2002

Bioinformatics Support of Genome Sequencing Projects

Xiaoqiu Huang
Iowa State University, xqhuang@iastate.edu

Bioinformatics Support of Genome Sequencing Projects

Abstract
The genome of an organism is the "book of life". It encodes the complete set of genetic instructions for the development of the organism. The structure of a genome is a linear sequence of nucleotides. Determination of the sequence of a genome lays the foundation for understanding biology at the molecular level. With the current biotechnology, it is a challenging task to determine the sequence of a genome. A sequencing machine can read the sequence of a piece of DNA for up to 1000 bp (base pairs). However, genomes are very huge. For example, the genome of the bacterium E. coli is about 4 Mb (million base pairs) in size, the genome of the nematode C. elegans is 100 Mb in size, and the human genome is 3 Gb in size. The inability to produce long sequences by sequencing machines requires that long sequences be produced from short sequence reads. A shotgun sequencing strategy is widely used to determine the sequence of a long segment of DNA. In this strategy, multiple copies of the DNA segment are randomly cut into small pieces. The sequence of each piece is read by an automated sequencing machine. The sequence of the large DNA segment is reconstructed by a computer program from short sequence reads. The sequence assembly problem is to assemble short reads into long sequences. What makes the sequence assembly problem non-trivial is that there is no information about how short sequence reads are ordered with respect to the DNA segment.

Disciplines
Bioinformatics | Biotechnology | Genetics and Genomics | Molecular Biology

Comments
This chapter was published as Huang, Xiaoqiu. "Bioinformatics Support of Genome Sequencing Projects." In Thomas Lengauer (Ed), Bioinformatics-From Genomes to Drugs (2002): 25-48.

2
Bioinformatics Support of Genome Sequencing Projects

Xiaoqiu Huang

2.1
Introduction

The genome of an organism is the "book of life". It encodes the complete set of genetic instructions for the development of the organism. The structure of a genome is a linear sequence of nucleotides. Determination of the sequence of a genome lays the foundation for understanding biology at the molecular level. With the current biotechnology, it is a challenging task to determine the sequence of a genome. A sequencing machine can read the sequence of a piece of DNA for up to 1000 bp (base pairs). However, genomes are very huge. For example, the genome of the bacterium *E. coli* is about 4 Mb (million base pairs) in size, the genome of the nematode *C. elegans* is 100 Mb in size, and the human genome is 3 Gb in size. The inability to produce long sequences by sequencing machines requires that long sequences be produced from short sequence reads. A shotgun sequencing strategy is widely used to determine the sequence of a long segment of DNA. In this strategy, multiple copies of the DNA segment are randomly cut into small pieces. The sequence of each piece is read by an automated sequencing machine. The sequence of the large DNA segment is reconstructed by a computer program from short sequence reads. The sequence assembly problem is to assemble short reads into long sequences. What makes the sequence assembly problem non-trivial is that there is no information about how short sequence reads are ordered with respect to the DNA segment.

In addition to the huge size of the genome, there are a number of difficulties with solving the sequence assembly problem. First, the genome contains repetitive regions (*repeats*), different parts of the genome that are highly similar in sequence. Methods for reconstructing the sequence of a DNA segment from short reads are based on finding overlaps between reads and using overlaps to guide assembly of reads into long sequences. Two reads have an *overlap* if a region of one read is highly similar to a region of the other read. If two reads come from overlapping parts of the DNA segment, then they have a true overlap. On the other hand, if two reads

come from two repetitive regions, then they have a false overlap. Repeats in the DNA segment lead to false overlaps between reads, which cause sequence assembly methods to make mistakes. Large genomes such as mammalian genomes contain more repeats than small genomes such as microbial genomes.

Second, the orientation of a read is unknown. The DNA segment consists of two complementary strands. It is not known whether a read is from the plus strand or the minus strand of the DNA segment. Third, sequencing machines make errors in reading DNA sequences. A good portion of a read contains errors at a rate of 1% to 5%. Much more errors occur at the ends of a read. Fourth, a read sometimes contains a contaminant at its left end, the sequence of a bit of foreign DNA, which is not from the DNA segment to be sequenced. This bit of foreign DNA comes from a cloning vector or other sources. Fifth, a read occasionally consists of two pieces which are from different parts of the DNA segment. This read is called a *chimeric read.* Sixth, the genomes of different individuals are not identical. For example, it is estimated that the genomes of two different persons differ at a rate of 1 in 1000 bp. This kind of differences is called *polymorphism.* Regions of a genome with a high rate of polymorphism pose a problem to assembly methods.

Two strategies are currently used in genome sequencing projects. In the first strategy called clone-based shotgun sequencing, a *physical map* of the genome is first constructed before sequencing. Here multiple copies of the genome are broken into DNA segments of sizes from 40 kb to 200 kb. Each DNA segment is characterized by a number of markers. A *marker* on a DNA segment could be any identification feature for the segment. Two DNA segments overlap if they share some markers. The DNA segments are ordered using overlaps between segments. A physical map of the genome is a collection of ordered DNA segments that cover the complete genome. Once a physical map of the genome is constructed, DNA segments from the map are selected for sequencing. The sequence of a DNA segment is determined by the shotgun sequencing strategy. The sequence of the genome is constructed from the sequences of the DNA segments according to the physical map. The current cloning vector of choice for carrying a DNA segment is the *bacterial artificial chromosome* (BAC). The advantage of this approach is obvious. It breaks the large genome into segments that can be easily handled by the current sequencing technology. While the genome may contain repetitive regions longer than DNA segments, DNA segments can only contain short repetitive regions. On the other hand, a major difficulty with this approach is in ordering the DNA segments. Because the markers for characterizing DNA segments are not precise and the genome contains long repetitive regions, it is very difficult to order the DNA segments with the markers. The genomes of the bacterium *E. coli*, the yeast *S. cerevisiae*, and the nematode *C. elegans* were sequenced using this clone-based approach with physical mapping [1, 2, 3].

The clone-based strategy is currently used to sequence the human ge-

nome by the public efforts called the Human Genome Project (www.nhgri.nih.gov). Because it was not possible to construct a physical map of the human genome before sequencing, the project went directly to sequencing without a prior physical map. In order to produce human sequences quickly and make them immediately available to the scientific community, the Human Genome Project decided to produce draft sequences of 99% accuracy first. To generate draft sequences, DNA segments were selected and sequenced at low coverage. Here every position of a sequenced DNA segment is covered by three to five reads on the average. Reads from a DNA segment were assembled into a number of contigs (continuous assemblies of DNA sequence). Consensus sequences of contigs were made immediately available on the web.

The initial phase of generating draft human sequences with the clone-based strategy was recently completed [36]. A total of 29,298 DNA segments were selected for sequencing. Of the 29,298 segments, 21,021 segments were sequenced at a low coverage depth of 2 to 5 and the rest at a high coverage depth of 8 to 12. A physical map of DNA segments was constructed simultaneously with the sequencing efforts. The map was used to place the sequenced DNA segments on the genome. Contigs from overlapping segments were merged. The clone-based strategy produced a total of 87,757 scaffolds and a total of 149,821 contigs. Half of all nucleotides of the genome are within a contig of at least 82 kb and within a scaffold of at least 274 kb. The Human Genome Project has moved to the next phase of producing human sequences of high quality (99.99% accuracy) by 2003.

