1 of 1

A Software Environment for Large-Scale Sequencing

**Comprehensive Progress Report
February 23, 1991 - July 15, 1993**

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

**Charles B. Lawrence
Department of Cell Biology
Baylor College of Medicine
Houston, TX 77030**

July , 1993

**Prepared for the U.S. Department of Energy
Under Grant Number DE-FG05-91ER61132**

MASTER

Appendix Contents

A Software Environment for Large-scale DNA Sequencing

Appendix Contents

- Comprehensive Progress Report for February 23, 1991 through July 15, 1993 prepared by Charles B. Lawrence, P.I.
- Subcontractor's Comprehensive Progress Report for February 23, 1991 through July 15, 1993 prepared by Eugene Myers.
- Removed* Lawrence, C.B, Honda, S., Parrott, N.W., Flood, T.C., Gu, L., and Myers, E.W. 1993. The Genome Reconstruction Manager (to be submitted to *Genomics*).
- Removed* Lawrence, C.B. and Solovyev, V.V. 1993. Using Position-Specific Error in DNA Sequence Reconstruction (to be submitted to *Genomics* or *Genome Science & Technology*).
- Removed* Honda, S., Parrott, N.W., Smith, R., and Lawrence, C. 1993. An object model for genome information at all levels of resolution. *Proceedings of the 26th Hawaii International Conference on System Sciences, Vol. 1* (eds. T.Mudge, V. Milutinovic, and L.Hunter) IEEE Computer Society Press, Los Alamitos, CA.
- Removed* Myers, E. Advances in sequence assembly. *Automated DNA Sequencing and Analysis Techniques* (ed. C. Venter) Academic Press, London, England (in press)(1993).
- Myers, E. *An interface for a fragment assembly kernel*. (Unpublished document describing the API to software component developed by Dr. Myers for this project.)

A Software Environment for Large-Scale Sequencing

Comprehensive Progress Report

1.0 Description of project

We are developing a next-generation software environment to support large-scale DNA sequencing for the Human Genome Project. The goal is to automate the data flow from its generation by the DNA sequencing hardware to the final reconstructed sequence. Thus, our emphasis is on automation while providing efficient graphical interfaces for interaction with or inspection of the data. A secondary goal is to develop a system flexible enough to support a range of sequencing strategies, including random, and various directed and mixed strategies. The project will result in a software product named the 'Genome Reconstruction Manager' (GRM).

The project has gone through three main phases. In the first phase we engaged in a thorough requirements analysis for the project by studying the strategies and practices used in key DNA sequencing laboratories. During this period we also evaluated the technology available for implementing this system. In the second phase, we developed proof-of-concept prototypes to test some of our ideas and the technology for system implementation. We are presently engaged in the final phase during which we plan to deliver a production quality system to a small number of DNA sequencing laboratories.

By the end of the project we will have accomplished the main objectives stated in our original proposal except that GRM will not include all of the analytical capabilities that were planned. The reason for this is that the system proved to be much more complex than we originally expected and required more effort to be devoted to system design and implementation. However, the design of GRM anticipates the addition of analytical capabilities and our strategy to focus on system design has resulted in a solid foundation on which to build.

2.0 Main results

2.1 Map model

An important result from our system analysis and design was the development of a data model for representing genome related data at all levels of resolution. This is realized as an object model and Class Library called the 'Genome Map Model'. This provides the basis for supporting different sequencing strategies and the ability to integrate the system conveniently with other genome related information resources.

2.2 Reconstruction paradigm

We have introduced a 'reconstruction' paradigm for large-scale sequencing in place of 'assembly'. This emphasizes to the user that the goal of the process is to reconstruct the target genomic sequence. Our system is designed to support this high-level goal and minimize the time spent on trivial details.

2.3 Strategy independence

Our design will successfully support different sequencing strategies. In fact, the two labs that we are initially targetting for deployment of the application are Richard Gibbs' lab at Baylor who is using almost a pure random strategy, and the LBL Genome Center who are using a pure directed strategy.

2.4 Client/server architecture

We have developed a unique client/server architecture that allows us to perform and distribute compute intensive steps on remote host computers.

2.5 Sequence preprocessing

GRM includes a flexible 'preprocessing' step for automatically identifying the subregion from primary sequence data that is useful for reconstruction. During preprocessing the quality of the primary data is assessed and set aside if it fails to meet a minimum standard, sequencing primer and cloning vector sequences are detected and removed, and the region where nucleotide bases are called with the highest reliability is determined.

2.6 Object database management system

GRM employs a commercial object DBMS (Gemstone, Servio, Inc.) to support object persistence.

2.7 Collaboration with Gene Myers

A subcontract on this grant supports the work of Gene Myers to implement algorithmic improvements to reconstructing sequence using the random strategy. We have incorporated the results of his work in a prototype of our system. His group will concentrate on improving the performance of his subsystem in the final year of this contract.

2.8 Position-specific error probability estimates

We have developed a methodology using multivariate discriminant analysis for assigning the probability of misscall, overcall and undercall errors to every nucleotide position in gel-based primary DNA sequence data (using ABI chromatograms as an example). We plan to refine this technology and introduced it into GRM. It will be a key part of our strategy to automate sequence reconstruction and final sequence generation.

3.0 Software

3.1 SeqPrep

A small application called 'SeqPrep' was completed in March, 1992. The purpose of SeqPrep was to test our ideas on sequence preprocessing, to test C++ and X Motif as an implementation technology, and to provide a practical tool for use in Richard Gibbs's laboratory.

SeqPrep has been used in Dr. Gibbs' laboratory since that time, but we have not distributed it because our cost of supporting this product would be too great a drain on our resources.

A significant result of this project was to abandon the use of C++ in favor Smalltalk 80, which we have found to be a vastly more productive implementation environment.

