An Editing Environment for DNA Sequence Analysis and Annotation
(Extended Abstract)

Edward C. Uberbacher, Ying Xu, Manesh B. Shah, Victor Olman,
Morey Parang, and Richard J. Mural[†]
Computer Science and Mathematics, and [†]Life Sciences Divisions
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6364, USA

This paper presents a computer system for analyzing and annotating large-scale genomic sequences. The core of the system is a multiple-gene structure identification program, which predicts the most "probable" gene structures based on the given evidence, including pattern recognition, EST and protein homology information. A graphics-based user interface provides an environment which allows the user to interactively control the evidence to be used in the gene identification process. To overcome the computational bottleneck in the database similarity search used in the gene identification process, we have developed an effective way to partition a database into a set of sub-databases of "related" sequences, and reduced the search problem on a large database to a signature identification problem and a search problem on a much smaller sub-database. This reduces the number of sequences to be searched from N to $O(\sqrt{N})$ on average, and hence greatly reduces the search time, where N is the number of sequences in the original database. The system provides the user with the ability to facilitate and modify the analysis and modeling in real time.

1 Introduction

As genome centers world-wide scale up to produce millions of bases of DNA sequence each day, developing fast and high-quality analysis and annotation tools becomes one of the most challenging problems facing the bioinformatics community. We have developed an editing environment for computationally analyzing and annotating genomic sequences. The core of the system takes as input the predicted exons, predicted gene boundaries, partial gene segments (possibly identified based on homology information), and the user's input, and constructs a most "probable" (multiple) gene structure. The user interface provides a graphical editing environment, which allows the user to interactively and iteratively control what and how the available information should be used in the gene modeling process, in an interactive and iterative manner. This environment allows the user to analyze and annotate a genomic sequence on a multi-megabase scale.

The system consists of five main modules. The *exon prediction module* predicts the coding exons; each exon is predicted with a confidence value and

a fixed translation frame. Currently this is done using the GRAIL II exon prediction program (version 1.3) [1,2], and it is being extended to include other exon prediction programs. The *database search module* locates sequences with high similarity scores with each of the predicted exons, using BLASTN (version 1.4.9) [3] and FASTA (version 2.0) [4] in both the dbEST database [5] and the Swissprot database, respectively. The *information extraction module* processes the database search results (alignments) to extract and organize the information related to gene modeling; The information may include indications of the boundaries of exons, evidence of a predicted exon being a true exon, the potential boundaries of genes, the minimal extent of a gene, and indications of false exon predictions and missing exons. The *graphical editor* allows the user to edit the extracted information and to add new information or constraints, based on the user's domain knowledge. The *gene modeling module* makes a prediction of the most "probable" (multiple) gene model based on the predicted exons, extracted database search information, and user input. The gene identification problem is formulated as a combinatorial optimization problem, and solved using a dynamic programming algorithm. Figure 1 schematically shows the structure of the system.

In an interactive environment, it is highly desirable to have the system respond quickly to user input. The slowest part of the system is the *database search module*. One of the main reasons for this is the amount of data a search program like BLASTN has to examine. For example, the dbEST database currently contains over one million entries, and it takes dozens of seconds on a SPARC 20 workstation to compare one typical exon to the database (long genomic regions may contain many candidate exons). One way to overcome this bottleneck is to preprocess the database in such a way that only small subset of the database needs to be searched for a given query sequence, and this small subset of the database can be quickly identified. We have implemented this idea on the dbEST database, and preliminary test results are very encouraging.

2 Sequence Analysis in an Interactive Environment

This section first briefly reviews the gene modeling algorithm (corresponding to the four modules described above except the interface module) [6,7] based on exons predicted by GRAIL II and database search results on dbEST and Swissprot, and then describes the graphic editing environment for sequence analysis and annotation.

