Advances in Sequence Assembly[†]

Eugene W. Myers[‡]

ABSTRACT

This article surveys the current state of affairs with regard to software for sequence assembly. We try to give some appreciation of the underlying combinatorial problems involved and the nature of the computer codes or algorithms used to solve them. The article further discusses the requirements that should reasonably be met by the user-interface that drives the computational components. This discussion is intended to give the reader a perspective on how to evaluate the software systems currently available to them. Finally, the article discusses current research on algorithms for fragment assembly and the nature of the improvements that can be expected in the near future.

1. Introduction

The scale of DNA sequencing projects has grown to the point that data analysis is becoming a significant aspect of the process. Shortly after the development of the sequencing techniques of Sanger et al. (1977) and Maxam & Gilbert (1977) just fifteen years ago, investigators were sequencing stretches of 1000 to 5000 nucleotides and were quite content with assembling the fragment data from their gel runs manually. Even as projects became more ambitious, the assembly step itself was not very time consuming, and simple software that basically assisted manual assembly by pointing out the overlaps between fragments was all that was required. But today, projects to sequence entire cosmids or small YACs of 40,000 to 200,000 nucleotides are routinely being undertaken, propelled by goals such as the Human Genome Initiative (U.S. Dept. of Energy, 1992).

[†] This work was supported in part by the National Library of Medicine under Grant R01 LM04960 and the Aspen Center for Physics.

[‡] Current Address: Dept. of Computer Science, University of Arizona, Tucson, AZ 85721.

A variety of schemes have been proposed for sequencing such large stretches. The traditional shotgun approach still has a great deal of appeal because it is economical, parallelizable, and automatable (Sanger et al., 1982). However, for the large stretches anticipated, coverage is a significant problem. Statistical considerations alone show that if one sequences six genome equivalents of 1kbp fragments of a 50kbp clone, then one can expect two to six gaps in the resulting assemblages of the data (Lander & Waterman, 1988). Of course, biological and experimental considerations tend to further increase this number. It is clear that in order to achieve closure other techniques such as directed sequencing or PCR must be employed near the end of a large-scale shotgun project. However, investigators aware of the problem have concocted the following variations that can reduce the number of contig gaps that need to be covered by more costly alternatives:

- **Dual End**: Sequence both ends of inserts that are designed to be larger than twice the average gel run length.
- **Sequencing "Noise"**: Run each gel run well past the point where accurate data is being gathered. The "noisy" tail still contains enough information for computing overlaps (but is not used for determining sequence content).
- **Sampling Without Replacement**: Sequence only those clones that do not hybridize with any sequenced so far. This accelerates one to the "knee" of the coverage probability curve.

It is thus clear that in any large-scale "shot-gun" style project, there will actually be additional information about the relative position and quality of sequence data that a good software solution should take into account.

Besides shotgun strategies, there are many other distinctive approaches that are very actively being pursued in an attempt to produce more economical, accurate, and faster methods. For example, there is much excitement about sequencing by hybridization (Drmanac et al., 1989). However, it should be noted that most of these proposals really amount to very rapid ways to produce fragment data, and still beg the problem of assembling the fragments. In another direction, improvements in the cost of producing primer oligonucleotides may make it quite economical to directly sequence large stretches via primer walking (Studier, 1989). PCR using oft-occurring subsequences or transposable elements as primers can be used to obtain multiple start points for such directed walks, so that a higher level of parallelism can be achieved. More exotic technologies, such as directly reading the sequence of a single strand with a scanning tunneling microscope (Allison et al., 1990), are contemplated but well into the future. While these methods produce data which is easier to "assemble", it seems that the most robust approach from the view of developing software is to build a system that handles the combinatorially most complex problem, i.e. "pure" shotgun, but which can gracefully handle problems for which more is known about how the fragments assemble. For example, even for directed sequencing, the computational aspect still presents the common problems of aligning walks in opposite directions and producing a consensus over all walks.