3.2 protoGRM

A prototype of our application, protoCRM, was finished in September, 1992 and demonstrated at the Hilton Head Conference. protoGRM was developed in Smalltalk 80. This experience confirmed that ST80 was an excellent environment with which to develop and deploy our application.

3.3 GRM

The first production release of GRM is presently being implemented. We expect to deploy the application in Richard Gibbs' laboratory by the end of the summer and at the LBL Genome Center in the fall.

4.0 Sequencing collaborators

Our initial strategy for deploying this application is to work with a small number of sequencing laboratories using different sequencing strategies. We plan to refine the design of the system by focussing on these groups.

4.1 Richard Gibbs, Baylor College of Medicine

Richard Gibbs sequencing core laboratory at Baylor has played a key role in our project by providing the basic requirements for the system we are developing. Dr. Gibbs has recently been awarded a grant to sequence 3 megabases from the human Xq28 region.

4.2 LBL, Genome Center

The Genome Center at Lawrence Berkeley Laboratories is committed to scale-up to greater than 1 megabase per year within the next two years using a directed sequencing strategy. We have talked extensively with Jasper Rine, Chris Martin and Ed Thiel at LBL and providing specific support for their strategy in GRM.

5.0 Plans for continuation of present objectives

We plan to submit a proposal to renew the project for another 3 years. During the next funding period, we will focus on improving GRM in response to the needs of the large-scale DNA sequencing community and develop a software distribution and support structure to disseminate the technology widely in the genome research community.

6.0 Students trained

No students or postdocs at Baylor are involved in this project. One graduate student in Eugene Myers' group at the Univ. of Arizona, a subcontractor on this project, has been working on the next generation fragment assembly system.

7.0 Opinion on present state of knowledge in this area of research

For the purposes of this section, I will define the area of research as "computational support for scale-up of laboratory processes in genome research". My opinion is that there is a considerable gap between the state-of-the-art in process and manufacturing control technology and the application of that technology to the genome program. The field would benefit by greater exchange of information among systems developers in the genome program and between the genome program and industry.

This area of research is important to progress in the genome project. Its significance in the field of environmental matters is related to the significance of the genome program to environmental matters.

8.0 Present division of Federal support for research program

This DOE grant represents approximately 75% of the funding for my lab's research and development program. The remaining 25% is for a NLM project that will terminate in March, 1994.

9.0 Bibliography

Honda, S., Parrott, N.W., Smith, R., and Lawrence, C. 1993. An object model for genome information at all levels of resolution. *Proceedings of the 26th Hawaii International Conference on System Sciences, Vol. 1* (eds. T.Mudge, V. Milutinovic, and L.Hunter) IEEE Computer Society Press, Los Alamitos, CA.

Honda, S., Parrott, N.W., and Lawrence, C. B. 1993. Computer-aided sequence reconstruction: Software support for multiple large-scale sequencing strategies. *Automated DNA Sequencing and Analysis Techniques* (ed. C. Venter) Academic Press, London, England (in press).

Lawrence, C.B, Honda, S., Parrott, N.W., Flood, T.C., Gu, L., and Myers, E.W. The Genome Reconstruction Manager (to be submitted).

Lawrence, C.B. and Solovyev, V.V. 1993. Using Position-Specific Error in DNA Sequence Reconstruction (to be submitted).

**A SOFTWARE ENVIRONMENT FOR
LARGE-SCALE SEQUENCING
ARIZONA SUBCONTRACT**

COMPREHENSIVE PROGRESS REPORT
FEBRUARY 23,'91 - JULY 15,'93

*GENE MYERS
DEPT. OF COMPUTER SCIENCE
UNIVERSITY OF ARIZONA
TUCSON, AZ 85721*

PREPARED FOR THE U.S. DEPT. OF ENERGY
UNDER GRANT DE-FG05-91ER61132

A SOFTWARE ENVIRONMENT FOR LARGE-SCALE SEQUENCING

ARIZONA SUBCONTRACT

COMPREHENSIVE PROGRESS REPORT

1. SUMMARY OF ORIGINAL OBJECTIVES

The Arizona team is responsible for producing a software library of routines, we call the assembly kernel, that perform the basic combinatorial tasks involved in solving the sequence reconstruction problem. This construction effort is these investigator's second such system. We proposed to start construction anew from scratch but to bring to bear our previous experience and many new ideas. Algorithmically the task breaks down into an *overlap phase*, than a *layout phase*, and a final *multi-alignment phase*.

The overlap phase compares all fragment sequences against each other looking for approximate overlaps between them that are statistically rare, and summarizes the results of this "all pairs" comparison in an *overlap graph* in which each vertex models a fragment and each significant overlap is modeled as an edge between the corresponding vertices. Our goal for this phase was to perform full sensitivity dynamic programming comparisons (and thus tolerate high sequencing error rates), with an algorithm based on a variation of a sublinear similarity search designed by this investigator. We expect this new algorithm to give us a 20-100 fold speed increase over our previous approach.

The layout phase produces an overlapping arrangement, or *layout*, of the fragments utilizing the overlaps of the first phase that is as short as possible. In order to support mixed mode sequencing strategies and to provide the ability to explore alternate assemblies we realized that we would need to be able to ask for the shortest assemblies subject to a *collection of constraints*. Originally we posited just the need for overlap and orientation constraints, but have since become aware that constraints specifying distance ranges between fragments are also necessary in order to model some experimental protocols. The layout problem is equivalent to the graph-theoretic problem of finding a maximum weight

Hamiltonian path in a simple translation of the overlap graph. We proposed to build a matchings-based relaxation algorithm to optimally solve this problem.