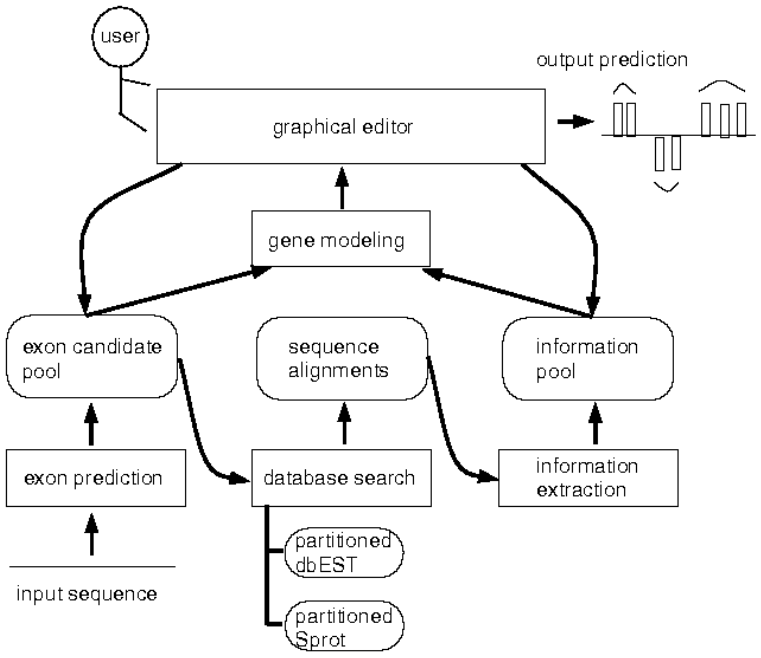

Figure 1: A schematic of the editing environment for genomic sequence analysis and annotation.

2.1 Reference-based gene modeling

Our gene modeling program uses GRAIL-predicted exons as the basic building blocks, and applies information extracted from database similarity search results to guide gene modeling. The extracted information can be used to better determine the correct boundaries of exons, to determine falsely-predicted exons and missing exons, and to suggest potential boundaries of genes.

Figure 2 can be used as an example to illustrate the basic idea of the gene modeling process of the system. Predicted exons are searched against the dbEST database and the Swissprot database using BLASTN and FASTA, respectively, and the matched ESTs or proteins are locally aligned against the genomic sequence (as shown in Figure 2). Overlapping ESTs determine the *minimal extent* of a gene, i.e., exons within each minimal extent region can only belong to one gene. Long stretches of ESTs (ESTs matching more than one exon) may provide indications of falsely-predicted exons and missing

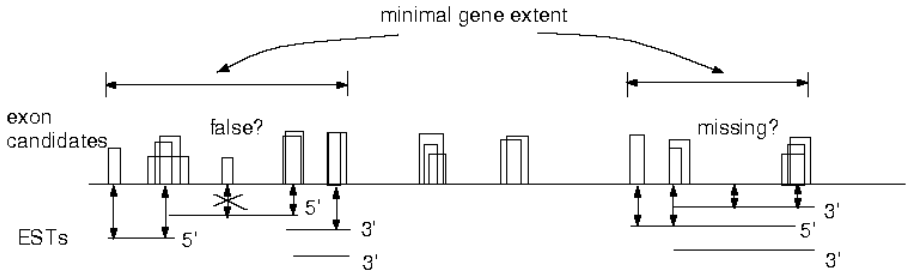

Figure 2: A schematic of information extraction. The X-axis represents the sequence axis. Each rectangle represents a predicted exon candidate, and overlapping rectangles represent different boundary predictions of one presumed exon. The height of a rectangle represent the confidence level of the prediction. Each horizontal line represents an EST sequence matching the corresponding part of the exons above it. ESTs can provide an indication of a gene extent, and verify or contradict an exon's prediction or edges. Homologous proteins can provide similar information.

exons if EST segments contradict predicted exons or exon edge predictions (as shown in Figure 2). The $3'$-most extent of a group of overlapping $3'$ ESTs generally indicate the $3'$-end of a gene though their $5'$ counter parts do not necessarily indicate the beginning of a gene. By combining the $3'$ ESTs, the minimal extent of genes, and a coding strand determination function [2], we can partition the given DNA sequence into a series of *maximal gene extents*, i.e., a gene within a maximal gene extent cannot extend beyond this region. Typically, a maximal gene extent corresponds to a region that usually contains one gene, but could possibly contain several genes, given the current coverage of ESTs. All this gene-structural information is extracted by the *information extraction module*. Using all this information, we have formulated the multiple gene modeling problem in **each** maximal gene extent region (see Figure 4) as the following optimization problem. We call each EST (or protein sequence) matching an exon a *reference model*.