An important issue in the cloned-based strategy is how to produce the complete sequence of the human genome without a physical map. A walking approach has been proposed to address the issue [4]. In this approach, a number of DNA segments are randomly selected and fully sequenced. Then the genome is walked by iteratively selecting and sequencing DNA segments that minimally overlap a fully sequenced DNA segment. To find DNA segments that overlap a fully sequenced DNA segment f, two reads are produced for every DNA segment, one from each end of the segment. A database of all end reads is constructed. The sequence of segment f is compared with the database of end sequences. A DNA segment overlaps segment f if an end sequence of the DNA segment is highly similar to a region of the sequence of segment f. A major problem in the cloned-based strategy is that the walking process can only be carried out in a small number of steps because it takes a month to fully sequence a DNA segment [5]. One way of avoiding many walking steps is to start with a large number of DNA segments as seeds and perform walking from the seeds in parallel. Parallel walking, however, causes a problem of redundant sequencing; DNA segments with large overlaps are sequenced at the same time. Some chromosomes of the plant *Arabidopsis* were sequenced using the walking method [6].

In the second strategy called whole-genome shotgun sequencing (WGS), the shotgun sequencing method is applied to the whole genome. Multiple

copies of the genome are broken into pieces. Both ends of every piece are read by an automated sequencing machine to produce two sequence reads, one from each strand of the piece. The size of the piece is measured, which is the distance between the two reads on the genome. The two reads along with their distance and orientation information form a constraint. Short reads are assembled into contigs. Constraints are used to order and orient contigs from unique and short repetitive regions of the genome into scaffolds. A scaffold is a collection of contigs that are ordered and oriented with respect to the genome. The advantage of the WGS method is that the method can quickly produce a draft picture of the genome that shows the order and orientation of unique and short repetitive regions of the genome. The disadvantage is that it cannot handle repetitive regions that are longer than constraints. A major challenge in the WGS strategy is development of a powerful sequence assembly program that can handle tens of millions of reads and use constraints to sort out short repeats such as the Alu repeat. In 1995, the WGS method was used to sequence the 1.8-Mb genome of the bacterium *H. influenzae* [7].

The WGS method was used to sequence the genome of the fruit fly *D. melanogaster* [8]. A total of 3.16 million reads were generated in the WGS sequencing phase. After trimming to an accuracy level of 98%, the average length of reads is 551 bp. Of the 3.16 million reads, 1.15 million pairs of reads form constraints, 57% of which have an average distance of 2 kb and the rest have an average distance of 10 kb. A total of 491 000 reads including 12 152 pairs of BAC end reads with an average distance of 130.2 kb, were produced by the public efforts using the clone-based method. The joint data set was assembled by the Celera Assembler into 838 firm scaffolds, where a *firm* scaffold contains a contig from a unique region of the genome. Of the 838 firm scaffolds, 134 scaffolds are considered to be the preliminary reconstruction of the euchromatic sequence. There are 25 scaffolds of sizes greater than 100 kb, which cover more than 95% of the assembled sequence. The largest scaffold is 24.3 Mb in size. The assembly is in excellent agreement with the maps of the *Drosophila* genome produced by the clone-based method. The assembly took less than a week on an eight-processor suit of Compaq Alpha ES40s with 32-Gb memory.

Draft sequences of the human genome were produced by the WGS method at Celera Genomics [37]. A random shotgun data set of 27 million reads with a coverage depth of 5 was produced at Celera. After quality and vector trimming, the average length of reads is 543 bp and the average accuracy level of reads is 99.5\%. The data set contains constraints of three distance classes: 2 kb, 10 kb, and 50 kb. The random shotgun data set was combined with a synthetic shotgun data set of 16 million reads. The synthetic shotgun data set was produced by shredding consensus sequences of BAC contigs from the Human Genome Project. No constraint information about the synthetic data set was generated. The combined data set was used for assembly by the WGS method. The WGS method produced a total of

118,968 scaffolds and a total of 221,036 contigs. The average length of scaffolds is 23,938 bp and the average length of contigs is 11,702 bp. The total amount of assembly work was about 20,000 hours for one processor. The computer platform for the assembly task consists of 10 Compaq ES40 servers, each with 4 processors and 4 Gb of main memory, and a Compaq GS160 computer with 16 processors and 64 Gb of main memory.

A localized assembly approach was also pursued by Celera. In this approach, consensus sequences of BAC contigs from the Human Genome Project were used to partition the random and synthetic reads into components such that the reads in a component come from a megabase region of the genome. Each component was assembled by the WGS method. The localized assembly approach produced slightly better results than the WGS method.

Because of major improvements in automated sequencing machines, hundreds of reads can be produced by a sequencing machine in a day. Computers are also much faster and have much more memory. It appears more cost-effective and efficient to produce a draft picture of a mammalian genome with the WGS method first. Then the clone-based method is used to walk the regions of the genome with long repeats, which cannot be handled by the WGS method. The combination of the WGS method and the clone-based method can work if a large portion of the genome are unique or contain only short repeats. The two methods complement each other nicely. The WGS method can produce correct assemblies for unique and short repetitive regions of the genome, which avoids the redundant-sequencing problem with the clone-based method. For every repetitive region of the genome that can not be handled by the WGS method, the clone-based method can sort it out in a few walking steps.

Throughout this chapter we will expect basic familiarity with approaches to the analysis and alignment of biological sequences. This knowledge can be drawn from Chapter 2 of Volume 1 of this book. In the following, we will use notation that is compatible with that chapter.

This Chapter focuses on DNA sequence assembly, a major phase in genome sequencing. The problem is to assemble short reads into long sequences. A *read* is a short DNA sequence. As mentioned before, some bases of a read are less accurate than others. One way of addressing sequencing errors in a read is to give to the assembly program the degree of accuracy of each base in the read. This enables the program to treat bases differently according to their degree of accuracy. The *quality value* of a base in a read is $q = -10 \times \log_{10}(p)$, where p is the estimated error probability for the base [9]. The way in which quality values are defined allows small positive integers to cover a wide range of error probabilities. For example, a quality value of 10 corresponds to an error rate of 1 in 10 and a quality value of 40 corresponds to an error rate of 1 in 10 000. Reads along with their quality values are produced by a base-calling program such as Phred [10]. Phred translates a trace of digital signals from a sequencing machine into a read

and computes a quality value for each base of the read. Input to the assembly program includes a set of reads, their quality values, and constraints.

Reads are assembled into contigs by the assembly program. A *contig* is an ordered list of overlapping reads. A contig is shown in two ways. A layout of all reads in a contig provides an overview of the contig. A multiple alignment of all reads in a contig gives a detailed view of the contig at the base level. The sequence of a contig is a consensus sequence of a multiple alignment of reads in the contig. Output from the assembly program includes the sequences of all contigs and their quality values. A small assembly example is given in this Chapter.

A number of DNA sequence assembly programs have been developed [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. These programs are for BAC and microbial genome assembly. We have developed three versions of the CAP assembly program [13, 21, 22]. In this Chapter, we describe the assembly techniques used in the CAP3 sequence assembly program [22], which is for BAC and microbial genome assembly. The CAP3 program includes a number of improvements and new features. A capability to clip 5′ and 3′ low-quality regions of reads is included in the CAP3 program. Base quality values are used in computation of overlaps between reads, construction of multiple sequence alignments of reads, and generation of consensus sequences. Efficient algorithms are employed to identify and compute overlaps between reads. Forward-reverse constraints are used to correct assembly errors and link contigs.

