JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 2, Number 2, 1995
Mary Ann Liebert, Inc.
Pp. 291–306

A New Algorithm for DNA Sequence Assembly

RAMANA M. IDURY and MICHAEL S. WATERMAN

ABSTRACT

Since the advent of rapid DNA sequencing methods in 1976, scientists have had the problem of inferring DNA sequences from sequenced fragments. Shotgun sequencing is a well-established biological and computational method used in practice. Many conventional algorithms for shotgun sequencing are based on the notion of pairwise fragment overlap. While shotgun sequencing infers a DNA sequence given the sequences of overlapping fragments, a recent and complementary method, called sequencing by hybridization (SBH), infers a DNA sequence given the set of oligomers that represents all subwords of some fixed length, k. In this paper, we propose a new computer algorithm for DNA sequence assembly that combines in a novel way the techniques of both shotgun and SBH methods. Based on our preliminary investigations, the algorithm promises to be very fast and practical for DNA sequence assembly.

INTRODUCTION

DRAMATIC PROGRESS IN MOLECULAR BIOLOGY has been driven and guided by knowledge of DNA. The modern history began in 1953 with the discovery of the three-dimensional structure of DNA by Crick and Watson. Isolation of restriction enzymes and DNA polymerases made possible DNA sequencing, the determination of the base sequence along a strand of DNA. The modern era of "rapid" DNA sequencing began in 1977 with the development of two techniques: dideoxy chain termination (Sanger *et al.*, 1977) and chemical degradation (Maxam and Gilbert, 1977). Beginning with Smith *et al.* (1986), DNA sequencing machines were developed that automated gel electrophoresis, data acquisiton, and base determination. Whether the sequence of the individual fragments comes from a gel read by humans or from a sequencing mahine, the general methodology of inferring the complete DNA sequence from which the sequence fragments are drawn has remained the same since 1977. We now describe the computational task in more detail.

The DNA molecule to be sequenced is L base pairs (bp) in length. The sequence of one strand is $\mathbf{a} = a_1 a_2 \cdots a_L$. Generated at random are N shorter fragments $f_1, f_2, \ldots, f_N$ each of approximate length $l \ll L$ whose base sequences can be determined experimentally. All of these fragments are contained in $\mathbf{a}$, but the relative order of these fragments is unknown. Further complications arise because each fragment f_i is only known approximately and because the orientation of f_i is unknown. That is, with probability 1/2,

Departments of Mathematics and Molecular Biology, University of Southern California, Los Angeles, CA 90089-1113.

f_i is written in the polarity of **a** and with probability 1/2 f_i is read as the reverse complement, f_i^r, of its location in **a**. In addition, many sequences are known to contain exact, as well as approximate, repetitive regions. If a repetitive region is long enough, then no single fragment may contain the entire region and we will not be able to place the repetitive region correctly with respect to the other regions. All of these features complicate sequence assembly.

To alleviate these complciations, the sequence **a** is redundantly sampled, at an average depth $c = Nl/L$. The experimentalist typically chooses $c \in [3, 10]$. Assuming that the locations of fragments are distributed uniformly along the sequence, the depth of coverage at a given base t is a Poisson random variable with mean c:

$$\mathbb{P}(\text{base } t \text{ is covered by } k \text{ fragments}) = \frac{e^{-c}c^k}{k!}.$$

The fraction of **a** that is covered by at least one fragment is $1 - e^{-c}$ (Lander and Waterman, 1988). This shows how the fraction of **a** covered increases with c.

If we have prior knowledge of the underlying sequence **a**, then we can determine the relative order and orientation of fragments, and also pinpoint their errors. In the absence of this knowledge, however, one must infer the relative order of fragments from the base sequences of fragments alone. One natural way to do this is by using the overlap information between the right end of one fragment with the left end of another fragment. The idea of shotgun sequencing is to look for overlapping fragments and solve the puzzle of **a**'s sequence by assembling the sequence from these overlaps.

Fragment assembly algorithms began with Staden (1980) and have been extended and elaborated by others (Peltola *et al.*, 1984; Kececioglu, 1991; Huang, 1992; Kececioglu and Myers, 1995). The general outline is much the same in these algorithms: (i) Examine all pairs (f_i, f_j) and (f_i, f_j^r) for significant overlap; (ii) form an approximate layout of the fragments using the data in (i); (iii) make a multiple alignment of the layout from (ii) and read **a** as a consensus sequence from the multiple alignment.

Steps (i), (ii), and (iii) need not be distinct. For example, fragments are overlapped and merged in a sequential manner in Staden's original algorithm, so that at any stage a new fragment is tested against all merged fragments. Using this method, the final alignment is automatic. This procedure is easier to code than other methods that consider all fragments simultaneously, but has the feature of being dependent on the order of entry of the fragments. The research on these algorithms has concentrated on ways to speed up steps (i), (ii), and (iii) as well as making them more rigorous.

It is well known that sequence assembly problem is NP-hard (Gallant *et al.*, 1980; Gallant, 1983; Kececioglu, 1991) so all existing methods rely on heuristics to combine fragments. As such, the success of any method is based on how well its heuristic takes advantage of the underlying structure of the problem and specific features peculiar to the problem at hand. Moreover, it must be emphasized that no single heuristic can outperform all others in all problem instances.

The tradition of mathematics and computer science is to solve precisely stated problems. Methods can then be compared to one another by optimality criteria and by running times. For example, in multiple sequence alignment, performance guarantee algorithms have been developed using the sum of pairs optimality criterion (Gusfield, 1991; Pevzner, 1992). This requires agreement on what an optimal multiple alignment is; in practice, these methods do not often produce alignments that are of interest to biologists studying a set of real sequences. Instead, the methods teach us new things about multiple alignment that are of interest in their own right and that may someday contribute to methods that will prove useful in practice. Sequence assembly provides another illustration of the difficulty of setting optimality criteria that "agree" with the biologist's criterion, which is simply that the assembly program produce a sequence identical with that of the DNA molecule **a**. Of course it is difficult to address directly this very natural but very illusive criterion. In the case of assembly by steps (i–iii) above, Kececioglu and Myers (Kececioglu, 1991; Kececioglu and Myers, 1995) have made a great deal of progress in identifying well-posed optimality criteria and in developing methods to solve them. This active area of research is of interest to computer scientists and also to biologists in that programs have been written that take fragment data and produce assembled sequence. Our intent in this paper is to describe a different heuristic method for DNA sequence assembly. Methods of computer science are employed but we do not pose any strict model of optimality. Our goal is to create a method that will correctly infer the correct sequence **a**, and we believe that the only tests are (i) the quality of the sequence produced by the program when it is measured against the correct sequence and (ii) the running time of the program. Naturally this can only be shown in practice.

In this paper, we propose a new algorithm for DNA sequence assembly using a different strategy from the previous methods. Based on preliminary investigations, our method promises to be very fast and practical for DNA sequence assembly. Our algorithm takes advantage of several key features of the sequence data. First of all, the redundancy or the depth of coverage c is often much larger than 2. Thus, when only pairwise comparisons are considered, information about multiple overlaps of fragments is ignored, which leads to complications in reconstruction process. Second, the fragments are sequenced with errors but usually are 98% accurate. Therefore, there are very likely to be long stretches with no errors. In considering the basic data for shotgun sequencing, we have come to a very distinct strategy that takes advantage of these properties. Before we describe our strategy, we will give a sketch of the computer science associated with what seems at first to be an entirely different problem, sequencing by hybridization.

