A Survey of Computational Techniques for Genome Sequencing

Sun Kim*

School of Informatics
Center for Genomics and Bioinformatics
Indiana University – Bloomington
sunkim@bio.informatics.indiana.edu

December, 2002

*This project is supported by Korea Insititute of Science and Technology Information

Contents

1 Introduction

The primairy goal of a genome project is to determine the complete sequence of the genome and its genetic content. Thus a genome project is carried out in two separate stages, one for genome sequencing and another for genome annotation. This report surveys computational techniques for genome sequencing.

The most common laboratory mechanism for reading DNA sequences, called *gel electrophoresis*, can determine sequence of up to approximately 10^3 nucleotides at a time. However, the size of an organism's genome is much larger; for example, a human genome consists of approximately 3 billion nucleotides. There are several ways to get around the length limitations inherent in gel electrophoresis. Among them, the most commonly used and most cost effective process is *shotgun sequencing*, which biologically breaks multiple copies (or *clones*) of a target sequence into short, readable, fragments and then reassembles the short fragments to recover the target sequence. The assembly of short fragments in shotgun sequencing was originally done by hand, but manual assembly clearly is not desirable since it is error prone and not cost effective. Due to these reasons, automatic assembly of DNA fragments has been studied for a long period of time [36, 44, 21, 19, 34, 13, 14, 15, 3]. These sequence assemblers contributed to the determination of many genome sequences, including the most recent announcement of the Human genome [48, 28].

The survey consists of three parts: (1) a survey of computational techniques for sequence assembly, (2) for detecting misassembly and (3) for the genome scaffold generation. This survey summarizes computational techniques in genome sequencing, thus the report is more suitable for computer scientists or bioinformaticians, who wants to develop a better genome sequencing environment, rather than biologists or bioinformaticians who intend to learn how to use genome sequencing software.

2 An overview of genome sequencing

Almost all large scale sequencing projects employ the shotgun sequencing strategy that assembles (deduces) the target DNA sequence from a set of short DNA fragments determined from a set of DNA pieces randomly sampled from the target sequence. The set of short DNA fragments, called *shotgun data*, are assembled into a set of *contigs*, or sets of aligned fragments, using a computer program, *sequence assembler*. The sequence assembly is a conceptually simple procedure that generates longer sequences by detecting overlapping fragments as shown in Figure 1. Shotgun data in Step (c) of Figure 1 consists of six fragments, $\{f_1, f_2, f_3, f_4, f_5, f_6\}$, and there is only one contig with all six fragments aligned togther in Step (d). The fragment alignment is based on the existence of overlapping subsequence. For example, f_5 and f_3 are aligned togther since they share the same subsequence, `ATC`. Since there are errors in fragment reads, two fragments can be aligned even when they share only similar – not same – sequences. For example, two fragments, f_6 and f_4, do not share the same sequences but they are aligned together. The sequencing error, `t` in f_4, can be corrected by the rule of majority, two `A`'s and one `t` in this case. That is one of the reasons why shotgun data is of multiple coverage of the target sequence. In this example, the *coverage* of shotgun data is 2.3 (= 31/13) where 31 is the sum of the number of characters in shotgun data and 13 is the length of the target sequence. In general, 6 to 10 coverage shotgun data is needed.

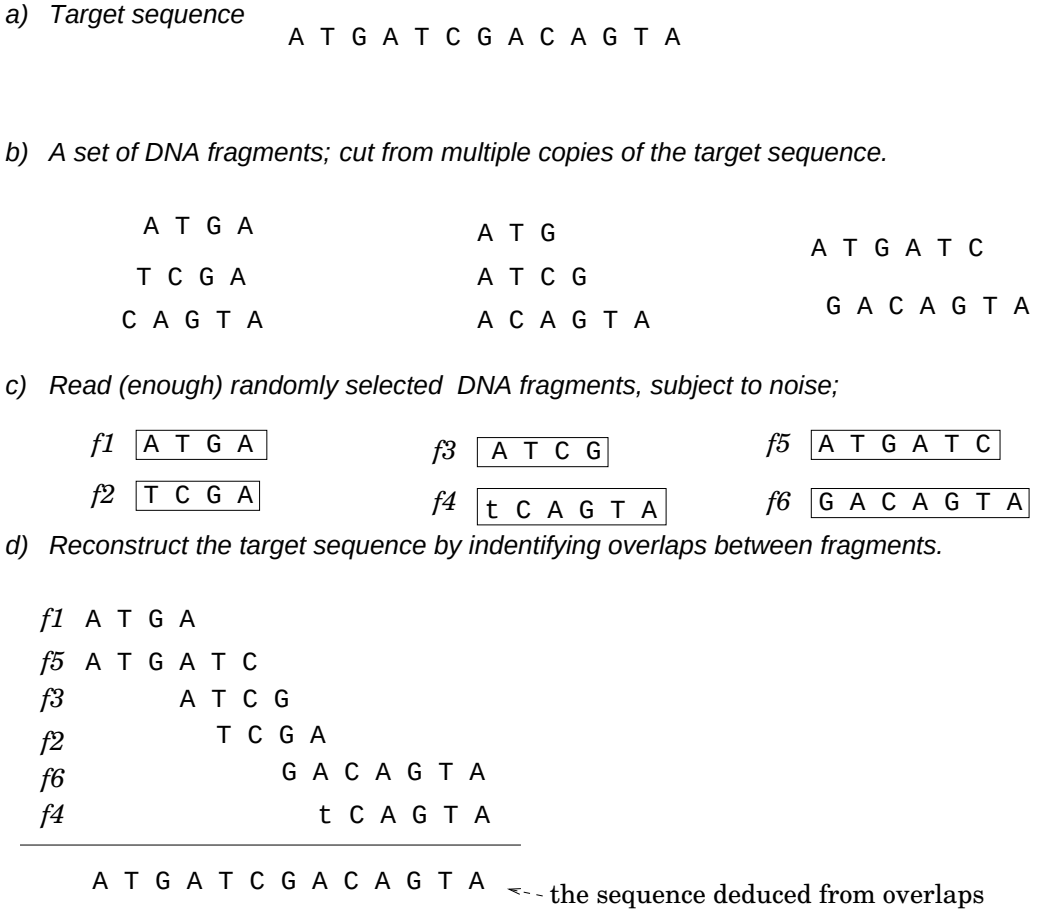

Figure 1: An illustration of a simple genome sequencing procedure. In case that there is no repetitive sequences in the target DNA, the genome can be sequenced by simply connecting overlapping fragments.

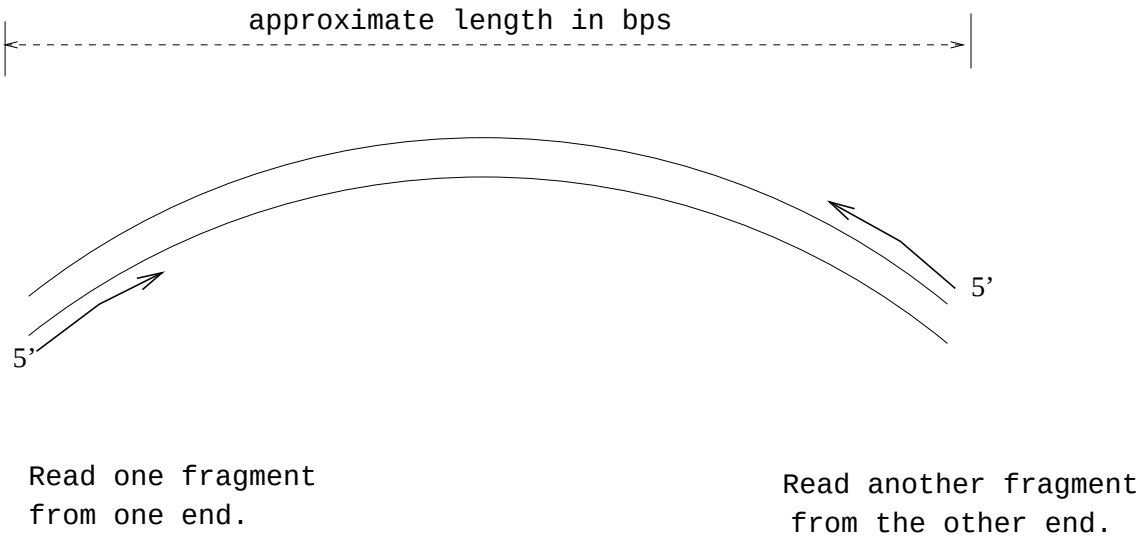

Figure 2: Clone mate information. Two fragments are read at both ends of the same clone.

Repetitive sequences and their effect in the sequence assembly In practice, there are extensive repetitive sequences, *repeats* in short, in a genomic sequence. A repeat is a collection of occurrences of the same or similar patterns in a sequence. Due to the nature of biological similarity, there is no formal definition of a repeat. However, the statistical theory of biological significance has been well developed; for example, see [18]. Thus, any patterns at different positions of a sequence, whose similarities are biologically significant, are defined as occurrences of the same repeat. A *repeat segment* is an occurrence of a repeat. The *copy number* of a repeat denotes how many repeat segments of a repeat exist in a target sequence. Repeats can easily mislead to false overlap detection since sequence assembly relies on the existence of overlapping subsequences among fragments. Two fragments that come from two different copies of the same repeat cannot be distinguished from two truly overlapping fragments that come from the same repeat copy. A recent paper [3] discusses the effect of repeats in the sequence assembly (Figure 3 in the paper).

Quality values and clone mate information In case that copies of repeats are similar but not identical, difference in base calls[1] can used to detect repeats. The most widely used sequence assembly program called Phrap relies on this apporach to discriminate fragments from different repeat copies. Base calls may not be correct and Phred generates numeric values to denote the confidence level of each base call. The quality value of a base is $q = -10 \times log_{10}(p)$ where p is the estimated error probability for the base [7]. Another important technique is to sequence both ends of a clone, generating two fragment reads per clone as shown in Figure 2. Since the insert size of clone is known, we know the approximate distance between two fragments. This technique is developed by Edwards and the colleagues [**?**] and is often known as *double barreled sequencing*.

[1]The DNA characters in a fragment are determined from *chromatogram* that can be viewed as a plot that shows possibilities of each of four DNA characters. The base call is a DNA character that determined from chromatogram and this process is done automatically by a computer program. The most widely used one is Phred [7].

The overlap-layout-consensus approach The most widely used "overlap-layout-consensus" approach to sequence assembly consists of three major steps: (1) the identification of candidate overlaps, (2) the fragment layout generation, and (3) the consensus sequence generation from the layout. The first step is achieved using string pattern matching techniques, generating possible overlaps between fragments. The second and third steps involve building models, implicit or explicit, for computing the layout of fragments and generating consensus sequences by enumerating the search space based on the model. Many successful sequence assembly algorithms have been developed based on this paradigm [21, 36, 13, 14, 15, 48, 44, 3]. There are also other approaches based on graph theory [17, 34]. These sequence assemblers contributed to the determination of many genome sequences, including the most recent announcement of the Human genome [48, 28]. However, sequence assemblers typically generate a large number of *contigs* rather than single contiguous sequences, due to repetitive sequences and technical difficulties encountered at different stages of a genome sequencing project. For example, the four most widely used assemblers generate 149 to more than 300 contigs for the *N. meningitidis* genome of 2.18Mb [34]. The complete determination of the target sequence from the set of contigs requires a significant amount of work, which is called *the post-assembly procedure*.[2]

The post-assembly procedure: The post-assembly procedure is labor intensive and constitutes a major bottleneck in the genome sequencing project. Two main problems are the verification of contig assemblies and the generation of scaffold of contigs. The verification of contig assemblies, despite its importance, has not been well studied and only a few computational approaches are known [24, 23, 42]. When fragments are read, two end sequences of the same clone are usually read and this *clone mate information* is valuable in generating scaffolds of contigs. Scaffolding contigs can be embedded in assemblers [44], or implemented as a separate package to order contigs [48] or BAC clones [20]. These scaffolding packages were successful in ordering a large number of contigs or BAC clones for large scale genome sequencing projects [20, 48]. However, the generation of the correct scaffolds of contigs is still quite challenging and more reliable algorithms need to be developed to handle a higher level of errors in contig assembly. Due to the difficulties, there are two different views in pursuing genome level sequencing, one in favor of the whole genome shotgun sequencing [50] and another in favor of smaller scale shotgun sequencing [32]. Figure 3 illustrates a sample genome sequencing project. In summary, there should be more efforts in developing frameworks for genome sequencing as well as component tools such as scalable, reliable sequence assemblers and contig assembly validation methods.

3 Computational Techniques for Genome Sequencing

3.1 Genome sequencing based on shotgun strategy

The number of successfully sequenced genomes based on shotgun strategy increases since the first successfully whole baterial genome, *H. influenzae*, was sequenced at TIGR in 1995 using the whole genome shotgun strategy [9]. However, there is also competing opinion advocating the BAC-size (about 100kb) shotgun sequencing [51, 52, 53]. Thus it is not known what is the appropriate size of the target DNA when the shotgun strategy is employed. In general, we can hardly expect to assemble single contiguous sequences by assembling the shotgun data. Instead, a large number

[2] A recent paper surveys issues in genome sequencing [38].

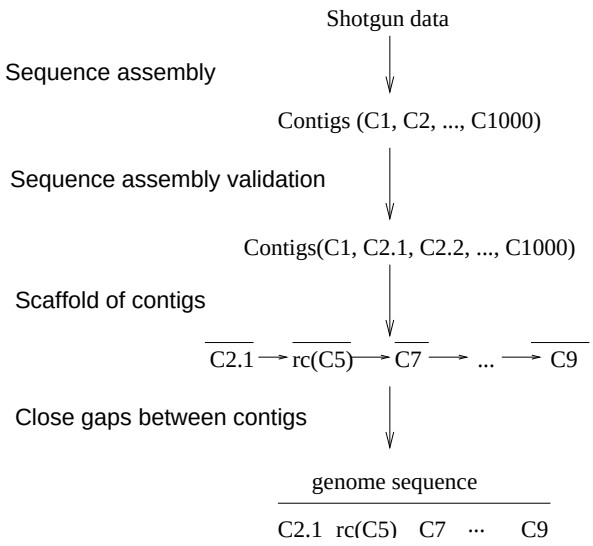

Figure 3: An illustration of a genome sequencing project. The shotgun data is assembled into 1000 contigs. Among them, C2 is found to be assembled incorrectly in the middle and separated into two contigs, C2.1 and C2.2. The orientation and ordering of 1001 contigs are determined. The orientation of C5 is determined in the reverse complement direction, denoted by rc(C5). With additional sequencing to close gaps between contigs in the scaffold, a single contiguous sequence is determined.

of contigs are generated and there are contigs that are misassembled probably due to repetitive sequences in the target DNA. As a result, genome sequencing is usually carried out in multiple steps and, unfortunately, there is no consensus on the steps of a genome sequencing project. We describe a general procedure for genome sequencing in the below, but we emphasize that procedures used at genome sequencing centers differ in detail.

1. **Reading fragment reads:** The sequence of each fragment are determined using an automatic base calling software. Phred [7, 8] is the most widely used program.

2. **Trimming vector sequences:** Sequence reads in the shotgun data contains part of the vector sequences that have to be removed before sequence assembly.

3. **Trimming low quality sequences:** Sequence reads contain poor quality base calls and removing or masking out these low quality base calls often leads to more accurate sequence assembly. however, this step is optional and some sequencing centers do not mask out low qualty base calls, relying on the sequence assmebler to utilize quality values to decide true fragment overlaps.

4. **Sequence assembly:** The shotgun data is input to a sequence assembler that automatically generates a set of aligned fragment called *contigs*.

5. **Sequence assembly validation:** Some contigs that assembled in the previous steps may be misassembled due to repeats. Since we do not have *a priori* knowledge on repeats in the target DNA, it is very difficult to verify the correctness of assembly of each contig and

this step is largely done manually. There are recent algorithmic development on automatic verification of contig assemblies (see Section 5).

6. **Scaffolding contigs:** Contigs need to be oriented and ordered. Clone mate information is a primary information source for this step, thus this step is not achievable if the input shotgun data is not prepared by reading both ends of clones (see Section 6 for more detail).

7. **Gap closing:** Assuming that all contigs are assembled correctly and contigs are oriented and ordered correctly, we can close gaps between two contigs by sequencing specific regions that correspond to the positions of gaps.

4 Survey of sequence assembly programs

4.1 Formulation As Shortest Common String Problem

One attempt to solve the shotgun sequencing problem is to formulate it as an instance of the "shortest common string problem" [45, 46, 49]. Such a formulation assumes that fragments are error free and fragment orientation is known. There is interesting theoretical work on the assembly result given by a greedy algorithm for such formulation of the sequence assembly problem. For example, [**?**] proves that the greedy algorithm delivers a superstring at most four times longer than the shortest superstring. Even though the shortest common substring problem has been theoretically well studied, there are two serious problems with this formulation. First, it is questionable whether the notion of "shortest" models the target sequence and the shotgun data. In particular, how to model repetitive sequences in this formulation has not been well studied. Second, we expect gaps in shotgun data for long sequences. The existence of gaps is problematic for the formulation, since there may be "weak" overlaps between non overlapping fragments either of which is followed by a gap in the shotgun data. Here, joining two fragments will result in contracting the target sequence or switching subsequences. Thus, approaches based on the shortest common superstring formulation are unlikely to be successful in assembling long sequences of real DNA.

4.2 CAP

CAP [13] operates in three phases: fragment overlap detection, contig formation, and consensus sequence generation.

4.2.1 Fragment Overlap Detection

CAP computes overlaps between every pair of input fragments and classifies the identified overlap into one of four overlap types: two containment overlaps and two non-containment overlaps (see Figure 4). A modified Smith-Waterman algorithm [41] is used to compute the best overlap score between a given pair of fragments. Since the overlap computation between every pair of input fragments is often computationally infeasible, an approximate string matching algorithm from Chang and Lawler [**?**] is used to speed up the overlap map construction; the Chang and Lawler algorithm is used to check if one fragment could possibly overlap with another fragment or be contained in another fragment.