For the remainder of the paper we consider the computational problem posed by a large shotgun sequencing project where the strategy is not "pure" in the sense that there is often times addition information about how fragments overlap. We divide our treatment into aspects that affect the underlying combinatorial algorithms, and those that are basically engineering issues essential for a high capability and user-useful system. Well-designed and practical algorithms for the fragment assembly problem should be capable of handling all of the following:

(1.a) **Errors**: Sequencing error generally runs at about 1% but can be as high as 5%. The ability to tolerate high error rates is desirable: all data is useful according to "noisy" sequencing advocates. Errors impact the determination of fragment overlap and require multi-alignment in high-coverage regions in order to determine an overall consensus reconstruction.

(1.b) **Unknown Orientation**: For many experimental protocols one does not know if a fragment is from the 5’ or 3’ strand.

(1.c) **Incomplete Coverage**: Needless to say one cannot expect a given data set to assemble into a single contig. One should be able to assemble the fragments available at any point during a project in order to get some sense of their progress in terms of coverage and the desirability of continuing to shotgun.

(1.d) **Vector Removal & Multiple Inserts**: Invariably some initial portion of a fragment sequence belongs to the vector, and occasionally, for very short inserts, some of the final portion as well. Also, for some experimental protocols, two or more inserts may be spliced into a single vector. These artifacts must be *detected* by the algorithm.

(1.e) **Overlap & Orientation Constraints**: As mentioned several times, a project may employ protocols for which there is additional information about where a fragment comes from or how it overlaps with others. For example, when sequencing without replacement one knows the fragments *don't* overlap, and when dual end sequencing one knows that the ends are in the same orientation, non-overlapping, and at some approximate distance apart. A method that accommodates such information, and even better is sped up by such information, is desirable.

(1.f) **Alternative Solutions**: While a given computer generated solution may "look" very reasonable, how do you know its clearly the best or that there aren't other assemblages that are also quite conceivable? An algorithm that has the built-in ability to generate alternative solutions in some rank order of "goodness" is very desirable. One can with confidence accept a given solution if the next best is clearly inferior.

Similarly, the following list enumerates a set of capabilities that should be available to the investigator using a software system that has at its heart a suite of algorithms meeting the specifications above.

(1.1) **Fragment Database**: When fragments are entered into the system, one should be able to enter additional information such as experimental conditions, responsible investigator, date of entry, and if at all possible a digitized image of the raw data. Over time one should be able to annotate the removal of vector sequence, the discovery of multiple inserts, and corrections to the raw data. The raw data must never be removed. All this requires a modest "database" of the fragment data.

(1.2) **Assembly Browser**: For a large project, the assemblages will be very large, so the interface must include an easy to navigate, window-based, browser for examining assemblages. Essential to this end, is auto-correlation amongst views of the data.

(1.3) **Multi-alignment editor**: The multi-alignments produced in high coverage regions will reveal sequencing errors. An editor for correcting these is a must. Such an editor should be able to automatically bring the user to a region that needs examination, and should be able to display the original raw data, e.g. an ABI waveform or digitized gel image, on screen and synchronized with the current editing focus.

(1.4) **Revision Control**: A simple revision control mechanism is needed to manage and keep track of refinements to the fragment data from (a) the removal of vector sequence, (b) the discovery of multi-inserts, and (c) manual correction with the sequence editor. Such revision control should be transparent to the user except when needed.

(1.5) **A posteriori constraints**: Any automated system should permit a user to override the decisions it makes. The user should be able to manipulate and constrain the assemblages produced a posteriori, based on human insight.

(1.6) **A priori constraints**: Mixed strategies must be employed for closure near the end of a sequencing project, and strategies such as sampling without replacement and dual end sequencing may be used to further alleviate the problem. In these cases there is additional a priori information that constrains the nature of the assembly and one should be able to signal these to the system.

(1.7) **Seamless Integration**: The software system should permit one to easily apply other sequence analysis tools to the reconstructions it produces, and to export said to other software, such as similarity searches against the national databases.