SEQUENCING BY HYBRIDIZATION

Recently, a new approach to sequencing DNA was presented, sequencing by hybridization or SBH for short (Drmanac and Crkvenjakov, 1987; Bains and Smith, 1988; Lysov *et al.*, 1988; Southern, 1988; Macevicz, 1989). The idea is to build a two-dimensional grid or matrix of all k-tuples or probes, where a k-tuple is a word of length k. This matrix of probes will be referred to as a sequencing chip. Then a sample of the single-stranded DNA to be sequenced is presented to the sequencing chip. This DNA is labeled with a radioactive or fluorescent material. Each k-tuple present in the sample is hybridized with its reverse complement in the matrix. Then, when unhybridized DNA is removed from the matrix, the hybridized k-tuples can be determined with a device detecting the labeled DNA. In Fig. 1, we present a grid of the $4^3 = 64$ 3-tuples along with the hybridization results from $\mathbf{a}$ = ATGTGCCGCA.

Although we assume perfect hybridization data for the sake of exposition in this paper, there are some technical difficulties with this method, both experimental and mathematical. A major experimental difficulty results from the hybridization between complementary k-tuples. Some k-tuples, such as those with high G-C content, hybridize more strongly than others. In addition, there can be imperfect hybridization involving mismatches. This contributes to errors in the SBH data, where k-tuples in the sequence are not hybridized or where hybridization signals occur where they should not. Another experimental implication for SBH data is that multiplicities are not detected. That is, when a k-tuple occurs more than once, we can only learn that it occurred at least once but cannot tell how many times it occurred. So far SBH is not a practical method to sequence DNA, although some limited success has been reported in pilot experiments (Drmanac, 1944).

The mathematical aspects of SBH are also nontrivial. The development we give is due to Pevzner (1989). For example, if k is too small then the SBH data becomes too dense to find the sequence. It is, for example, not useful to learn that all 16 2-tuples are present in a DNA sequence 100 bases in length. In these cases, the data will not allow us to resolve the sequence. Obviously, it is an advantage to take k as large as possible. In fact, if the sequence to be determined is of length L, it would be ideal to have a grid of all 4^L L-tuples. Then the sequence could be read from just one hybridization on the matrix. Obviously, this is experimentally impractical, and the goal is to resolve as much sequence as possible from the k-tuple matrix. Technologists work on increasing the feasible size of k, currently $k = 8$ has been constructed and perhaps $k = 10$ can be obtained in the near future.

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

FIG. 1. Matrix of 3-tuples and hybridization results.

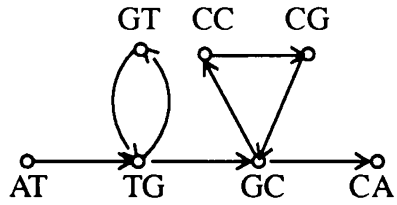


FIG. 2. Graph G for ATGTGCCGCA.

To summarize, we have data $\mathbb{1}_w = 1$ if w is a substring of **a** and $\mathbb{1}_w = 0$ if w is not, for each k-tuple w. Hybridization data is nonrandom and of quite distinct character. We define the spectrum of **a,** denoted by $S(\mathbf{a})$, as the set of k-tuples w such that $\mathbb{1}_w = 1$. In other words, $S(\mathbf{a}) = \{w : w = a_i a_{i+1} \cdots a_{i+k-1}, 1 \leq i \leq L - k + 1\}$. We build a graph on the spectrum of **a** as follows.

The directed graph G has as vertex set V, the set of $(k-1)$-tuples from the spectrum, and as edge set E, the set of (k-tuples from the spectrum, where each k-tuple of the spectrum contains two $(k-1)$-tuples. The $(k-1)$-tuple u is joined by a directed edge to v if the spectrum S contains a k-tuple whose first $(k-1)$-tuple is u and second $(k-1)$-tuple is v. An example of the directed graph G is shown in Fig. 2. We call G a spectrum graph for convenience. There is a natural interpretation from paths in G to sequences. For example, while an edge, which is a path of length one, specifies a sequence of length k, a path of length two specifies a sequence of length $k+1$, and in general, a path of length m specifies a sequence of length $k+m-1$. An Eulerian path visiting every edge exactly once gives a solution to the SBH problem.

Euler's theorem for directed graphs gives conditions for the existence of an Eulerian path. Define for vertex v,

$$in(v) = |\{u \mid [u, v] \in E\}|,$$
$$out(v) = |\{w \mid [v, w] \in E\}|.$$

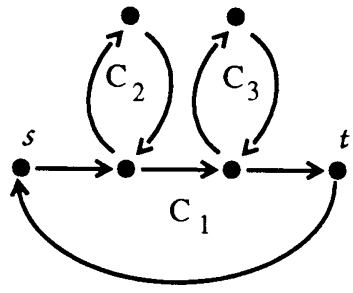
Label the starting vertex s and the terminal vertex t. There is an Eulerian path starting at s and ending at t, if and only if

$$in(v) = out(v) \quad \forall v \neq s, t,$$
$$out(s) - in(s) = 1,$$
$$out(t) - in(t) = -1.$$

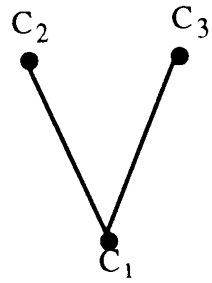
Since, each Eulerian path corresponds to a different DNA sequence, we can infer the sequence unambiguously if and only if the number of Eulerian paths in G is exactly one. We can reduce the problem of determining the number of Eulerian paths from s to t in G to the problem of determining the number of Eulerian cycles in the graph $G \cup [t, s]$, which has a simple solution described below.

We define an intersection graph on the cycles of $G \cup [t, s]$ as follows. First decompose $G \cup [t, s]$ into simple cycles: $v_{i_1} \to v_{i_2} \to \ldots \to v_{i_k} = v_{i_1}$, where no $v_i = v_j$ except for $v_{i_k} = v_{i_1}$. An edge can be used in at most one cycle C but a vertex can be used in arbitrarily many cycles. For these cycles, define the intersection graph G_I of the cycles $C_1, C_2, \ldots, C_m$ where cycles correspond to vertices, and if cycles C_i and C_j have l vertices in common, we connect them by l edges in G_I.

Returning to graph G in Fig. 2, the simple cycles in $G \cup [t, s]$ look like



so the graph G_I is



which is a tree, because G_I has no cycles. One may verify that $G \cup [t, s]$ above has only one Eulerian cycle and hence G in Fig. 2 has only one Eulerian path. The following theorem gives the necessary and sufficient conditions for general graphs.