The final multi-alignment phase aligns the fragment data in regions covered by more than two fragments in order to correct errors and produce a consensus sequence that is the most likely estimate of the original DNA sequence. We proposed to realize superior approximation algorithms for this phase by improving upon the second-sweep refinement strategy used in our first system.

At the level of system construction, we proposed to build a well-engineered kernel that adhered to object-oriented design principles and that was as flexible and open-ended as possible. We planned to support version control, provide greater control over the construction of the overlap graph, and in contrast to our first system, shield users from the internal details of the data structures modeling assemblies by realizing them as first-class objects.

2. ACCOMPLISHMENTS TO DATE

The team that has performed all the work on constructing the new assembly kernel consists of systems programmer Susan Larson, doctoral student Mudita Jain, and myself. Susan was hired into the systems programmer position in August of the first year of the grant, and the majority of the funds allocated to the Arizona subcontract are devoted to covering her position. Mudita and I have been supported by my principle NIH grant (see below). Each of us has been responsible for developing and coding a phase of the computation: I've been responsible for the overlap phase, Susan for the layout phase, and Mudita for the multi-alignment phase. The overall design and the algorithmic decisions for each phase has been my responsibility as P.I.

We first designed the interface for the kernel and wrote it up as an internal working document. This was submitted to the Baylor group for their review and after a couple of minor revision cycles a consensus between the two groups was reached.

We then proceeded to develop code realizing the functionality of the kernel using at every step the simplest possible algorithms. This allowed us to construct a kernel the Baylor group could use in the shortest possible time frame. In this way we made a functioning kernel available to them that met the interface specification early on in the project. This minimized the extent to which their development efforts were impeded and encouraged them to build on top of the actual interface. Since the interface hasn't changed as we've improved and refined the software in the kernel, our teams were able to proceed independently thereafter. Whenever we have achieved a new construction milestone, the

Baylor group has simply had to drop the new version of our "engine" in their "chassis". Once this initial prototype was constructed we went on to work on improving the software for each of the individual phases as described below.

The overlap phase of the new kernel is very open ended: users can have explicit control over the inclusion or exclusion of overlaps and can add overlaps constructed by procedures other than those provided in the kernel. Support for version control is available via the introduction of the concept of fragment classes. The initial prototype simply used a basic dynamic programming algorithm for detecting overlaps and large lookup tables to assess their significance. Unlike our first kernel whose overlap algorithm had the idiosyncrasy of only permitting indels, this one allows substitutions in the overlap alignments. The first layer of improvements were (1) to install an improved version of the Four Russians pre-screener to reduce the time taken to compute overlaps, and (2) to replace the large lookup tables with spline encodings that compress the tables by a factor of 50 with an insignificant loss of accuracy. The only remaining refinement is to install the sublinear comparator. The required variation has been designed and involves a more intricate extension of the underlying principle used in the database searching algorithm than originally expected. The code is currently under development and is expected to be completed this Summer.

My perspective on what is important for the layout phase has changed during the grant period. In the last several years, experimentalists have been devising variations on the "pure" shotgun strategy in order to accelerate the process to closure and improve coverage. The essence of these variations is that they provide additional information about the position, overlap, and orientation of fragments with respect to each other. The number of such techniques and the extent to which they are being used in large projects makes it clear that support for such "constrained" problems should be a priority for our new kernel. We've long known that our relaxation algorithm could handle overlap and orientation constraints and have pressed such ability as an asset of our approach. However, it has become clear that constraints on the distance between fragments are also very important and augmenting the relaxation algorithms to handle them appears exceedingly difficult if not simply insoluble. Thus we have returned to the popular greedy algorithm for producing layouts and focused on modifying it to handle constraints of all three types, which can be done although it's still a non-trivial extension. We think that it is more important to properly handle constraints than it is to handle pure shotgun optimally.

For the initial prototype, Susan built just the simple greedy algorithm and all kernel primitives associated with constraints were realized as "no-ops". In a subsequent refinement she went on to add overlap and orientation constraints. We have also realized

that the number of edges considered in producing a layout can be significantly reduced by ignoring all *transitively-implied* edges. An edge from fragments *A* to *C* is transitively-implied if there is a fragment *B* and edges from *A* to *B* and *B* to *C* that place *A* and *C* in the same relationship. This refinement has also been implemented and found to yield a 50% to 70% reduction in the number of edges that are considered. The only remaining refinement requiring completion is to realize distance constraints. We have a preliminary design and plan to finish by Summer's end.

For the final multi-alignment phase, Mudita prototyped contigs as large two dimensional arrays and produced the multi-alignments by simply merging the pairwise alignments of the corresponding layout. Thereafter she went on to produce a space-efficient encoding multi-alignment that employed an AVL-dag of alignment blocks and alignments encoded by the Δ 's of each fragment. A fragment's Δ is the position of all the dashes needed to align a fragment with all others. Such a representation takes only a small percentage of the space required for the fragments themselves and required fairly sophisticated algorithms for searches and updates. Next, Mudita went on to optimize the code for producing the multi-alignments in a way that provided a 5-fold speedup. She is currently working on a second sweep algorithm that further refines the multi-alignment into one that is expected to be optimal almost everywhere when experimental error in the raw sequence is 5% or less. We also expect to complete this final coding development by the end of the Summer.

We have hired a Masters student, Ron Schmidt, to beta test our software during this summer. We are finding it very useful to have such a hired "antagonist". His efforts have revealed numerous bugs in our code and by removing them at our end first, we hope to reduce any exacerbation the Baylor group may experience in their beta tests.