Given are a set of K predicted exons with $score_p(E)$ representing the prediction score of a predicted exon E, and a list of M reference models $\{R_1, ..., R_M\}$. Each exon E has a score $score(E, R)$ with respect to each of its reference model R[a]. For the simplicity of discussion, we define $score(E, \emptyset) = score_p(E)$ and always use R_0 to represent $\emptyset$ as a special reference model. The goal is to select a list $\{E_1, ..., E_n\}$ of nonoverlapping exon candidates from

[a] $score(E, R) = -\infty$ if R does not match E.

the given exon set, a mapping $\mathcal{M}$ from $\{E_1, ..., E_n\}$ to the (extended) reference model list $\{R_0, R_1, ..., R_M\}$, and a partition of $\{E_1, ..., E_n\}$ into D (not predetermined) sublists $\{E_1^1, ..., E_{n_1}^1\}, ..., \{E_1^D, ..., E_{n_D}^D\}$ (corresponding to D (partial) gene models) in such a way that the following function is maximized,

$$
\begin{array}{ll}
\text{maximize} & \sum_{g=1}^{D}(\sum_{i=1}^{n_g} score(E_i^g, \mathcal{M}(E_i^g)) + \sum_{i=2}^{n_g} link(\mathcal{M}(E_{i-1}^g), \\
& \quad \mathcal{M}(E_i^g)) + \mathcal{P}_s(E_1^g) + \mathcal{P}_t(E_{n_g}^g)) \\
\text{subject to:} & (1)\ l(E_1^{g+1}) - r(E_{n_g}^g) \geq \mathcal{L}, \text{ for } g < D, \\
& (2)\ E_{n_g}^g \text{ and } E_1^{g+1} \text{ not belonging to the same minimal} \\
& \quad \text{gene extent, for } g < D, \\
& (3)\ E_i^g \text{ is spliceable to } E_{i+1}^g, \text{ for all } i \in [1, n_g - 1] \\
& \quad \text{and } g \leq D
\end{array}
\qquad (P)
$$

where $l(E)$ and $r(E)$ are the left and right edges of an exon candidate, respectively, $link(X, Y)$ is a reward factor when $X = Y$ and $X \neq R_0$, and is zero otherwise, $\mathcal{P}_s()$ and $\mathcal{P}_t()$ are two penalty factors for a gene missing the initial or terminal exon, respectively, and $\mathcal{L}$ is the minimum distance between two genes. For the definition of two exons being spliceable, we refer the reader to [6].

By solving this problem, we can construct an optimal gene model within each maximal gene extent, based on the given information. This problem can be solved by a fast dynamic programming algorithm[6].

2.2 The graphical editor

The system first constructs an optimal multiple-gene structure fully automatically using the predicted exons and the information extracted from the database search results, and display the predicted gene structure (see Figure 4) to the graphical editor as the baseline for the further refinement, which is done interactively between the system and the user. The graphical editor provides an environment for the user to easily edit the information to be to used in the gene modeling process. Currently the system allows the user to (a) modify the exon candidate pool generated by the exon prediction module; (b) modify the predicted gene boundaries by the information extraction module; (c) to threshold the alignments to be used; (d) select or eliminate ESTs or reference components. As the user makes the modifications, the system automatically updates its information pool by executing the information extraction module. Figure 3 shows the flow of control and data of the system. Figure 4 illustrates a part of the graphical interface.

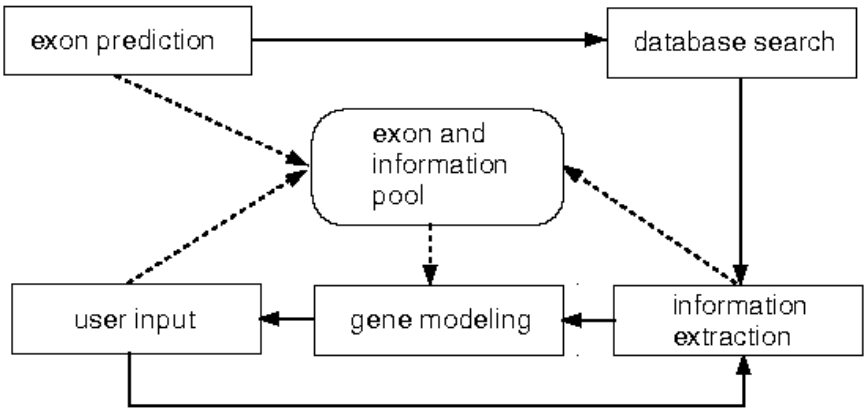

Figure 3: Control and data flow. The solid and dotted arrows represent the flow of control and data, respectively.