Theorem 1 (Pevzner, 1989). *There is a unique Eulerian cycle in G if and only if the intersection graph G_I of simple cycles from G is a tree.*

A NEW ALGORITHM FOR SHOTGUN SEQUENCING

The basic idea of our new algorithm is to combine the mathematical ideas of SBH with those of shotgun sequencing. Because we already know the fragments, we can just read along a fragment f_i and determine all its k-tuples or the spectrum $S(f_i)$, where k is chosen by the person setting up the algorithm. If the error rate is 2% then, on the average, one expects one error in 50 bp. This means that we will have $50-k$ correct k-tuples in every 50 bp of fragment data. With $k \in [10, 20]$ this gives a substantial fraction of correct data. If we have the ideal data, that is, if (i) there are no sequencing errors, (ii) the entire sequence **a** is covered with fragments, and (iii) the overlap between adjacent fragments is at least k, then the union $\cup_i S(f_i)$ of all k-tuples obtained from all fragments will be the same as the spectrum $S(\mathbf{a})$. With reasonable depth of coverage and error rate, $\cup_i S(f_i)$ can be expected to contain a very high fraction of the k-tuples of $S(\mathbf{a})$.

Our idea is to build a graph similar to the spectrum graph on the k-tuples of $\cup_i S(f_i)$. Once we have such a graph, we can perform Eulerian tours to infer the underlying sequence **a**. Because we generate the k-tuples directly from the fragments ourselves, we can set k as high as feasible even though an experimentalist cannot possibly obtain a corresponding sequencing chip because of technological limitations. Moreover, for each k-tuple or "probe," we know the fragment(s) and position(s) where it has come from. As such, we can label our k-tuples with this information. With this additional information, our scheme still retains all the information present in the original fragments. Current technologies of SBH can only tell whether a given k-tuple is present in a sequence but cannot give the location(s) of occurrence. In this regard, our approach attempts to combine the advantages of both shotgun and SBH methods while keeping their disadvantages to a minimum.

If we have the ideal data as described by the conditions (i), (ii), and (iii) above, and the underlying sequence contains no repeats of length k or greater, then we can reproduce the underlying sequence by choosing an appropriate value of k. However, we need to deal with the problems of sequencing errors, longer repetitive regions, and unknown fragment orientation to make our method practical.

To avoid the troublesome feature of fragment orientation, we simply enlarge the fragment data by a factor of two to include f^r as well as f. As we shall see later, our algorithm will then produce two complementary sequences that correspond to the complementary strands of the DNA sequence. This slows the algorithm by a factor of 2 but provides a very simple solution to the orientation problem. In the rest of this paper, we assume that we produce only one strand of the sequence.

Repetition of a k-tuple can occur for different reasons. First, a k-tuple may occur at multiple locations within a single fragment. This problem can be handled with the help of fragment and position information. When we perform the Eulerian tour, we can make sure that edges corresponding to adjacent positions of a fragment are visited successively. Moreover, we can modify the Eulerian tour by allowing multiple visits to the edges corresponding to repetitions. Second, a k-tuple may repeat because it corresponds to a repetitive region in different fragments. In this case, the corresponding edge contains significantly more number of fragment positions than its nonrepetitive neighbors.

Sequencing errors create extraneous edges that correspond to tuples resulting from those errors. Thus, while repetitions cause multiple visits to some edges, sequencing errors create some edges that should not be visited at all. Therefore, what we perform is not an exact Eulerian tour, but a variation of it, in which we make multiple visits to some edges and do not visit some edges at all. Later, we describe how we make these decisions when visiting the edges of the graph.

Once we perform a variant of the Eulerian tour sketched above, we can print the corresponding inferred underlying sequence **a**. Biologists, however, require a multiple alignment, both to check the algorithm and to check their basic fragment data. We can do this by a three-step process as follows. First, for each fragment, we apply the hashing methods (Dumas and Ninio, 1982; Wilbur and Lipman, 1983; Pevzner and Waterman, 1995) to see where a fragment might align well to the inferred sequence. This gives the candidate diagonals for alignment. For moderate sequencing error rates, the fragment fits into the underlying sequence with a very high similarity score. This means that we will be able to perform dynamic programming to align the fragment to the sequence by restricting ourselves to a narrow band along the diagonal. Our second step, then, is to apply dynamic programming restricted to a narrow band along a diagonal for each fragment. Once we have fitted each fragment to the inferred sequence, we can tell, for each position of the inferred sequence, whether or not the fragment is in agreement with the inferred sequence at that location. For our third and last step, we look at each position of the inferred sequence, and take a consensus among the fragments covering that position. Because we only perform the dynamic programming along diagonal bands rather than whole matrices, the multiple alignment step is very fast, proportional to the sum of lengths of fragments times the average width of the band.

We now summarize our algorithm.

Algorithm 1 (Assembly) `Set.` N, k: `input:` $f_1, f_2, \ldots, f_N$.

1. Obtain the union of spectrum of all fragments and their reverse complements.
2. Construct the spectrum graph on $(k-1)$-tuples for the k-tuples from 1. Each edge is labeled with the fragment and position information.
3. Perform a variant of Eulerian tour(s) and infer the sequence(s).
4. Align the fragments to sequence(s) produced by (3).

THE SEQUENCE GRAPH

Because the graph that we construct in our algorithm is not exactly the same as the spectrum graph constructed in an SBH problem, we call it a sequence graph to make the distinction explicit. The distinction comes from the fact that: (i) our k-tuples are generated by several small fragments rather than a single large fragment (as would be the case in SBH), and (ii) the edges in a sequence graph are labeled with the fragment and position information which is nonexistent in SBH.

We define $G = (V, E)$ on the fragments as follows. The vertices of the graph correspond to distinct $(k-1)$-tuples occurring in the fragments. That is, for any $v \in V$, there exists at least one fragment f and position i such that the $(k-1)$-tuple corresponding to v is $a_i^f \ldots a_{i+k-2}^f$ or $a_{i+1}^f \ldots a_{i+k-1}^f$. We let $tup(v)$ denote the tuple corresponding to v.

The edges of a sequence graph initially correspond to distinct k-tuples occurring in the fragments. That is, for any $e \in E$, there exists at least one pair of fragment-position values (f, i) such that the k-tuple corresponding to e is $a_i^f \ldots a_{i+k-1}^f$. We define $tup(e)$ as the string that corresponds to the k-tuple initially. When we make certain reductions on the sequence graph, to be described later, we may replace the edges on a path in the initial sequence graph with a single new edge. As a result, $tup(e)$ can, in general, represent a string obtained by visiting the path.

Edges are directed such that if an edge $e = [u, v]$ is directed from the vertex u to the vertex v and $tup(e) = a_1 \ldots a_m (m \geq k)$, then $tup(u) = a_1 \ldots a_{k-1}$ and $tup(v) = a_{m-k+2} \ldots a_m$.

We let $occ(e)$ denote the set of all regions or substrings (of length $\geq k$) of fragments contained in $tup(e)$. That is, $occ(e) = \{(f, i, j) \mid a_i^f \ldots a_j^f$ is a substring of $tup(e)\}$. Each edge e is labeled with the set $occ(e)$. Note that if a substring of length $\geq k$ is repeated in a fragment, then all its occurrences are recorded in the occ information. The weight of an edge e, denoted by $wt(e)$, is usually set to the sum $\sum_{(f,i,j) \in occ(e)} (j - i - k + 2)$. Note that this scheme gives weight to regions or substrings solely based on their length. It is possible to give different weights to regions of same length, depending on the fragment and

position they come from. For example, one may assign more weight to middle locations of the fragments than the ends since most of the sequencing errors are known to occur at the ends.