3. CONTINUATION AND FUTURE PLANS

As indicated in the previous section, we plan on finishing all of the primary coding tasks by the end of this Summer and have a well-tested version of the assembly kernel that achieves full functionality. This requires that we (1) complete the sublinear overlap comparator for the overlap phase, (2) realize distance constraints in the layout phase, and (3) add a second sweep refinement algorithm to the multi-alignment phase. During the rest of the grant period which ends this coming February, we are reserving the time that is left to improve efficiency by virtue of simply profiling our code and tuning the "hot spots". We also figure that there will be additional debug work and perhaps slight alterations required as the Baylor group gains experience with our kernel.

For the future we would like to continue the development of our kernel and to further begin the construction of our own fragment assembly GUI (graphics user interface). A preapplication was favorably reviewed by the DOE's Human Genome Program and we will be applying for a 3-year grant independent of the Baylor group's planned continuation application. The two groups have worked well together and we intend to continue to collaborate. The preapplication outlined our reasons for proceeding independently, including a cost savings to the DOE, the leeway to explore some of our own design ideas, and a number of additional interactions. The preapplication went on to describe in detail our future plans which we just quickly summarize as follows. In terms of algorithms for the kernel we will continue to explore building a better layout algorithm in conjunction with Jim Orlin at MIT, we will add primitives for vector prescreening and chimera detection as part of a properly architected kernel, we will permit and incorporate information on base call reliability in collaboration with Bruce Roe at Oklahoma, and we will work on variations that handle repetitive sequence in collaboration with Lee Hood. Unlike the Baylor effort we wish to build a GUI that is focused exclusively on just fragment assembly, and propose to include features such as revision control, hierarchical and auto-correlated browsing, intuitive ways of describing the constraints of a given protocol at the level of the GUI, and automatic focusing to problematic sequencing segments.

4. STUDENT TRAINING

No students were supported by the grant. However, Mudita Jain has been engaged on the project for the last two years. During the grant period the P.I. will have had three doctoral students obtain degrees under his direction (John Kececioğlu, Gerhard Mehldau, and Jim Knight, who is expected to graduate this August).

5. PUBLICATIONS SUPPORTED BY THE GRANT

Kececioğlu, J.D. and E.W. Myers, "Exact and Approximate Algorithms for the Sequence Reconstruction Problem," *Algorithmica*. In press.

Myers, E.W., "Advances in Sequence Assembly," *Automated DNA Sequencing and Analysis Techniques* (ed. C. Ventner), Academic Press, London, England. In press.

Larson, S.J., Mudita, J. and E.W. Myers, "A Kernel in Support of Fragment Assembly," Technical Report TR93-??, Dept. of Computer Science, University of Arizona, Tucson, AZ 85721.

Lawrence, C.B., Honda, S., Parrot, N.W., and E.W. Myers, "The Genome Reconstruction Manager." In progress.

OTHER RELEVANT PAPERS (not directly supported by the grant):

Mehldau, G and E.W. Myers, "A System for Pattern Matching Applications on Biosequence," *CABIOS*. In press.

E.W. Myers, " A Sublinear Algorithm for Approximate Keyword Matching," *Algorithmica*. In press.

Wu, S., Manber, U. and E.W. Myers, "A Sub-Quadratic Algorithm for Approximate Limited Expression Matching," *Algorithmica*. Submitted for publication.

E.W. Myers, "Algorithmic Advances for Searching Biosequence Databases," *Proceedings of the International Symposium on Computational Methods in Genome Research* (Heidelberg, Germany 1992). In press.

E.W. Myers, "Approximate Matching of Network Expressions with Spacers," *Bulletin of Mathematical Biology*. Submitted for publication.

6. OPINION ON ADVANCES IN FRAGMENT ASSEMBLY

The enclosed survey article, "Advances in Sequence Assembly", written for *Automated DNA Sequencing and Analysis Techniques* (ed. C. Ventner) gives my detailed opinion on the current state of affairs with regard to software for sequence assembly. The article lists requirements for algorithms and for systems, and indicates that very few efforts meet all or even a significant number of the specifications. Roger Staden's software is perhaps the best system in use in that it has a decent GUI, but it is strapped on top of the simple-minded, FORTRAN-coded algorithms he developed in the early 80's. The GCG and Intelligenetics packages have poorer interfaces and are no more sophisticated than Roger's work at the algorithmic level. I would evaluate the few other systems in general use similarly to the three above. In general software development for handling the computational aspects of sequence assembly lag behind the continued innovation of new

experimental protocols. To my knowledge, we are the only group that has seriously considered and begun the construction of a "constraint-based" approach for the efficient handling of a wide variety of current and *future* methods. But our efforts are not yet sufficiently mature, a situation we hope to change in the near future.

7. DIVISION OF CURRENT FEDERAL SUPPORT

Problems in Discrete Pattern Matching, Co-PI, National Science Foundation:

Provides 1 month of salary support for the PI. Ends this month (June, 1993) with no plans to renew. (~ \$9,000 in direct costs went to this investigator this year.)

Efficient Software for the Analysis of Biosequences, PI, National Library of Medicine:

PI's main grant. Provides 2 month of salary support for the PI, money for 3 research assistants, and 1/4 of a secretary. Other expenses such as travel, etc. are at the customary levels for the number of people involved. There are no significant moneys for equipment. This grant expires December, 1995. (~ \$105,000 in direct costs went to this investigator this year.)

Software Environment to Support Large Scale Sequencing, Subcontractor, DOE:

The grant that is the subject of this report. It has provided salary of one systems programmer, along with modest supplies and travel moneys for said position. (~ \$65,000 in direct costs went to this investigator this year. Note that Arizona overhead was applied to this amount which was then considered direct costs to Baylor.)

A Laboratory for Programming Languages and Software Systems, PI, NSF:

A large infra-structure grant to the department for equipment purchases. No moneys directly benefit this investigator, but the grant has purchased equipment that the P.I. and his personnel use.