There is no current system and/or algorithm suite that meets all the requirements in the two lists above, and most meet very few. However, the level of sophistication is rising rapidly and current systems under development should come very close to meeting all the capabilities above. The sections that follow, review the progress to date on algorithms and software systems, and we conclude with future developments.

2. Algorithms for Fragment Assembly

Given short fragment sequences randomly sampled from a long unknown DNA sequence, the shotgun fragment assembly problem is to determine the most likely reconstruction of the original sequence. The earliest work was so ad hoc in nature that a formal statement of the problem the software was attempting to solve was not given and the underlying algorithms did little more than assist in "melding" fragments together (Staden, 1979, Gingeras et al., 1979). As work progressed it became clear (Peltola et al. 1984, Turner, 1986, Kececioglu 1991) that the formal problem for a shotgun problem addressing items (1.a), (1.b), and (1.c) was in essence a "noisy" shortest common superstring problem:

> **The Fragment Assembly Problem**: Given a collection of fragments $\mathcal{F}$ and a small error tolerance $\varepsilon \in [0,1]$, find the *shortest* string R such that for every fragment $F \in \mathcal{F}$, F or its Watson-Crick complement aligns with a substring of R with $\varepsilon|F|$-or-less differences (insertions, deletions, and substitutions).

R is a common superstring in that every fragment F or its complement is (approximately) a substring of R. The "noise" is modeled by ε which is an upper bound on the sequencing error rate, e.g. $\varepsilon = .05$ asserts that no gel run has more than 5% of its base calls in error. The criterion that R be as short as possible is an appeal to the principle of parsimony: R is the shortest possible explanation of all the data $\mathcal{F}$.

Theoretically the problem is in the class of NP-hard problems (Maier, 1978) implying that a procedure that is guaranteed to be computationally efficient for all problem instances probably does not exist. However, empirically the problems that arise in practice can be solved quite well by sufficiently powerful algorithms. Such algorithms generally decompose the problem into the three phases depicted in Figure 2.1. We discuss each phase and in particular our approaches to them (Kececioglu & Myers, 1992).