2.2 Methods

The sequence assembly problem is to reconstruct the sequence of a DNA segment from short reads, where each read is approximately the sequence of a short piece of the DNA segment. Note that reads contain sequencing errors and some reads are chimeric. A mathematical formulation of the problem, called *shortest common superstring problem* (SCS), is to compute a shortest sequence such that each read is exactly identical to a region of the sequence. SCS is a simplification of the assembly problem because it does not address sequencing errors, chimeric reads, or repeats. Nevertheless, it has been shown that SCS is NP-hard, that is, it is likely that there is no efficient algorithm for solving SCS exactly [23]. An approximation algorithm with guaranteed performance has been developed for the SCS problem [24]. The algorithm is guaranteed to produce a sequence of length $O(n \log n)$ for an optimal sequence of length n. Note that constraints are not included in the SCS formulation. A mathematical formulation of the sequence assembly problem that includes constraints, called SCS with constraints, is to compute a shortest sequence such that each read is very sim-

ilar to a region of the sequence and the maximum number of constraints are satisfied by the sequence. Because SCS is a special case of SCS with constraints, it is likely that there is no efficient algorithm for solving SCS with constraints exactly. For SCS with constraints, no approximation algorithm is known that guarantees its performance in terms of percent identity between a sequence produced by the approximation algorithm and an optimal sequence. In practice, an engineering approach has been taken to address the assembly problem. A sequence assembly program is developed, evaluated, and improved on real data. The program is evaluated by comparing the sequences produced by the program with the answer sequence that is verified by experimental laboratory techniques. Evaluation of the program on a large number of real data is very helpful for finding and addressing deficiencies of the program.

The assembly algorithm of CAP3 works in three major phases. In the first phase, pairs of similar reads are identified. Poor end regions of each read are identified and removed. Overlaps between reads are computed. False overlaps are identified and removed. In the second phase, reads are joined to form contigs in decreasing order of overlap scores. Then forward-reverse constraints are used to make corrections to contigs. In the third phase, a multiple sequence alignment of reads is constructed and a consensus sequence along with a quality value for each base is computed for each contig. Note that base quality values are used in computation of overlaps and construction of multiple sequence alignments. We describe each step in detail below.

2.2.1 Fast identification of pairs of similar reads

Pairs of similar reads are identified using a fast database search technique. Let $f_1, f_2, \ldots$ be all the input reads in given orientation and let r_x be the reverse complement of read f_x. The goal of this step is to find quickly pairs of similar reads f_x and f_y with $x < y$ and pairs of similar reads r_x and f_y with $x < y$. Note that each identified pair of reads represents two symmetric pairs. Because reads r_x and r_y are the reverse complements of reads f_x and f_y, respectively, a similarity between reads f_x and f_y is complementary to one between reads r_x and r_y and a similarity between reads r_x and f_y is complementary to one between reads f_x and r_y. Two types of complementary similarities are shown in Figure 2.1.

First consider how to determine quickly if two reads are similar. An alignment between two reads is simplified as an ordered chain of segment pairs, where each segment pair is an ungapped portion of sufficient length of the alignment. A part of the alignment between two adjacent segment pairs is considered as a gap in the chain. Two reads are similar and hence have a potential overlap if the reads contain a chain of similarity score

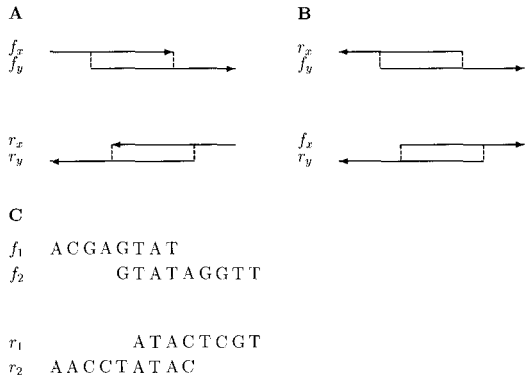

Fig. 2.1
Pairs of complementary overlaps. Read r_x is the reverse complement of read f_x and read r_y is the reverse complement of read f_y. (**A**) A similarity between two reads is indicated by a pair of broken lines. The similarity between reads f_x and f_y and the similarity between reads r_x and r_y are complementary, and (**B**) so are the similarity between reads r_x and f_y and the similarity between reads f_x and r_y. (**C**) An example pair of complementary overlaps.

greater than a score cutoff. The score of a chain is the sum of scores of each segment pair and each gap in the chain. Segment pairs between two reads are computed by extending exact matches between the two reads [25]. See section 2.4 in Chapter 2 of Volume 1 of this book for details of computing segment pairs in the BLAST program. Segment pairs are combined into high-scoring chains by a dynamic programming algorithm as follows [26, 27, 28]. The last antidiagonal of a segment pair is the sum of the end positions of the two segments in the pair. The segment pairs are considered in increasing order of their last antidiagonals. The maximum score of chains ending at the current segment pair is computed. In order to obtain independent high-scoring chains, chains are partitioned into classes such that all chains in a class begin with the same segment pair. For each class, a chain with the maximum score in the class is selected as the representative for the class.

Next consider, for each read f_x, how to find all reads f_y with $x < y$ such that f_x and f_y are similar (have a potential overlap). One approach to addressing this problem is to go through each read f_y with $x < y$ and determine if f_x and f_y are similar. However, this approach is not efficient because it has to go through every read f_y that is not similar to read f_x. Below we describe a hashing technique that allows us to find directly reads f_y that are similar to read f_x.

The sequences of all reads $f_1, f_2, \ldots$ are concatenated together with a special character inserted at every read boundary. The resulting sequence is called the *combined sequence*. A *hash table* is constructed for the combined sequence, where for every word of length w, the table gives the positions of

each occurrence of the word in the combined sequence. Each read f_x is compared with the combined sequence to find pairs of reads f_x and f_y, $x < y$, with a potential overlap as follows. A word of length w at each position of read f_x is considered. The hash table is used to find each occurrence of the word in the combined sequence. Note that reads f_y that have no common word of length w with read f_x are no longer considered. All exact matches of length w between read f_x and the combined sequence are extended into longer segment pairs. Segment pairs are combined into high-scoring chains by the dynamic programming method described above. The special boundary character is used to make sure that only segment pairs from the same read in the combined sequence are combined in a chain. Every high-scoring chain between read f_x and the combined sequence indicates that read f_x is similar to a region of the combined sequence. We need to determine the corresponding read f_y from which comes the region of the combined sequence. This is done as follows. Let j be the position of the region in the combined sequence. Let E be a list such that for each index z, $E(z)$ is the end position of read f_z in the combined sequence. A binary search for position j is performed in the list E to find the smallest index y such that $j \leq E(y)$. Since read f_x is similar to the region of the combined sequence, which comes from read f_y, reads f_x and f_y are similar and have a potential overlap. Note that the chains between read f_x and any read f_y with $x \geq y$ in the combined sequence are ignored. Similarly, each read r_x is compared with the combined sequence to find pairs of reads r_x and f_y, $x < y$, with a potential overlap.

The criterion for selecting a value for the word length w is that the hashing method performs well and requires a reasonable amount of memory. The number of entries in the hash table is 4^w, the number of all words of length w. We suggest that w be set to a value such that 4^w is close to the total length of all reads. This choice requires a linear amount of memory for the hash table and allows the hashing method to perform effectively because each entry in the table is likely to contain a small number of word occurrences. The default value for w in CAP3 is 11.