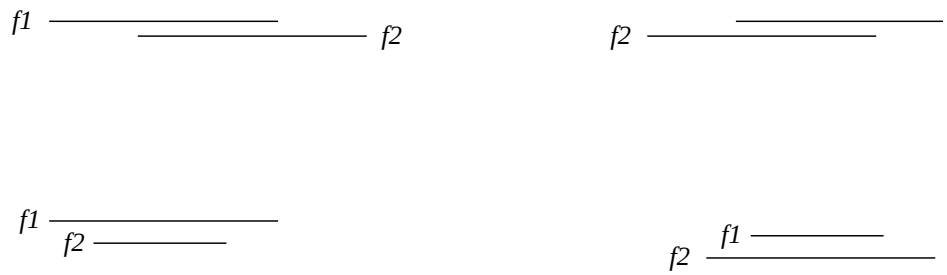

Figure 4: Four types of alignments of two fragments

4.2.2 Contig Formation

Contigs are assembled in a greedy fashion, by adding the best overlapping fragment (from the fragment overlap detection phase) one at a time. The overlap score is the similarity of the overlapping alignment between two fragments; the length of overlaps are not considered in computing the overlap score.

Containment trees are computed first to reduce the complexity of the assembly task. A containment tree is a tree whose the root is a noncontained fragment and a fragment $f1$ is a child of a fragment $f2$ if the best alignment of $f1$ and $f2$ is that $f2$ contains in $f1$. Then, noncontainment fragments are used for the greedy contig assembly.

4.2.3 Consensus Sequence Generation

To generate consensus sequences from the contigs, CAP merges each pairwise alignment into a multiple alignment of fragments in a similar fashion to [19]. A multiple alignment of fragments for a contig is generated one section at a time, where each section consists of a fixed number of nucleotides, say 60. A working set of fragments that appears in the current working section is maintained. As a section finishes, fragments which end within the section are removed from the working set of fragments and fragments which start within the section are added to the working set of fragments.

4.3 Kececioglu and Myers

Kececioglu and Myers approach [19] is best summarized as follows:

1. Construct a graph of approximate overlaps between every pair of fragments.
2. Assign an orientation to each fragment, i.e., choose the forward or reverse complement sequence for each fragment.
3. Select a set of overlaps that induces a consistent layout of the oriented fragments.
4. Merge the selected overlaps into a multiple sequence alignment, and vote on a consensus.

Each of the four phases can be viewed as a separate problem. Unfortunately, the last three problems are NP-complete and are handled by heuristic measures in the Kececioglu and Myers approach .

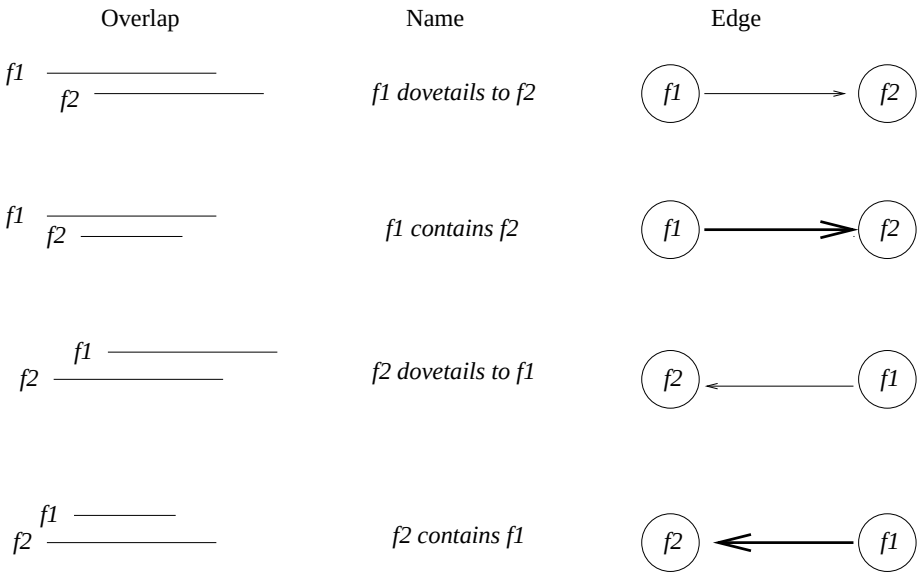

Figure 5: Four possible overlaps between fragments $f1$ and $f2$

4.3.1 Overlap Graph Construction

There are four possible overlap types between two fragments $f1$ and $f2$ (see Figure 5). Fragment $f1$ *dovetails to* fragment $f2$ when a suffix of $f1$ aligns to a prefix $f2$.[3] $f1$ *contains* $f2$ when $f2$ is a substring of $f1$.[4] There are $O(mn)$ dovetails and $O(m^2)$ containments for a given pair $f1, f2$ of sequences of length m and n.

An overlap graph $G = (V, E, w)$ represents fragments with vertex set V, and overlaps with edge set E. An edge weight function w gives the plausibility metric for an overlap. An *unoriented overlap graph* V contains two vertices for every fragment F. The two vertices are one for the original fragment, and one for the reverse complement of the fragment. Figure 6 shows an overlap graph.

To choose which of the possible overlaps actually appears between each vertex pairs in the overlap graph, Kececioglu and Myers choose the maximum likelihood overlap among all possible overlaps based on the edit distance [5] and the probabilistic metric from Chvatal and Sankoff [**?**]. The probabilistic metric is computed as follows: given an overlap between two fragments, the overlap of length $l + d$ is a string containing l exact matches(substrings that both fragments agree) and d mismatches. Chvatal and Sankoff compute the probability that a string of length $l + d$ which contains the *fixed* substring of length l occurs.

Finally, Kececioglu and Myers cull overlap edges since most of the edges represent chance alignment rather than true overlaps. There are two criteria for culling overlaps; (1) match significance, and (2) error distribution. An overlap with match probability less than *match significance threshold* is rejected. Assuming a binomial distribution in the error, they also reject some edges by *error*

[3]Note that a suffix of $f1$ may not match exactly with a prefix of $f2$ due to the reading error.

[4]Again the containment does not imply $f2$ is a exact substring of $f1$ due to the error in reading.

[5]We say informally that a string $S1$ is d edit distance from a string $S2$ when $S1$ can be transformed to $S2$ by d operations which consist of either inserting a character, deleting a character, or replacing a character by another character.

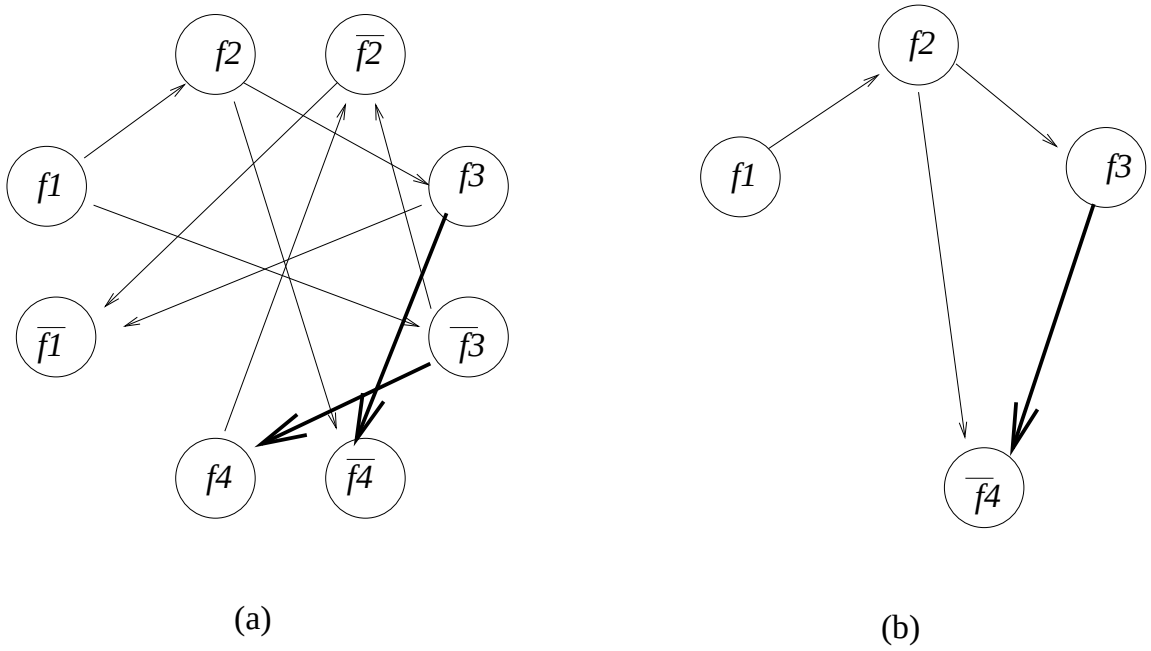

Figure 6: Overlap graphs. (a) unoriented graph. (b) oriented graph

distribution threshold. Roughly speaking, an overlap with highly clustered errors is rejected by a probabilistic argument.

4.3.2 Fragment Orientation

Once an overlap graph has been constructed, the next step is to determine the orientation of each fragment. Let G be an unoriented overlap graph with weight function w, and $\overline{f1}$ be a reverse complement of a fragment $f1$. Given two fragments $f1$ and $f2$, there are four possible overlaps; $(f1, f2), (f1, \overline{f2}), (\overline{f1}, f2)$ and $(\overline{f1}, \overline{f2})$. The basic idea is to determine one edge among the four possible edges between fragments $f1$ and $f2$ so that G has the maximum overlap score in total. Before defining the orientation problem formally, let's define some notation. Let F be a set of all input fragments, O a subset of F containing fragments in one direction, and $\overline{O}$ be $F - O$ containing fragments in the other direction. For two fragments $f1$ and $f2$, let

$$same(f1, f2) = max(w(f1, f2), w(f2, f1))$$

and

$$opp(f1, f2) = max(w(f1, \overline{f2}), w(\overline{f2}, f1)).$$

Then the fragment orientation problem is defined as follow.

$$maximize \quad w(O) = \sum_{(f1,f2)\in(O,\overline{O})} opp(f1, f2) + \sum_{(f1,f2)\notin(O,\overline{O})} same(f1, f2)$$

This problem is NP-complete, shown by the reduction from the maximum weight problem [19]. Thus Kececioglu and Myers proposed a greedy approximation algorithm that works as follows.

Given an ordering $F_1, F_2, ..., F_n$ of F, and an initially empty set O, at each step i, consider adding F_i to O where $(O, \overline{O})$ currently partitions $\{F_1, ..., F_{i-1}\}$. If $w(O \cup \{F_i\}) \leq w(\overline{O} \cup \{F_i\})$, then $O = O \cup \{F_i\}$. Otherwise, O is unchanged, and F_i is added to $\overline{O}$. Kececioglu and Myers proved that the greedy heuristic finds an orientation of weight at least $\frac{1}{2}w(O^*)$ where O^* is an optimal solution, for *any* ordering of fragments. Kececioglu and Myers also proposed a fragment ordering heuristic.

4.3.3 Fragment Layout

After the fragment orientation phase, the overlap graph G is an oriented graph. In the fragment layout phase, a layout of fragment overlaps is generated by traversing the oriented graph according to the four constraints given below.

1. Every vertex in G has at most one incoming edge.
2. The edges do not form cycles.
3. No two dovetail edges leave the same vertex.
4. No containment edge $f1 \Rightarrow f2$ is followed by a dovetail edge $f2 \rightarrow f3$

A set of edges that satisfies constraints 1 and 2 is called a *branching*. A branching can be seen as a collection of trees. A set of edges that also satisfies constraints 3 and 4 is called *a dovetail-chain branching*. Its dovetail edges form disjoint chains that originate at the root. Let IN be a set of edges which should be included in the dovetail chain branching, and OUT be a set of edges which should be excluded in the dovetail chain branching. The fragment layout phase repeats the following two steps.

1. Given a graph G, generate the maximum weight branching β, satisfying constraints 1 and 2.
2. If β is a dovetail-chain branching, return its weight. Otherwise, identify a pair of edges (e, f) violating constraints 3 and 4, and generate two subgraphs; G_1 with IN and $OUT = OUT \cup \{e\}$ and G_2 with $IN = IN \cup \{e\}$ and $OUT = OUT \cup \{f\}$. Solve both G_1 and G_2 and select the dovetail chain branching with higher weight.

Algorithms for generating a branching in order of decreasing weight can be found in the literature [**?**, **?**]. Thus a solution to the maximum dovetail branching problem can be found by repeating the generation of the maximum weight branching and then generating two subproblems by removing a pair of edges violating constraints 3 and 4.

Obviously, it grows exponentially. Kececioglu and Myers presented two techniques to reduce search space.

4.3.4 Multiple Sequence Alignment

Now the final step is to determine the DNA sequence from the layout of fragment overlaps. Note that from the fragment layout phase, we know how *each pair* of fragments overlap if they do. However, simply merging all pairwise overlaps does not lead to a consistent sequence of nucleotides as shown in Figure 7. The graph (b) corresponding to the graph (a) shows that there is a cycle.

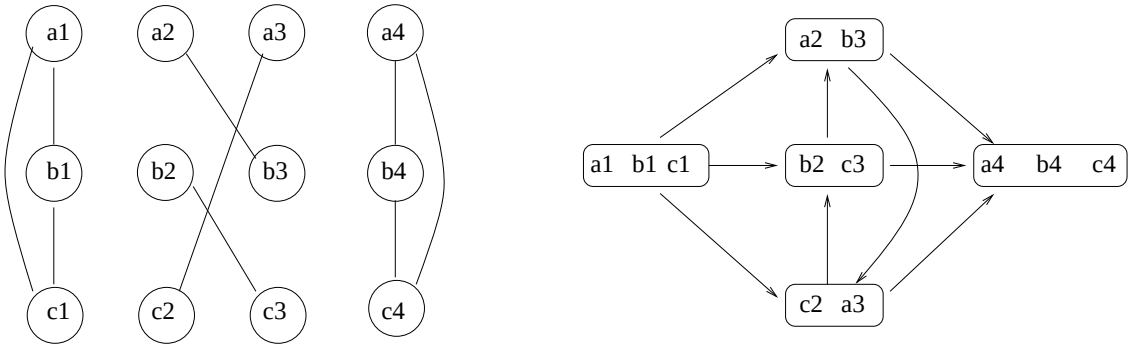

Figure 7: Pairwise alignments may not form a multiple alignment. (a) Three pairwise alignments of $a_1a_2a_3a_4$, $b_1b_2b_3b_4$, and $c_1c_2c_3c_4$. Edges join matched characters. (b) The connected components contains a cycle. A directed edge from v_1 to v_2 is constructed if any character in v_1 precedes any character in v_2.

A graph like (a) is called *trace* when the corresponding graph of connected components like (b) does *not* have a cycle. A trace gives us a sequence of characters. Thus the multiple alignment phase determines a trace first and then generates a sequence by topologically sorting connected components in the trace.

The main problem is how to find the best trace from a graph of characters. The basic idea is that any pairwise alignment that is already determined forms a trace. In Figure 7, any two sequences form a trace, although all three sequences do not. Given the pairwise alignments, compute a *supergraph* whose *supernodes* are the sequences to be aligned and whose *superedges* are defined by pairwise alignments. A superedge is weighted by a similarity metric which is the sum of similarities between characters in two sequences.[6] Now the goal is to find a *tree of pairwise traces* with the maximum weight for a supergraph. The heuristic proposed by Kececioglu and Myers is:

1. Compute a closure between pairwise traces and add them to the superedges.
2. Compute the maximum weight spanning tree.

From the observation that errors are clustered at the ends of a fragment, Kececioglu and Myers proposed a variant of the heuristic based on a *window* containing a subset sequence. In this variant, supernodes become the subset sequences in the window, and searching for the maximum weight spanning tree generates a column of characters.[7] The application of the observation is the use of a larger window in the middle of a sequence and of a smaller window at the end. The detailed description of the variant of the heuristic can be found in [19].

4.4 Idury and Waterman

Idury and Waterman [17] proposed a totally new way to assemble fragment data by combining *sequencing by hybridization* [5, 35] and shotgun sequencing. Before describing their new approach, the hybridization technique and sequencing by hybridization are described.

[6]The *similarity* between two character sets X and Y is defined by $|X \cap Y|/|X \cup Y|$. At first, X and Y are singletons, but they become non-singleton sets as sequences are added to the alignment.

[7]A column of characters becomes a character in the final sequences determined by voting between fragments to remove ambiguity.

4.4.1 Hybridization Technique

Hybridization technique is to prepare probes(k-tuples of nucleotides) and apply them to the set of DNA pieces. The probes hybridize (or bind) with any fragment having the reverse complement nucleotide sequences. The existence of hybridized probes can be identified with a radioactive or fluorescent material. The hybridization data is then used to construct a map so that positional information among fragments can be obtained.[8] The map is used for determining sequences of fragments or identifying genes, etc.

The major problems here is that (1) hybridization data may contain errors, and (2) we do not know how many occurrences of a probe or in what order they occur when there are multiple occurrences of probes. Thus it is often not possible to determine a complete hybridization map. Determination of a hybridization map is another research topic in computational biology(see [**?**]).

4.4.2 Sequencing By Hybridization

Biologists came up with an approach to determine the DNA sequence by hybridization(SBH) [5] and Pevzner [35] further developed the approach. This section will describe Pevzner's idea. The approach consists of three major steps.

1. Prepare a collection of all the instances of k-tuple probes, called a *sequencing chip*.
2. Apply all of the sequencing chip to each fragment being sequenced.
3. Construct a graph from the resulting hybridization data and determine the sequence.

The example of 3-tuple sequencing chip and its application to a sequence $\mathbf{a}$ = ATGTGCCGCA is presented in Figure 8. The reason for preparing *all* instances is that we do not know the original sequence so there is no way to tell in advance which subset of a sequencing chip will be useful. Obviously it is advantageous to prepare larger sequencing chips, but there is a technical limitation since to prepare a sequencing chip of size k means to generate 4^k biological probes. (The current technology can accommodate up to the $k = 8$.)

The next step is to construct a graph using the hybridization data. Let's assume that we used a sequencing chip of size k. In the graph, each probe becomes a node, and an edge from a node N_1 to N_2 is constructed if a suffix of N_1 of size $k-1$ matches exactly a prefix of N_2 of size $k-1$. Returning to the example in Figure 8, the sequence $\mathbf{a}$ contains instances of probes ATG, TGC, GCA, CAG, AGG, GGT, GTC, TCC. There is a directed edge from ATG to TGC since the suffix TG of ATG is equal to the prefix TG of TGC.