2.3 System implementation

The system has been implemented within the JavaGRAIL architecture, a Java application, at ORNL. JavaGRAIL provides an interactive environment for several types of analysis including homology-based multiple-gene modeling for very large sequences. The analysis modules are encapsulated as Java Remote objects and as CORBA objects, since it is likely that both kinds of distributed object environments will become prevalent. Thus, irrespective of the protocol used, other developers can incorporate access to our analysis modules into their systems. In addition, a traditional socket- based client/server system is also implemented.

Exon prediction and homology searches involved in the gene modeling generate a substantial amount of information which needs to be stored for use in gene modeling and refinement. The bulk of this data is retained by the server and only the necessary data and context information is sent back to the client. During the process of gene model refinement by the user, the client needs to send to the server only the context information and the gene modeling parameters selected by the user. The user can perform several iterations of gene model refinement. At the end of the analysis session, this data is passed back to the client, which the client saves on disk for future use.

JavaGRAIL represents the next generation of the XGRAIL sequence analysis system. It is built using the platform-independent, web-based Java lan-

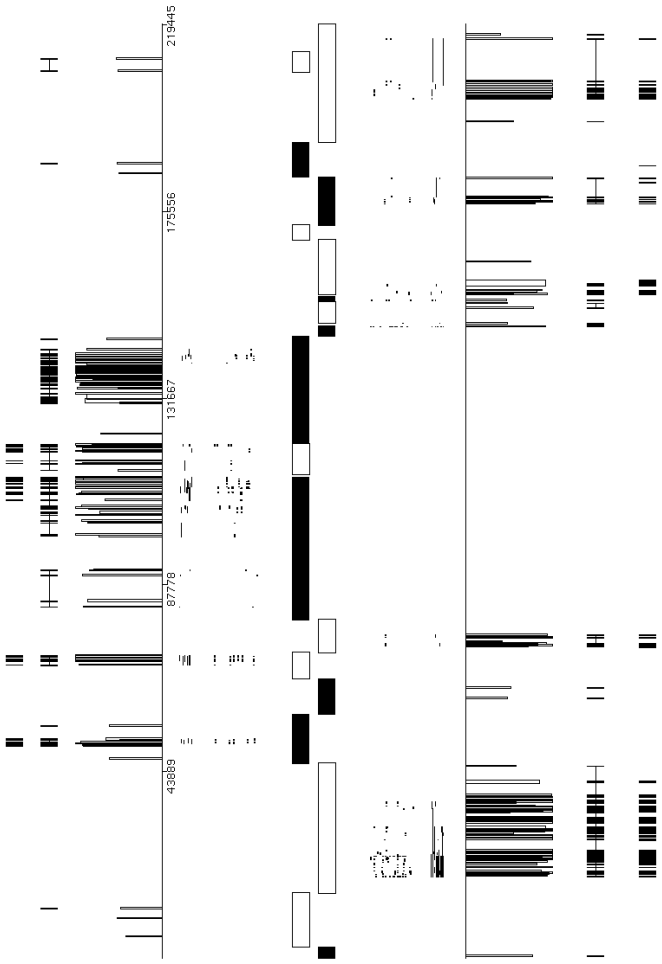

Figure 4: Information editing. The solid bars in both the top and the bottom represent the positions of the annotated GenBank coding exons in the forward and reverse strand, respectively. The solid bars in the next-to-top and next-to-bottom rows represent the exons in the predicted gene models, including non-coding exons. Each set of bars connected through a line represent one gene model. The hollow rectangles represent the predicted GRAIL exons. The short lines (or dots) represent the matched ESTs. The boxes (alternately hollow and solid) in the middle represent boundaries of maximal gene extent, within which gene models are built. A user can modify the boundaries of genes, bboundaries of maximal gene extent, threshold the ESTs to be used, select individual ESTs or reference models, modify the exon candidates or their boundaries, etc.