For a pair of reads with a potential overlap, as indicated by high-scoring chains of segment pairs, the overlap can be computed by the Smith–Waterman algorithm described in Chapter 2 of Volume 1. However, it is slow and not necessary to perform the computation over the full matrix because the overlap is of strong similarity and is located in a small area of the matrix. Here we describe a way of determining the small area of the matrix for the overlap using high-scoring chains computed previously for a pair of reads. A *diagonal* of the matrix is named by a number k, such that the diagonal consists of all entries (i,j) with $j - i = k$. A segment pair beginning with position i of one read and position j of the other read is said to occur on diagonal $j - i$. A *band of diagonals* from numbers *low* to *upp* with $low \leq upp$ consists of diagonals $low, low + 1, low + 2, \ldots, upp$. A band of diagonals in the matrix covers a chain of segment pairs if each segment

pair in the chain occurs on a diagonal inside the band. For a pair of reads with high-scoring chains of segment pairs, a minimum band of diagonals from *low* to *upp* is determined, where *low* is the small number of diagonals on which segment pairs in the high-scoring chains occur and *upp* is the largest number of diagonals on which segment pairs in the high-scoring chains occur. Clearly, the band from diagonals *low* to *upp* covers all the high-scoring chains of segment pairs between the reads.

For each pair of reads with a potential overlap, a minimum band of diagonals is determined to cover all the high-scoring chains of segment pairs between the reads using the method mentioned above. Later the computation of the overlap between the reads is carried out over the band of diagonals in the matrix.

2.2.2 Clipping of poor end regions

Both base quality values and sequence similarities are used to identify poor end regions of full-length reads. Using base quality values alone is not sufficient for finding the exact positions of poor end regions since poor end regions of chimeric reads may have high base quality values. Any sufficiently long region of high quality values that is highly similar to a region of another read is defined to be good. In addition, any sufficiently long region that is highly similar to a good high-quality region of another read is defined to be good. The 3′ clipping position of a read is the maximum of 3′ end positions of good regions of the read. The 5′ clipping position of a read is the minimum of 5′ end positions of good regions of the read.

The high-quality region of each read is identified by computing a region of the read with the maximum quality score using a one-dimensional dynamic programming algorithm. The quality score of a base is its quality value minus a quality value cutoff and the quality score of a region is the sum of quality scores of each base in the region. The quality value cutoff is often an integer between 10 and 20, which corresponds to an error rate between 10% and 1%.

Regions of reads with a strong sequence similarity to other reads are identified by computing the start and end positions of an optimal local alignment for each pair of similar reads. An *overlap* between two reads is defined as an optimal local alignment between the reads, that is, a local alignment with the maximum score. However, if there are two optimal local alignments between the reads as shown in parts A and B of Figure 2.2, which of the two alignments is likely to represent the true overlap? The alignment in part B of Figure 2.2 is likely to represent the true overlap because in the true overlap, an end of a read is likely to be similar to an end of the other read and therefore there is likely to be a short terminal region

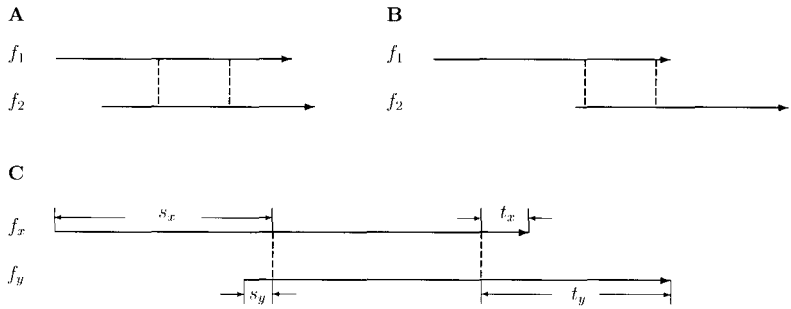

Fig. 2.2
Definition of an overlap as a local alignment. (**A**) An optimal local alignment between f_1 and f_2 with two long terminal regions at each end. (**B**) An optimal local alignment between f_1 and f_2 with a short terminal region at each end. (**C**) A local alignment between reads f_x and f_y. Let s_x and t_x be the lengths of the terminal regions of read f_x before and after the alignment, respectively.

before and after the overlap. The score of a local alignment is defined as follows so that an optimal local alignment (one with the maximum score) represents the real overlap between the two reads. Our strategy is to reflect the lengths of the terminal regions before and after an alignment in the score of the alignment. The score of a local alignment is the sum of scores of each match and each difference in the alignment plus the 5′ and 3′ gap penalty scores for the terminal regions not included in the alignment. Let $term(s) = -(go + ge \times s)$ be the penalty score of a terminal region of length s, where non-negative integers go and ge are gap open and gap extension penalties. The values go and ge should be much smaller than the gap open and extension penalties used to score gaps within the alignment in order to tolerate terminal regions due to contamination. For the alignment in part C of Figure 2.2, the 5′ gap penalty score is the smaller of $term(s_x)$ and $term(s_y)$, and the 3′ gap penalty score is the smaller of $term(t_x)$ and $term(t_y)$. The algorithm of Smith and Waterman [29] is modified to handle penalties for terminal regions. For efficiency, computation with the modified algorithm of Smith and Waterman [29] is restricted to a band of diagonals in the dynamic programming matrix [30, 19]. Recall that our fast method for finding each pair of reads with a potential overlap reports a band of diagonals for the pair. The band is expanded in both directions by a certain number of bases, which can be specified by the user. A local alignment computation is performed over the expanded band of diagonals.

Base quality values are incorporated into the local alignment algorithm. Scores of matches and differences are weighted by the quality values of the bases involved. Specifically, the score of a match or a difference involving two bases of quality values q_1 and q_2 is weighted by $\min(q_1, q_2)$.

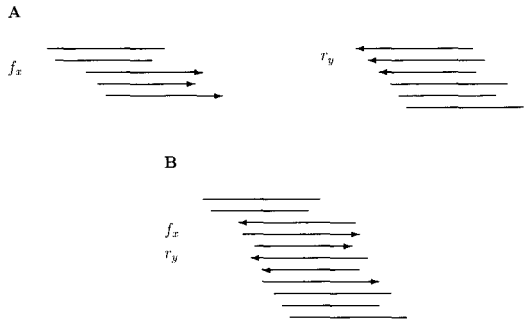

Fig. 2.3
Merging of two contigs into a new contig. (**A**) The current overlap involves read f_x in one contig and read r_y in the other contig. (**B**) A new contig is obtained by merging the two contigs.

2.2.3
Computation and evaluation of overlaps

Overlaps between clean reads are computed as follows. An overlap between two reads is defined as a global alignment of the reads with the maximum similarity score. Match and difference scores are weighted by base quality values. For efficiency, the computation of an overlap between two reads is restricted to a band of diagonals centered at the start position of the optimal local alignment computed previously. The width of the band is the number of diagonals required to cover the optimal local alignment plus the band expansion size specified by the user.

Each overlap is evaluated by several criteria. Any overlap that does not satisfy any of the requirements is not considered in construction of contigs. Requirements are imposed on the length, percent identity, and similarity score of the overlap. In addition, if the overlap contains a sufficient number of differences at bases of high quality values, then the overlap is not used.