After constructing a graph, the sequence is determined by finding a Hamiltonian path through the graph. Since constructing a Hamiltonian path is NP-complete, Pevzner came up with an alternative formulation of the graph. In the new graph G, each $k-1$ tuple becomes a node, and an edge is set from node N_1 to N_2 when there is an instance of a probe whose prefix of size $k-1$ is N_1 and whose suffix of size $k-1$ is N_2. With the new graph, the sequence can be determined by a finding Eulerian tour, for which a simple linear time algorithm is known (Figure 9 shows a Hamiltonian graph H and an Eulerian Graph G for the sequence ATGTGCCGCA). Another advantage of the Eulerian graph approach is that there might be cases where the Eulerian tour

[8] Thus we do not determine the DNA sequence by hybridization data.

AAA	ACA	AGA	ATA
AAC	ACC	AGC	ATC
AAG	ACG	AGG	**ATG**
AAT	ACT	AGT	ATT
CAA	CCA	CGA	CTA
CAC	CCC	**CGC**	CTC
CAG	**CCG**	CGG	CTG
CAT	CCT	CGT	CTT
GAA	**GCA**	GGA	GTA
GAC	**GCC**	GGC	GTC
GAG	GCG	GGG	**GTG**
GAT	GCT	GGT	GTT
TAA	TCA	TGA	TTA
TAC	TCC	**TGC**	TTC
TAG	TCG	TGG	TTG
TAT	TCT	**TGT**	TTT

Figure 8: Sequencing chip and hybridization result

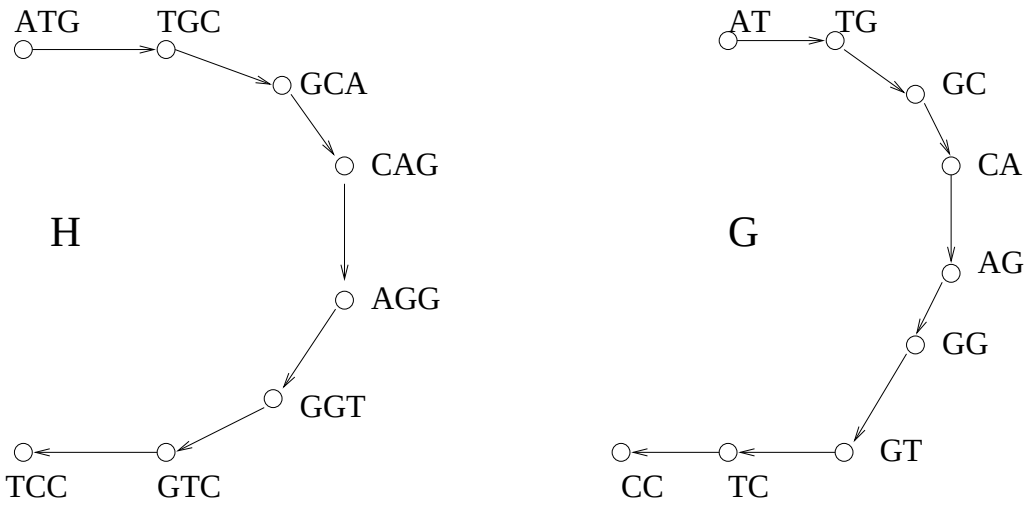

Figure 9: Graphs H and G for ATGCAGGTCC

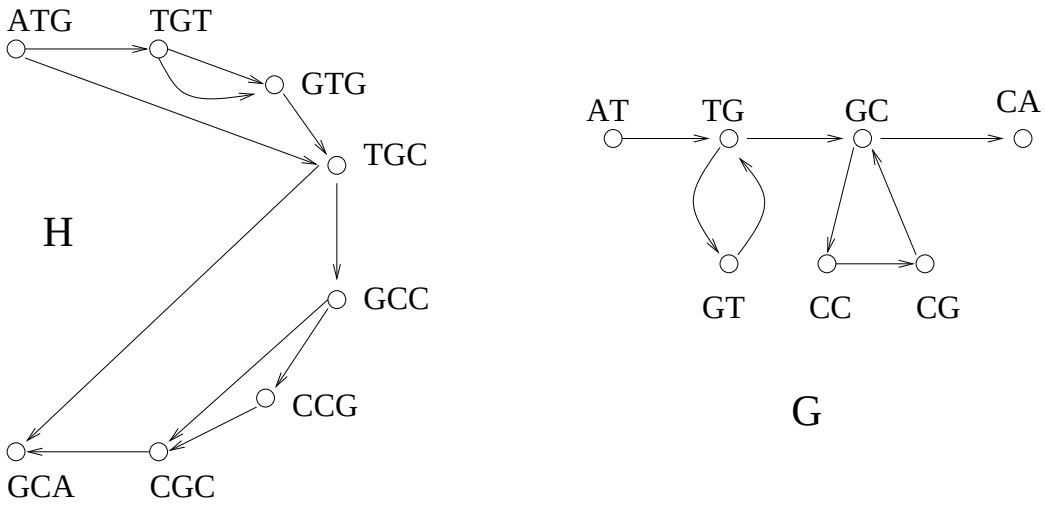

Figure 10: Graphs H and G for ATGTGCCGCA

determines one sequence, but Hamiltonian tour cannot determine a sequence due to the existence of the multiple Hamiltonian tours (Figure 10 shows such an example).

4.4.3 Idury and Waterman's Approach

As explained before, SBH is limited by the size of the sequencing chip and also by the possibility of errors in the hybridization data. Idury and Waterman [17] proposed an approach combining SBH and shotgun sequencing that sidesteps the first of these problems.

4.4.4 Overview of Idury and Waterman's Approach

The basic idea is to use artificial probes generated from shotgun sequencing data, instead of biological probes. The advantage is that there is now no restriction on the size of a probe since a probe comes from a sequence which is already determined by shotgun sequencing. However, shotgun sequencing data still contains errors, so that occurrences of a probe may depend on the presence of errors. Idury and Waterman argue that the error rate is not that high so there is a sufficient number of probe occurrences even in the presence of errors. For example, if there are 2% errors, on the average there will be $50 - k$ correct tuples in every 50 nucleotide sequence [**?**]. Idury and Waterman rely on the fact that there is multiple coverage to sidestep these errors.

The algorithm is in Figure 11. In the algorithm, Idury and Waterman handle the unknown orientation problem by simply enlarging the input fragment set by a factor of 2 by including all reverse-complemented fragments f_i^r. Step 1 is to convert a fragment into a sequence of occurrences of probes of size k. The start and end of each probe occurrence is marked by i_α and j_α. In step 2, an edge is weighted according to the number of occurrences. Step 3 is to construct *super edges* to reduce the size of the graph (graph reduction technique will be explained later in detail). In step 4, the greedy Eulerian tour begins with the heaviest edge(s). The example in Figure 12 shows the construction of the sequence.

input: N(coverage), k(probe size), $f_1, f_2, ..., f_n$(fragments)
output: sequence assembly

1. Convert $f_1, f_2, ..., f_n$ $f_1^r, f_2^r, ..., f_n^r$ into $(w_\alpha, i_\alpha, j_\alpha)$ for all α occurrences of w, when $|w| = k$.
2. Construct the Euler graph on $(k-1)$-tuples for the k-tuples from (1). Each edge w has $\alpha = 1$ to $n(w)$ pairs of (fragment=i_α, position=j_α).
3. Reduce the Graph.
4. Perform greedy Eulerian tour(s)
5. Align fragments to sequence produced by (4).

Figure 11: Idury and Waterman's algorithm

ATG	(3,1)		
TGT	(3,2)	(4,1)	
GTG	(1,1)	(3,3)	(4,2)
TGC	(1,2)	(4,3)	
GCC	(1,3)	(2,1)	
CCG	(1,4)	(2,2)	
CGC	(2,3)		
GCA	(2,4)		

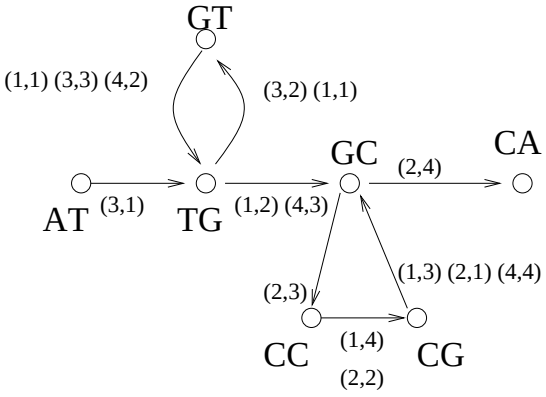

Figure 12: An Eulerian graph for ATGTGCCGCA; an Eulerian tour is constructed, starting with the heaviest edges and going backwards and forwards.

4.4.5 Graph Reduction and Repeating Sequences

There are three types of reducible nodes according to their classification; *singletons, forks* and *crosses*. Before performing an Eulerian tour, Idury and Waterman reduce the graph by removing all edges of these three types.

Elimination of singletons A singleton v is a node having only one incoming edge (u, v) and only one outgoing edge (v, w). The reduction removes the node v and collapses two edges into one, (u, w). Let the tuples represented by edges, (u, v) and (v, w), be $a_1...a_p$ and $b_1...b_q$ respectively. Then the tuple represented by the edge (u, w) becomes $a_1...a_p, b_k...b_q$ since $a_{p-k+2}...a_p = b_1...b_{k-1}$. The new edge (v, w) is called *a super edge*. Idury and Waterman observed that 90 % of edges in the graph are singletons and the length of a super edge often is more than 50 nucleotide long.

Elimination of forks A fork is a node which has one incoming edge and multiple outgoing edges. For example, consider a case in which v has one incoming edge $e = (u, v)$ and two outgoing edges, $e1 = (v, w_1)$ and $e2 = (v, w_2)$. If we visit v once, then we will have to visit w_1 or w_2, but not both. Idury and Waterman remove forks by selecting the edge with the heaviest weight. Removing forks in turn creates new forks since some crosses become forks by elimination of existing forks.

Elimination of crosses A cross is a node with multiple incoming edges and multiple outgoing edges. Elimination of crosses is done in a fashion similar to the elimination of forks, that is, by selecting the heaviest incoming and outgoing edges. Note that elimination of crosses is performed only after elimination of singletons and forks; after elimination of crosses, the first two reductions are repeated if applicable.

Note that real DNA sequences contain repeating regions, that is, occurrences of the same pattern at different positions. The three graph reduction techniques just described cannot handle repeating regions.

4.5 Parsons, Forrest and Burks

Parsons, Forrest, and Burks [33] proposed an approach based on genetic algorithms. Since the sequence assembly problem is NP-complete, the application of genetic algorithms makes sense. Furthermore, as Parsons, Forrest and Burks argue, the basic idea underlying all genetic algorithms (to generate a better composite solution from good partial solutions) is well suited to the sequence assembly algorithm.

The basic principles of genetic algorithms can be found in many Artificial Intelligence text books and articles, e.g. [47].

4.5.1 Genetic Algorithms

A genetic algorithm starts with a random population, which is allowed to evolve over many generations. In other words, given a population, we apply genetic operators like crossover and mutation to increase the population, which is the next generation, and then select a fixed number of individuals in the population by a selection function called a *fitness function*. When a genetic algorithm technique is applied to a specific domain, the issues are as follows;

1. *The representation* of a specific problem in genetic algorithm terms is not trivial. Usually a set of binary bits is used to encode the suitable information.

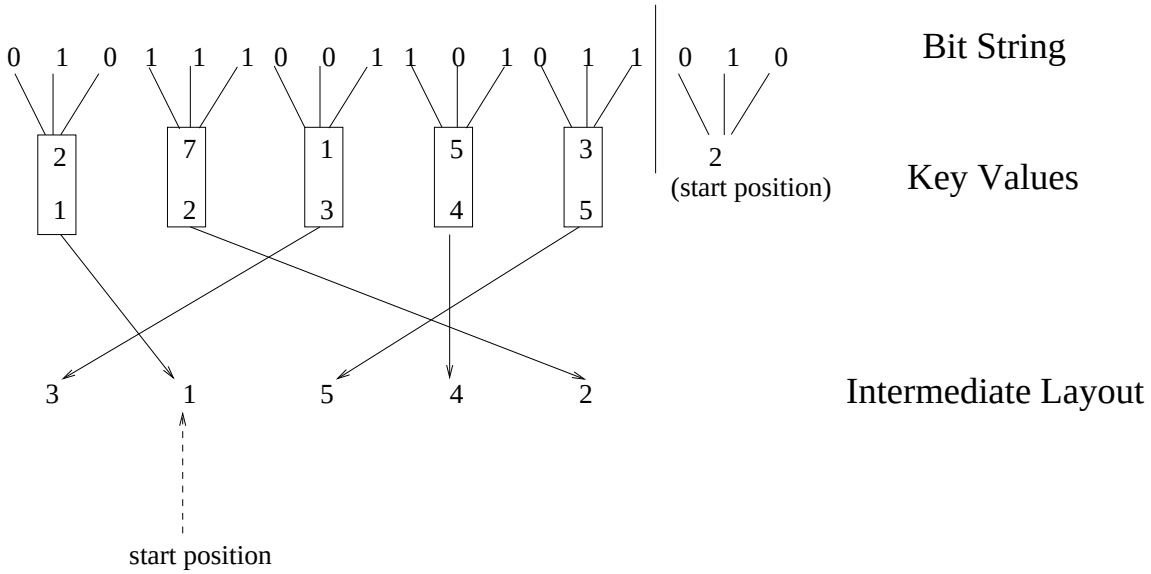

Figure 13: How to get a permutation ordering, 1 5 4 2 3.

2. *A fitness function* is the metric to evaluate the goodness of an individual; such a function is necessarily domain dependent.

3. *Genetic operators* are used to generate new individuals. In general, there are two kinds of operators, *crossover* and *mutation*. The crossover operator generates a new individual by exchanging substrings in parents. The purpose of the crossover operator is to allow partial solutions to evolve in different individuals and then combine them to produce, sometimes, a better solution. The mutation operator alters one individual by exchanging a primitive element of that individual(e.g., flipping one bit). The *rate* of what percentage of individuals is formed through an operator is one of the important parameters.

4.5.2 Representing the Sequence Assembly Problem

A permutation of integers represents a sequence of fragment numbers where successive fragments overlap if there is evidence for overlapping.[9] They used the *sorted-order representation* for the permutation ordering [43]. There are *two requisite properties for a legal ordering*: (1) all fragments should be present in the ordering and (2) there should be no duplication in the ordering. These properties are ensured through the use of a sort. Consider the example in Figure 13. In the example, each 3 successive bits represent a fragment number in the bit string. The bit string represents a sequence, 2 7 1 5 3. Each number is associated with another number (associate) representing its position in the ordering; The position of the number '2' is 1st, so '2' is associated with '1'. Then the sequence is sorted in ascending order, becoming a new sequence 1 2 3 5 7. The permutation is a sequence of associates, that is, 3 1 5 4 2. The final step is to read the sequence starting at the position 2 designated as a starting position by the last number in the original bit string. This reading gives us the permutation, 1 5 4 2 3.

[9]Parsons, Forrest and Burks did not explain in the paper how to compute an overlap score between fragments. We might compute it as in [19].

4.5.3 Fitness Function

The choice of a fitness function is crucial to the success of a genetic algorithm on a particular problem. Parsons, Forrest and Burks used two fitness functions; `F1` and `F2`.

Let $I = f[0], f[1], ...f[n-1]$ be an ordering of fragments, where $f[i] = j$ denotes that fragment j appears in position i of the ordering. The first fitness function `F1` is defined by

$$\texttt{F1}(I) = \sum_{i=0}^{n-2} w_{f[i],f[i+1]}$$

where $w_{i,j}$ is the pairwise overlap strength of fragment i and j. Note that this function examines only the overlap strength of directly adjacent fragments in the ordering. The optimization process finds a layout that maximize this function.

The second fitness function `F2` considers the overlap strength among all possible pairs, in which fragments with strong overlap are far apart.

$$\texttt{F2}(I) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |i-j| \times w_{f[i],f[j]}$$

4.5.4 Genetic Operators

Parsons, Forrest and Burks used four genetic operators: order crossover, edge recombination, inversion, and transposition. For the brevity of this survey paper, I will explain the operators briefly rather than at length. Detailed description of the operators can be found in [33].

- `Order crossover` selects two random points in the permutation ordering. Unlike traditional crossover operators, it copies a subsequence between the two points from a parent into the same positions in an offspring. Then the remaining sequence is filled by copying numbers from another parent, in accordance with the two requisite properties for a permutation ordering(see Section 6.2).

- `Edge recombination` is another crossover operator which attempts to preserve adjacencies in the parents. In particular, it preserves adjacencies that are common to both parents.

- `Inversion` is a mutation-like operator that reverses the order of the fragments in a contig. Parsons, Forrest and Burks argue that this operator is useful for handling unknown orientation problem.

- `Transposition` is another mutation-like operator. It moves a contig to a position between two adjacent contigs that may allow the extension of a contig.

An important property in the operators is that they try to preserve or extend existing contigs. Especially, the two mutation-like operators do not contract existing contigs but only try to extend them. The basic idea is to allow for the evolution of larger contigs by treating smaller contigs as basic building blocks.

4.6 TIGR Assembler

The TIGR Assembler [44] of Sutton *et al.* has successfully assembled very long sequences. The TIGR Assembler operates in two phases: pairwise comparison of fragments and fragment assembly. A consensus sequence is automatically determined as fragments are assembled (see Section 4.6.3). Handling repetitive sequences is discussed.

4.6.1 Pairwise Comparison

TIGR Assembler computes the best alignment of two fragments using a modified Smith-Waterman algorithm [41]. Since pairwise comparison of all input fragments is often impractical due to the quadratic complexity of the Smith-Waterman algorithm, they locates all patterns of length n bps and restricts pairwise overlap comparison only to pairs of fragments with the same patterns of n bps; thus any pair of fragments without common patterns of n bps are determined not to overlap.

4.6.2 Merging Fragments with Assemblies

After computing pairwise overlaps among input fragments, a fragment is merged with the current assembly if the match criteria are satisfied. There are four elements of match criteria: minimum length of overlap, minimum similarity in the overlap region as a percentage of the best possible score, maximum length of overhang, and maximum number of local errors. Overhang is the region in the alignment where two fragments do not match, and the length of overhang is the maximum number of overhang at both ends of the alignment. Local errors are the base pair mismatches between the fragment and the assembly in any given contiguous 32 bp in the overlap region. The maximum number of local errors is used to reject overlap with clustered errors, which might survive the similarity test. For example, 16 errors spread out evenly over a 400 bp overlap (96% similarity) is more likely to be a real overlap than 8 errors clustered in a 32bp window of 400 bp overlap (98 % similarity).