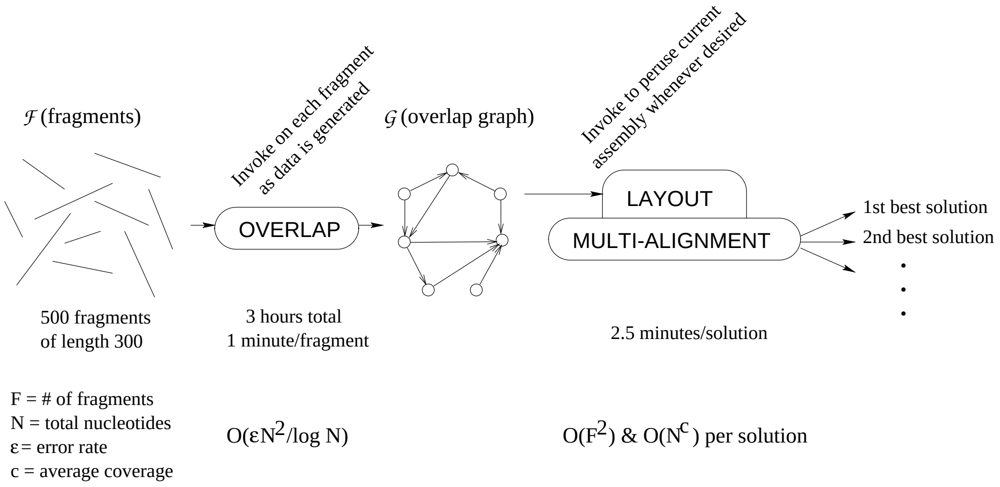

Figure 2.1: A 3-Phase Sequence Assembly Algorithm

2.1. **Overlap Phase**: This first phase compares all pairs of fragments to determine significant approximate overlaps. We advocate a full-sensitivity approach, as opposed to heuristics which occasionally miss significant overlaps. Our own algorithms score each overlap based on its statistical significance. We are the only ones to do so and can do it because of an incremental alignments algorithm we developed specifically for this problem (Myers, 1986). Efficiency, without losing sensitivity, is afforded by the further use of a Four-Russians prescreener (Wu et al., 1992). For a problem with 500 fragments of average length 300, we can perform this computationally intensive step in 3 hours on a 20MHz workstation. Moreover, this computation can be amortized over the period of data entry which for the project above was six months. The results of the overlap comparisons are encoded as edges in a directed, weighted *overlap graph*: each fragment is modeled as a vertex; an edge from *A* to *B* models an overlap between *A* and *B*; and its weight is the score of the overlap. In our model, a fragment is "entered" into the overlap graph once by comparing it against those already in the graph. The computation need not be repeated and as an example it took less than two minutes to enter the last fragment into the graph for the project above.

2.2. **Layout Phase**: This phase takes the overlap graph, call it 𝒢, as input and generates a series of alternate assemblies or layouts of the fragments based on the pairwise overlaps therein. A layout specifies the relative locations and orientations of the fragments with respect to each other and is typically visualized as an arrangement of overlapping, directed lines, one for each fragment.

One desires a layout that is as short as possible. In terms of the overlap graph this problem reduces to one of finding a maximum weight *Hamiltonian path* through $\mathcal{G}$ (Turner, 1986, Kececioglu, 1991). Whereas previous investigators have used a simple greedy heuristic (Staden, 1982, Peltola et al., 1984, Huang, 1992) to find a Hamiltonian path whose weight is near the optimal, we are the first to use a relaxation approach: we took an efficient algorithm for the computationally tractable problem of generating *directed spanning trees* in order of score, and ran this generator until (1) either a tree that was a path was generated, or (2) until too much time had elapsed. If one terminates with case (1) then one has the optimal solution in hand. In case (2), a post-process converts the generated spanning trees into paths by locally optimal edge exchanges, and then reports the highest scoring "repaired" tree as the solution. It is always the case that these greedily repaired spanning trees produce layouts whose scores are at least as good as those produced by the simple greedy heuristic. Moreover, in practice we find that our algorithm usually terminates via case (1) with an optimal solution. And in those cases where it has to repair, we are at least able to report how far from optimal our reported solution might be. Even more important, is that our approach is fundamentally generative due to the use of the generative spanning tree algorithm and so our software suite meets criterion (1.f). By simply continuing to run the generator, the approach can deliver a second-best solution, then a third-best solution, and so on.

2.3. **Multi-Alignment Phase**: The final phase simultaneously aligns the sequences of the fragments in a layout produced by the layout phase giving a final consensus sequence as the desired reconstruction of the original strand. This phase, like the previous one, represents another intractable problem requiring time exponential in the overlap depth to solve optimally. We proceed by producing an initial alignment consistent with all the pairwise alignments of the edges in a branching of the previous phase. This is always possible, computationally efficient, and since the error rate is typically less than 10% produces a very good first approximation (unlike the related work on protein sequences that are 70% different (Feng & Doolittle, 1987)). In a second step, a "window" is swept over this initial alignment to optimize the alignment in subregions where the use of global overlap alignments produced locally nonoptimal subalignments. With this window-sweep we empirically find the resulting multi-alignment to be almost-everywhere optimal, especially when the error rate is less than 5%.