2.2.4
Construction of contigs

A greedy method is used to construct contigs as follows. Initially, each read is a contig by itself. The overlaps are considered in decreasing order of their scores. Consider the current overlap involving reads f_x and r_y. If reads f_x and r_y are in separate contigs and the contigs can be merged into a longer contig using the overlap, then the two contigs are merged. Otherwise, no action is taken. Figure 2.3 shows an example of merging two contigs into a longer one. The step is repeated until all the overlaps are considered.

Forward-reverse constraints are used to correct errors in the construction of contigs and to produce a partial order of contigs. A *forward-reverse constraint* consists of two reads, and two integers that specify a range for the

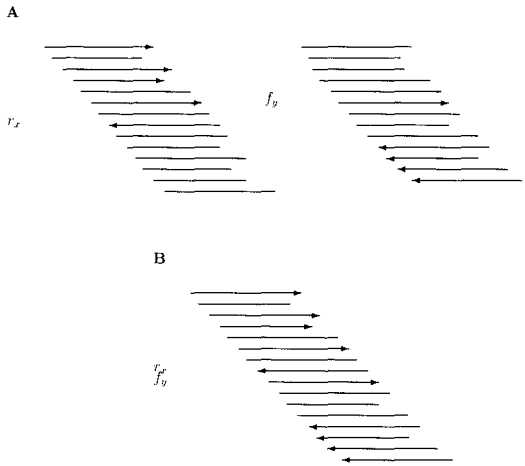

Fig. 2.4
Correction to contigs. (**A**) An unused overlap between reads r_x and f_y with each indicated by a line with an arrow head is supported by four unsatisfied constraints. The eight remaining lines with arrow heads indicate the eight reads involved in the four constraints. (**B**) A new contig is obtained by making a correction to the contigs in **A**. The contig with read r_x is disconnected right after r_x, the contig with read f_y is disconnected right before f_y, and the portion with r_x and the portion with f_y are merged into the new contig.

distance between the reads. Constraints are produced by sequencing both ends of each subclone. A constraint is satisfied by a contig if the two reads occur in the contig, the upstream read is in forward orientation, the downstream read in reverse orientation, and the distance between the two reads is within the given range. Otherwise, the constraint is unsatisfied. An overlap is unused if the overlap is not used in any contig. A group of unsatisfied constraints support an unused overlap if, after a correction is made to the current set of contigs to use the overlap in a resulting contig, all the constraints in the group are satisfied with respect to the resulting contig.

The algorithm for using constraints consists of the following steps. In step 1, each constraint is evaluated with respect to the current contigs to determine if it is satisfied or not. Unsatisfied constraints are partitioned into groups, where all constraints in a group support an unused overlap. In step 2, an unused overlap supported by the largest number of unsatisfied constraints is selected for consideration. If the number of unsatisfied constraints supporting the unused overlap is sufficiently larger than the number of satisfied constraints against the unused overlap, then a correction is made to the current set of contigs to implement the unused overlap. Figure 2.4 shows how a new contig is obtained by making a correction to existing contigs. If no correction is made for the selected overlap, then other unused overlaps are considered until a correction is made to the current set of

contigs or no more unused overlap is available for selection. In step 3, if no correction is made in the last step, then the algorithm is terminated. Otherwise, steps 1 and 2 are repeated.

Contigs are ordered using constraints as links. Unsatisfied constraints are partitioned into groups, where all constraints in a group link a pair of contigs. The groups are considered in decreasing order of group size. For each group of constraints that are associated with a pair of contigs, if the number of constraints in the group is sufficiently large and neither of the two contigs is already linked to other contigs at the corresponding end, then a link between the contigs is established.

2.2.5 Construction of consensus sequences

A multiple sequence alignment of reads is constructed for each contig. The construction is performed by repeatedly aligning the next read with the current alignment. The reads are considered in increasing order of their positions in the contig.

We consider aligning one read with the current alignment of reads. For clarity, the current alignment is called a block. Although the block may be very long, a major portion of the block remains unchanged for the rest of the construction. Thus, only the 3′ portion of the block that gets changed is used for alignment. This 3′ portion of the block is replaced by the resulting alignment. In the following, the block means the 3′ portion. To produce an accurate alignment, base quality values are used in the construction. In this scheme, seven average quality values are computed for each column of the block: five for substitution, one for deletion, and one for insertion. Of the five average quality values for substitution, four values are for the four regular base types and one value is for the ambiguous base type. The average quality value for a base type is the signed sum of quality values of each base in the column divided by the number of bases in the column. The signed sum is computed by considering every base in the column. If the base is of the base type, its quality value is added to the sum. Otherwise, its quality value is subtracted from the sum. The averages quality values of the block are used along with the quality values of the read to weight match and difference scores. A global alignment of the block and the read with the maximum score is computed in linear space using a divide-conquer technique [31, 32, 33]. Since the pair-wise alignment computation is performed at most once for each read, it is affordable to carry out the computation over the entire dynamic programming matrix for best results. The average quality values for the block are precomputed so that each entry in the dynamic programming matrix is calculated in a constant time.

After an alignment is constructed, a consensus sequence along with a quality value for each base is computed for the contig. For each column of

the alignment, a weighted sum of quality values is calculated for each base type and the base type with the largest sum of quality values is taken as the consensus base for the column. The quality value for the consensus base is the sum of quality values for the consensus base type minus the sum of quality values for every other base type. However, if the column contains two base types, each with a very large sum of quality values, then a very low quality value is assigned to the consensus base. The assignment of the low quality value indicates a potential problem often caused by polymorphism or collapsing of highly similar copies of a repetitive element.

2.3 An example

We present a sequence assembly example. The data set in this small example contains 6 real reads of lengths from 799 bp to 1022 bp. The reads are named R1 through R6 with R+ denoting read R in given orientation and R− denoting the reverse complement of read R+. Read R− is obtained by reversing read R+ and exchanging bases A and T, and exchanging bases C and G. Each read consists of a sequence of bases and a sequence of quality values. The base sequence and the quality value sequence are of the same length and the accuracy of a base at position i in the base sequence is indicated by a quality value between 0 and 99 at position i in the quality sequence with 0 being the lowest quality and 99 the highest quality. An exception is that quality value 98 is reserved for an ambiguous base call indicated by the letter N [19]. This exception provides a way for the user to inform the assembly program that some bases are ambiguous. In the assembly program, bases of quality value 98 are not treated as bases of good quality. The base sequences of the reads are in one file and the quality values are in another file. The sequence and quality files are available at http://genome.cs.mtu.edu/cap/data.

The six reads were assembled into one contig by the CAP3 program, with three reads in given orientation and the other reads in reverse orientation. Below we go through the assembly procedure on the example in detail.

A high-quality region of each read was computed using the quality values of the reads. The quality value cutoff was set to 10. Table 2.1 shows the start and end positions of the high-quality region of each read and the length of the read. Fifteen pairs of reads with a similarity were identified. Table 2.2 shows the start and end positions of the similar regions of the reads in the proper orientation for each pair. For example, the first pair in Table 2.2 involves the region from bases 2 to 427 of read R1−, which corresponds to the region from bases 434 to 859 of read R1+. The 5′ and 3′ clipping positions of each read were computed from the pairs of similar regions in Table 2.2. For example, the five pairs of regions involving read R1 give the following

Tab. 2.1
The start and end positions of high-quality read regions.

Read	*Start*[a]	*End*	*Read length*
R1+	5	849	860
R2+	55	888	1022
R3+	55	870	918
R4+	4	790	799
R5+	17	599	920
R6+	70	789	850

[a] Positions are relative to raw reads in given orientation.