For any vertex v, we define $In(v) = \{u \mid [u, v] \in E\}$, and similarly $Out(v) = \{w \mid [v, w] \in E\}$. For any edge e, let $begin(e) = \{(f, i, j) \in occ(e) \mid i = 1\}$. That is $begin(e)$ denotes those regions of $occ(e)$ that occur at the beginning of a fragment. Similarly, we can define $end(e)$ to denote the substrings that occur at the end of a fragment.

Let $ie = [u, v]$ and $oe = [v, w]$ be two edges incident on the vertex v in a sequence graph. Given a subset $il \subseteq occ(ie)$, we define the continuation subset of $occ(oe)$ with respect to il, denoted by $cont(il, oe)$, as $\{(f, i, j) \in occ(oe) \mid (f, m, i + k - 2) \in il$ for some $m\}$. In other words, the positions of the region of any fragment in the continuation subset must immediately follow the positions of a region of the same fragment in il (the regions overlap in $k - 1$ bases that correspond to the $(k - 1)$-tuple of v). We define the overlap of oe with respect to il, denoted by $ov(il, oe)$ to be $|cont(il, oe)|$. Similarly, we can reverse the direction of continuity, and define $cont(ie, ol)$ and $ov(ie, ol)$, for a subset $ol \subseteq occ(oe)$.

Continuation information enables us to perform the required Eulerian tour. Let us assume that there are no sequencing errors. During the tour, we maintain a list of active regions that is initially set to the occ of the first edge on the tour. Suppose we have reached the vertex v from u using ie during the tour, and $il \subseteq occ(ie)$ is the active list of regions. If there is only one outgoing edge $[v, w]$ from v, then we take that edge next and change the active list to $cont(il, [v, w]) \cup begin([v, w])$; $begin([v, w])$ is added because new fragments can begin anywhere along the path. If there are multiple outgoing edges from v then we take the edge oe with maximum overlap $ov(il, oe)$, and change the active list to $cont(il, oe) \cup begin(oe)$. Because continuation information assures that edges corresponding to adjacent regions of fragments are visited successively, fragments will be aligned continuously in the inferred sequence. Since we allow subsets (as opposed to the entire set) of $occ(e)$ to be put on the active list during a visit to e, multiple visits to e can be made, each time with an active subset disjoint from the previous ones.

As noted earlier, sequencing errors create extraneous edges that must be detected and eliminated. At first glance, it may seem that it is extremely difficult, if not impossible, to perform this task. However, in most sequencing projects, the structure of the sequence graph is much simpler than it may seem. We have observed, both from simulated and real data, that in almost all cases the number of vertices with indegree and outdegree of at most one each, is at least 90% of the total number of vertices generated. Even among the remaining vertices, the number of vertices with either indegree or outdegree of only one is very often at least 70%. The details of these structural aspects will be presented in the next section.

These observations enable us to perform a series of reductions on the sequence graph, to be described below. These reductions eliminate most of the extraneous edges. The heart of the algorithm performs these reductions until no further reductions are possible. The resulting graph, after making all possible reductions, is often very small and simple in structure. This graph, called the irreducible sequence graph, has some very important consequences which we shall discuss in the section Irreducible Sequence Graphs.

STRUCTURE OF A SEQUENCE GRAPH

In this section, we state and prove bounds on the number of several types of vertices. We also give some experimental results to validate our claims. We first give the definitions of some types of vertices.

A singleton is a vertex whose indegree and outdegree are at most one each. A fork is a vertex for which one of indegree or outdegree is at most one, and the other is more than one. We also define the identifying edge of a fork as that edge corresponding to the degree of one. Finally, a cross is a vertex for which both indegree and outdegree are more than one.

For the purpose of analysis, we shall make certain simplifying assumptions, some of which can be relaxed with a deeper analysis. Although the analysis that we present using these assumptions is simple and preliminary, it provides a basis for the observations made earlier that lead to the graph reductions in the next section. Moreover, the estimated values obtained from our analysis are consistent with observed values from experiments on simulated as well as real sequencing data. The assumptions are listed below.

1. Errors are uniformly distributed both over all fragments, and over the entire length of each fragment.
2. The error rate is small.

3. For any position i of **a**, the number of fragments covering the region $i \cdots i+k-2$ is a Poisson random variable.
4. There are no repeats of length k or greater in **a**.
5. The only sequencing errors are substitution errors.

We use the following symbols in our analysis:

k = tuple size corresponding to each edge,
L = length of the original sequence,
N = number of fragments,
l = average length of each fragment,
c = mean depth of coverage,
T = number of $(k-1)$ bp regions, $(|f_1| + \cdots + |f_N|) - (k-2)N = N(l-k+2)$,
r = error rate

We are now ready to give our bounds:

Theorem 2. *Let $L' = L - k + 2$ and $R = 1 - (1-r)^{k-1}$. Then, in a sequence graph:*

1. *The expected number of vertices* $\mathbb{E}(|V|) = RT + [1 - e^{-c(1-R)}]L'$.
2. *The expected number of singletons* $\mathbb{E}(|S|) = RT + e^{-c(1-R)}[e^{c(1-R)(1-r)^2} + c(1-R)r(2-r) - 1]L'$.
3. *The expected number of forks* $\mathbb{E}(|F|) = 2e^{-c(1-R)}[e^{c(1-R)(1-r)} - e^{c(1-R)(1-r)^2} - c(1-R)r(1-r)]L'$.

Proof. The depth X_α or the number of fragments that cover the region $\alpha \cdots (\alpha + k - 2)$ of **a** is Poisson with mean c. Note that $\mathbb{E}(\sum_{\alpha=1}^{L'} X_\alpha) = T$.

Recall that any $k-1$ bp-long substring $a_i^f \cdots a_{i+k-2}^f$ of a fragment f gives rise to a $(k-1)$-tuple in the sequence graph. With uniform error rate, r, the probability that a base is read correctly is $1-r$. Therefore, the probability that an entire $k-1$ bp region is read correctly is $(1-r)^{k-1}$ and the probability that it is read incorrectly is $1-(1-r)^{k-1} = R$. Thus, r is the single-base error rate while R is the $(k-1)$-tuple error rate.

Let us classify the $(k-1)$-tuples of the sequence graph as true if they are derived from a correctly read fragment region, and false otherwise. Let X be Poisson with mean c.

Because the error rate is assumed to be small, we further assume that no two fragments generate the same false tuple, that is, false tuples appear only once among fragments. Therefore, the expected number $\mathbb{E}(\textit{False})$ of false tuples is equal to RT.