In closing this section we note several interesting points to ponder in considering algorithm designs for this problem. First, most complaints about the quality of solutions produced are attributable to weak algorithms in the overlap and multi-alignment phases. Some have asserted that it is not worth the seemingly prohibitive amount of time required to do a full-sensitivity dynamic programming comparison of fragments, or a second sweep in the multi-alignment phase. But it is exactly failures here — poor or undetected alignments, or obviously improvable multi-alignments — that most annoy the end-user. Moreover, the argument against the amount of time required diminishes with each year as the power and speed of machines continues to rise. The second point concerns the layout phase. While the simple greedy heuristic works on most occasions, it does occasionally fail. The question to ask is can one afford these failures and what is one willing to pay to reduce them. We think that if the process is to be automated, then the automaton must be as reliable as possible. So we again will choose more sophisticated approaches in an attempt to increase reliability. Given that the time to gather the data is measured in units of days and months, why should one be disturbed that the accompanying computation takes a few minutes?

3. Software Systems for Fragment Assembly

Early software systems did little more than assist in melding fragments together (Staden, 1979, Gingeras et al., 1979). The first software to be based on a reasonably firm analytic foundation was the SEQAID tool (Peltola, 1984). These systems were not highly interactive and designed for a simple ASCII terminal interface. While the most recent systems have significantly improved window-based, menu-driven interfaces (e.g. Dear & Staden, 1991), the capabilities of the current workstation technology have not been fully utilized. Moreover, in many cases the underlying algorithm suite is still quite naive and facilities such as revision control (1.4), constraints (1.5, 1.6), and seamless integration (1.7) to other tools are not realized. Mega-base sequencing projects can benefit, and arguably require, the robust software solutions and sophisticated user interfaces specified in the introduction. We are currently in the process of building such a system (our second), and we discuss some of our design decisions here.

The first and most important design decision was to separate issues of interface and environment from the suite of algorithms solving the underlying combinatorial problem. The suite of algorithms have been packaged into an object-oriented "kernel" of routines that manipulate overlap graph, sequence, and multi-alignment objects. This internal interface divides the software into two layers that may be modified or upgraded independently of each other.

As a preliminary design experiment we built a simple browsing system (Miller & Myers, 1991) we call FAB (Fragment Assembly Browser) that realizes capabilities (1.1) through (1.4). FAB is built upon the X-11 windowing system. Once data has been entered and system parameters set through a number of other menu options, one can elect to browse the current set of fragment assembly solutions. An initial window lists the alternate assemblages for a given sequencing project. Selection of one of the lines summarizing an assemblage opens a window containing a listing of the contigs or islands for that assemblage. Selection of one of these contig lines opens a window graphically portraying the corresponding layout as a series of lines. Finally, a menu selection from a layout opens a window on a multialignment showing the individual bases in the layout. Thus the overall organization allows one to start at the highest conceptual view of a solution and navigate down towards the most detailed level. All windows are scrollable, for example, multialignments (multi's for brevity) typically do not fit on a screen and so the user can scroll it either left or right to reveal different portions within the window. X-11 permits one to move windows about, change their size, reduce them to icons, and change their relative depth.

One may open as many windows on each object — contig lists, layouts, and multi's — as they wish, including several windows on the same object if desired. A window opened as the result of a selection on another is termed the child of the originating or parent window. Among layouts and multi's a box in the parent shows the region of the child currently being displayed in the child's window. Scrolling the child moves the box in the parent, and either dragging the box or scrolling the parent updates the view in the child. This *auto-correlation* feature greatly facilities navigation and prevents one from "getting lost". One can zoom in or out on layouts, compare two layouts, and there is a limited editing capability for multi's. All the interaction is mouse/button driven with menus much in the "Macintosh" style.

Conceptually FAB maintains a project database consisting of a collection of fragments and an overlap graph modeling the approximate overlap relationships between them. As new fragment data becomes available it is entered in this database, given a name, and compared against all other fragments in order to keep the overlap graph up to date. Additional information such as the date of entry and sequencing conditions are also associated with each fragment. (Ultimately FAB will also be capable of maintaining a digitized gel image or other representation of the raw data.) At any time during the course of the project, a user can invoke the assembler upon the fragments or a subset of the fragments currently in the database. The user is immediately placed in the browser with the results of the assemblage as described above. While browsing the results, they can edit multi-alignments (ultimately in consultation with the stored digitized gel images). Further, they can explore alternate layouts of the fragments by asking where a fragment or contig of fragments might otherwise be incorporated, and then constraining the assembler to utilize such an overlap relationship. When such a session is over, the user is asked if they wish to save the changes to a fragment's sequence implied by a multi-edit, and if they wish to save a set of assembly constraints used in editing a layout. These changes if saved must be reflected back into the project database, thus closing a feedback cycle.