Tab. 2.2
The start and end positions of similar read regions.

Read	*Start*[a]	*End*	*Read*	*Start*	*End*
R1−	2	427	R4+	374	799
R1−	2	239	R5+	390	628
R1−	2	311	R6+	518	827
R1+	12	829	R2+	55	873
R1+	12	829	R3+	55	874
R2−	27	545	R4+	282	799
R2−	34	357	R5+	304	628
R2−	27	429	R6+	426	827
R2+	55	838	R3+	55	838
R3−	4	441	R4+	364	799
R3−	90	253	R5+	464	628
R3−	4	325	R6+	508	827
R4+	46	611	R5+	52	628
R4+	18	683	R6+	152	827
R5+	52	628	R6+	181	755

[a] Positions are relative to raw reads in specified orientation.

good regions of read R1+: bases 434–859, 622–859, 550–859, 12–829, and 12–829. The 5′ clipping position of read R1+ is 12, the smallest start position of the good regions, and the 3′ clipping position of read R1+ is 859, the largest end position of the good regions. Table 2.3 shows the clipping positions of each read in forward orientation. The poor regions of each read were removed. The resulting reads are called clean reads.

For each pair of similar regions, an overlap between the clean reads was computed. The overlaps were evaluated. No overlap was removed. Table 2.4 shows the overlaps in deceasing order of scores. A containment is an overlap where one clean read is entirely similar to a region of another clean read. The overlaps were considered in decreasing order of scores for construction of contigs.

Tab. 2.3
The start and end positions of clean regions.

Read	*Start*[a]	*End*	*Read length*
R1+	12	859	860
R2+	55	996	1022
R3+	55	915	918
R4+	18	799	799
R5+	52	628	920
R6+	152	827	850

[a] Positions are relative to raw reads in given orientation.

Tab. 2.4
The positions and scores of overlaps.

Read	*Start*[a]	*End*	*Length*[b]	*Read*	*Start*	*End*	*Length*	*Score*
R1+	1	848	848	R3+	1	850	861	53378
R3+	1	861	861	R2+	1	861	942	51554
R1+	1	848	848	R2+	1	849	942	50668
R6+	1	676	676	R4+	1	666	782	36206
R4+	357	782	782	R1−	1	426	848	19848
R4+	265	782	782	R2−	1	519	942	19133
R4+	347	782	782	R3−	1	438	861	18011
R5+	1	577	577	R4+	29	594	782	10311
R6+	275	676	676	R2−	1	403	942	10308
R5+	1	577	577	R6+	30	604	676	10211
R6+	367	676	676	R1−	1	310	848	10047
R6+	357	676	676	R3−	1	322	861	8516
R5+	246	577	577	R2−	1	331	942	3021
R5+	339	577	577	R1−	1	238	848	2473
R5+	327	577	577	R3−	1	250	861	1871

[a] Positions are relative to clean reads in specified orientation.
[b] Length is the length of a clean read without poor end regions.

A layout of reads in a contig is represented by an ordered list of containment trees. A child read of a parent read in a containment tree means that the child read is completely covered by the parent read in the layout. The root read of a tree is not covered by any other read in the layout. The root reads of two adjacent containment trees overlap in the layout. A doubly linked list of root reads is used to represent both strands of a contig, one direction for one strand.

Initially, each read was a tree by itself. The overlap between reads R1+ and R3+ was first considered. This overlap shows that read R1+ is completely covered by read R3+. Read R1+ was made a child of read R3+ and

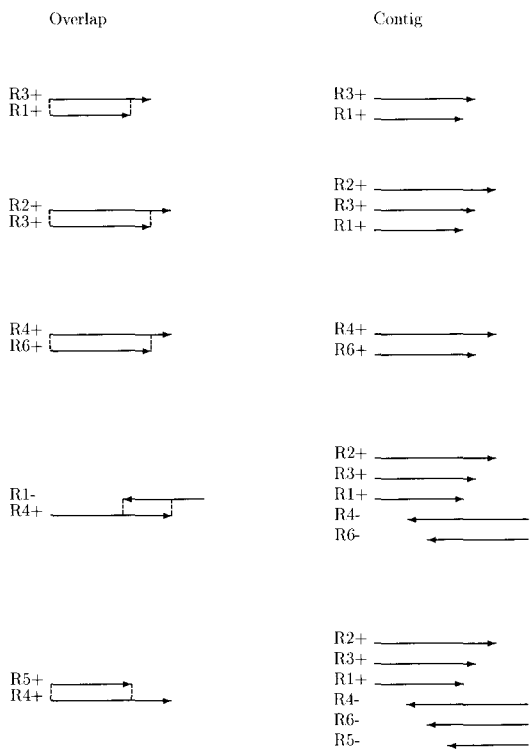

Fig. 2.5
The current overlap and the resulting contig after the overlap is processed. The overlaps are considered in the top-down order.

R1− a child of R3−. The overlap between reads R3+ and R2+ was considered next, which shows that read R3+ is completely covered by read R2+. Read R3+ became a child of read R2+ and R3− a child of R2−. No action was taken on the overlap between reads R1+ and R2+ since reads R1+ and R2+ were already in the same contig. Because of the overlap between reads R6+ and R4+, read R6+ became a child of read R4+ and R6− a child of R4−. The overlap between read R4+ and R1− was next considered. Since R1− was not a root, the root of the tree containing read R1− was located, which was read R2−. A search was performed to see if there was an overlap between R4+ and R2−. Since there was an overlap between R4+ and R2−, a link from R4+ to R2− and a link from R2+ to R4− were set. No action was taken on the overlap between reads R4+ and R2− since they were already in the same contig. There was no action on the overlap between reads R4+ and R3−. The next overlap caused read R5+ to become a child of read R4+ and R5− a child of R4−. No action was taken for each of the remaining overlaps. Figure 2.5 shows the current overlap and the resulting contig for each assembly step.