Note that, when a fragment is considered for assembly, match criteria are computed against the current assembly, thus the assembly doesn't rely on pairwise alignment, although selection of a fragment for assembly does.

4.6.3 Building a Consensus Sequence

When a fragment is added to the current assembly, TIGR does not commit to a consensus sequence but keeps a history what bases have been aligned to that position in the past. This results in a column or profile where a count of bases plus gaps is recorded. From this profile, a consensus sequence is generated after the assembly is completed, selecting the most frequent bases for consensus sequence.

4.6.4 Handling Repetitive Sequences

TIGR uses two strategies to handle repetitive sequences: increased match criteria for fragments possibly from repeats, and clone length constraints.

Fragments from repetitive sequences are identified by the number of potential overlaps each fragment has based on pairwise comparisons. Any fragment with the number of overlaps more than a threshold is labeled as being from repetitive sequences. When a repeat fragment is added

to the current assembly, the stringency of the match criteria is increased so that nonexact repeats (for example, 80% similarity for Alu repeats) can be distinguished.

Even with strict match criteria, it is impossible to avoid false overlap identifications when repeats are longer than the average fragment size. TIGR uses *clone length information* that maintains distance between the fragments that read from the same clone.

4.7 CAP2

CAP2 [14] is an extended version of CAP (see Section 4.2). Three major improvements are (1) performance improvement by filtering out potentially non-overlapping fragments, (2) identification of chimeric fragments, and (3) handling repetitive sequences.

4.7.1 Performance Improvement

CAP2 uses a modified Smith-Watermn algorithm [41] to test overlaps between a pair of fragments. Since applying the Smith-Waterman algorithm for every pair of fragments to examine possible overlaps is costly and often not practical in terms of computing time, CAP2 filters pairs of fragments that are unlikely to overlap by locating common patterns. First all common patterns of 9bps are searched among the input fragments, and then patterns of 9bps are extended to see if there are gap-free common patterns of 20bps with at most 2 substitutions. Any pair of fragments without such common patterns of 20bps are determined not to overlap without applying the expensive Smith-Waterman algorithm.

4.7.2 Identification of Chimeric Fragments

First of all, CAP2 computes an *error rate vector* for each input fragment when constructing pairwise overlap comparison. The error rate vector of a fragment is a list of the error rates of each regions of length $r$bps; the error rate $E_p(i)$ is defined as an estimated number of errors between i-th base position and $(i + r)$-th base position of a fragment f_p. Roughly speaking, the error rate is computed as the minimum difference while computing pairwise comparisons of fragments. The error rate vector is used to evaluate overlaps among fragments when constructing contigs and to identify chimeric fragments.

A chimeric fragment is a fragment read from a chimeric clone which is a joining of two or more clones. Thus, a chimeric fragment contains two or more non-contiguous segments of the target sequence. A position separated by non-contiguous segments of the target sequence is called a *separation point (sp)*. A fragment f_q is a chimeric fragment if it satisfies the following three conditions:

1. The maximum difference

$$max\{E_q(i - r/2) - E_q(i) \mid r < i < length(f_q) - r\}$$

   is greater than a threshold.

2. There is a fragment $f1$ (other than f_q) such that the overlapping region between $f1$ and f_q ends near the maximum difference position in condition (1). There also is a fragment $f2$ (other than f_q) such that the overlapping region between $f2$ and f_q starts near the maximum difference position in condition (1).

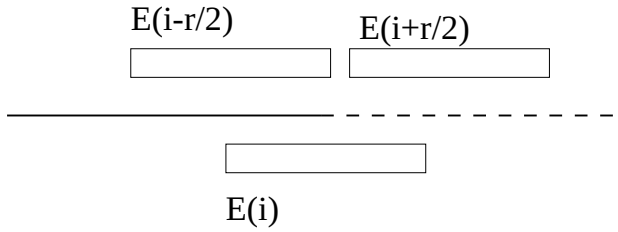

Figure 14: Difference in error rates in a chimeric fragment (where real and dotted lines denote two segments which come from different parts of the target DNA). Two error rates $E(i - r/2)$ and $E(i + r/2)$ are expected to be small since they are contiguous regions of the target DNA sequence, thus it is high likely that a fragment covers the entire region for the two error rate estimations. However, the error $E(i)$ is large since it contains two non-contiguous regions of the target DNA sequence, thus there is no fragment which covers the entire region for the error rate estimation.

3. No fragment overlapping with f_q crosses over the maximum difference position.

The maximum difference position in condition (1) can be illustrated in Figure 14.

4.7.3 Handling Repetitive Sequences

When constructing contigs, the best remaining overlap from pairwise comparison is selected in a greedy fashion. Thus, several contigs may be present at some point. The selection of an overlap between two fragments that belong to two different contigs results in merging the two contigs into one contig. When merging two different contigs, inconsistent overlaps in fragments lead to construction of *repetitive contigs* which are essential to identifying repeats.

Let's assume that the best remaining overlap is $f1 \rightarrow f2$, and $f1$ and $f2$ belong to two different contigs $C1$ and $C2$ repetitively. Then, a temporary contig that includes the current best overlap $f1 \rightarrow f2$ is constructed from the set of fragments in $C1$ and $C2$, by adding consistent overlaps. If the two contigs does not contain repeats, the temporary contig will include all fragment in $C1$ and $C2$, since all overlapping fragments in $C1$ and $C2$ are consistent. Otherwise, the temporary contig will contain only a subset of the set of fragments in $C1$ and $C2$. In that case, a *repetitive contig* is constructed from the temporary contig. Later, merging ordinary contigs and repetitive contigs is considered to complete the assembly of the target sequence.

4.8 AMASS

Kim and Segre [21] proposed a sequence assembly algorithm that uses exact matches of short patterns randomly selected from fragment data to identify fragment overlaps, construct an overlap map, and finally deliver a consensus sequence.

4.8.1 Overview of AMASS

AMASS starts with sampling patterns of fixed length from shotgun data, and then find all occurrences of sampled patterns in shotgun data. The occurrences of patterns are used to detect overlapping fragments and to handle repeats which is the major hurdle to sequence assembly. Two distributions, *the pattern distribution* and *the fragment distribution*, are used to identify fragments

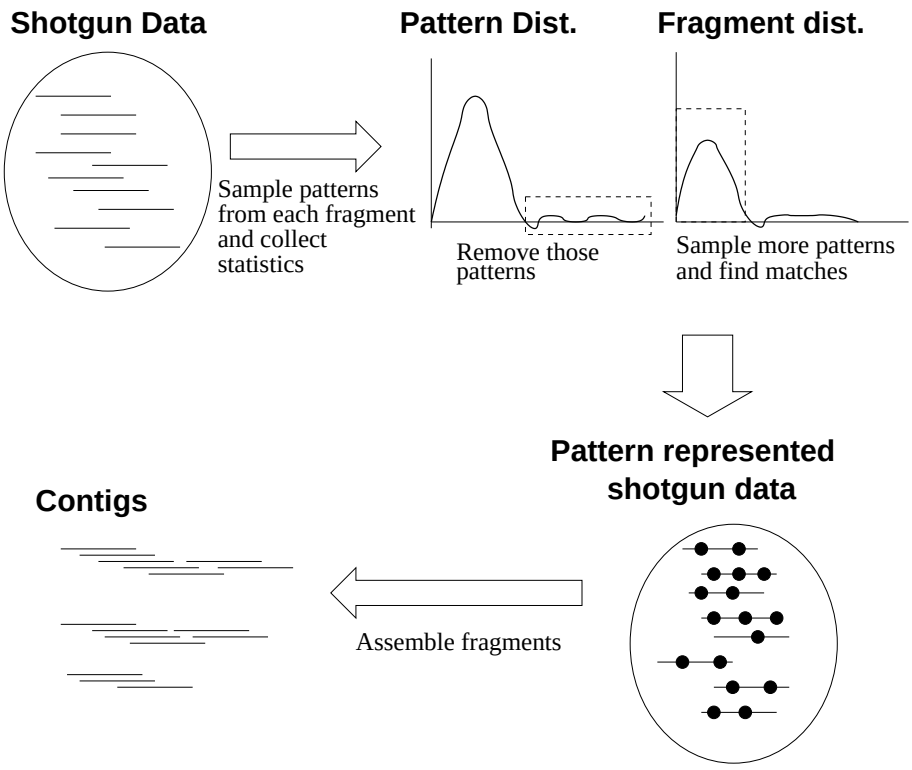

Figure 15: An overview of the AMASS sequence assembler.

that come from repeats. Then, assembly of fragments from non-repeat regions are assembly first and later contigs are extended to include repeat regions. Figure 15 illustrates how AMASS works.

4.8.2 Building contigs

AMASS represents a fragment as an ordered set of pattern occurrences (probes) and their interprobe spacings and fragment overlaps are determined by this representation. (see Figure 16). Contig is built in a greedy fashion, adding a fragment with the best score at a time as summarized below.

1. Build a pairwise overlap table.
2. Take best pair of unassigned fragments and build an initial contig.
3. Compute overlap score between the current contig and remaining fragments.
4. Selecting a fragment with the highest score, grow the current contig by adding the fragment.
5. Repeat Step 3 & 4 while overlap scores are still significant.
6. Goto Step 2 while unassigned fragments remain.

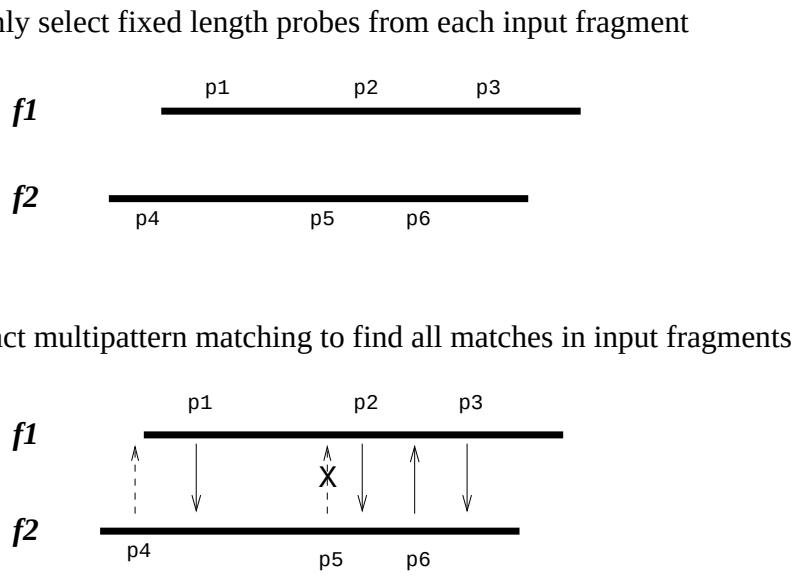

Figure 16: Fragments are represented as ordered sets of pattern occurrences (probes) and their interprobe spacings. Overlaps between fragments or between a fragment and a contig are determined by this representation.

4.8.3 Handling repeats

Repeats are handled in three different ways: (1) use of satellite matching, (2) the pattern distribution and (3) the fragment distribution. To handle very short repeats, AMASS employs a technique called satellite matching which tests for co-existence of very short patterns around a pair of common pattern (probe) occurrences. If probes are from two different repeat copies, their surrounding regions will be different for short repeats and the satellite matching test can identify probes from different repeat copies quickly and effectively. (see Figure 17) Any probe occurrences that fail for the satellite matching test will be discarded so that they are not used for contig construction.

Probe occurrences are further screened using the pattern distribution that shows the distribution of the number of probe occurrences v.s the number of probes. Any probe that occurs unusually many times are obviously from repeats and discarded. This technique is used by nearly all sequence assemblers to quickly filter out probes from repeats.

AMASS also attempts to identify fragments that comes from repeats using the fragment distribution that the distribution of the number of fragments v.s the number of probe occurrences in a fragment. Figure 18 shows a fragment distribution from *M. genitalium* shotgun data obtained from TIGR. The basic idea is that fragments from long repeats will exhibit a significant increase in the number of probe occurrences since probes from a repeat copy will occur in other repeat copies of the same repeat. AMASS marks fragments in the tail region as repeat fragments. Contigs are constructed for non-repeat regions using the set of fragments that are not marked as repeat fragments and repeat regions are expanded later whenever ambiguity in overlapping regions are acceptible.

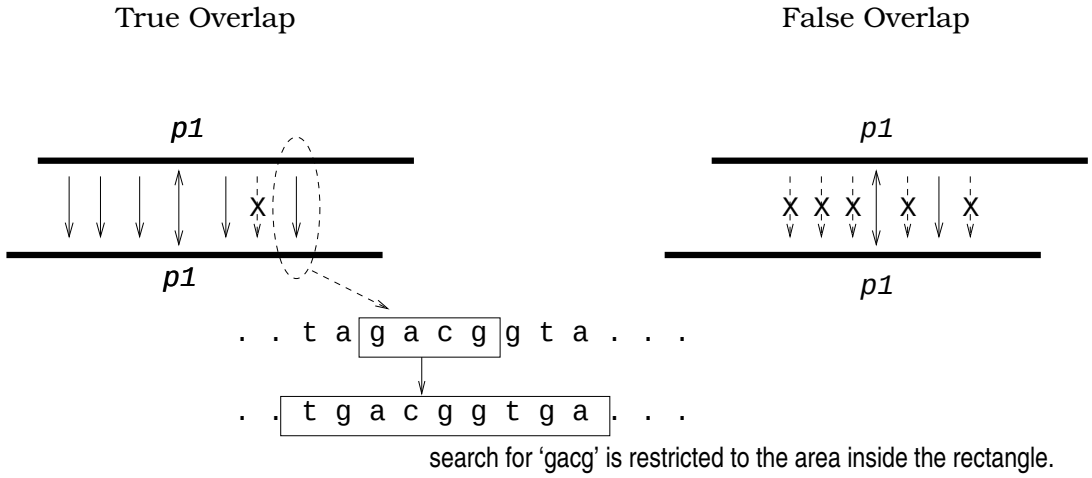

Figure 17: Given a pair of common probe occurrences, p1 in this case, the satellite matching test can identify probes from different repeat copies quickly and effectively.

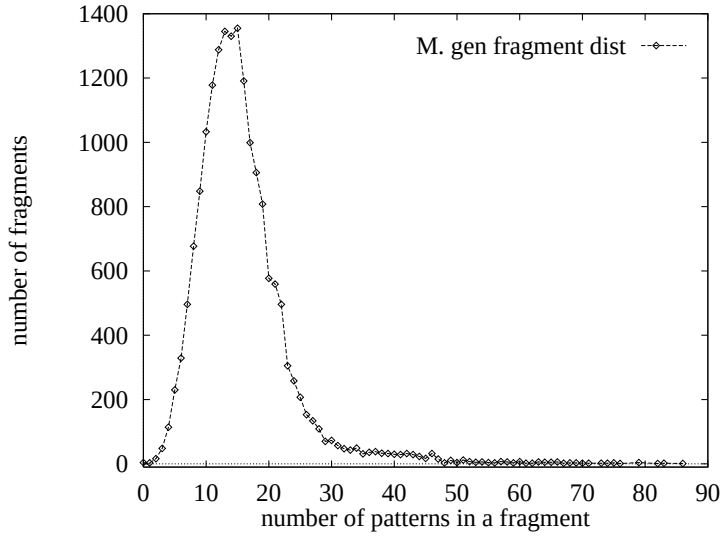

Figure 18: Fragments in the tail region are most likely come from repeats.

4.9 PHRAP

PHRAP is the most widely used sequence assembler. However, there is no published paper that described the algorithm for PHRAP. We will briefly described the algorithm based on the information on the web site at `http://www.phrap.org`.

1. A sorted list of all matching words of length at least `minmatch` between any pair of sequences is constructed.

2. For each such pair, a band of a specified width, centered on the diagonal defined by the matching words, is defined and then overlapping bands are merged. A "recursive" SWAT (an implementation of Smith-Waterman algorithm) search of each band is used to find matching segments with score greater than or equal to `minscore`. The recursive search of SWAT, which is intended to find all matches between two matches, is done by repeating masking out the current matched region and performing SWAT.

3. A match is associated with LLR score, which is a log-likelihood ratio for comparing the hypothesis that the reads truly overlap to the hypothesis that they are from 95repeats. The point is that discrepancies between overlapping reads are due to base-calling errors and thus tend to occur in low-quality bases, whereas reads from different repeats can have high-quality discrepancies that are due to sequence differences between the repeats; and the probability of the observed data under each hypothesis can be quantified using the interpretation of the phrap qualities in terms of error probabilities. A pairwise match tends to have a positive LLR score if the two reads overlap, whereas it tends to have a negative LLR score if the two reads are from different repeats (unless the repeats are nearly identical).

4. Sort all matching pairs by decreasing LLR score, and assemble layout by progressively merging pairs with high score.

5. Construct contig "consensus" as a mosaic of individual reads. This is done by building a weighted graph using selected positions of matches as vertices (ends of alignments, and midpoints of perfectly matching segments of sufficient length). Unidirectional edges from 5' to 3' positions within a single read, with weight equal to the total quality of the sequence between the two nodes. To crosslink matches, bidirectional edges with weight 0 between aligned bases in overlapping reads are added. Then, a standard maximum weight single source maximum weight path is computed (the complexity is linear in relation to the number of vertices).

4.10 CAP3

CAP3 is an improved version of CAP2 [14] (see Section 4.7 also). Three major improvements are:

1. Automatics clipping of $5'$ and $3'$ poor quality regions of fragment reads.

2. Use of base call quality values in alignment of fragment reads.

3. Use of clone-mate information (forward-reverse constraints) to correct assembly errors and link contigs.

4.10.1 Automatics clipping of $5'$ and $3'$ poor quality regions

Clipping is done using both base quality values and sequence similarities, and based on the notion of *good regions of a read* as opposed to the poor quality regions. Good quality regions are defined with two criteria.

1. Any sufficiently long region of high-quality values that is highly similar to a region of another read is defined to be good.
2. Any sufficiently long region that is highly similar to a good high-quality region of another read is defined to be good.