guage, making it portable. It is a front-end to a distributed object architecture, using Java (and CORBA) Remote Objects for accessing sequence analysis services and data. Distributed objects make it convenient to develop complex systems using plug-and-play paradigm, by combining several simpler components or objects together. JavaGRAIL currently supports the services developed and maintained at ORNL, namely, GRAIL exon prediction, GenQuest database search engine, homology-based multiple gene structure identification. Access to other exon and gene prediction systems, and other kinds of analysis engines will be added later.

3 Database Partitioning for Fast Similarity Search

The computational bottleneck of the system is sequence similarity search. To solve this problem, we pre-process the database in such a way that only a small (related) subset of the database needs to be searched for a given query. This section outlines this implementation on the dbEST database.

3.1 Partition of dbEST database

The EST database can be partitioned into a set of sub-databases such that no two sequences from different sub-databases have a BLASTN score higher than some pre-set parameter s (we also used the p-score as a part of the thresholding), and two sequences S and S' in the same sub-database either have an BLASTN score above s, or there exits a chain of sequences $S_1, ...S_n$ such that the BLASTN scores between S and S_1, S_n and S', and all S_i and S_{i+1}, $i \in [1, n-1]$, are above the threshold s. Note that if we have such a database partitioning, searching the EST database can be done by first identifying one or potentially several sub-databases and then searching the identified sub-database(s).

We have implemented this idea on the EST database as follows. We first randomly select one sequence in dbEST, and run BLASTN using it as the query sequence to find all the as yet unfound sequences with BLASTN alignment scores above s (we call them *related* sequences); Then select one of the newly found sequence as the query, and repeat this process. This process continues until no more sequences can be added to the tree. This tree forms one sub-database. Then we can repeat the above by selecting an unfound sequence as the query until no unfound sequences are left. This process can be illustrated by the following tree-based search structure, where each node represents a query sequence, and all its children nodes represent its matched (unfound) sequences.

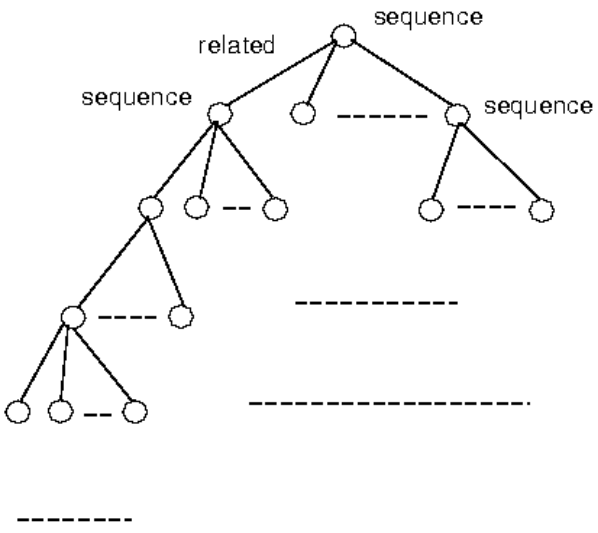

Figure 5: Construction of sub-database of "related" sequences. Each node in the tree represents a sequence, and the children of each node represent all the (as yet) unfound sequences with alignment scores above some threshold.

The following pseudo-code describes this procedure.

Procedure tree_based_search (*query*)

Step 1: get_all_unfound_related_sequences (*query*, *list_of_sequences*).
Step 2: store_all_sequences_into_subdatabase (*list_of_sequences*).
Step 3: **while** (*list_of_sequences* $\neq \emptyset$)
    query $\leftarrow$ first_sequence (*list_of_sequences*);
    remove *query* from *list_of_sequences*;
    tree_based_search (*query*).