No forward–reverse constraints were used for this data set. One direction of the doubly linked list of roots was selected for construction of a multiple

```
                .    :    .    :    .    :    .    :    .    :    .    :
R2+             CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C
R3+             CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C
R1+             CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C
R4-             CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C
R5-                   GCGCACCGGACCAGCCCCTATAGAAACCTATGAAGAGTCACCTTCTCAA
R6-             CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C
                ____________________________________________________________
consensus       CCCTCGTGTGCGCGCACCGGGCCAGCCCCTATAGAAACATCTGAGGAGTCACTTCCTC-C

                .    :    .    :    .    :    .    :    .    :    .    :
R2+             CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG
R3+             CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG
R1+             CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG
R4-             CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG
R5-             CTTGACTATCGCCCGCCCGGCCGGATGTAGTCGCCTCCTGGCAAGCTTCAGGCACCTCAG
R6-             CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG
                ____________________________________________________________
consensus       CATGACTCTCGCCCGCCCGGCCGGCTGGAGTCGGCTCCTGGCAAGCTTCAGGCACCTCAG

                .    :    .    :    .    :    .    :    .    :    .    :
R2+             TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTCCAGTTCTC
R3+             TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTTCAGTTCGT
R1+             TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTCCAGTTCTC
R4-             TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTCCAGTTCTC
R5-             TTGTCATGATTACCCACAGCCCACTTTCCTTATCGAAGCCCTTGAGAGCCTCCAGTCCTC
R6-             TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTCCAGTTCTC
                ____________________________________________________________
consensus       TTGTCCTGAATACACACAGCACCCTTTCCTTACTGAAGCCCCTGAGAGCCTCCAGTTCTC

                .    :    .    :    .    :    .    :    .    :    .    :
R2+             CCTCCTTGCT-CACCCAGACTTCCTCCCTCCCGGCCCCTTCAGCGTTCCAAAGGGTAAAT
R3+             CCTCCTTGCTTCACCACACTTTCCTCCTTCCGGGCCCCTTCAGCGTTCCAAAGGGGAAAT
R1+             CCTCCTTGCT-CACCCACACTTCCTCCCTCCCGGCCCCTTCAGCGTTCCCAAAGGTAAAT
R4-             CCTCCTTGCT-CACCCACACTTCCTCCCTCCCGGCCCCTTCAGCGTTCCAAAGGGTAAAT
R5-             CCTACACGTT-CCCACACAATACCTCCATCCCGGCCCCCTCAGAGTTCCAAAGGGTAAAT
R6-             CCTCCTTGCT-CACCCACACTTCCTCCCTCCCGGCCCCTTCAGCGTTCCAAAGGGTAAAT
                ____________________________________________________________
consensus       CCTCCTTGCT-CACCCACACTTCCTCCCTCCCGGCCCCTTCAGCGTTCCAAAGGGTAAAT

                .    :    .    :    .    :    .    :    .    :    .    :
R2+             -GTGGGCTGGGAAGACAGCTCAGCAGTGGAGAAATCGGGGTGCAGG-TCAGTTGCTGCAT
R3+             TGTGGGCTGGGAAGACAGCTCA
R1+             -GTGGGCTGGGA
R4-             -GTGGGCTGGGAAGACAGCTCAGCAGTGGAGAAATCGGGGTGCAGG-TCAGTTGCTGCAT
R5-             -GTGGGCTGGGAAGCCAGCTCAGCAGTGGAGACACCGGGGTGCAGAATCAGTCGCTGTAT
R6-             -GTGGGCTGGGAAGACAGCTCAGCAGTGGAGAAATCGGGGTGCAGG-TCAGTTGCTGCAT
                ____________________________________________________________
consensus       -GTGGGCTGGGAAGACAGCTCAGCAGTGGAGAAATCGGGGTGCAGG-TCAGTTGCTGCAT
```

Fig. 2.6
A portion of a multiple alignment of reads and a consensus sequence produced by CAP3 on the example data set.

alignment of reads in the contig, say, the direction from read R2+ to R4−. The final list consists of two roots R2+ and R4−. The tree rooted at R2+ has the edges from child R1+ to R3+ and from child R3+ to R2+. The tree at R4− has the edges from child R5− to R4− and from child R6− to R4−. The list is shown in Figure 2.5.

A multiple alignment of reads in the contig was constructed. Initially read R2+ was an alignment by itself. Then read R3+ was aligned with read R2+. Read R1+ was next added to the alignment. Then reads R4−, R5−, and R6− in this order were added to the alignment. A portion of the multiple alignment is shown in Figure 2.6. A consensus sequence along with its quality values was produced from the multiple alignment.

2.4 Other assembly programs

We briefly describe computational techniques used in four other sequence assembly programs: Celera Assembler [8]. GAP4 [34], Phrap [19], and TIGR Assembler [18]. Then we indicate the strengths of each program.

Celera Assembler is designed to use many computers to assemble millions of reads into long sequences. In Celera Assembler, a BLAST-like method is used to find quickly pairs of reads with a potential overlap. Overlap and coverage information are used to partition all the reads into unique reads and repetitive reads, where unique reads are from unique regions of the genome and repetitive reads are from repetitive regions of the genome. Unique reads are assembled into contigs called *unitigs*. Uniques are linked into scaffolds of contigs using constraints. Some gaps between unitigs are filled by repetitive reads linked by constraints to unique reads, where the proper place for a repetitive read is determined according to the location of the unique read and the distance in the constraint. Features of Celera Assembler are that it can handle millions of reads and that it makes extensive use of forward-reverse constraints to address the problem of repeats.

The GAP4 program is a comprehensive sequence assembly package. It supports directed and interactive assembly in addition to random shotgun assembly. A few external assembly programs (CAP2, FAKII, and Phrap) can be used in GAP4. Below we describe the internal assembly program in GAP4. Low-quality ends of reads are determined using quality values and removed in a preprocessing step. The clean reads are given as input to the assembly program in GAP4. In the assembly program, the overlap computation and construction of contigs are interleaved. The reads are processed one at a time as follows. The current read is compared with the consensus sequences of the current contigs. If the read overlaps well with some contigs, then a best overlap between the read and a contig is identified and the read is added to the contig by aligning the read with the consensus sequence of the contig. Otherwise, a new contig consisting only of the read is formed. The consensus sequence of a contig is obtained from a multiple alignment of reads in the contig using quality values. A major difference between the internal assembly program in GAP4 and the other assembly programs is that overlap computation and construction of contigs are interleaved, which is suitable for directed and interactive assembly. In the other assembly programs, overlap computation is before construction of contigs, which is suitable for random shotgun assembly.

The Phrap program takes as input full-length reads and their base quality values. The program works in three phases. In the overlap computation phase, a banded Smith–Waterman algorithm is used to compute overlaps between reads. In the contig construction phase, the overlaps are considered in decreasing order of scores. For the current overlap between reads f and g,

if the contig containing read f overlaps the contig containing read g, then the two contigs are merged into one. Ends of the two contigs may need to be clipped in order to make the merging possible. Each contig is represented as a profile. In the last phase, a sequence along with its quality values is generated for each contig. The sequence is a mosaic of read regions of the highest quality values. In other words, the sequence consists of many short pieces, each of which is a region of a read with the highest quality values. A multiple alignment of reads in a contig is constructed by aligning each read with the sequence of the contig. Unique features of Phrap are that poor regions of reads are dynamically determined and removed during construction of contigs and that the sequence of a contig is constructed without using a multiple alignment of reads in the contig.

TIGR Assembler takes as input clean reads, which are produced from raw reads in a preprocessing step. The assembly program works in three phases. In phase 1, overlaps between reads are computed using a fast method based on counting of exact short word matches between reads. In phase 2, contigs are constructed one at a time as follows. A seed read is selected to start a new contig. The contig is grown by repeating the following step. A read that has a best overlap with the current contig is chosen and merged with the contig by aligning the read with a profile of the contig with the Smith–Waterman algorithm. Forward-reverse constraints are also used in the selection of a read to add it to the current contig. In phase 3, for each contig, a consensus sequence is generated from the profile of the contig. TIGR Assembler is the first program to perform whole-genome assemblies of microbial genomes.

Celera Assembler can handle a large mammalian genome. The other assembly programs are for small genomes. There are few performance comparisons of the assembly programs. Performance comparison of CAP3 and Phrap showed that Phrap often produces fewer, longer contigs than CAP3 [22]. Evaluation of CAP3, Phrap, and TIGR Assembler on public ESTs without quality values showed that CAP3 produces more accurate assemblies of ESTs [35]. No comparison of the assembly programs on their efficiency has been published. However, we found that Phrap is a few times faster than CAP3 on genomic BAC data sets and that Phrap requires much more memory than CAP3 on EST data of very deep coverage. CAP3 takes 1–3 hours on a genomic BAC data set [22]. Phrap works well on full-length reads with quality values. TIGR Assembler works well on clean reads with or without quality values. CAP3 works well on full-length reads without quality values. On data sets with constraints, CAP3 and TIGR Assembler make more use of constraints than Phrap. GAP4 works well on directed assembly. The strengths and availability of the assembly programs are summarized in Table 2.5. Because improvements are continually made to the assembly programs, heavy users of assembly programs should obtain the latest version of each program and evaluate the programs on their data. For instance, the fourth version of the CAP program, CAP4, has been de-

Tab. 2.5
The strengths and availability of the sequence assembly programs.