The expected number of true tuples, $\mathbb{E}(\textit{True})$ is equal to the number of positions α such that at least one fragment contains no sequencing errors in the region $\alpha \cdots \alpha + k - 2$. Therefore,

$$\begin{aligned}\mathbb{E}(\textit{True}) &= L' \sum_{i=1}^{\infty} (1 - R^i)\mathbb{P}(X = i) \\ &= L' \sum_{i=1}^{\infty} \left(\frac{e^{-c}c^i}{i!} - \frac{e^{-c}(cR)^i}{i!} \right) \\ &= L'(1 - e^{-c(1-R)}).\end{aligned}$$

1. Since vertices are generated one per each true or false tuple, the expected number of vertices, $\mathbb{E}(|V|)$, is:

$$\begin{aligned}\mathbb{E}(|V|) &= \mathbb{E}(\textit{False}) + \mathbb{E}(\textit{True}) \\ &= RT + L'[1 - e^{-c(1-R)}].\end{aligned}$$

2. A vertex is a singleton if the $(k-1)$-tuple appears only once among the fragments, or appears at least twice among the fragments (correctly) and each time it appears it is preceded (followed) by the same base. All vertices corresponding to false tuples are singletons because they are assumed to appear only once. At each position, the probability the true tuple at that position appears in only one fragment is:

$$\sum_{i=1}^{\infty}\binom{i}{1}(1-R)R^{i-1}\mathbb{P}(X=i)$$
$$= c(1-R)\sum_{i=1}^{\infty}e^{-c}\frac{(cR)^{i-1}}{(i-1)!}$$
$$= c(1-R)e^{-c}e^{cR}$$
$$= c(1-R)e^{-c(1-R)}.$$

Therefore, the expected number of tuples appearing exactly once is equal to the sum:

$$RT + c(1-R)L'e^{-c(1-R)}.$$

Next, we consider tuples that appear at least twice (correctly) and are preceded (followed) by the same base. Because all false tuples are assumed to appear only once, the same base preceding (following) all the occurrences of the tuple must be the true base. That is, the corresponding fragments have no sequencing error at that position of the base. Therefore, the expected number of these tuples is:

$$L'\sum_{i=2}^{\infty}\sum_{j=2}^{i}\binom{i}{j}(1-R)^jR^{i-j}(1-r)^j\mathbb{P}(X=i)$$
$$= L'\sum_{i=2}^{\infty}e^{-c}\frac{c^i}{i!}\sum_{j=2}^{i}\binom{i}{j}(1-R)^jR^{i-j}(1-r)^{2j}$$
$$= L'\sum_{i=0}^{\infty}e^{-c}\frac{c^i}{i!}\Big([(1-R)(1-r)^2+R]^i - i(1-R)(1-r)^2R^{i-1} - R^i\Big)$$
$$= L'[e^{c(1-R)(1-r)^2} - c(1-R)(1-r)^2 - 1]e^{-c(1-R)}.$$

Adding the above quantity to the expected number of tuples that appear exactly once gives the expected number $\mathbb{E}(|S|)$ of singletons as $RT + e^{-c(1-R)}[e^{c(1-R)(1-r)^2} + c(1-R)r(2-r) - 1]L'$.

3. A fork corresponds to a tuple that appears in at least two fragments (correctly), and is preceded by the same base but followed by at least two different bases, or *vice versa*. As in the previous case, this is the number of true tuples that are preceded by the same true base, and followed by at least two different bases (multiplied by two to handle the symmetric case). Suppose there are i fragments covering the region $\alpha\cdots\alpha+k-2$, and j of these fragments are read correctly in the region. The probability that are all preceded by the same true base in position $\alpha-1$ is $(1-r)^j$. The probability that they are followed by at least two bases in position $\alpha+k-1$ is same as the probability that there is at least one false base in that position which is equal to $1-(1-r)^j$. Using these probabilities, we can compute the expected number $\mathbb{E}(|F|)$ of forks as:

$$\mathbb{E}(|F|) = 2L'\sum_{i=0}^{\infty}e^{-c}\frac{c^i}{i!}\sum_{j=2}^{i}\binom{i}{j}(1-R)^jR^{i-j}(1-r)^j[1-(1-r)^j]$$
$$= 2L'\sum_{i=0}^{\infty}e^{-c}\frac{c^i}{i!}\left\{\left(\sum_{j=0}^{i}\binom{i}{j}(1-R)^jR^{i-j}(1-r)^j[1-(1-r)^j]\right) - i(1-R)r(1-r)R^{i-1}\right\}$$
$$= 2L'\sum_{i=0}^{\infty}e^{-c}\frac{c^i}{i!}\Big([(1-R)(1-r)+R]^i - [(1-R)(1-r)^2+R]^i - i(1-R)r(1-r)R^{i-1}\Big)$$
$$= 2L'[e^{-c}e^{c[(1-R)(1-r)+R]} - e^{-c}e^{c[(1-R)(1-r)^2+R]} - c(1-R)r(1-r)e^{-c}e^{cR}]$$
$$= 2L'e^{-c(1-R)}[e^{c(1-R)(1-r)} - e^{c(1-R)(1-r)^2} - c(1-R)r(1-r)].$$

This completes our proof. □

TABLE 1. ESTIMATED AND OBSERVED VALUES ON SIMULATED DATA WITH $L = 20,000, l = 400$

		v vertices		s singletons		f forks		s/v %		$f/(v-s)$ %	
c	r %	*est*	*obs*	*est*	*obs*	*est*	*obs*	*est*	*obs*	*est*	*obs*
10	10	316,042	303,842	300,706	284,223	12,330	14,623	95.1	93.5	80.4	74.6
8	6	201,786	196,161	187,858	180,394	11,920	13,046	93.1	91.9	85.6	82.7
8	3	134,874	131,592	123,738	119,760	10,050	10,536	91.7	91.0	90.2	89.0
6	1	66,252	66,838	62,256	62,572	3,868	4,086	94.0	93.6	96.8	95.8

We have performed several simulations to test the accuracy of these estimates. We have written a computer program that takes as input, the length L of a DNA fragment to be sequenced, the number N of fragments to be generated, the length l of each fragment, the error rate r of the experiment, and an integer seed for random number generation. The program then produces a random DNA string of length L along with N fragments that contain errors (misreads, insertions, and deletions) with a uniform rate r. We then ran a counting program which, for a given value of k, gives a count of the number of vertices, singletons, and forks generated by the simulated sequencing project.

Using the above scheme, we ran two sets of simulations. In the first set, the length of the underlying sequence is 20,000 and the length of each fragment is 400. In the second set, the length of the underlying sequence is 30,000 and the length of each fragment is 500. The value of k is taken to be 13 for all simulations. In each case, we estimated the number of vertices, singletons, and forks using Theorem 5.1, and compared them against the corresponding observed values from the experiments. Each entry gives the results of a single data set. Each data set was generated with a different seed value. Because we include the reverse complements of all fragments in our method, we have written our counting program to include reverse complements as well. To account for this fact, we doubled all our estimates from Theorem 2.

Table 1 shows the results of simulations from the first set. We show results for four different error rates ranging from 1% to 10%. In addition to the number of vertices, singletons, and forks, we also report the percentage of singletons among all vertices, and the percentage of forks among all nonsingleton vertices. In all experiments, our estimates for vertices and singletons are within 10% of the observed values, with the estimates improving with decreasing error rates. The estimates for forks are not so impressive, but this is understandable because the number of forks is very small compared to the number of vertices and singletons. The key point to observe is that in all cases, singletons comprise at least 90% of all vertices, and forks comprise at least 70% of all nonsingletons. This is true with both estimated and observed values. This is precisely the claim we made above in the section, The Sequence Graph.