Once good regions are defined, the 3' clipping position of a read is the maximum of 3' end positions of good regions of the read and the 5' clipping position of a read is the minimum of 5' end positions of good regions of the read. High-quality regions are easily defined based on base call quality values, but how can sequence similarity be defined? Regions of a read with a strong similarity to other reads are computed using a extended version of the banded Smith-Waterman algorithm to incorporate base quality values.

4.10.2 Computation and evaluation of overlaps

A global alignment with the maximum similarity score is computed over the band determined by the optimal local alignment while clipping poor quality regions. Then each overlap is evaluated by five measures.

1. The minimum length
2. The minimum percent identity
3. The minimum similarity score
4. The fourth measure examines the differences of the overlap at bases of high quality values. If the overlap contains a sufficient number of differences at bases of high-quality values, then the overlap is probably false. This is done by computing the quality difference score of the overlap that is the sum of scores at each difference
5. The difference rate of the overlap is examined with respect to the sequencing error rates of the two regions involved in the overlap. The error rate of any region of a read is estimated using the error vector method [14] (see Section 4.7 also).

4.10.3 Use of constraints in construction of contigs

The procedure for using constraints in construction of contigs consists of four steps:

1. An initial layout of reads is performed by use of a greedy method in decreasing order of overlap scores.
2. The quality of the current layout is assessed by checking constraints.

3. A region of the current layout with the largest number of unsatisfied constraints is located such that the unsatisfied constraints are satisfiable by making corrections to the region. If such a region exists, corrections to the region are made and steps 2 and 3 are repeated. Otherwise, the correction procedure is terminated.

4. Contigs are linked with constraints.

More details for each step are given below.

Terminologies The clone-mate information of two reads are called *a forward-reverse constraint*. The constraint is *satisfied* by a layout if the two reads occur in the same 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 the current layout. Let f $\rightarrow$ g denote an overlap from read f to read g.

Step 2: grouping unsatisfied constraints Unsatisfied constraints are partitioned into two types of groups, where all constraints in a group are associated with an unused overlap or a pair of contigs. For each unsatisfied constraint that is satisfiable by unused overlaps, a unique overlap with the maximum score is selected from the unused overlaps, and the constraint is associated with the overlap. Additional criteria are used to choose only one winner in the case that there are two or more unused overlaps with the maximum score.

Step 3: constraint satisfaction The unsatisfied constraint groups are attempted to be satisfied. There are two cases. The first case is that a group involves unused overlaps. A group with the largest number of unsatisfied constraints is selected for consideration. First consider the case where the group is associated with an unused overlap f$\rightarrow$ g. The number of unsatisfied constraints in the group is examined to be greater than the sum of the following three; the parameter u, the number of satisfied constraints supporting the used overlap involving f, and the number of satisfied constraints supporting the used overlap involving g. If this is the case, the layout is corrected by breaking the used overlaps involving f and g, and joining f and g with the overlap f $\rightarrow$ g. The second case is that a group involves two contigs. Reads from other regions that are associated by constraints with reads in the two contigs are used to close the gap.

Step 4: contig ordering Unsatisfied constraints are used to orient and order contigs. The groups are considered in decreasing order of group size and the order is determined when the number of unsatisfied constraints is above a threshold.

4.10.4 Construction of alignments and 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. To produce an accurate alignment, the base quality values of the reads are used in the construction.

Aligning a read to a multiple sequence alignment is conceptually the same as aligning two reads. The only difference is how to define a scoring scheme, i.e., match score, mismatch score, gap open penalty, and gap extension penalty. The most important issue is how to incorporate base quality

scores for the scoring scheme. This is done as follows. Consider a column of the block. Let the column consist of k nonblank characters $c_i, 1 \leq i \leq k$. Each c_i is in one of A, C, G, T, N, and -, where N denotes any base other than the four regular bases and is a gap symbol. Let q_i be the quality value of c_i. If c_i is a gap symbol, then q_i is the quality value of the base immediately before c_i in the same row if such a base exists and the quality value of the base immediately after c_i otherwise. Seven average quality values are computed for the column: five for substitution, one for deletion, and one for insertion. Each of the five values for substitution corresponds to one of the five base types. Let $q_s(d)$ denote the average quality value for substitution involving base type d of the read, let q_d denote the average quality value for deletion, and let q_n denote the average quality value for insertion. The values are defined by the following formulas, where d is a regular base.

$$q_s(d) = ((\sum_{1 \leq i \leq k \ \& \ c_i = d} q_i) - (\sum_{1 \leq i \leq k \ \& \ c_i \neq d} q_i))/k$$

$$q_d = \sum_{1 \leq i \leq k \ \& \ c_i \neq -} q_i/k$$

$$q_n = \sum_{1}^{k} q_i/k$$

$$q_s(N) = -q_n$$

where k is the number of non-blank characters in the alignment.

The scores of a substitution, a deletion, and an insertion are defined as follows. Consider a substitution involving the column of the block and a base d of the read with quality value q_r. If $q_s(d) > 0$, then the substitution is considered as a match and its score is $m \times min(q_s(d), q_r)$, where the positive integer m is a match score factor. If $q_s(d) < 0$, then the substitution is considered as a mismatch and its score is $n \times min[q_s(d), q_r]$, where the negative integer n is a mismatch score factor.

The score of a gap is the gap open score plus the gap extension score of each element in the gap. For simplicity, the gap open score is a small negative number independent of quality values. However, the extension score depends on quality values. Let a positive number g be a gap extension penalty factor. The extension score of a column with average deletion quality value q_d in a deletion gap is $g \times min(q_d, q_r)$, where q_r is the quality value of a base of the read immediately before or after the gap. The extension score of a base of the read with quality value q_r in an insertion gap is $g \times min(q_n, q_r)$, where q_n is the average insertion quality value of a column of the block immediately before or after the gap.

4.11 CELERA

The CELERA Whole Genome Assembler(WGA) is the first sequence assembler that assembled large eucharyotic genomes (>100Mbp) successfully using the whole genome shotgun approach. The description of WGA is based on the article for the Drosophila genome assembly [**?**]. The main design principle is as follows.

1. The clone mate information is used to generate scaffold of the genome, not just contigs.

2. The assembler is capable of utilizing external information.

3. The assembler places fragment reads in a series of stages, starting with the safest moves to more aggressive ones. The stage and evidence for a read's placement are open to inspection, providing an audit trail of the assembler's decision.

4.11.1 Data preparation

To achieve a very large scale genome sequencing, it is very important to ensure the quality of the input shotgun data. Errors in the input shotgun data will certainly compound the already difficult assembly problem. The data used for the Drosophila genome is very high quality, less than 2% errors in reads, 0.34% error rate in clone mate information. In addition, poor quality regions are trimmed out before sequence assembly.

4.11.2 The overview of WGA

WGA employs the "overlap-layout-consensus" approach. However, note that every assembler employing the approach differs in detail. A figure from Figure 19 shows an overview of the WGA, which will be detailed in the subsequence sections. One distinguishing feature would be *scaffolder*.[10] The more important feature would be the quality control of input data. As explained in Section 4.11.1, the input data was of very high quality. Furthermore *Screener* module masked known repeats.

4.11.3 Overlapper

Fragment overlaps are detected with a criteria requiring fewer than 6% difference and at least 40bp of unmasked sequence. The methodology is similar to the well known seed-and-extend idea developed for BLAST [1]. The average number of overlaps per fragment read was 33.7, which is a lot higher than the coverage of the shotgun data (14.6). Like many other assemblers, unusually high number of fragment overlaps can be used for detecting repeats.

4.11.4 Unitigger

A collection of fragments whose arrangement is uncontested by overlaps from other fragments were assembled into *unitigs*, which are usually called contigs in other assemblers. Then each unitig were examined to see whether it contains repeats or not, with a method that will be explained below. Unitigs that represent unique DNA, i.e., no repeats, are called *U-unitigs*. Potential repeat boundaries in U-unitigs are computed and then U-unitigs were extended up to this repeat boundaries. To detect misassembled unitigs, they used *A-statistic* that is the log-odds ratio of the probability that the distribution of fragment start points is representative of a correct versus an overcollapsed unitig of two repeat copies. U-unitigs are unitigs with A-statistic > 10. A-statistic is calculated as follows.

Let F be the number of fragments in the database and G be the (estimated) size of a genome. Assume that fragments are not oversampled. For an unitig with k fragments and distance ρ between the start of its first fragment and the start of the last fragment, the probability of seeing $k-1$

[10] Several assemblers generate scaffolds as of now, but WGA is probably the first one that attempted to generate scaffolds of the whole genome.

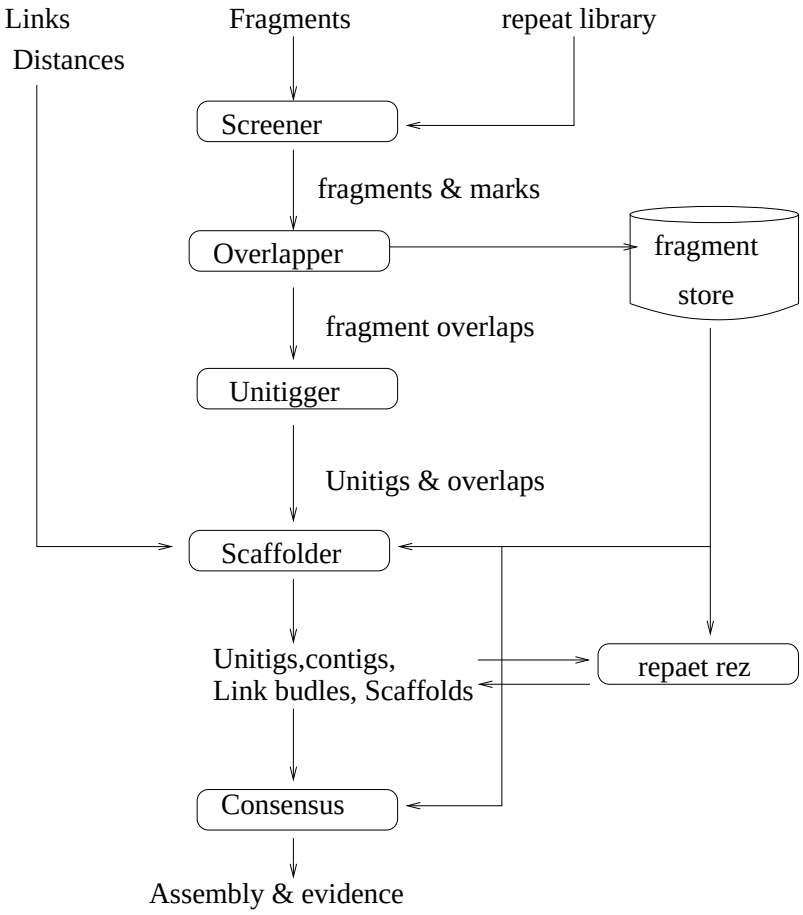

Figure 19: The overview of WGA

start points in the interval of length ρ is $[(pF/G)^k/k!]exp(-pF/G)$. If the unitig was the result of collapsing two repeats, the probability is $[(2pF/G)^k/k!]exp(-2pF/G)$. A-statistic is the log of the ratio of the two probabilities.

By detecting repeat boundaries, unitigs are extended on the order of a fragment length, i.e., extension of an unitig up to a fragment length. Repeat boundaries are detected as follows. Whenever a unitig A overlaps with two different unitigs B and C, but B and C overlapping regions fail to overlap, a repeat boundary can be detected by aligning B and C using a dynamic programming technique.

4.11.5 Scaffolder

Using clone-mate information or BAC ends, unitigs are oriented and ordered. More discussion can be found in Section 6.2.

4.11.6 Repeat resolution: rocks, stones, and pebbles

Both intra- and inter-scaffold gaps are filled with a series of three, increasingly more aggressive, levels of repeat resolution.

1. The rock phase: unitigs are placed when at least two clone mate pairs are consistent.
2. The stone phase: unitigs are placed when a single clone mate pair and overlap tiling path confirm the positioning.
3. The pebble phase: a best tiling path based on quality values is computed.

4.12 ARACHNE

ARACHNE [3] is a system for assembling genome sequence. It employs a traditional "overlap-layout-consensus" approach (look at Euler in Section 4.13 for an alternative approach), but has a number of well designed features for large scale genome projects; an efficient and sensitive procedure for finding real overlaps, a procedure for scoring overlaps that achieves high accuracy by correcting errors *before* assembly, read merger based on forward-reverse links, and detection of repeat contigs by forward-reverse link consistency. The outline of the ARACHNE assembly algorithm is:

1. Overlap detection and alignment: Pairwise overlaps are computed by a three step process: (a) identify all k-mers (they used $k = 24$) and merge overlapping shared k-mers, (b) extend the shared k-mers to alignments, and (c) refine the alignment by dynamic programming.
2. Error correction: Sequencing errors are detected and corrected by generating multiple alignments among overlapping reads using a majority rule based on quality base score (by Phred).
3. Evaluation of alignments: Each alignment is evaluated by an overall penalty score for an alignment which combines individual discrepancy in base calls (considering the quality score). Overlaps incurring to a high penalty score are discarded. This step detects repeats and chimerical reads.

4. Identification of paired pairs: Complexes of paired pairs are built by starting with combining two overlapping paired pairs at both ends and iteratively adding paired reads matching both ends.

5. Contig assembly: First, potential repeat boundaries are identified by trying to align fragments that extend the same fragment. The all fragment reads are merged and extended until a repeat boundary is reached.

6. Detection of repeat contigs: Once contigs are assembled, contigs that are potentially misassembled due to repeats are identified by two criteria, one to check the depth of coverage and another to check the consistency of links to other contigs.

7. Creation of supercontigs: After marking repeat contigs, unmarked contigs are oriented and ordered based on forward-reverse links.

8. Filling gaps in supercontigs: Once the scaffold of contigs are generated, gaps between CONTIGS are attempted to be filled using repeat contigs.

We will review techniques used in ARACHNE in detail below.

4.12.1 The alignment module

The alignment module is used to align fragment reads to reads, reads to contaminants, contigs to each other. It works in four phases as below.

1. The phase 1 makes a list of all k-mers that appear in the sequences and their reverse complements.

2. After sorting the k-mer records, each record is processed as below (any k-mer that occurs more than a specified threshold times is not processed). Each k-mer hit is extended to a long, imperfect, but gap-free partial alignments (no mismatches at the ends of the extended match).

3. The phase 3 generates a full alignment of a pair of reads from the extended k-mer hits by filling in bases between consecutive extended k-mer hits.

4. Using the banded Smith-Waterman algorithm, each full alignment in the phase 3 are refined. Each final alignment is assessed using the quality scores, a penalty score is computed. An alignment with a penalty score higher than a preset threshold is discarded.

4.12.2 Contig assembly

Reads are merged into contigs up to potential repeat boundaries. Repeat boundaries are detected by looking for inconsistency in extending overlaps. For example, suppose that a read r can be extended by both x and y, but x and y do not overlap each other. Then a repeat boundary can be detected by aligning x and y. However, this could result in detecting too many potential repeat boundaries. Thus ARACHNE uses two techniques, dominated reads and subreads, to eliminate spurious repeat boundaries.

4.12.3 Detecting repeat contigs and repeat supercontigs

Two heuristics are used to detect contigs and supercontigs that are misassembled due to repeats.

1. Density of reads: ARACHNE computes the log-odds ratio,

$$log(\frac{a\ given\ density\ of\ reads\ representing\ a\ unique\ region\ of\ genome}{a\ density\ representing\ at\ lest\ two\ copies\ of\ a\ repeated\ regions})$$

   Any contigs with a log-odds ratio less than 1 are marked as repeated. This technique is used in the CELERA assembler (see Section 4.11).

2. Consistency of forward-reverse links: For two contigs A and B, the distance between two is estimated whenever there are at least two links. The mean and standard deviation, denoted by $d(A, B)$ and $ERR(A, B)$ respectively, are computed and used for detecting repeat (super)contigs with two rule explained below.

   (a) Rule1: If $d(A, B) < -2000 - 4 \times ERR(A, B)$ then the (super)contigs are marked as repeated.

   (b) Rule2: if contig A is linked to both B and C, then from $d(A, B)$ and $d(A, C)$ and the lengths of B and C, we can compute $d(B, C)$, the estimated gap or overlap length between B and C. If $d(B, C) < -2000 - 4 \times ERR(B, C)$ then A is marked as repeated.

4.12.4 Supercontig assembly

Once contigs are assembled and repeat contigs are marked, supercontigs are built incrementally using unmarked contigs. To make this procedure recursive, every contig becomes a supercontig and then merged incrementally using a priority queue Q. During each round of merging, the following steps are performed.

1. Extract from Q the pair of supercontigs, S1, S2, with the highest priority score.
2. Merge S1, S2, creating supercontig T.
3. Remove all pairs in Q where one of the two supercontigs is either S1 or S2.
4. Find all supercontigs W that share forward-reverse links to T, and insert (T,W) into Q.
5. Apply Rules 1 and 2 for marking repetitive supercontigs, to mark T or any supercontig W linked to T. For any supercontig marked as repetitive, remove all pairs in Q containing it.

4.12.5 Filling gaps in supercontig

Gaps are filled using graph based algorithms.

1. First a Graph $G = (V, E)$ is defined, where V is the set of contigs and an edge connects two contigs if they are known to overlap. Then for every pair of contigs, the shortest *path* between contigs A and B, $d_p(A, B)$, is computed.
2. The shortest path information is stored in a new graph, $G_{paths}(V, E_{paths})$, where E_{paths} correspond to the computed shortest paths.

3. For every pair of contigs A,B that are consecutive in a supercontig S, ARACHNE attempts to fill the gap as follows.

   (a) If (A,B) is in E_{PATHS}, that is, a path has already been computed for A,B, then simply fill the gap with the contigs in the path.

   (b) Otherwise, first find every contig T that shares forward-reverse links with W, and such that from these links T is positioned between A and B in W (Fig. 12B). The set of all such contigs is called the set of targets, V_T.

   (c) Using breadth-first search, look in G_{PATHS} for a path from A to B that uses only nodes in V_T. If a path is found in G_{PATHS} from A to B, this corresponds to a path p in G from A to B. Use the ordered contigs in p to fill the gap between A and B.

