CONF-930117--5

Overview of the Integrated Genomic Data System (IGD)*

Ray Hagstrom* George S. Michaels† Ross Overbeek* Morgan Price*
Ronald Taylor†

*Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4801

†Division of Computer Resources and Technology
National Institutes of Health
Bethesda, MD 20892

ANL/CP--77546

DE93 004805

Abstract

In previous work, we developed a database system to support analysis of the *E. coli* genome. That system provided a pidgin-English query facility, rudimentary pattern-matching capabilities, and the ability to rapidly extract answers to a wide variety of questions about the organization of the E. coli genome. To enable the comparative analysis of the genomes from different species, we have designed and implemented a new prototype database system, called the Integrated Genomic Database (IGD). IGD extends our earlier effort by incorporating a set of curator's tools that facilitate the incorporation of physical and genetic data, together with the results of genome organization analysis, into a common database system. Additional tools for extracting, manipulating, and analyzing data are planned.

1 Introduction

With the current advances in molecular biology and genome science, the ability to perform comparative analysis of the genome organization between species is rapidly becoming a reality. Such comparative analysis requires access to a variety of genomic map information [4]. This map information includes genetic maps as defined by genetic linkage analysis, genome mapping data such as clone hybridization results, restriction enzyme fragment mapping (high-resolution maps from clones and electrophoresis fragment maps for a chromosome), and physical data from DNA sequencing.

Providing flexible access to such diverse data presents a challenging problem in data representation. In earlier work on the *E. coli* genome [1-3], we designed a database based on the programming language Prolog. We chose Prolog because of its inherent relational database characteristics, powerful query capabilities, and ability for rapid prototyping. After we added a pidgin-English query interface, biologists were able to begin analyzing the integrated clone, physical map, sequence fragment, and genetic data.

*This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

To enable comparative analysis of genomes from different species, we have designed and implemented a new prototype database called the Integrated Genomic Database (IGD). IGD initially will help those users attempting to organize and curate genomic data, including maps, sequence, and alignments. Eventually, the system will include facilities for easy visual access and pidgin-English access to a wide variety of types of data relating to a collection of genomes. Ultimately, we envision that IGD will be used both by research biologists and by teachers in classroom instruction.

1.1 Components of the IGD System

The IGD system is currently composed of a number of distinct tools that we are integrating into a coherent framework:

1. A set of tools designed to extract data from archival databases (such as GenBank [5], EMBL [6], and PIR [7]). These tools reformat the data into Prolog for insertion into our database.

2. A Prolog database that supports flexible access to the data, along with a toolkit designed to support curatorial functions (such as redundancy control, recognition of similarity, meld support, and positioning of sequence fragments on chromosomes). There is also a pidgin-English command framework for common queries).

3. GenoGraphics [8], a system that supports visual display and reasoning about the data in the Prolog database. (Note that GenoGraphics can function as a stand-alone tool, accepting data from outside the Prolog database as long as it is properly formatted. GenoGraphics has its own internal database system, separate from the Prolog database. We transfer data from the Prolog database to this database when we want visual display.) It can produce displays of physical maps, genetic maps, cytogenetic maps, sequence fragments, attached features, GC content, and a variety of other forms of the data. In addition, it includes facilities for searching sequence for patterns (not as sophisticated as what can be done in the Prolog database, but quite good) and for attaching comments to objects (including graphics images, a feature of interest in documenting specific curatorial decisions).

Each of these three components addresses a critical problem facing anyone attempting to curate or access genomic data. We will maintain basic documentation on all three components as they evolve.

1.2 Use of the IGD System

The IGD system is intended to be used as follows. First, genomic data is extracted and reformatted into Prolog clauses, which constitute the "logical integration" of the data. This initial step converts data from available archival sources into a consistent representation that offers flexible access.

Then, using the tools offered by the Prolog database component, a curator goes through the following steps:

1. generation of a non-redundant data collection by removing redundant sequence fragments from the different data collections (often making decisions concerning which of several possible versions of the data to keep), and then

2. placing sequence fragments on locations or intervals of the relevant genome (using information in physical maps, genetic maps, and sequence similarity to related genomes). Clearly, issues relating to multiple versions of sequence fragments are controversial (when to keep alleles, when to discard possible sequencing errors, identification of included vector sequences, and so on). The proper handling of these "social" issues is not completely worked out yet. These are problems for a domain expert to decide. However, we are making every effort to produce a framework where such issues can be addressed in a productive manner.