APPENDIX

An Interface for a Fragment Assembly Kernel

1. Overview

The Fragment Assembly Kernel facilitates the creation of three types of objects: overlap graphs, assemblies, and constraint sets. Functions are provided to create and destroy these objects, and to read and write them to and from disk files. Each object created by a Fragment Assembly Kernel function is persistent until it is destroyed by another function.

2. Initialization and Error Handling

Function `fa_startup` initializes the fragment assembly system and must be the first routine called. If `trapflag` is non-zero then `fa_startup` uses the C Library `setjmp/longjmp` mechanism to allow control to return to the point of the call to `fa_startup` in the event of an error. The first call to `fa_startup` returns a 0 but has the important side effect of establishing itself as the return point for error exceptions. Thereafter, whenever an error is detected, control is transferred to the call as if it had just returned, but this time with a non-zero value indicating the type of error. If `fa_startup` is called with `trapflag` set to zero, then on the detection of an error, an error message is output on `stderr` and execution terminates.

```
int    fa_startup (trapflag);
char *fa_error_msg ();
```

Function `fa_error_msg` returns a pointer to a string containing the error message for the most recently detected error.

3. Constructing Overlap Graphs

An overlap graph is constructed by performing a series of additions and deletions of fragments, and of edges between them. In support of version control, fragments may be grouped together into classes with one "representative" fragment being the active fragment of the class at any given time. Many of the operations that follow operate only on the active fragment of each class, which is the desired interpretation when one is thinking of a class as representing a slightly different version of the same raw sequence. However, the primitives below are rich enough to permit the user to override this default mechanism whenever desired.

```
FA_GRAPH *fa_create_graph ();
void      fa_destroy_graph (FA_GRAPH *graph);
```

Function `fa_create_graph` returns a pointer to an empty overlap graph. The routine `fa_destroy_graph` frees the memory consumed by an overlap graph.

```
FA_NAME fa_add_class (FA_GRAPH *graph, char *sequence, int ext_id, int cmp_type,
                     float error_rate, float overlap_threshold);
void     fa_del_class (FA_GRAPH *graph, FA_NAME frag);
```

```
FA_NAME fa_add_frag (FA_GRAPH *graph, FA_NAME frag, char *sequence, int ext_id,
                    int cmp_type, float error_rate, float overlap_threshold);
void     fa_del_frag (FA_GRAPH *graph, FA_NAME frag);
```

```
void fa_active (FA_GRAPH *graph, FA_NAME frag);
```

```
void fa_list_class (FA_GRAPH *graph, FA_NAME frag, void (*handler)());
void handler (FA_NAME fname);
```

Function `fa_add_class` establishes a new fragment class whose sole and active member is the supplied sequence with associated user-supplied id, `ext_id`. The fragment is assigned a name of type `FA_NAME` that is the return value of the call and that must subsequently be used to refer to the fragment. The remaining parameters control the addition of edges between this new fragment and the active members of other classes in the graph. If `cmp_type` is `FA_COMPARE_NONE` then no edges are added and the remaining parameters need not be specified. If `cmp_type` is `FA_COMPARE_ALL` then the new fragment is compared against all active fragments. For a given active fragment, the most significant overlap with score greater than `overlap_threshold` and fewer than

error_rate errors is computed and if one exists an edge representing the overlap is added to the graph. The fa_del_class procedure removes from the overlap graph all fragments in a class and the edges incident to these fragments.

Function fa_add_frag is identical to fa_add_class except that (1) the supplied fragment sequence is added to the pre-existing class containing fragment frag, and (2) the parameter cmp_type may be specified as FA_COMPARE_CLASS. Let A be the active fragment in the class. If cmp_type is FA_COMPARE_CLASS, then the new fragment is compared against the active fragments of the classes containing fragments to which A has an edge. Just before return, the new fragment is made the active fragment of the class. The procedure fa_del_frag removes the specified fragment frag and incident edges from its class and the overlap graph. If frag was the active fragment of the class, then another fragment in the class is randomly selected to be active.

The routine fa_active makes the specified fragment the active fragment of the class containing it. Recall that only one version of a fragment can be active at any given time. The procedure fa_list_class is used to list all the fragments in the class containing fragment frag. The routine handler, passed to fa_list_class, is called once with the FA_NAME of each fragment in the class.

```
int *fa_compare (FA_GRAPH *graph, FA_NAME frag1, FA_NAME frag2,
                float error_rate, float *score, float *erlang,
                int orient, int coord1, int coord2);
```

Function fa_compare returns a list of integers representing an optimal overlap alignment between frag1 and frag2. The overlap is computed using the specified error rate (unless coord1 and coord2 are nonzero, see below). The highest scoring overlap (meeting the constraints, if any) is returned and its score and erlang probability are returned via the pointers score and erlang. The score is $-\log(4)$ of the probability that such an alignment would occur at random. This is a prior odds ratio. The erlang probability is the odds of seeing the number of differences in the alignment given that errors are distributed exponentially with arrival rate $1/\text{error_rate}$.

If the fragments are to be compared in both the same and opposite orientations, the orient parameter should have a zero value. If one wishes to constrain the orientation of the fragments considered, the parameter orient is used as follows: Setting orient to FA_SAME specifies that frag1 and frag2 are to be compared in the same orientation. If orient is FA_COMP, then frag1 is compared to the reverse complement of frag2. If the orientation constraint is given but the last two arguments have a zero value then the best overlap between the fragments in the given orientation is returned.