4.13 EULER

In Section 4.4, we reviewed an approach, proposed by Idury and Waterman [17], to assemble fragment data by combining *sequencing by hybridization* [5, 35] and shotgun sequencing. This method has recently been further developed by Pevzner, Tang, and Waterman [34]. The proposed approach uses the Eulerian superpath approach and does not employ the traditional "overlap-layout-consensus" approach. As in the Idury and Waterman approach, a de Bruijn graph, denoted by G, is constructed using tuples of length l and the Eulerian path of the graph is computed for sequence assembly (see Section 4.4). However, there are two major differences in two approaches, of EULER and of Indury and Waterman.

1. Errors in fragments results in a tangle of erroneous edges, which becomes a major hurdle in computing the Eulerian path. EULER tries to correct as many errors in fragments as possible before computing the Eulerian path by solving the error correction problem.

2. EULER solves the Eulerian superpath problem where a series of transformations of an Eulerian graph and collection of paths to a new graph and collection of paths are made to compute a solution to the sequence assembly problem.

4.13.1 Error correction and data corruption

To correct errors in the input shotgun data, we need to know the target sequence. However, the target sequence is supposed to be assembled using the (erroneous) input shotgun data. This seemingly impossible task is nicely formulated as computing G_l, the set of all tuples in G. Of course, G_l is unknown but their strategy is to approximate G_l without knowing the sequence of G. Two problem formulations are proposed to eliminate errors in the shotgun data, *the spectral alignment problem* and *the error correction problem*. According to the authors' experience, the error correction problem formulation captures the specifics of DNA sequencing and it is more effective that the the spectral alignment problem formulation. Thus we discuss the error correction problem only.

The error correction problem Let $S = \{s_1, ..., s_n\}$ be a collection of reads, and Δ be an upper bound on the number of errors in each DNA read, and l be an integer for the length of a pattern. given S, Δ and l, introduce up to Δ corrections in each read in S in such a way that $|S_l|$, i.e., the set of all l-tuples from S, is minimized.

The basic idea comes from an observation that a single error in a read s affects at most l l-tuples in s and l l-tuples in the reverse complement of s that point to the same sequencing error ($2d$ for positions within a distance $d < l$ from the endpoint of the reads). EULER employs the greedy approach that looks for error corrections in the reads that reduce the size of S_l by $2l$ (or $2d$ for positions close to the endpoints). This greedy procedure sometimes introduces errors rather than correcting errors. For example, this procedure corrected 234,410 errors and introduced 1,452 errors in the *N. meningitidis* genome project. the authors argue that introducing errors is not bad as long as consistent overlaps can be detected so that false edges in the de Bruijin graph can be reduced (the errors can be corrected later).

4.13.2 Eulerian superpath

Once the $|S_l|$, the set of all l-tuples from $S = \{s_1, ..., s_n\}$, is computed by solving the error correction problem, the de Bruijin graph $G(S_l)$ with vertex set S_{l-1} (the set of all $l-1$-tuples from $S = \{s_1, ..., s_n\}$) is computed as follows. A directed edge from an $l-1$-tuple $v \in S_{l-1}$ to another $w \in S_{l-1}$ is added if S_l contains an l-tuple whose the first $l-1$ prefix coincides with v and whose the last $l-1$ suffix coincides with w.

Terminologies A path $v_1, ..., v_n$ in the graph is called *repeat* if indegree(v_1) > 1, outdegree(v_n) > 1, and indegree(v_1) = outdegree(v_i) $= 1$ for $1 \leq i \leq n-1$. A *read-path* covers a repeat if it contains an entrance into and an exit from this repeat. A repeat is called *tangle* if there is no read-path containing this repeat.

Eulerian superpath problem Given an Eulerian graph and a collection of paths in this graph, find an Eulerian graph path in this graph that contains all these paths as subpaths. The Eulerian superpath problem is solved by transforming both the graph G and the system of paths P into a new graph G_1 and a new system of paths P_1. Such transformation is called *equivalent* if there exists a one-to-one correspondence between Eulerian superpaths in (G, P) and (G_1, P_1). The goal is to make a series of transformations

$$(G, P) \rightarrow (G_1, P_1) \rightarrow ... \rightarrow (G_k, P_k)$$

where the final system of paths P_k are single edge. Due the equivalent transformations, every solution of the Eulerian superpath problem in (G_k, P_k) provides a solution of the Eulerian superpath problem in (G, P), i.e, contigs.

The transformations are implemented using two techniques, *detachment* and *cut*. The x,y-detachment is a transformation that adds a new edge z=(v_{in}, v_{out}) and deletes the edges x and y from G (see Figure 20 for more detail). The x-cut is a transformation that removes the edge x from all paths that contain the edge (see Figure 21 for more detail). The authors claims that detachments and cuts are powerful techniques to untangle the graph and to reduce the fragment assembly to the Eulerian Path Problem for bacterial genomes that the authors used for.

4.13.3 Use of clone-mate information

There are two EULER programs that utilize the clone mate information, EULER-DB and EULER-SF. EULER-DB tries to untangle repeats using the clone-mate information by treating each clone-

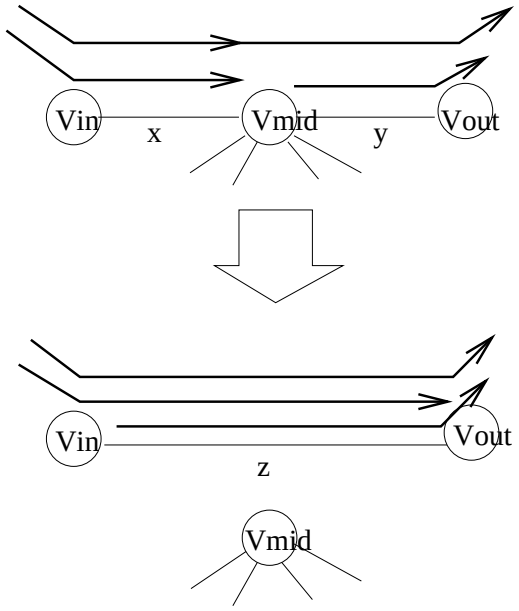

Figure 20: The x,y-detachment: edges x and y are substituted with a new edge z. As a result, all three paths have direct paths from x to y.

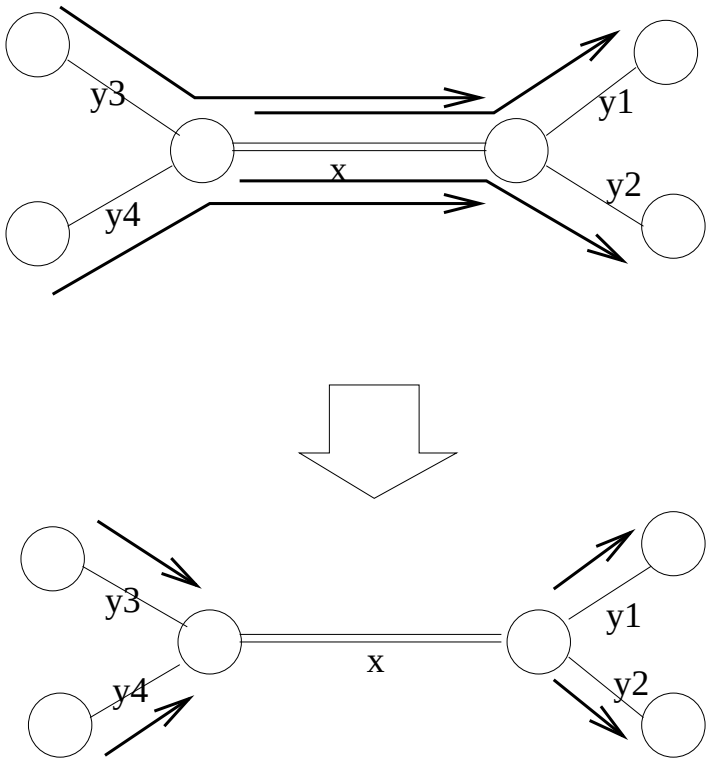

Figure 21: The x-cut is used for a tangle, i.e., a repeat that does not have a read-path. The edge x is deleted from all paths that contains x.

mate pairs as artificial paths in the graph with their expected lengths. EULER-SF generates scaffolds using the clone mate information.

5 Methods for Detecting Misassembly

The existence of repeats in the genome can easily lead to misassembly of contigs. The most accurate method to detect such misassembled contigs is to perform wet-lab experiments. However, this is time-consuming and requires a carefully designed experiments. Rouchka and States [42] summarizes such techniques and proposed a computational technique for wet-lab experimental result for contig assembly verification. The wet-lab experiments discussed in the paper include high clone coverage maps, multiple complete digest mapping, optical restriction mapping, and ordered shotgun sequencing [42]. Recently, several computational techniques without using wet-lab experiments have been developed. These techniques can be developed as separate computational tools [23, 24] or embedded in the sequence assemblers. The assembly validation techniques embedded in the sequence assemblers are reviewed in Sections 4.12, 4.11, and 4.10.

5.1 A clone coverage analysis approach

A sequence assembly validation method based on the coverage of clone is proposed recently [23]. This approach works in three steps as below.

1. Contigs are oriented and ordered.
2. The estimated lengths for all library clone types are computed and clones are classified into two classes, *bad clones* whose length deviate much from the expected clone length and *good clones* whose length is within an acceptable range of the expected clone length.
3. A *good-minus-bad clone coverage plot* is computed for each contig by subtracting the number of bad clones from the number of good clones.

The basic idea is simple. Any region where more bad clones are aligned than good clones is likely to be misassembled. This approach successfully identified positions of misassembly, i.e., where the misassebly begins and ends while many other approaches are intended to identify whether a contig as a whole is misassebled or not.

5.1.1 Determination of contig orientations and orders

We define terminologies that denote different regions of a clone.

Definition 1 `The forward fragment`, **C.f**, *is the sequence read at one end of a clone* **C** *and* `the reverse fragment`, **C.r**, *is the sequence read at the other end of the clone* **C**. `The sequenced region` *of a clone is a region covered by the fragments at both ends of the clone.* `The unsequenced region` *of a clone is a region not covered by either C.f or C.r.* `Clone length` *of a clone C,* `length(C)`, *is the sum of the sequenced and unsequenced regions of the clone.* `Inner-clone length` *is the length of unsequenced region. See Figure 22.*

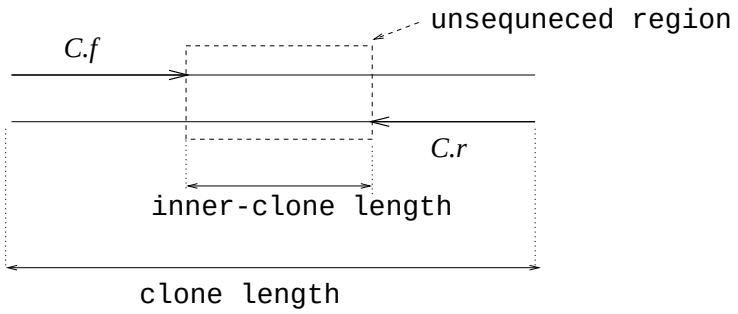

Figure 22: Illustration of clone regions showing the sequenced regions (`C.f` and `C.r`) , unsequenced region, clone length, and inner-clone length.

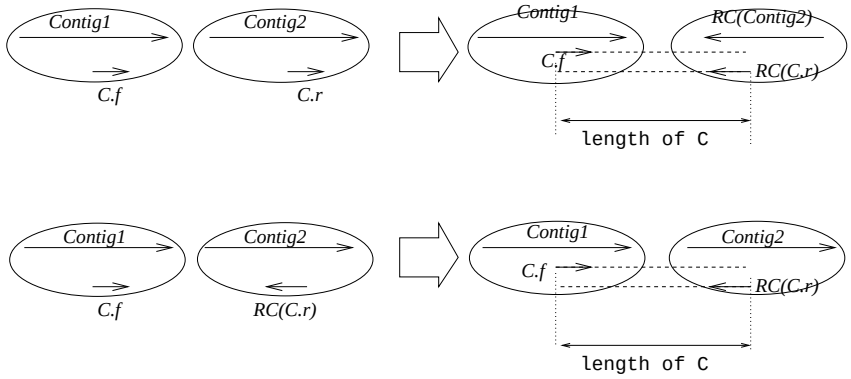

Figure 23: Contig ordering and determination of inter-contig clone length. RC(`seq`) denotes the reverse complement of a sequence (`seq`).

Definition 2 `An embedded clone`*, C, is a clone where C.f and C.r are aligned in the same contig.* `An inter-contig clone`*, C, is a clone where C.f and C.r are aligned in two different contigs.*

We compute *the average clone length* for each library clone type by computing the mean value of all embedded clones. Because inter-contig linkages are not reliable, inter-contig clones are not included when the average clone length is computed. However, all clones, including inter-contig clones, are considered when we classify clones into *good clones* and *bad clones* (see Definition 4). To compute the clone length of an inter-contig clone, orientation of contigs is first defined.

Definition 3 `Contig ordering rule`*: Assume an inter-contig clone C that links two contigs, Contig1 and Contig2. The two contigs can be ordered with respect to clone C. There are four possible cases in terms of alignments of C.f and C.r in the contigs. If C.f is aligned in Contig1 and C.r is aligned in Contig2, then the order is Contig1 followed by the reverse complement of Contig2. If C.f is aligned in Contig1 and the reverse complement of C.r is aligned in Contig2, then the order is Contig1 followed by Contig2. In the remaining two cases, contigs can be similarly ordered.*

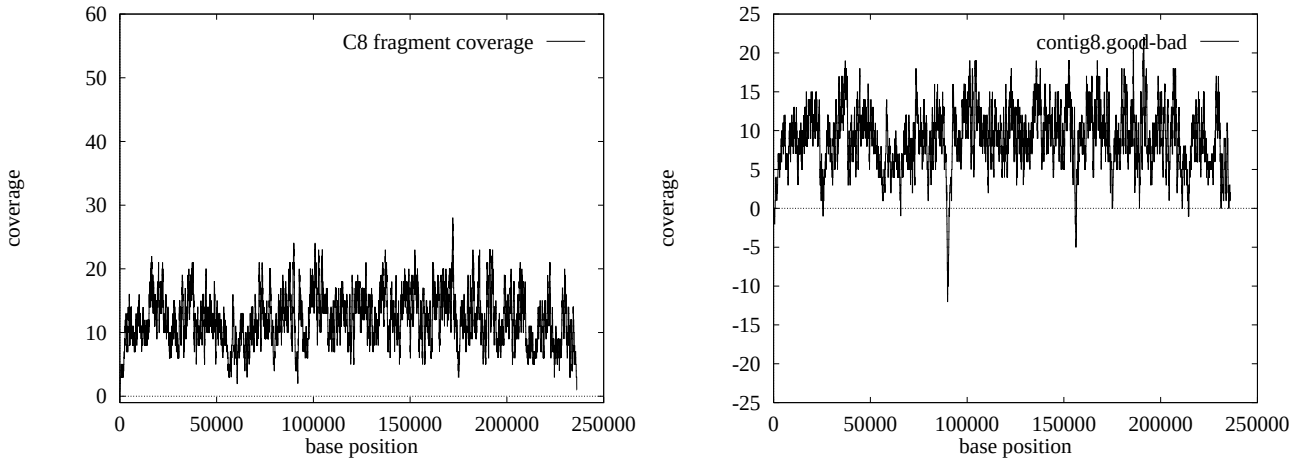

Figure 24: The fragment coverage (left) and the clone coverage plot (right). There is a misasembled region from 89,415 to 90,332 where the fragment coverages are not distinctly high but the valleys in the clone coverage plot are distinct, effectively identifying the misassembled region.

5.1.2 The classification of clones

Given an order of contigs with respect to a clone C, the length of C can be easily computed (see for Figure 23). Then, clones can be classified into *good clones* and *bad clones* with respect to the clone length v.s the average clone length as described below.

Definition 4 *An embedded clone C is a* `good clone`*, if length(C)* $< 2\times$ *the average clone length. Otherwise, C is a* `bad clone`*.*
An inter-contig clone C is a `bad clone` *if length(C)* $\geq 2\times$ *the average clone length.*

Because the length of a gap between two contigs is not known, good clones cannot be reliably defined for inter-contig clones.

Definition 5 `seq(x,y)` *denotes a subsequence of a sequence* `seq` *from base position* `x` *to base position* `y`*.*

5.1.3 The good-minus-bad clone coverage analysis

Good clone coverage is computed by counting how many sequenced regions of good clones cover the base position. Bad clone coverage is computed by counting how many sequenced regions of bad clones cover each base position. The goal of this analysis is to compute how many good (trustworthy) clones and bad (untrustworthy) clones are aligned in contigs. This simple analysis is effective in identifying positions where misassembly begins and ends. For example, there is a misassembled region from 77222 to 78496 base positions in the contig 7 in the assembly of *Mycoplasma genitalium* genome generated using Phrap [36] with default parameters. Almost exact positions can be identified using the good-minus-bad clone coverage plot in Figure 25. As shown in Figure 24, this approach clearly discriminate misassembled regions from correctly assembled regions.

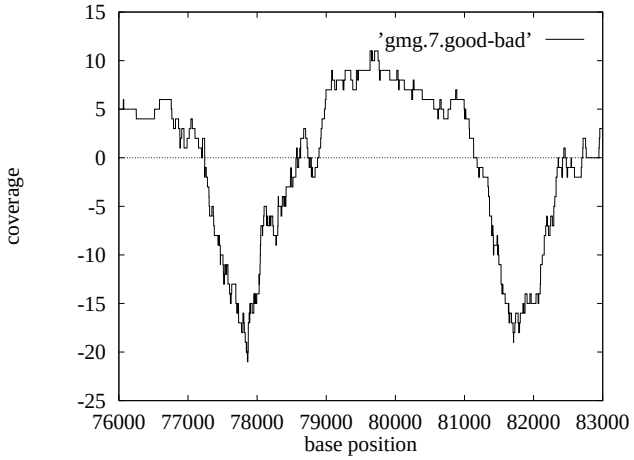

Figure 25: Expanded view of the two valley regions in Contig 7. CROSS_MATCH found three parts of C7 match different segments of the published genome: C7(34,78496) matches (351169,429610), C7(77222,82533) matches (226188,231457), and C7(81660,117375) matches (314173,349879). The boundaries of the misassembled regions are 77222, 78496, 81660, and 82533 base positions. The start and end positions of the valley region (< 0) correspond to boundaries of misassembled regions.