Table 2 shows the results of simulations from the second set. In this case, we show results for the same error rates of the first set but with slightly smaller average depth. The results here are consistent with those from the first set.

We also ran experiments on real sequencing data. Table 3 shows the results of these experiments. We obtained three different data sets, which are shown as G6PD, TV1, and STAN in Table 3. The first two sets were obtained from Dr. Ellson Chen's laboratory at Applied Biosystems Division, and the third set from

TABLE 2. ESTIMATED AND OBSERVED VALUES ON SIMULATED DATA WITH $L = 30,000, l = 500$

		v vertices		s singletons		f forks		s/v %		$f/(v-s)$ %	
c	r %	*est*	*obs*	*est*	*obs*	*est*	*obs*	*est*	*obs*	*est*	*obs*
8	10	389,902	378,482	371,650	354,700	15,066	17,762	95.3	93.7	82.5	74.7
6	6	240,688	232,860	224,868	215,021	13,926	15,062	93.4	92.3	88.0	84.4
6	3	166,636	163,025	153,780	149,590	11,856	12,014	92.3	91.8	92.2	89.4
4	1	84,882	84,397	80,916	80,329	3,872	3,872	95.3	95.2	97.6	95.2

TABLE 3. OBSERVED VALUES ON REAL SEQUENCING DATA

Data	*L*	*l*	*N*	*c*	*v* *vertices*	*s* *singletons*	*f* *forks*	*s/v* %	*f/(v − s)* %
G6PD	11,781	283	165	3.96	27,396	26,076	1,158	95.2	87.7
TV1	36,357	398	579	6.33	101,335	93,842	6,294	92.6	84.0
STAN	44,180	268	1905	11.56	182,332	168,782	11,742	92.6	86.6

Dr. David Botstein's laboratory at Stanford University. The three sequencing data sets essentially capture a spectrum of the size of sequencing projects. Moreover, the data set G6PD is known for the presence of several *alu* repeats (Chen *et al.,* 1991). In each case, we have reported the length of the underlying sequence and the number of fragments. We have also computed and reported the average length of each fragment and the mean depth. Because we do not accurately know the error rates of these projects, we have reported only the observed values. Once again, one may see that at least 90% of the vertices are singletons and at least 70% of the nonsingletons are forks.

SEQUENCE GRAPH REDUCTIONS

As mentioned above, we perform a series of reduction steps on the sequence graph G until no further reductions are possible. The resulting graph, called the irreducible sequence graph, has several interesting properties that will be explored in the next section.

All reductions replace a path or a series of edges with a new edge. The goal of performing these reductions is to eliminate extraneous edges generated by sequencing errors, and collapse singleton vertices to obtain a simple and compact graph that can then be toured to infer the underlying sequence.

There are three types of reductions that are described in increasing order of complexity. We perform all lower-order reductions before any higher-order reduction. It is possible that when we perform a reduction of order i, we may have created situations where reductions of order $i - 1$ or lower are now possible. In such cases, it is understood that we perform those lower-order reductions before proceeding with any higher-order reductions.

Recall that we include fragments as well as their complements in the sequence graph to handle the problem of unknown orientation. As a result of this, we perform two Eulerian tours that are complementary to each other. To preserve this complementarity, we make sure that whenever we perform a reduction on a set of edges, we also perform the complementary reduction on the corresponding complementary set. We also take special care to avoid placing both a fragment and its complement on the same tour.

We now describe our reductions starting with the simplest one.

Reduction 1: Elimination of singletons

Our first reduction eliminates all singletons. As mentioned above, these constitute about 90% of the vertices in many sequencing projects, so eliminating them greatly reduces the size of the sequence graph.

By definition, a singleton v can have only one incoming edge $[u, v]$ and one outgoing edge $[v, w]$. The reduction is to remove the vertex v along with the edges $[u, v]$ and $[v, w]$, and replace the edges with the edge $[u, w]$. If $tup([u, v]) = a_1 \ldots a_p$ and $tup([v, w]) = b_1 \ldots b_q$, then we set $tup([u, w]) = a_1 \ldots a_p b_k \ldots b_q$, since $a_{p-k+2} \ldots a_p = b_1 \ldots b_{k-1}$. We set $wt[(u, w]) \leftarrow wt([u, v]) + wt([v, w])$. We set $occ([u, w])$ to

$$\{(f, i, m) \mid \exists j, (f, i, j + k - 2) \in occ([u, v]) \text{ and } (f, j, m) \in occ([v, w])\} \cup end([u, v]) \cup begin([v, w])\}$$

Once we set $occ([u, w])$ we can set $begin([u, w])$ and $end([u, w])$ accordingly.

After eliminating all singletons, we are often left with a graph which is about one-tenth of the original size. We call the remaining edges *super edges.*

Apart from reducing the size of the graph significantly, the above reduction offers another advantage. It has been found from both real and simulated data that in many sequencing projects, one often encounters

super edges of length 50 or more. Put in a different way, even in a sequencing project with a moderate number of errors, one can find regions of length 50 bp or longer, where no error occurs. If the average depth of coverage in the project is big enough, say at least 4, then we have super edges of weight 200 or more. Because these edges represent local regions of perfect alignment among fragments, they can be included in any tour with a very high confidence. In fact, the use of the super edges will be readily evident in the next reduction step.

The next two reductions are applicable when a vertex v has multiple incoming and outgoing edges. Both the reductions, when applicable, eliminate one incoming edge along with one outgoing edge from v, thereby reducing the indegree and outdegree of v. Unlike the first reduction, these reductions are performed only when certain conditions, described below, are met.

Consider a vertex v along with an incoming edge $e = [u, v]$. Suppose v has more than one outgoing edge. For each outgoing edge $e_i = [v, w_i]$ of v, compute its continuation subset $cont(occ(e), e_i)$. Assuming that there are no sequencing errors then all but one of the continuation subsets will be empty unless e corresponds to a repetitive region. Even when e is repetitive, if we reach the vertex v from u using the edge $e = [u, v]$ during an Eulerian tour with $il \subseteq occ(e)$ as the active list of regions, only one of $cont(il, e_i)$ will be nonempty and the corresponding edge unambiguously becomes the next edge on the tour.

In the presence of sequencing errors, however, the above scenario does not hold perfectly. First, it is not easy to tell if e corresponds to a repetitive region. Second, even if we know that e does not correspond to a repetitive region, we cannot readily decide the next edge on the tour because, very often, more than one outgoing edge will have nonempty continuation subsets. Therefore, we need to resort to a heuristic overlap test to decide whether e corresponds to a nonrepetitive region, and if so, which of the candidate edges to visit next on the tour.

Based on our experience with simulated as well as real sequencing data, we have come up with the following heuristic for performing the overlap test. Because we expect our algorithm to run very fast, we believe that we can let users specify heuristics of their own and interactively assemble a sequence based on their own overlap criterion. In this sense, our algorithm is not based on a strict model of optimality.