Finally, if the best alignment subject to a given orientation and overlap interval is desired, then one may further specify the interval with the parameters coord1 and coord2. These parameters specify the beginning and ending positions of the overlap interval. Positive values represent positions in frag1, and negative values indicate positions in frag2. If the product of the values for coord1 and coord2 is positive, a containment overlap is indicated; if this product is negative, it represents a dovetail overlap. For example, if frag1 is 110 characters long and frag2 is 60 characters long, then the (orient, coord1, coord2) triples at left give rise to the overlaps at right:

```
(FA_SAME, 100, -10) <==> frag1[100..110] dovetails frag2[1..10]
(FA_SAME, 20, 85) <==> frag1[ 20.. 85] contains frag2
(FA_SAME, -45, 20) <==> frag2[ 45.. 60] dovetails frag1[1..20]
(FA_SAME, -1, -60) <==> frag2 contains frag1 (with lots of errors)
(FA_COMP, 100, -10) <==> frag1[100..110] dovetails FRAG2[1..10]
(FA_COMP, 20, 85) <==> frag1[ 20.. 85] contains FRAG2
```

where FRAG2 is the reverse complement sequence of frag2. Notice that FRAG2[1..10] is the reverse complement of frag2[51..60]. Also notice that the overlaps may contain errors. For example, the first triple above implies that the last 11 characters of frag1 overlap with the first 10 characters of frag2, so there must also be an insert in frag2 in order for it to align with frag1.

The list of integers returned by fa_compare represents the alignment as follows. The first three integers indicate the orientation and overlap interval, exactly as do the parameters to fa_compare. That is, the first integer in the list is FA_SAME or FA_COMP, indicating whether or not frag2 is reverse-complemented. The second and third integers indicate the overlap interval as above. The remaining integers in the list indicate the positions at which to insert dashes into the two sequences so as to produce the encoded alignment. Specifically, a positive integer, k, indicates that a dash should be inserted before the k'th symbol of frag1, and a negative integer, -k, indicates that a dash should be inserted before the k'th symbol of frag2. The list is terminated with a 0. For example, if frag1 = 'acggtacgttacgatag' and frag2 = 'gtaaacttaagaacgtaa' then the alignment:

```

acggt--acgttacgatacg
gtaaac-ttaaga-acgtaa

```

is specified by the list <FA_SAME,4,-15,6,6,-7,-13,0>.

```

void fa_add_edge (FA_GRAPH *graph, FA_NAME frag1, FA_NAME frag2,
                  int *alignment, float score);
void fa_del_edge (FA_GRAPH *graph, FA_NAME frag1, FA_NAME frag2,
                  int orient, int coord1, int coord2);

void fa_list_edges (FA_GRAPH *graph, FA_NAME frag, void (*handler)());
void handler (FA_NAME fname1, FA_NAME fname2, int *alignment, float score);

```

The procedure `fa_add_edge` adds an edge from `frag1` to `frag2`, using the specified alignment, to an overlap graph. The alignment is represented by a list of integers such as those returned by `fa_compare`. The edge is assigned the designated score for the purposes of computing best layouts. It may be the score returned by `fa_compare` or whatever the user desires (e.g., the length of the overlap). The `fa_del_edge` routine removes from an overlap graph all edges between `frag1` and `frag2` with the specified orientation (FA_SAME or FA_COMP) and overlap interval (`coord1` and `coord2` as described for `fa_compare`). That is, the edge is identified by the first three integers in the list of integers encoding it. If there is more than one edge between `frag1` and `frag2` satisfying the description (but possibly differing in the exact alignment between the overlapped intervals), they all are removed.

The routine `fa_list_edges` can be used to obtain information about all edges incident to the specified fragment in an overlap graph. The routine `handler`, passed to `fa_list_edges`, is called once for each edge incident to the fragment, and is passed the FA_NAMES of the overlapping fragments represented by the edge, the integer list encoding of the alignment (as described for `fa_compare`), and the score of the edge. The value of `fname1` passed to the handler is always the value `frag` passed to `fa_list_edges`.

```

char *fa_sequence (FA_GRAPH *graph, FA_NAME name);
int   fa_length   (FA_GRAPH *graph, FA_NAME name);
int   fa_ext_id   (FA_GRAPH *graph, FA_NAME name);

```

Function `fa_sequence` returns a pointer to the character string for the sequence in the overlap graph with the associated name, or the null pointer, if no such sequence exists. Function `fa_length` returns the length of the named sequence, or 0 if there is no such sequence. Function `fa_ext_id` returns the integer id supplied by the developer when the specified sequence was inserted into the overlap graph. If there is no such sequence, `fa_ext_id` returns 0.

```

void      fa_write_graph (FA_GRAPH *graph, FILE *stream);
FA_GRAPH *fa_read_graph (FILE *stream);

```

Procedure `fa_write_graph` stores an overlap graph in a file. The specified file must be opened for writing when `fa_write_graph` is called. Function `fa_read_graph` reads a previously stored overlap graph from a file which has been opened for reading.

4. Constraints

Constraints can be used to specify that given fragments or edges in an overlap graph are used in an assembly in a particular way, or are not included in an assembly. The following functions can be used to build constraint sets:

```

FA_CSET *fac_all_frgs();

FA_CSET *fac_frag_in (FA_NAME frag);
FA_CSET *fac_frag_out(FA_NAME frag);

FA_CSET *fac_edge_in (FA_NAME frag1, FA_NAME frag2,
                      int orient, int coord1, int coord2);
FA_CSET *fac_edge_out(FA_NAME frag1, FA_NAME frag2,
                      int orient, int coord1, int coord2);

FA_CSET *fac_orient_same(FA_NAME frag1, FA_NAME frag2);

```

```
FA_CSET *fac_orient_opp (FA_NAME frag1, FA_NAME frag2);
```

```
FA_CSET *fac_distance(FA_NAME frag1, FA_NAME frag2, int anchor1,
                      int anchor2, int mingap, int maxgap);
```

Each of the above constraint functions returns a reference to a constraint set containing a single constraint. Constraint sets consisting of collections of these "singleton" constraints can be built using the function `fac_union` to be described later. Note that it is possible for inconsistent constraint sets to be created by unioning conflicting constraints. If such an inconsistency is detected, a warning will be issued, and the assembly will proceed without using the most recently added constraint which caused the inconsistency to be discovered. Also note that when a fundamental constraint above is evaluated, references to fragments are to the active member of the class containing it (see `fa_active` above).