This feedback cycle is potentially dangerous. What if a user later decides that the multi changes were not correct? or that a layout rearrangement was not advisable based on new sequence data? If the project database is updated destructively, then this previous state is lost. Thus we argue that revision control is necessary in an assembly environment. Our context presents a much simpler problem than that classically required for software maintenance. Revision control for fragment assembly need only support the model that a series of progressive refinements lead ultimately to correct sequence. In this model, the *history* of a fragment is a *linear* series of revisions leading from the raw data to the current revision. Each revision, except the current one, has exactly one successor. Using standard technology, such revision control typically requires only an additional 20% to 30% space overhead over holding just the raw data.

All raw fragment data entered into the system is placed in project revision 0 and all other revisions already extant (conceptually only). Each set of edits performed and saved in a session with the browser creates a new project revision that supersedes the previous one. Revisions of a project are numbered sequentially and time stamped by default. They can be given explicit names, and comments can be associated with them. By default a user is always working with the most recent revision, so that revision control is effectively *invisible* until a user explicitly needs to utilize the capability, say to undo some changes or re-examine a series of earlier corrective decisions. The most recent set of revisions may be removed, changes to an individual fragment can be undone, and histories of fragments or groups of fragments may be "rolled back".

Finally, we turn attention to constraints (1.5 and 1.6). Such capabilities must be supported by the underlying algorithms for assembly (1.e). To date, no system has such capabilities and we are the first to address it at the algorithmic level (Kececioglu & Myers, 1992). Our relaxation approach to the layout phase naturally supports constraints: the spanning tree generator algorithm already produces alternate trees by placing constraints on the *edges* (i.e. overlaps) that must be in or not in the next spanning tree. In a completely analogous manner one can modify such generators to produce solutions with the fragments in given orientations. This algorithmic capability will translate at the system level into a user being able to assert that they wish to see all solutions in which certain overlaps are "IN", certain overlaps are "OUT", and certain pairs of fragments are in either the "SAME" or "OPPOSITE" orientation. A user may use such a feature to override decisions automatically made by the assembly kernel whose parsimony-based optimization criterion may produce erroneous results in, for example, highly repetitive regions. We qualify such constraints as *a posteriori*, because they are imposed after an investigator has viewed the results of an assembly and applied his human expertise.

A priori constraints are those that are intrinsic to the data collection strategy, such as dual-end sequencing or the use of directed sequencing or nested deletions near the end of a project when investigators move to other techniques to span gaps between contigs. All such protocol constraints can be mapped internally to a set of overlap and orientation constraints, with the exception of dual end sequencing which requires one to be able to assert that a given pair of fragments occur at a certain distance range from one another. With this additional internal constraint mechanism a rich set of a priori constraint scenarios can be handled. At the interface level we plan to support a menu of protocol types, such as "NESTED", "DIRECTED", "DUAL", etc. that can be applied to subsets of the fragments and are automatically translated into the appropriate set of overlap, orientation, and distance constraints.

4. Future Developments

Software development for handling the computational aspects of an experimental protocol typically lags behind the innovation of the methods. In the case where techniques are not well established, no one is willing to invest the man-power to produce a production quality software system. However, shotgun assembly and its variation has become so common place that a significant number of computational scientists have studied the problem and a number of commercial concerns have perceived enough of a market to produce products. None currently achieves all the specifications of the list in the introduction, but there are no technical barriers to such an end. As the nature of the needed computational support becomes better delineated such products will appear on the scene. Hopefully, our group is only one of several attempting to do so.