The purpose of the heuristic overlap test is to decide if a given edge $e = [u, v]$ corresponds to a nonrepetitive region, and if so, which of the candidate edges $e_i = [v, w_i]$ can follow e on an Eulerian tour. The decisions are based on the overlaps $ov(occ(e), e_i)$. The overlap test takes two threshold parameters: *ncov* (*noncandidate overlap*) and *cndiff* (*candidate-noncandidate difference*). If e corresponds to a nonrepetitive region then there must be a candidate edge $e_c = [v, w_c]$ for continuation. The criterion for selecting a candidate e_c is:

$$ov(occ(e), e_i) \leq ncov \text{ and } ov(occ(e), e_c) - ov(occ(e), e_i) \geq cndiff, \quad \forall i \neq c.$$

If the criterion is satisfied for a candidate, then the overlap test succeeds and returns the candidate. Otherwise, the test fails, indicating that e probably corresponds to a repetitive region. It is possible to start the graph reductions with stringent values for the parameters and progressively relax them. We found that, the values $ncov = 1$ and $cndiff = 2$ reduce the graph to a few hundred vertices in many sequencing projects.

We also reverse the direction of an Eulerian tour and define a symmetric overlap test which decides the continuation for a given edge $e = [v, w]$ among the candidates $e_i = [u_i, v]$.

Reduction 2: Elimination of forks

Our second reduction can be used to eliminate most of the forks. As mentioned above, these constitute about 70% of the nonsingleton vertices in many sequencing projects. This will further reduce the size of the sequence graph. Moreover, eliminating forks, in turn, creates more forks by changing crosses into forks, which leads to further reduction of the graph.

Let us assume, without loss of generality, that a given fork v has one incoming edge $e = [u, v]$ and two outgoing edges $e_1 = [v, w_1]$ and $e_2 = [v, w_2]$. Suppose that we have placed e on the tour. It is evident that unless we visit e twice, which happens only if e corresponds to a repetitive region, only one of e_1 or e_2 can follow e on the tour so the other must be an extraneous edge. If the overlap test fails then we leave the fork as it is because e probably corresponds to a repetitive region. Suppose, without loss of generality, that the overlap test succeeds and returns e_1 as the candidate for continuation. Then we eliminate the edge

e_2 because e_2 is more likely to be the extraneous edge than e_1. Notice that eliminating e_2 changes v from a fork to a singleton so we can eliminate v using Reduction 1 and create the edge $[u, w_1]$. We also reduce the indegree of w_2 by one and change its status accordingly. We can similarly extend the reduction to the case where there are more than two outgoing edges from v.

To make this reduction more effective and robust, we always consider the fork with the heaviest identifying (super) edge for the next reduction. Because heavier superedges are very likely to correspond to regions of true underlying sequence, this choice improves our chances of finding the true sequence.

Reduction 3: Elimination of crosses

After the first two reductions, the remaining graph will be very small compared to its original size. Our final reduction can be used to eliminate most of the crosses, which are vertices with both indegree and outdegree of more than one.

Let us assume that a given cross v has two incoming edges $ie_1 = [u_1, v]$ and $ie_2 = [u_2, v]$, and two outgoing edges $oe_1 = [v, w_1]$ and $oe_2 = [v, w_2]$. If the overlap test returns oe_1 as the candidate for continuation with ie_1, and ie_1 as the candidate for continuation with oe_1 (on the reversed path) then we can merge ie_1 with oe_1 because doing so does not affect other edges incident on v. We can treat other cases similarly. Doing this reduction decreases the indegree and outdegree of v by one and may change its status. We perform any further reductions that may result in this process.

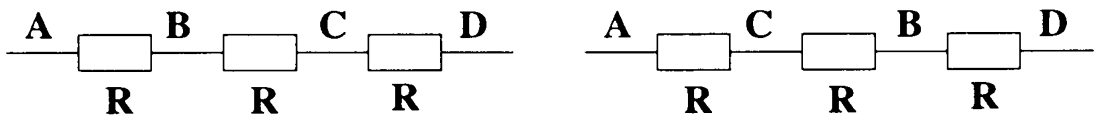
As in the case of Reduction 2, we always perform this reduction starting with heaviest edges to increase its effectiveness.

IRREDUCIBLE SEQUENCE GRAPHS

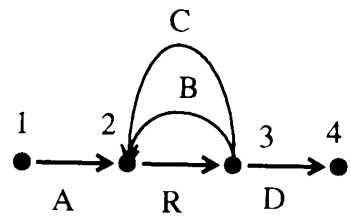
As described above, we perform all possible reductions on the sequence graph until no further reductions are possible. The resulting graph, called the irreducible sequence graph, is free of most of the extraneous edges. The primary advantage of this graph is that it is a very compact and straightforward way of representing plausible sequences without listing them exhaustively.

Let us look at some examples. If we have the ideal data described in the section A New Algorithm for Shotgun Sequencing, the irreducible graph will be just a super edge e such that $tup(e)$ is the underlying sequence. This can be achieved if the sequencing error rate is very small (say $< 1\%$), and the underlying sequence does not contain long repeats (such as *alu*). In fact, we have achieved this on random sequencing data with 1% error rate. Even if the sequence contains some long repeats, we can still expect to deduce the underlying sequence unambiguously as long as there are fragments bridging repeats and their adjacent nonrepetitive regions.

However, real sequences do contain repeats (typically 30% of many sequences are made of *alu* repeats) and we may not have bridging fragments to decide which repeat goes where. In such situations we will have not just one but many possible underlying sequences. Such a situation is given below.



In the above sequences, the regions A, B, C, D are nonrepetitive regions and R is a repetitive region. It is not immediately clear whether $ARBRCRD$ or $ARCRBRD$ is the right sequence. If there are fragments overlapping the repetitive regions, that can be uniquely aligned and if there are mutations to distinguish between the three R's, we can resolve the ambiguity. In some cases, fragments will bridge R by overlapping A and B, for example. This makes the resolution of the sequence easy to establish. In the absence of such information, we cannot uniquely report an underlying sequence. Such a situation would arise if $|R| = 800$ and $l = 300$, and all three R's are identical in the underlying sequence. In such a situation, the irreducible sequence graph looks as follows.



In the above graph, the edge [2, 3] corresponding to the repetitive region R is adjacent to four other edges corresponding to the nonrepetitive regions A, B, C, and D. It is not difficult to see that the size of $occ([2, 3])$ will be significantly more than the sizes of occ's of other edges. If we start an Eulerian tour with the edge [1, 2], it will be possible to visit the edge [2, 3] three times because $occ([2, 3])$ will have disjoint continuation subsets for the three edges representing A, B, and C.

It is not difficult to see that this graph has two possible Euler tours corresponding to the two possible underlying sequences. When R is long and repeated exactly, we cannot uniquely determine an underlying sequence, and we print the irreducible sequence graph leaving it to the users to decide which sequence they want. One advantage of this approach is that it exposes "ambiguous" regions where further sequencing can be done to resolve ambiguities. We believe this to be a very valuable tool in improving the efficiency of a sequencing project.