Function `fac_all_fragments` asserts that all (active) fragments are to be considered in the assembly. If two constraint sets are merged using the `fac_union` function and either set contains the 'all_fragments' constraint, the resulting constraint set will assert that all fragments except those referenced by 'frag_out' constraints will be considered in the assembly.

Function `fac_frag_in` asserts that the specified fragment is to be among those assembled. Function `fac_frag_out` asserts that the specified fragment is not included in the assembly. Thus one may specify a set of fragments to be assembled either by listing which ones are in, or by listing which ones are out. That is, in the first case, one builds a constraint set of "fac_frag_in"s, and in the latter case one builds a constraint set of "fac_frag_all" and "fac_frag_out"s.

Function `fac_edge_in` asserts that one of the edges (if any) between `frag1` and `frag2` meeting the the orientation and overlap conditions imposed by `orient`, `coord1`, and `coord2` will be used to overlap the fragments in the resulting assembly. The `orient`, `coord1`, and `coord2` parameters optionally specify the relative orientation and the overlap of the edges to be considered. As in `fa_compare`, one may use a nonzero value for none of these parameters, just the orientation, or all three, in each case specifying a progressively more restricted set of edges to consider. Function `fac_edge_out` asserts that all edges between `frag1` and `frag2` satisfying the orientation and overlap constraints (if in effect), will not be used to overlap the two fragments in the resulting assembly.

Function `fac_orient_same` asserts that the two fragments specified will be in the same orientation in the resulting assembly (if they are in the same contig). Function `fac_orient_opp` asserts that the two fragments specified will be in the opposite orientation in the resulting assembly.

Function `fac_distance` asserts that the `anchor1`'th and `anchor2`'th symbols of `frag1` and `frag2` are to be separated by at least `mingap` characters and at most `maxgap` characters in the resulting assembly. The fragments may be in either orientation with respect to their anchors and the anchors do not necessarily have to be positions in the fragment, e.g., -10 specifies a position 10 characters to the left of the first character of the relevant fragment.

```
FA_CSET *fac_union(FA_CSET *cset1, FA_CSET *cset2);
FA_CSET *fac_copy(FA_CSET *cset);
void      fac_destroy(FA_CSET *cset);
```

Function `fac_union` returns a reference to the constraint set that results from merging two specified constraint sets. When two constraint sets with 'edge in' constraints are merged and the resulting constraint set is applied to an assembly, the 'in-edges' in `cset1` are considered in the assembly before those in `cset2`. Merging constraint sets consumes the references to `cset1` and `cset2`. To retain a reference to either of these constraint sets, `fac_copy` must be called on the reference. Function `fac_copy` creates a new reference to a constraint set and returns the newly created reference. Procedure `fac_destroy` consumes a reference to a constraint set and frees the memory associated with the constraint set if this was the last reference to the object.

As implied by the naming of pointers to constraint sets as references, a reference counter mechanism is used to manipulate constraint sets. While initially a bit awkward, we have found that the conventions above are very flexible and are best illustrated with an example.

```
FA_CSET *cf, *ce, *ca;
FA_NAME f1, f2, f3, f[n+1];

cf = fac_all_fragments();
for (i = 1; i <= n; i++)
    cf = fac_union(cf, fac_frag_out(f[i]));
```

```

ce = fac_union(fac_union(
    fac_edge_in(f1,f2),
    fac_edge_in(f1,f3)),
    fac_edge_in(f2,f3));
ca = fac_union(cf,ce);
fac_destroy(ca);

```

cf becomes a reference to a constraint set specifying that all fragments except f[1..n] should be assembled. Within the loop, fac_union consumes the reference to cf and that returned by fac_frag_out and returns a new one to an object modeling their union. Thus later, when fac_destroy is called on ca, all the objects created are destroyed. If instead one had set "ca = fac_union(fac_copy(cf),ce)", then after the code is executed, cf will still be a valid reference, but all constraints associated with ca and ce will have disappeared.

From another point of view, the fundamental primitives and fac_union return pointers to objects whose reference count is 1. Moreover, fac_union does not modify the reference counts of its operands but since it needs to point at them, it effectively consumes one of the counts. fac_copy increments the reference count. fac_destroy decrements the reference count and if it becomes zero, recursively garbage collects all objects that become unreferenced as a result.

```

void      fac_write(FA_CSET *cset, FILE *stream);
FA_CSET *fac_read(FILE *stream);

```

Procedure fac_write writes a constraint set to a file. The file must be open for writing. Function fac_read returns a reference to a constraint set read from a file. The file must be open for reading.

5. Generating Assemblies

An assembly consists of a layout of fragments and a multi-alignment. The following functions can be used to generate assemblies from an overlap graph:

```

void      fa_init_assemble (FA_GRAPH *graph, FA_CSET *cset);

FA_ASSEMBLY *fa_gen_assembly (FA_GRAPH *graph);
void      fa_destroy_assembly(FA_ASSEMBLY *asm);

void      fa_finis_assemble (FA_GRAPH *graph);

```

Procedure fa_init_assemble prepares for the computation of assemblies over an overlap graph, using a constraint set. (See section on Constraints above). The fa_init_assemble routine must be called before the first call to fa_gen_assembly. The graph is locked by fa_init_assemble, and remains locked until fa_finis_assemble is called. Calling fa_init_assemble on a graph that has already been locked causes an error trap as described at the start of the document.