5.2 A probabilistic approach

This approach[24] identifies misassembled regions using entropy plots that are computed using statistics on the number of patterns per fragment. To compute entropy of fragments, we need to construct a probability model that measures how much each aligned fragment contributes to misassembly. The probability model is built using the *fragment distribution*, a measure used for repeat handling in a sequence assembler called AMASS[21]. We first describe how to collect the fragment distribution from a shotgun data.

5.2.1 Fragment distribution

A fixed number of non-overlapping patterns from each fragment are selected at random positions. Once all patterns are selected from all fragments, occurrences of the selected patterns are found in the entire shotgun data using a fast multiple string pattern matching algorithms[26]. A plot of the number of patterns in a fragment *vs.* the number of fragments is then generated. Figure 26 shows how to calculate the fragment distribution. Note that the fragment distribution is a fragment-centric distribution that counts multiple pattern occurrences within a fragment.

The rationale behind using this distribution for repeats handling is as follows. We select patterns that are long enough and therefore unlikely to occur by chance. For instance, if we use patterns of 16 bp, then patterns of that size occur only once by chance in a sequence of 4,294,967,296 bp ($= 4^{16}$) as there are four characters in DNA. Thus, the common occurrences of a pattern reflect with high probability true overlapping fragments. What we are collecting is a distribution of the number of such patterns within a fragment. Because we consider multiple patterns and their positions (within a fragment), the distribution should reflect the characteristic of the shotgun sequencing strategy, *i.e.,* random selection of fragments. Thus, the distribution is expected to look like a normal distribution centered around a mean value. Repeats, of course, will distort the distribution

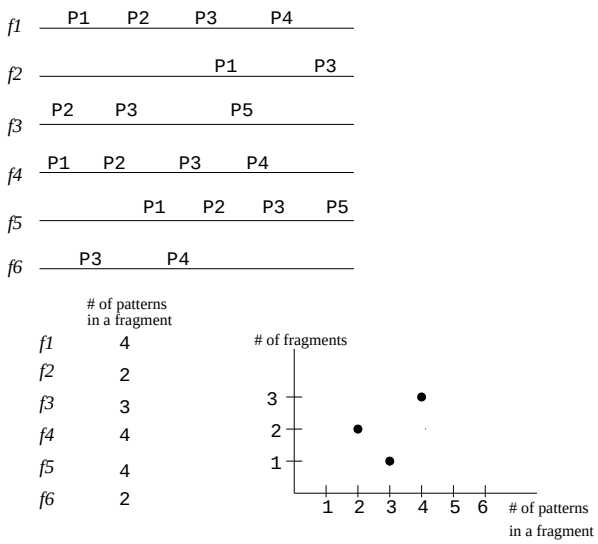

Figure 26: A sample fragment distribution

since any pattern that belongs to a repeat will exhibit an increased number of pattern occurrences. Thus, abnormally high number of pattern occurrences in a fragment with respect to the peak value of the fragment distribution indicates possible repeats. Consequently, the higher the number of patterns occurrences in a fragment, the more likely this fragment will contribute to misassembly. The probability model simply reflects this observation.

5.2.2 The probabilistic model

From the fragment distribution, we construct a probability function that models how much each fragment contributes to misassembly. As we discussed in the previous section, we expect the fragment distribution to reflect the characteristic of the shotgun data, *i.e.,* randomness of fragment selection, when there are no significant repeats. Thus the contribution of a fragment to misassembly can be seen by how much the fragment is deviated in terms of randomness of fragment selection. One way to measure the degree of deviation is to compare the fragment distribution to the fragment coverage distribution. Unfortunately, the coverage distribution is not known. However, it could be approximated by sampling coverage distributions from many short assembled regions.

For now, we use an *ad hoc* probability model based on the observation that a fragment with more pattern occurrences is likely to lead to misassembly. The probability that a fragment f contributes to misassembly is computed as below:

$$
\begin{aligned}
prof(f_i) = & 0.001 \quad \text{if number of patterns in } f_i < 2 \times p_v \\
& 0.01 \quad \text{if } 2 \times p_v \leq \text{number of patterns in } f_i < 3 \times p_v \\
& 0.1 \quad \text{if } 3 \times p_v \leq \text{number of patterns in } f_i < 4 \times p_v \\
& 0.889 \quad \text{if } 4 \times p_v \leq \text{number of patterns in } f_i
\end{aligned}
$$

where p_v denotes the peak value in the fragment distribution.

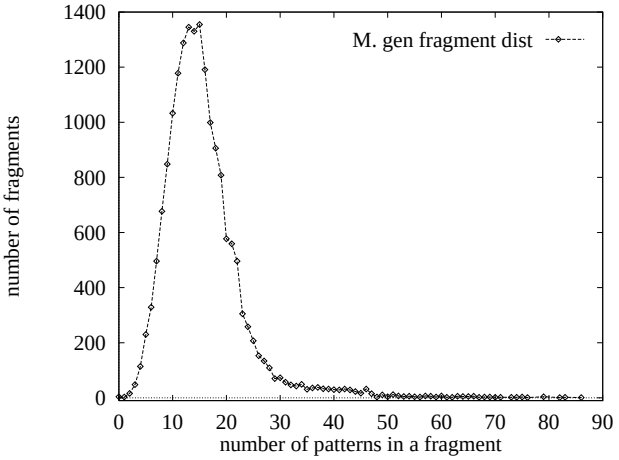

Figure 27: A fragment distribution from gmg shotgun data with 2 patterns of 16 bp per fragment.

5.2.3 Computing entropy

From the probability model, we compute entropy at base position p in a contig as below:

$$entropy(p) = \sum_{p-\delta \leq pos(f_i) \leq p+\delta} -prob(f_i) \times log(prob(f_i))$$

where $pos(f_i)$ denotes the left end position of f_i in the contig and δ is a user input parameter (by default, it is the same as the window size used for the fragment distribution calculation).

As shown in Figure 28, this approach clearly discriminate misassembled regions from correctly assembled regions.

6 Methods for Scaffold Generation

Recently developed sequence assembler packages include a scaffold generation module that generate scaffolds of (pre)assembled contigs [34, 20, 30, 3, 15]. Here, we survey three scaffolding techniques, CELERA scaffolder [16], and Gigassembler [20] (see Section 4.10 for CAP3 [15]). In contrast, there is a proposal for a genome sequencing framework to enumerate scaffolds in an exploratory, hierarchical fashion [25], which has been tested with much smaller genome, *Agrobacterium* [27].

6.1 Gigassembler

Gigassembler [20] is an algorithm to assemble contigs, map, mRNA, EST, and BAC end data and generate scaffolds of a genome. It was used for the Human genome assembly on the public side [28]. This is a second-stage assembler after PHRAP [36] or any assemblers. The overview of assembly process is as follows.

1. Decontaminating and repeat masking the sequence. Repeatmasker [40] is used to mask known repeats and contaminants from bacteria, vectors, and others.

2. Aligning mRNA, EST, BAC end, and paired plasmid reads against initial sequence contigs. See the paragraph "Bridging rafts with mRNA, ESTs, paired plasmid reads, BAC end data."

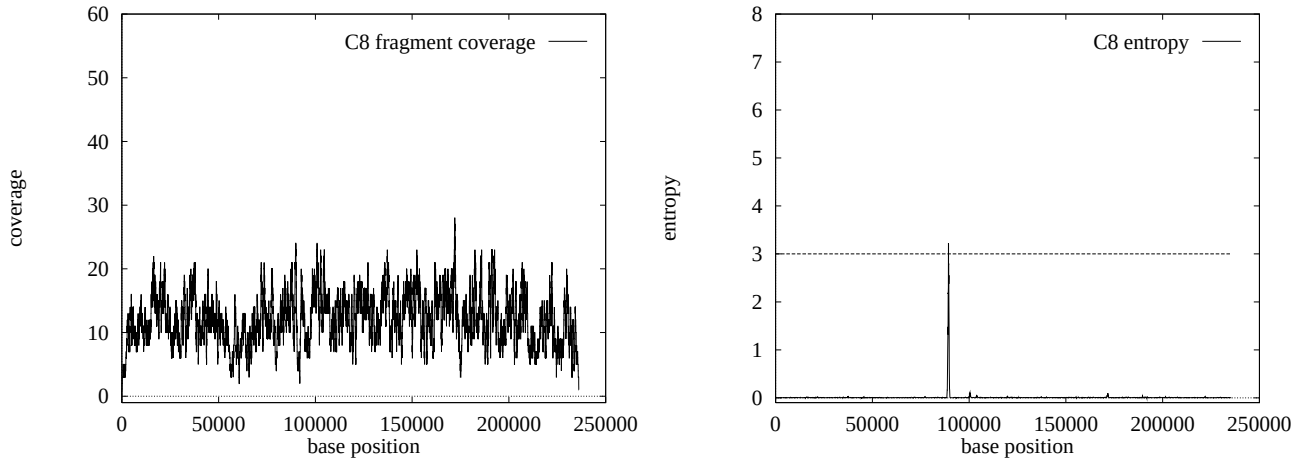

Figure 28: The fragment coverage (left) and the entropy plot (right). There is a misaasembled region from 89,415 to 90,332 where the fragment coverages are not distinctly high but the peaks in the entropy plot are distinct, effectively identifying the misassembled region.

3. Creating an input directory structure using map and other data. For the Human genome, they used Washington University map data. A directory is created for a chromosome and a subdirectory for each fingerprint clone contig.

4. For each fingerprint clone contig, aligning the initial sequence contigs within that contig against each other.

5. Using the GigAssembler program within each fingerprint clone contig to merge overlapping initial sequence contigs and to order and orient the resulting sequence contigs into scaffolds. See the paragraph "Building a sequence path."

6. Combining the contig assemblies into full chromosome assemblies.

6.1.1 Alignment of mRNA, ESTs, BAC Ends, and Paired Reads

Contigs are oriented and ordered by aligning mRNA, ESTs, BAC ends, and paired reads to contigs using a program called psLayout, which works as follows.

1. Build up an index of all of the 10-mers in a 30-Mb subset of the target sequence, excluding simple repeating elements from this index.

2. Break up the query sequence into overlapping 500-base regions.

3. Look up the 10-mers that occur in the 500-base region in the index and identify clusters of matching 10-mers in the target sequence that represent likely areas of homology.

4. Do a detailed alignment between the 500-base region and the sections of the target sequence identified in step 3. In the case of mRNAs and ESTs this alignment is done in a fashion that tolerates introns.

5. Stitch together the alignments from overlapping 500-base regions using a dynamic programming algorithm.

PsLayout reports all matches above a certain minimal quality between query sequences and database sequences. To reduce the effects of repeats, two techniques are used. The first technique is to use repeats sequences (repeats are not masked although they are detected by RepeatMasker). The second technique is to maintain "near best" matches only, which means discard matches (even good ones) if they are below the best match.

6.1.2 The Gigassembler

The Gigassembler operates in many steps. Before detailing the each step, an overview is given below.

1. Build merged sequence contigs, called *rafts*, from overlapping initial sequence contigs. See the paragraph "The Construction of rafts."
2. Build sequenced clone contigs, called *barges*, from overlapping clones. See the paragraph "Ordering clones for constructing barges."
3. Assign default coordinates to initial sequence contigs.
4. Build a *raft-ordering* graph. This is a directed graph with two types of nodes: rafts and sequenced clone end points. See the paragraph "The raft-ordering graph."
5. Add information from mRNAs, ESTs, paired plasmid reads, BAC end pairs, and ordering information from the sequencing centers. The resulting graph is called a *bridge graph*. See the paragraph "Bridging rafts with mRNA, ESTs, paired plasmid reads, BAC end data."
6. Walk the bridge graph to get an ordering of rafts. Each bridge is walked in the order of the default coordinates assigned in step 3 subject to the constraint that if a raft has predecessors, all the predecessors must be walked before the raft is walked.
7. A sequence path through each raft is built. See the paragraph "Building a sequence path."
8. Build the final sequence for the fingerprint clone contig by inserting the appropriate number of Ns between raft sequence paths.

The Construction of rafts A score to each aligning pair is assigned, and then the alignments are processed from the best scoring alignment to least ones. The raft building procedure is similar to the contig building procedure with analogy of rafts to contigs and with analogy of input contig sequences to fragment reads. The main issue is the scoring function. The scoring function strongly favors overlaps that are unique, weakly favors overlaps that are repeat masked, strongly discourages sequence mismatches and inserts within the aligning blocks, and moderately discourages tails (the shorter end of a contig in an alignment).

Ordering clones for constructing barges Barges are constructed in a similar fashion where the clone overlap is the sum of all initial sequence contig overlaps. The overlaps are used to order the clones in the following manner.

1. Clones that are completely enclosed by another clone are put aside.

2. A clone is selected and the most overlapping other clone is joined with it to initialize an ordered list of clones.

3. Given an ordered list of clones ABCD, though there are still clones in the fingerprint clone contig that significantly overlap clones in this list, the clone X that overlaps as much as possible with another element on the list is selected and inserted into the list as follows:

   (a) If X overlaps A but not B, the order becomes XABCD.

   (b) If X overlaps A and B, and overlap(A,B) $<$ overlap(A,X) and overlap(A,B) $<$ overlap(B,X) then the order becomes AXBCD. Here, overlap(A,B) is the total number of bases in the overlaps between the initial sequence contigs of A and the initial sequence contigs of B.

   (c) Otherwise, above two steps are repeated, shifting the list so that C is considered in place of B and B in place of A.

   (d) If there are still clones left in the fingerprint clone contig that have not been added to the ordered list after the iterations of the above steps cease, then a new barge is started to accommodate the remaining clones.

4. The order of the clones in each barge is compared to the fingerprint map, and if the barge looks backwards the order of its clones is reversed.

5. Barge coordinates are given in the following manner:

   (a) The first clone is given an offset of zero.

   (b) For the nth clone, Offset(n + 1) = Offset(n) + Size(n) Overlap(n,n + 1), where the size of a clone is defined as the total length of all its initial sequence contigs.

6. Clones that are completely enclosed are given the coordinate: Offset(inner) = Offset(outer) + (Size(outer) Size(inner))/2

The raft-ordering graph Once the orientation and order of clones are determined while constructing barges, rafts (merged sequence contigs) can be ordered. To understand what is happening at this stage, see Fig. 29.

Bridging rafts with mRNA, ESTs, paired plasmid reads, BAC end data This information is used to connect rafts in the raft ordering graph in a three-step process. The resulting graph is called *a bridge graph*.

1. Build a *bridge* out of alignments of other data with initial sequence contigs.

2. Scoring the bridge. The score function for bridges is the sum of two factors. The first factor is based on the type of the information. mRNA information is given the highest weight, then paired plasmid reads, information provided by the sequencing centers, ESTs, and BAC end matches, in that order.

3. Add bridges one at a time, best scoring first, to the ordering graph.

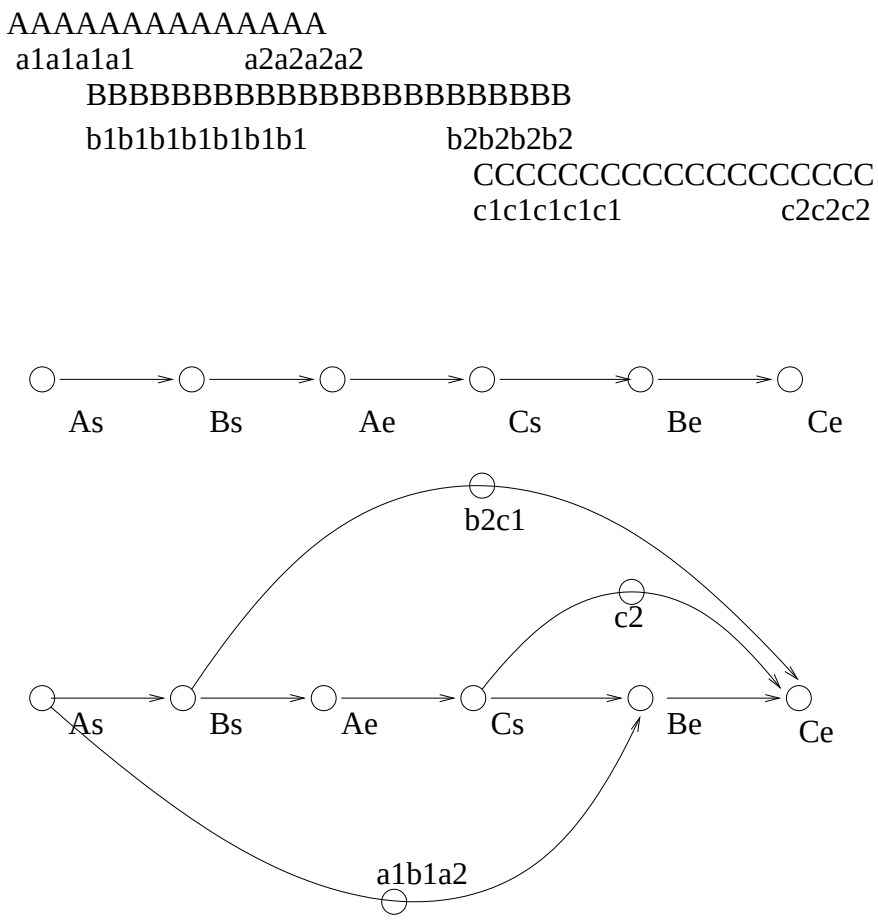

Figure 29: How to build a raft-ordering graph. Six initial contigs (a1,a2,b1,b2,c1,c2) are aligned to three clones (A,B,C) (top figure), an ordering graph of clone starts and ends is given (middle figure), and the final raft-ordering graph after adding in rafts to the ordering graph (bottom figure). Form the top figure, we can construct three rafts, a1b1a2, b2c1, and c2, based on their overlaps. Then an ordering graph of clone starts and ends can be constructed based on the positions of clone start and end positions as in the middle figure. The node names As and Ae denotes the start and the end of a clone A respectively. Finally, the three rafts are added to the ordering graph as in the bottom figure.

Building a sequence path

1. Find the longest, most finished initial sequence contig that passes though each section of the raft.
2. Put the best initial sequence contig for the first part of the raft into the sequence path.
3. Find an alignment between the best initial sequence contig for the first part of the raft and the best initial sequence contig for the second part of the raft.
4. Search for a 'crossover point' in the alignment where it would be reasonable to switch the sequence path to the next initial sequence contig. This crossover point is ideally 250 bases from the end of the larger, more finished initial sequence contig, but may be adjusted depending on the exact alignment.
5. Repeat steps 3 and 4 to extend the sequence path until the end of the raft is reached.