CONCLUSIONS

We have proposed a new algorithm for DNA sequence assembly that combines the features of shotgun sequencing and sequencing by hybridization. The algorithm takes advantage of high coverage and low sequencing error rates made possible with the advent of advanced DNA sequencing machines. We are still in the process of developing complete software that incorporates all features mentioned in this manuscript. As such, we cannot provide experimental results on a number of real sequencing projects and compare our method with other methods. However, we developed a prototype that has some of the key features of the algorithm, and used this prototype to assemble simulated sequencing data. We report the results of such an experiment below, mainly as a demonstration of the potential of our approach.

Table 4 shows the results of our experiment. First, we generated a random DNA sequence of length 20,000 bp. We then generated a fragment set of mean length 400 bp for this sequence, with a mean depth of coverage of 7 using the `enzyme` program of the `GenFrag` utility (Engle and Burks, 1993), version 2.1 (Engle and Burks, 1994). Sequencing errors were simulated on this fragment set with error rates ranging from 0.5% to 3.0% using the `mutate` program of the same utility. Each entry in Table 4 corresponds to a different error rate. These mutated fragments are then given as input to our prototype program. All runs were performed with $k = 15$. All runs were completed under ten seconds on a SUN SPARCstation 10.

Our prototype builds the sequence graph on a given fragment set and performs the graph reductions mentioned above. It does not perform the Eulerian tours to handle repeats, and reports the inferred sequence on each edge remaining in the irreducible graph. It does not yet perform the multiple alignment on the fragments. The quality of the inferred contigs is judged by running the dynamic programming algorithm

TABLE 4. RESULTS OF SEQUENCE ASSEMBLY ON A SIMULATED DATA SET

Error rate (%)	*Number of contigs*	*Similarity score*	*Length of inferred sequence*	*Number of correct calls*	*Lengths of contigs*
0.5	1	19,987	19,995	19,991	19,995
1.0	1	19,962	19,972	19,968	19,972
1.5	1	19,950	19,964	19,958	19,964
2.0	3	19,913	19,949	19,934	11,520, 6,058, 2,335
2.5	2	19,904	19,945	19,927	13,147, 6,757
3.0	2	19,899	19,981	19,945	13,063, 6,836

described in the section A New Algorithm for Shotgun Sequencing that fits a given contig to the true sequence (that we know already) at the appropriate location.

For each entry in Table 4, we show the number of contigs generated (the prototype actually generates twice the number of contigs because of the complementarity of edges). For each contig, we computed a similarity score from the dynamic programming algorithm by giving a positive score of 1 for each match, and a negative score of 1 for each mismatch or indel. We also computed the number of matches or correct base calls for each contig. When there are multiple contigs, we reported the sum of similarity scores, and total number of correct calls. The length of inferred sequence covered is the sum of lengths of all contigs.

Table 4 shows that our prototype, even without performing any multiple alignment and consensus, has a very high percent of correct calls ($>99.8\%$), and a very high similarity score in all cases.

ACKNOWLEDGMENTS

We are grateful to Dr. Pavel Pevzner for helpful discussions, and to the referees for their useful comments. We also thank Dr. Ellson Chen and Dr. David Botstein for providing us with the sequencing data. This work was supported by grants from the National Science Foundation (DMS-90-05833) and the National Institutes of Health (GM-36230). Some of this work was also done while we were visiting DIMACS at Rutgers University, and was supported by National Science Foundation (STC-91-19999 and BIR-94-12594).

REFERENCES

Bains, W., and Smith, G.C. 1988. A novel method for DNA sequence determination. *J. Theoret. Biol.* 135, 303–307.

Chen, E.Y., Cheng, A., Lee, A., Kuang, W-J., Hillier, L., Green, P., Schlessinger, D., Ciccodicola, A., and D'Urso, M. 1991. Sequence of human glucose-6-phosphate dehydrogenase cloned in plasmids and a yeast artificial chromosome. *Genomics* 10, 792–800.

Drmanac, R. 1994. DNA sequence determination by hybridization: a strategy for efficient large-scale sequencing. *Science* 263, 596–596.

Drmanac, R., and Crkvenjakov, R. 1987. Yugoslav Patent Application 570.

Dumas, J.P., and Ninio, J. 1982. Efficient algorithms for folding and comparing nucleic acid sequences. *Nucleic Acids Res.* 10, 197–206.

Engle, M.L., and Burks, C. 1993. Artificially generated data sets for testing DNA sequence assembly algorithms. *Genomics* 16, 286–288.

Engle, M.L., and Burks, C. 1994. GENFRAG-2.1: new features for more robust fragment assembly benchmarks. *Computer Applic. Biosci.* 10, 567–568.

Gallant, J. 1983. The complexity of the overlap method for sequencing biopolymers. *J. Theoret. Biol.* 101, 1–17.

Gallant, J., Maier, D., and Storer, J. 1980. On finding minimal length superstrings. *J. Computer System Sci.* 20, 50–58.

Gusfield, D. 1991. Efficient methods for multiple sequence alignment with guaranteed error bounds. Tech. Report, Computer Science Division, University of California, Davis, CSE-91-4.

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

Kececioglu, J. 1991. "Exact and approximation algorithms for DNA sequence reconstruction." Ph.D. Thesis. Department of Computer Science, University of Arizona, Tucson, AZ.

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

Lander, E.S., and Waterman, M.S. 1988. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics* 2, 231–239.

Lysov, Y.P., Florentiev, V.L., Khorlin, A.A., Khrapko, K.R., Shik, V.V., and Mirzabekov, A.D. 1988. DNA sequencing by hybridization with oligonucleotides. A novel method. *Dokl. Acad. Sci. USSR* 303, 1508–1511.

Macevicz, S.C. 1989. International Patent Application PS US89 04741.

Maxam, A., and Gilbert, W. 1977. A new method for sequencing DNA. *Proc. Natl. Acad. Sci. USA* 74, 560–564.

Peltola, H., Söderlund, H., and Ukkonen, E. (1984). SEQAID: A DNA sequence assembly program based on mathematical model. *Nucleic Acids Res.*, 12, 307–321.

Pevzner, P.A. 1989. *l*-tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.* 7, 63–73.

Pevzner, P. 1992. Multiple alignment, communication cost, and graph matching. *SIAM J. Applied Math.* 52, 1763–1779.

Pevzner, P.A., and Waterman, M.S. 1995. Multiple filtration and approximate pattern matching. *Algorithmica* 13, 135–154.

Sanger, F., Nicklen, S., and Coulson, A. 1977. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. USA* 74, 5463–5467.

Smith, L.M., Sanders, J.Z., Kaiser, R.J., Hughes, P. Dodd, C., Connell, C.R., Heiner, C., Ken, S.B.H., and Hood, L.E. 1986. Fluorescence detection in automated DNA sequence analysis. *Nature* 321, 674–679.

Southern, E. 1988. United Kingdom Patent Application GB8810400.

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

Wilbur, W.J., and Lipman, D. 1983. Rapid similarity searches of nucleic acid and protein databanks. *Proc. Natl. Acad. Sci. USA* 80, 726–730.

Address reprint requests to:
Dr. Michael S. Waterman
Department of Mathematics
University of Southern California
Los Angeles, CA 90089-1113

msw@hto.usc.edu