In the actual implementation[b], instead of selecting one newly found sequence, we append a number of sequences to form the query sequence. This speeds up the process. In our current implementation, we append 30 sequences to form the next query sequence. By repeating the above procedure, we can

[b] Repeats are removed before we run the partitioning algorithm.

partition a database into a set of subdatabases of related sequences. Currently it takes about one day to partition 10% of the dbEST database on a single-processor SPARC 20 workstation. As new sequences are added to dbEST, we don't need to re-partition the database, instead we only need to assign the new sequences to the "correct" sub-databases, and probably have to merge some of the sub-databases if a new sequence matched sequence from different sub-databases. Overall repartitioning can be carried out periodically.

3.2 Searching the partitioned database

The key for identifying the "correct" sub-database(s) to search for a given query sequence is to design a distinguishing signature for each sub-database. We have used the list of the M most frequent twelve-mers of each sub-database as the signature of the sub-database (in our current implementation, $M \leq 10000$). When given a query sequence, we calculate the number of exact matches of the twelve-mers of the query sequence with each of the signatures, and use the one with the highest number of matches as the identified sub-databases (to be safe, we can search a few sub-databases with the highest matches). Test results have shown that this signature is quite effective in identifying the "correct" sub-database(s).

To make the signature identification process fast, we have pre-processed all the signatures using a technique described in [8] so that each comparison between the query and a signature takes only linear time of the query size (independent of the signature size).

Note that the total search time depends on the total time for searching all the signatures and the total time for searching the identified sub-database(s). A balance between these two times will make our search very efficient. Based on our tests, many of the sub-databases contain only a few sequences, and hence this makes the number of sub-databases large. We are currently experimenting with different methods to merge some of the small sub-databases to make their size as close to $\sqrt{N}$ as possible, without losing the identity of the sub-database. After merging, we have roughly about $O(\sqrt{N})$ sub-databases, and on average, each of them has about $O(\sqrt{N})$ sequences. Hence on average, we can expect that for a given query, we need to compare about $O(\sqrt{N})$ signatures, and search about $O(\sqrt{N})$ sequences in the identified sub-database(s). On a test on 10% of the dbEST database (about 100000 sequences), we partitioned it into about 1000 sub-databases.

4 Summary

We have developed an interactive environment for genomic sequence analysis and annotation, which allows the user to incorporate his/her domain knowledge into the gene modeling process in an interactive manner. By combining statistical information-based prediction methods, sequence alignment information with ESTs and protein homolog, and human domain expertise, this user-friendly environment should provide a powerful tool to molecular bioloists for analyzing and annotating DNA sequences in a timely fashion.

Acknowledgements

This research was supported by the United States Department of Energy, under contract DE-AC05-84OR21400 with Lockheed Martin Energy Systems, Inc.

1. E. C. Uberbacher and R. J. Mural, "Locating Protein-coding Regions in Human DNA Sequences by a Multiple Sensors-neural Network Approach", *Proc. Natl. Acad. Sci. USA*, Vol. 88, pp. 11261 - 11265, 1991.
2. Y. Xu, R J. Mural, J. R. Einstein, M. B. Shah, and E. C. Uberbacher, "GRAIL: A Multi-Agent Neural Network System for Gene Identification", *Proceedings of The IEEE*, Vol. 84, pp. 1544 - 1552, 1996.
3. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tools", *J. Mol. Biol.*, Vol. 215, pp. 403 - 410, 1990.
4. W. R. Pearson and D. J. Lipman, "Improved Tools for Biological Sequence Comparison", *Proc. Natl. Acad. Sci. USA*, Vol. 85, pp. 2444 - 2448, 1988.
5. M. S. Boguski, T. M. Lowe, and C. M. Tolstoshev, "dbEST – Database for Expressed Sequence Tags", *Nat. Genet.*, Vol. 4, pp. 332 - 333, 1993.
6. Y. Xu and E. C. Uberbacher, "Automated Gene Identification in Large-scale Genomic Sequences", *Journal of Computational Biology*, Vol. 4, 3. pp. 325 - 338, 1997.
7. Y. Xu, R. J. Mural and E. C. Uberbacher, "Inferring Gene Structures in Genomic Sequences Using Expressed Sequence Tags", *The Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pp. 344 - 353, AAAI Press, 1997.
8. A. V. Aho and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", *Communications of ACM*, Vol. 18, 6, pp. 333 - 340.