6.2 The CELERA Scaffolder

The CELERA Scaffolder[16] uses a greedy algorithm using the graph structure. The input data is CELERA scaffolds (see Section 4.11) and BAC sequences available in the public domain. The goal is to generate scaffolds of a genome using the both data. We briefly sketch the algorithm.

1. The bactig graph is constructed by a transitive reduction technique on bundled clone mate edge sets. See "Bactig graph construction" below.
2. Bactig orders are determined by a greedy path-merging algorithm. See "The path-merging algorithm" below.

6.2.1 Bactig graph construction

Bactigs are contiguous sequences from a common source region obtained by shotgun strategy and assembly process. In the input data, there are four different types of library clones of lengths, 2k, 10k, 50k, and 100k. The graph is constructed in three steps.

1. The initial graph: A bactig graph G is a weighted, undirected multi-graph, without self loops. Each bactig Bi gives rises to precisely two nodes v, w in the graph and an edge e connecting v and w. Then, an edge e is associated with a standard deviation $\sigma(e)$ of length variation.
2. Edge bundling: Let M denote the set of mate edges between nodes, v and w. Edges are bundled by first greedily choosing a median-length mate edge $e' \in M$, whose length $l(e')$ is within $3\sigma(e)$ of $l(e)$ and then bundling all of these with e. The weight of an edge e $w(e)$ is 1 if e is a simple mate edge and $\sum_{i=1}^{k} w(e_i)$ if e is obtained by bundling k mate edges $\{e_1, ..., e_k\}$.
3. Transitive reduction: This step is to transitively reduce *long* mate edges. Recall that there are four types of library clones of lengths, 2k, 10k, 50k, and 100k. By transitively reducing long edges from larger clone library, there is no jumping edges in the ordering, which can cause ambiguity in ordering. A mate edge e from v to w can be *transitively reduced* on to the path p if e and P have approximately the same length (three times of variation). If this is the case, the edge e is removed from the graph and the weight for every mate edge m_i in P is increased by $w(e)$. In this way, the final bactig graph is constructed.

6.2.2 The path-merging algorithm

First, *the bactig ordering problem* is defined and a heuristic algorithm called *a greedy path-merging algorithm* is proposed. They classified edges into *happy edges* and *unhappy edges*. The happy edge is an edge whose length is within three times of length variations. Otherwise, an edge is unhappy.[11]

The bactig ordering problem Given a bactig graph G, find an ordering ϕ of G that maximizes the sum of weights of happy edges.

The path-merging algorithm works as follows. Initially, all bactigs are selected. An edge e is selected in the descending order of weight. Then two paths $P1$ and $P2$ incident to e are considered for merging. If merging the two paths gains as much weight for happy edges as for unhappy edges, the two paths are merged. Refer to [16] for more detail).

6.3 An exploratory genome sequencing framework

A genome sequencing framework[25] is developed as an effort for microbial genomics project at DuPont, and has been successful in assembling several bacterial genomes including *Agrobacterium* [27]. A schematic overview of the whole procedure is depicted in Figure 30.

This search framework is not designed to determine single base reads accurately but to *search* for a set of correctly assembled contigs and their orderings. To achieve this task, two issues should be addressed:

1. How do we know whether contigs are assembled correctly? We developed several methods for identifying misassembled regions in contig assembly: clone coverage analysis [23] and a probabilistic approach [24].

2. How can we utilize clone mate information? When fragments are read, two end sequences of the same clone are usually read; this method is known as *double barreled sequencing* [31]. Several assembly packages were successful in incorporating the clone mate information into the assembler [44] and generation of scaffolds of contigs [48]. However, utilization of this information is quite limited mainly due to the fact that two end fragments of a clone, say of 10kb, are not connected and connecting the two end fragments by aligning other fragments becomes a small scale sequence assembly problem itself. In general, multiple library clone types of different sizes (2kb, 10kb, 40kb, and 100kb) are strategically used for genome-scale sequencing. This makes the utilization of clone information even harder. The framework in Figure 30 is designed to combinatorially enumerate sequence assemblies in an iterative, hierarchical fashion. The hierarchy is necessary to utilize clones of various length. An iterative procedure is necessary since we are not sure of contig ordering or clone layout. As the iterative procedure proceeds, it is the users who make a decision on whether the current assembly is correct or not; thus, the procedure needs user input.

This approach can be viewed as "*a hypothesis generation and validation paradigm*" in search of a set of correctly assembled contigs and their ordering. All decisions made at user interaction points are *hypotheses* that will be subsequently *tested* with larger clones in the next step. This approach was successful in assembling several genome sequences. For example, 502 contigs in the

[11] Interestingly, a contig validation method [23] classifies clones into *bad clones* and *good clones* (see Section 5.1 also).

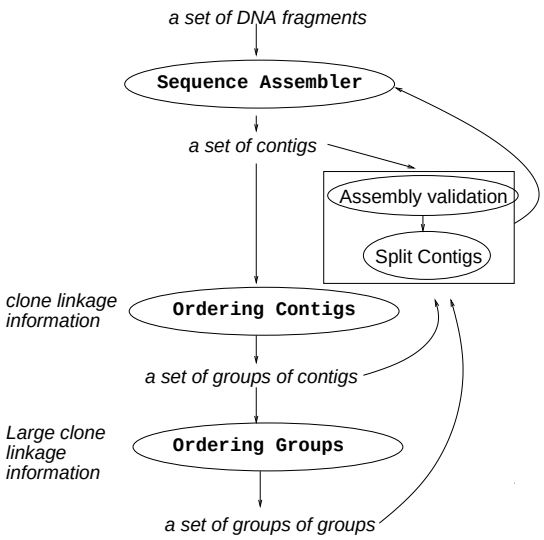

Figure 30: A framework for genome sequencing. This framework is to search for a set of correctly assembled contigs and their ordering in an iterative fashion.

Phrap assembly of the *Agrobacterium* shotgun data were grouped and ordered into only 15 sets of contigs (the largest set longer than 2Mb) using a web interface in a single iteration of our genome sequencing framework. Given that there are four replicons in *Agrobacterium* genome, this result is very encouraging and closer to complete sequencing.

7 Discussion

7.1 Trends in Genome Sequencing

Genome sequencing still remains an open problem. However, recent advances in computational techniques made it possible to sequence very large eucharyotic genomes including *Drosophila melanogaster* [30] and the Human genome [28, 48].

We will discuss recent recent trends in genome sequencing strategies.[12]

1. To achieve very large scale genome sequencing, it is necessary to have the input shotgun data of very high quality (see [30]). Otherwise, is is not possible to distinguish repeats from errors in the shotgun data. What is really interesting is that recent assemblers attempt to correct errors in the input shotgun data *before* sequence assembly [34, 3]. There is no guarantee to correct errors without knowing the target sequence. Indeed, EULER names this procedure as *data corruption* instead of error correction. Nonetheless this is a proven technique that works for difficult sequence assembly.

2. Repeat boundaries are identified before sequence assembly and contigs are assembled up to the boundaries [30, 3]. Like the error correction, there is no guarantee to identify the

[12]This discussion is preliminary and is not intended to extensively cover all issues in achieving such very large scale genome sequencing.

repeat boundaries correctly without knowing the target sequence, but this is another proven technique.

3. Computational techniques to ensure the correctness of contigs assembly becomes more important. Correctness can be checked using the characteristics of the shotgun data, i.e., random sampling.[13] There are two ways to utilize the characteristics of data, on the fragment level [30, 3, 15, 44] and on the clone level [3, 15, 23]. There is also an interesting approach based on pattern statistics [24].

4. Clone-mate information and base call quality values become an essential data for genome sequencing.

7.2 Incompleteness of the survey

Genome sequencing is a very important problem in biology but it is a very difficult computational problem. As a result, there has been a lot of research on computational techniques. Although we surveyed 13 different assembly programs and several computational techniques on sequence assembly validation and scaffold generation, this survey is not complete by any means. There are many assemblers we wished to include. Among them are STROLL [6], a trie-based sequence assembler, and the Staden package [39], which is also widely used sequence assembly package. In addition, the genome sequencing is still unresolved[14] and more sophisticated techniques will be continuously developed.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) "Basic local alignment search tool," *Journal of Molecular Biology* **215** 403-410.

[2] Altschul, S.F., Madden, T.L., Schffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, **25** 3389-3402.

[3] S. Batzoglou, D. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J.P. Mesirov, and E.S. Lander, "Arachne: A Whole-Genome Shotgun Assembler," *Genome Research*, vol. 12, no. 1, 2002, pp. 177-189.

[4] Bonfield, J. K., K. Smith, and R. Staden. 1995. "A new DNA sequence assembly program," *Nucleic Acids Research* 23:4992-4999.

[5] C. Cantor, M. S. Mirzabekov, and E. Southern, 1992, "SBH: An idea whose time has come", *Genomics*, 11.

[6] Ting Chen and Steven Skiena, "Trie-based data structures for fragment assembly," *The Eighth Symposium on Combinatorial Pattern Matching*, Aarhus, Denmark, June 30 - July 2, 1997.

[13] There are regions where sampling is biased due to the biological reasons. However, random sampling can be assumed as a whole shotgun data.

[14] Interested users may read three articles on Human genome sequencing [51, 52, 53].

[7] Brent Ewing, LaDeana Hillier, Michael C. Wendl, and Phil Green, "Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment," *Genome Research* 1998 8: 175-185

[8] Brent Ewing and Phil Green "Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities," *Genome Research* 1998 8: 186-194

[9] Fleischmann, R.D., Adams, M.D., White, O., Clayton, R.A., Kirkness, E.F., Kerlavage, A.R., Bult, C.J., Tomb, J.F., Dougherty, B.A., Merrick, J.M., McKenney, K., Sutton, G., FitzHugh, W., Fields, C., Gocayne, J.D., Scott, J., Shirley, R., Liu, L.I., Glodek, A., Kelley, J.M., Weidman, J.F., Phillips, C.A., Spriggs, T., Hedblom, E., Cotton, M.D., Utterback, T.R., Hanna, M.C., Nguyen, D.T., Saudek, D.M., Brandon, R.C., Fine, L.D., Fritchman, J.L., Fuhrmann, J.L., Geoghagen, N.S.M., Gnehm, C.L., McDonald, L.A., Small, K.V., Fraser, C.M., Smith, H.O. and Venter, J.C. (1995). Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science* 269(5223): 496-512.

[10] David Gordon, Chris Abajian, and Phil Green "Consed: A Graphical Tool for Sequence Finishing," *Genome Research* 1998 8: 195-202

[11] David Gordon, Cindy Desmarais, and Phil Green "Automated Finishing with Autofinish," *Genome Research* 2001 11: 614-625

[12] Dan Gusfield, *Algorithms on strings, trees, and sequences* Cambridge University Press, 1997

[13] X. Huang, 1992, "A Contig Assembly Program Based on Sensitive Detection of Fragment Overlaps", *Genomics*, 14.

[14] X. Huang, 1996, "An Improved Sequence Assembly Program ", *Genomics*, 33.

[15] X. Huang and A. Madan, "CAP3: A DNA Sequence Assembly Program," *Genome Research*, vol. 9, no. 9, 1999, pp. 868-877.

[16] D. H. Huson, K. Reinert, and E. Myers, "The Greedy Path-Merging Algorithm for Sequence Assembly," RECOMB 2001, pp 157 - 163

[17] Ramana Idury and Michael S. Waterman, "A New Algorithm for DNA Sequence Assembly," *Journal of Computational Biology*, 2: 2, 291-306

[18] S. Karlin and S. Altschul, 1990, "Methods For Assessing The Statistical Significance Of Molecular Sequence Features By Using General Scoring Scheme", *Proceedings of National Academy of Science*, 87

[19] J. D. Kececioglu and E. W. Myers, 1995, "Combinatorial Algorithms for DNA Sequence Assembly", *Algorithmica*, 13

[20] 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.

[21] Sun Kim and Alberto Maria Segre "AMASS: A Structured Pattern Matching Approach to Shotgun Sequence Assembly," *Journal of Computational Biology*, Vol 6 (4), 1999

[22] Sun Kim, Li Liao, and Jean-Francois Tomb, "Enumerating Repetitive Sequences From Pairwise Sequence Matches," manuscript, DuPont Central Research, 2000

[23] Sun Kim, Li Liao, Michael Perry, Shping Zhang, and Jean-Francois Tomb, "A Computational Approach to Sequence Assembly Validation," manuscript, DuPont Central Research

[24] Sun Kim, Li Liao, and Jean-Francois Tomb, "A Probabilistic Approach to Sequence Assembly Validation," *ACM SIGKDD Workshop on Data Mining in Bioinformatics (BioKDD2001)*, 2001, pp 38-43

[25] Sun Kim, "The AMASS Genome Sequencing Package," *Advances in Genome Biology and Technology Conference*, Feb. 2002

[26] Sun Kim and Yanggon Kim, "A Fast Multiple String-Pattern Matching Algorithm," *Proc. of 17th AoM/IAoM Conference on Computer Science*, August 1999

[27] Derek W. Wood, Joao C. Setubal, Rajinder Kaul, Dave E. Monks, Joao P. Kitajima, Vagner K. Okura, Yang Zhou, Lishan Chen, Gwendolyn E. Wood, Nalvo F. Almeida Jr., Lisa Woo, Yuching Chen, Ian T. Paulsen, Jonathan A. Eisen, Peter D. Karp, Donald Bovee Sr., Peter Chapman, James Clendenning, Glenda Deatherage, Will Gillet, Charles Grant, Tatyana Kutyavin, Ruth Levy, Meng-Jin Li, Erin McClelland, Anthony Palmieri, Christopher Raymond, Gregory Rouse, Channakhone Saenphimmachak, Zaining Wu, Pedro Romero, David Gordon, Shiping Zhang, Heayun Yoo, Yumin Tao, Phyllis Biddle, Mark Jung, William Krespan, Michael Perry, Bill Gordon-Kamm, Li Liao, Sun Kim, Carol Hendrick, Zuo-Yu Zhao, Maureen Dolan, Forrest Chumley, Scott V. Tingey, Jean-Francois Tomb, Milton P. Gordon, Maynard V. Olson, and Eugene W. Nester, "The Genome of Agrobacterium tumefaciens C58: Insights into the evolution and biology of a natural genetic engineer," *Science* , Dec 14 2001: 2317-2323

[28] Eric S. Lander, Lauren M. Linton, Bruce Birren, *et. al.* "Initial sequencing and analysis of the human genome," *Nature*, 409, 860 - 921 (15 Feb 2001)

[29] Gene Myers, Susan Miller, and Eric Anson, `http://www.cs.arizona.edu/faktory/`

[30] G. Myers, G. Sutton, A. L. Delcher, et. al., "A Whole-Genome Assembly of Drosophila," *Science*, Vol 287, March 2000, pp 2196 - 2204.

[31] J.C.Roach, C. Boysen, K. Wang, and L. Hood, "Pairwise end sequencing: a unified approach to genome mapping and sequencing," *Genomics*, 26:345-353, 1995

[32] Maynard Olson and Phil Green, "A "Quality-First" Credo for the Human Genome Project," *Genome Research*, Vol. 8, Issue 5, 414-415, May 1998

[33] R. J. Parsons, S. Forrest, and C. Burks, 1995, "Genetic Algorithms, Operators, and DNA Fragment Assembly", *Machine Learning*, 21.

[34] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman "An Eulerian path approach to DNA fragment assembly," *PNAS* 2001 98: 9748-9753.

[35] P. A. Pevzner and R. J. Lipshutz, 1994, "Towards DNA Sequencing Chips", *19th Symposium on Mathematical Foundation in Computer Science*, 841.

[36] Phil Green, `http://www.phrap.org`

[37] Pearson, W. R. and Lipman, D. J. (1988) "Improved tools for biological sequence comparison," *Proc. National Academy of Science*, **85** 2444-2448

[38] Mihai Pop, Steven L. Salzberg, and Martin Shumway "Genome Sequence Assembly:Algorithms and Issues," *IEEE Computer*, 35(7), pp. 47-54, July 2002 (Vol. 35, No. 7)

[39] Rodger Staden, David P. Judge and James K. Bonfield "Sequence Assembly and Finishing Methods," In *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, Second Edition Eds. Andreas D. Baxevanis and B. F. Francis Ouellette. John Wiley & Sons, New York, NY, USA.

[40] Arian Smit, `http://repeatmasker.genome.washington.edu/`

[41] T. F. Smith and M. S. Waterman, 1981, "Identification of Common Molecular Sequences", Journal of Molecular Biology, 147

[42] Eric C. Rouchka and David J. States, "Sequence Assembly Validation by Multiple Restriction Digest Fragment Coverage Analysis" *Proc. of Intelligent Systems for Molecular Biology (ISMB)*, 1998, pp 140 - 147, AAAI Press

[43] G. Syswerda, 1989, "Uniform Crossover in genetic Algorithm", *Proc. of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann

[44] G.G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage, "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects," *Genome Science and Technology*, 1995, vol. 1, pp. 9-19

[45] J. Tarhio and E. Ukkonen, 1988, "A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings", *Theorectical Computer Science*, 57.

[46] J. Turner, 1989, "Approximation Algorithms for the Shortest Common Superstring Problem", *Information and Computation*, 83.

[47] P. H. Winston, 1992, *Artificial Intelligence*, 3rd ed. Addison Wesley.

[48] J. Craig Venter, Mark D. Adams, Eugene W. Myers , et. al. "The Sequence of the Human Genome," *Science* 2001 February 16; 291: 1304-1351.

[49] E. Ukkonen, 1990, "A Linear Algorithm for Finding Approximate Shortest Common Superstings", *Algorithmica*, 5.

[50] James L. Weber and Eugene W. Myers "Human Whole-Genome Shotgun Sequencing," *Genome Research*, 1997 7: 401-409.

[51] Robert H. Waterston, Eric S. Lander, and John E. Sulston, "On the sequencing of the human genome," *PNAS* 2002 99: 3712-3716

[52] Phil Green, "Whole-genome disassembly," *PNAS* 2002 99: 4143-4144

[53] Eugene W. Myers, Granger G. Sutton, Hamilton O. Smith, Mark D. Adams, and J. Craig Venter "On the sequencing and assembly of the human genome," *PNAS* 2002 99: 4145-4146