research interests include gene finding, genome sequence alignment, genome archaeology, and evolution. He developed one of the first computational gene-finding systems in the mid-1990s. Salzberg received a PhD in computer science from Harvard University. He is a member of the AAAS and the ACM. Contact him at salzberg@tigr.org.*

***Martin Shumway** is a software manager at the Institute for Genomic Research, with interests in measurement. He received an MSc in computer science from Colorado State University and is a member of the IEEE. Contact him at shumwaym@tigr.org.*