On the purely algorithmic and analytic front, refinements to the accuracy and speed of the underlying computation continue to be made. For example, our next suite of kernel algorithms will improve on the speed of the overlap computation by at least a factor of ten by virtue of further algorithmic advances. By moving from a relaxation approach using spanning trees to one involving the graph-theoretic concept of a *matching*, we expect to greatly increase the reliability of the layout phase. Finally, analytic advances in the analysis of multi-alignments will improve the error correcting/detecting accuracy of the multi-alignment phase. All these advances are currently in development and we anticipate their arrival over the course of the next few years.

References

Allison, D.P., Thompson, J.R., Jacobson, K.B., Warmack, R.J. and Ferrell, T.L. (1990). "Scanning tunneling microscopy and spectroscopy of plasmid DNA". *Scanning Microsc.* **4**, 517-522.

Dear, S., and Staden, R. (1991). "A sequence assembly and editing program for efficient management of large projects". *Nuc. Acids Res.* **19**, 3907-3911.

Drmanac, R., Labat, I., Brukner, I., and Crkvenjakov, R. (1989). "Sequencing of megabase plus DNA by hybridization: Theory of the method". *Genomics* **4**, 114-128.

Feng, D. and Doolittle, R. (1987). "Progressive sequence alignment as a prerequisite to correct phylogenetic trees", *J. Mol. Evol.* **25**, 351-360.

Gingeras, T.R., Milazzo, J.P., Sciaky, D. and Roberts, R.J. (1979). "Computer programs for the assembly of DNA sequences". *Nuc. Acids Res.* **7**, 529-545.

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

Kececioglu, J.D. (1991). "Exact and approximate algorithms for DNA sequence reconstruction". Ph.D. Thesis. Technical Report 91-26, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721.

Kececioglu, J.D. and Myers, E.W. (1992). "Combinatorial algorithms for DNA sequence assembly". Accepted for publication in *Algorithmica*.

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

Maier, D. (1978). "The complexity of some problems on subsequences and supersequences". *J. of the ACM* **25**, 322-336.

Maxam, A.M., and Gilbert, W. (1977). "A new method for sequencing DNA". *Proc. NatL. Acad. Sci. USA* **74**, 560-564.

Miller, S. and Myers, E.W. (1991). "A fragment assembly project environment". Technical Report 91-17, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721.

Myers, E.W. (1986). "Incremental alignment algorithms and their applications". Technical Report 86-2, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721.

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

Sanger, F., Nicklen, S., and Coulson, A.R. (1977). "DNA sequencing with chain-terminating inhibitors". *Proc. NatL. Acad. Sci. USA* **74**, 5463-5467.

Sanger, F., Coulson, A.R., Hong, G.F., Hill, D.F., and Petersen, G.B. (1982). "Nucleotide sequence of bacteriophage λ DNA". *J. Mol. Biol.*. **162**, 729-773.

Staden, R. (1979). "A strategy of DNA sequencing employing computer programs". *Nuc. Acids Res.* **7**, 2601-2610.

Staden, R. (1982) "Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequence". *Nuc. Acids Res.* **10**, 4731-4751.

Studier, F.W. (1989). "A strategy for high-volume sequencing of cosmid DNAs: Random and directed priming with a library of oligonucleotides". *Proc. Natl. Acad. Sci. USA* **86**, 6917-6921.

Turner, J. (1986). "Approximation algorithms for the shortest common superstring problem". Technical Report WUCS-86-16, Dept. of Computer Science, Washington University, St. Louis, MO 63130.

U.S. Dept. of Energy (1992). *Human Genome: 1991-92 Program Report*. Available from the National Technical Information Service, U.S. Dept. of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

Wu, S., Manber, U. and Myers, E.W. (1992). "A sub-quadratic algorithm for approximate limited expression matching". Technical Report 92-36, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721. Also submitted to *Algorithmica*.