Program	*Strength*	*Availability*
CAP3	Assembly of ESTs	genome.cs.mtu.edu/cap/cap3.html
Celera Assembler	Assembly of large genomes	www.celera.com
GAP4	Interactive assembly	www.mrc-lmb.cam.ac.uk/pubseq
Phrap	Assembly of reads with quality values	www.phrap.org
TIGR Assembler	Assembly of clean reads	www.tigr.org

veloped. On five BAC and microbial data sets with quality values and constraints, CAP4 produced fewer, longer contigs than Phrap and was close to Phrap in running time (www.paracel.com).

2.5 Conclusion

The genome community has made tremendous progress in sequencing the genomes of humans and model organisms. High quality sequences of the model organisms and draft sequences of the human genome have been produced. Efforts are under way to sequence the genomes of a number of important animals and plants. However, with the current sequencing technology, it will take a few years to produce high quality sequences of the human genome. Continued improvements to the sequencing technology are required to meet the challenges in those genome projects. Sequence assembly software, an important component of the sequencing technology, can have a major impact on a genome project, as demonstrated by Celera Assembler. We think that it is necessary to make continued improvements to the efficiency and accuracy of the existing sequence assembly programs in order to produce quickly high quality sequences of a large genome.

Acknowledgments

I would like to thank Thomas Lengauer for many helpful suggestions on the presentation of this Chapter. The work on CAP3 was supported in part by National Institutes of Health Grant R01 HG01502-03 from National Human Genome Research Institute.

References

1 Blattner, F. et al. The complete genome sequence of *Escherichia coli* K-12. *Science* **1997**, 277, 1453–1474.

2 Goffeau, A. et al. Life with 6000 genes. *Science* **1996**, 274, 546–563.

3 The *C. elegans* Sequencing Consortium. Genome sequence of the Nematode *C. elegans*: A platform for investigating biology. *Science* **1998**, 282, 2012–2018.

4 Venter, J. C., H. O. Smith, and L. Hood. A new strategy for genome sequencing. *Nature* **1996**, 381, 364–366.

5 Batzoglou, S., B. Berger, J. Mesirov, and E. S. Lander. Sequencing a genome by walking with clone-end sequences: A mathematical analysis. *Genome Research* **1999**, 9, 1163–1174.

6 Lin, X. et al. Sequence and analysis of chromosome 2 of *Arabidopsis thaliana*. *Nature* **1999**, 402, 761–768.

7 Fleischmann, R. D. et al. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* **1995**, 269, 496–511.

8 Myers, E. W. et al. A whole-genome assembly of *Drosophila*. *Science* **2000**, 287, 2196–2204.

9 Ewing, B. and P. Green. Base-calling of automated sequencer traces using Phred. II. Error probabilities. *Genome Res.* **1998**, 8, 186–194.

10 Ewing, B., L. Hillier, M. C. Wendl, and P. Green. Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Res.* **1998**, 8, 175–185.

11 Staden, R. A new computer method for the storage and manipulation of DNA gel reading data. *Nucleic Acids Res.* **1980**, 8, 3673–3694.

12 Peltola, H., H. Soderlund, and E. Ukkonen. SEQAID: A DNA sequence assembling program based on a mathematical model. *Nucleic Acids Res.* **1984**, 12, 307–321.

13 Huang, X. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics* **1992**, 14, 18–25.

14 Smith S., W. Welch, A. Jakimciu, T. Dahlberg, E. Preston, and D. Van Dyke. High throughput DNA sequencing using an automated electrophoresis analysis system and a novel sequence assembly program. *Biotechniques* **1993**, 14, 1014–1018.

15 Gleizes, A. and A. Henaut. A global approach for contig construction. *Comput. Appl. Biosci.* **1994**, 10, 401–408.

16 Lawrence, C. B., S. Honda, N. W. Parrott, T. C. Flood, L. Gu, L. Zhang, M. Jain, S. Larson, and E. W. Myers. The genome reconstruction manager: A software environment for supporting high-throughput DNA sequencing. *Genomics* **1994**, 23, 192–201.

17 Kececioglu, J. D. and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica* **1995**, 13, 7–51.

18 Sutton, G. G., O. White, M. D. Adams, and A. R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Technology* **1995**, 1, 9–19.

19 GREEN, P. http://www.phrap.org.

20 CHEN T. and S. S. SKIENA. A case study in genome-level fragment assembly. *Bioinformatics* **2000**, 16, 494–500.

21 HUANG, X. An improved sequence assembly program. *Genomics* **1996**, 33, 21–31.

22 HUANG, X. and A. MADAN. CAP3: A DNA sequence assembly program. *Genome Res.* **1999**, 9, 868–877.

23 GALLANT J., MAIER, D. and STORER, J. On finding minimal length superstring. *J. Comput. Sys. Sci.* **1980**, 20, 50–58.

24 JIANG, T. and LI, M. DNA sequencing and string learning. *Math. Systems. Theory* **1996**, 29, 387–405.

25 ALTSCHUL, S. F., W. GISH, W. MILLER, E. W. MYERS, and D. J. LIPMAN. 1990. Basic local alignment search tool. *J. Mol. Biol.* **1990**, 215, 403–410.

26 WILBUR, W. J. and D. J. LIPMAN. Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA* **1983**, 80, 726–730.

27 CHAO, K.-M. and W. MILLER. Linear-space algorithms that build local alignments from fragments. *Algorithmica* **1995**, 13, 106–134.

28 HUANG, X. Fast comparison of a DNA sequence with a protein sequence database. *Microbial & Comparative Genomics* **1996**, 1, 281–291.

29 SMITH, T. F. and M. S. WATERMAN. Identification of common molecular subsequences. *J. Mol. Biol.* **1981**, 147, 195–197.

30 PEARSON, W. R. and D. LIPMAN. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* **1988**, 85, 2444–2448.

31 HIRSCHBERG, D. S. A linear space algorithm for computing maximal common subsequences. *Commun. Assoc. Comput. Mach.* **1975**, 18, 341–343.

32 MYERS, E. W. and W. MILLER. Optimal alignments in linear space. *Comput. Applic. Biosci.* **1988**, 4, 11–17.

33 HUANG, X. On global sequence alignment. *Comput. Appl. Biosci.* **1994**, 10, 227–235.

34 BONFIELD, J. K., K. SMITH, and R. STADEN. A new DNA sequence assembly program. *Nucleic Acids Res.* **1995**, 24, 4992–4999.

35 LIANG, F., I. HOLT, G. PERTEA, S. KARAMYCHEVA, S. L. SALZBERG, and J. QUACKENBUSH. An optimized protocol for analysis of EST sequences. *Nucleic Acids Res.* **2000**, 28, 3657–3665.

36 International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature* **2001**, 409, 860–921.

37 VENTER, J. C. et al. The sequence of the human genome. *Science* **2001**, 291, 1304–1351.