Function fa_gen_assembly generates the next best fragment assembly over a graph, using the constraint set specified in the call to fa_init_assemble. A pointer to an object of type FA_ASSEMBLY is returned, or the null pointer if there is no next best assembly. The same constraint set is used for each assembly; to use a different constraint set, fa_finis_assemble must be called, followed by a call to fa_init_assemble with the new constraint set. If fa_gen_assembly is called on a graph that has not had assembly initialized via a call to fa_init_assemble, the error is handled as described above. Procedure fa_destroy_assembly frees the data structures associated with an assembly.

Procedure fa_finis_assemble terminates assembly generation over a graph, frees the associated data structures, and disassociates the constraint set specified in the call to fa_init_assemble from the graph.

```

int      fa_num_contigs(FA_ASSEMBLY *asm);
int      fa_contig_height(FA_ASSEMBLY *asm, int ctg);
int      fa_contig_width(FA_ASSEMBLY *asm, int ctg);
float    fa_contig_score(FA_ASSEMBLY *asm, int ctg);

int      fa_contig_view(FA_ASSEMBLY *asm, int ctg, int beg_col, int end_col,
                        int consensus, void (*handler)());
void handler(int row, int col, int frag, int len, int pos);

```

```

int      fa_num_fragments(FA_ASSEMBLY *asm, int ctg);
FA_NAME fa_frag_id(FA_ASSEMBLY *asm, int ctg, int frag);
void     fa_frag_loc(FA_ASSEMBLY *asm, int ctg, int frag,
                    int *row, int *bcol, int *ecol);
char     *fa_frag_eseq(FA_ASSEMBLY *asm, int ctg, int frag);
int      fa_frag_overlap(FA_ASSEMBLY *asm, int ctg, int frag);
int      fa_frag_orient(FA_ASSEMBLY *asm, int ctg, int frag);

```

```

FA_CSET *fa_assembly_seed(FA_ASSEMBLY *asm);

```

Function `fa_num_contigs` returns the number of contigs in an assembly. Function `fa_contig_height` returns the number of rows needed to display an assembly contig. Function `fa_contig_width` returns the length in characters of an assembly contig. Function `fa_contig_score` returns the score of an assembly contig.

Function `fa_contig_view` produces the requested columns of an assembly contig, and calls a user-supplied routine, handler, once for each fragment in the window. If the consensus argument is non-zero, a consensus row will be computed and passed to the handler routine after the last fragment in the window. The `fa_contig_view` function returns a non-zero result if no errors occur while producing the contig rows, and zero otherwise. The handler routine must accept arguments specifying the row being passed, the starting column, the index of the fragment in the contig (or -1 if the consensus row is being passed in), the length in characters of the portion of the fragment in the window, and the first position in the fragment that is within the window.

Function `fa_num_fragments` returns the number of fragments in an assembly contig. Function `fa_frag_id` returns the `FA_NAME` for the fragment with index `frag` in the assembly contig indexed by `ctg`. Procedure `fa_frag_loc` passes back the row, `*row` and beginning and ending column positions, `*bcol` and `*ecol`, of the fragment indexed by `frag` in the assembly contig indexed by `ctg`.

Function `fa_frag_eseq` returns a pointer to a character buffer containing the aligned sequence for the fragment with index `frag` in the assembly contig indexed by `ctg`, or if `frag` is -1, the consensus sequence is returned. The aligned sequence for a fragment is the sequence of characters (including dashes) representing the fragment in the multi-alignment. The buffer containing the aligned sequence is overwritten each time `fa_frag_eseq` is called. Function `fa_frag_overlap` returns `FA_CONTAIN`, `FA_DOVETAIL`, or 0 to indicate whether the fragment indexed by `frag` in assembly contig `ctg` is contained in another fragment, dovetailed with another fragment, or not overlapped by a preceding fragment. If `fa_frag_overlap` indicates that a fragment is not overlapped, the fragment is the root (first fragment) of the contig. Function `fa_frag_orient` returns 1 if the fragment indexed by `frag` in the assembly contig with index `ctg` is reverse-complemented, 0 otherwise.

Function `fa_assembly_seed` returns a reference to a constraint set that contains an edge constraint for the seed edge used to generate the assembly.

```

void      fa_write_assembly(FA_ASSEMBLY *asm, FILE *stream);
FA_ASSEMBLY *fa_read_assembly(FA_GRAPH *graph, FILE *stream);

```

Procedure `fa_write_assembly` writes an assembly to a file. The file must be open for writing, and the graph from which the assembly was generated must be saved independently by calling `fa_write_graph`. Function `fa_read_assembly` reads an assembly in terms of an existing graph. The file from which the assembly is to be read must be open for reading.

```

void      fa_swap_rows(FA_ASSEMBLY *asm, int ctg, int row1, int row2);
void      fa_delete_col(FA_ASSEMBLY *asm, int ctg, int col);
void      fa_insert_col(FA_ASSEMBLY *asm, int ctg, int col, char *seq);
char     *fa_get_col(FA_ASSEMBLY *asm, int ctg, int col);
void      fa_undo_edit(FA_ASSEMBLY *asm, int ctg);

```

The multialignment edit functions operate on the assembly `asm`, in the contig indexed by `ctg`. Procedure `fa_swap_rows` swaps `row1` and `row2`. Function `fa_delete_col` deletes the column specified by `col`. Function `fa_insert_col` inserts the column of characters specified by the null-terminated string `seq` before column `col`. The length of the string `seq` must be the same as the number of rows in the contig. Valid characters for `seq` include blanks and characters representing an encoded nucleotide set. Function `fa_get_col` returns a null-terminated string containing the symbols from column `col`. The string is overwritten each time `fa_get_col` is called. Function `fa_undo_edit` reverses the last edit made to the assembly contig.

**DATE
FILMED**

2/28/94

END