Once an initial version of the data has been determined, a visual presentation of the integration is created by using GenoGraphics. We have found that numerous curatorial decisions are more easily made when the effects of such decisions can be visually summarized. (We are tightly integrating GenoGraphics to the Prolog database component. The flow of data from the Prolog database into GenoGraphics is already automated and simple, but options will be added so that the transfer will become even easier to perform for display of features on the genome that contain complex patterns.) Once an initial integration has been achieved, the curator can construct a variety of detailed GenoGraphic maps that visually portray many useful aspects of the data, including the distribution of interesting sequence features and the GC content of regions.

1.3 Availability

We are making stand-alone versions of GenoGraphics available in the public domain. These will be the main form of mass distribution of the integrated data from multiple genomes. GenoGraphics runs both under MS-DOS on PCs and on Sun workstations under Unix and X-Windows. The MS-DOS version of GenoGraphics contains its own windowing environment and does not need Microsoft Windows.

We also plan to make the Prolog database, along with the GenoGraphics tool, available to labs capable of supporting the associated computational environment. (Unix is needed at present, together with Quintus prolog and, as a matter of practicality, some sort of Unix shell such as Gnu-Emacs, which allows *post facto* access to the screen output stream; we are, however, actively pursuing the transfer of the Prolog database to a version of Prolog that runs under MS-DOS so that the entire IGD system will run on an IBM PC or PC-clone.) The initial forms of this release will be of a system that is still openly viewed as a prototyping environment and will be intended for users that are comfortable with Unix workstation computing environments.

Since the IGD system is in an initial and evolving stage, original users must be aware of the basic function of each of the distinct tools and how they interact. The remainder of this document addresses these points.

2 Converting Archived Data into the Prolog Representation

The conversion of archived data to IGD system format is a three-step process.

1. GenBank or EMBL locus entries are converted to Prolog clause format. Essentially, each field of an entry (locus, definition, comment, features, etc.) is translated into one argument of a clause. Also, some information is extracted from the surrounding text and stored in separate arguments: the locus id, the (first) accession number, and the base listing concatenated together as one unbroken string, among others. The locus entries taken as input can be in any of four formats: NCBI GenBank, Intelligenetics GenBank, GCG GenBank, or EMBL. In addition, we are currently implementing the same sort of import/export facility to the new "acedb" database [9], which has been adopted by the *C. elegans* and *Arabidopsis thaliana* research communities.

   Optional step: At this point, the user may enter into the IGD some subset of the original loci data (now converted into Prolog format) rather than all of the data. A procedure exists that allows the user to select out only those loci that meet one (or a combination) of the following criteria:

   - A specific locus_id is found. A match is looked for in the locus_id field against a user-defined list of locus_ids.

- A specific accession number is found. A match is looked for in the accession number field against a user-defined list of accession numbers.
- A specific source organism is found. A match is looked for in the source_organism field against a user-defined list of organisms. Example of such a list: ['E.coli','E. coli','Esch coli'].
- A specific string is found. A match is looked for across all fields against a user-defined list of strings.

2. Each feature and associated sequence from the feature table for each locus (now in Prolog format) is extracted and converted to Prolog clause format. The base sequence of the feature is analyzed and information is stored on whether (a) a join was used, (b) the sequence was an integer multiple of three, (c) the "direct" or "complement" strand was used, (d) a start codon begins the sequence, and (e) a stop codon appears before, at, or immediately after the end of the sequence (or is missing entirely).

3. The Prolog-formatted locus entries and features are entered into the actual IGD system by conversion into IGD objects. Locus entries become objects of type "sequence_fragment". The original locus id is assigned as the unique id of each such object. Feature objects fall into many classes or types, depending on how they were labeled in the original feature table in the archived data. For example, a coding-sequence feature labeled "cds" in the feature table would be given the type "cds". Similarly, we would obtain these types: "mRNA", "tRNA", "prim_transcript", "misc_feature", "promoter", etc. If the feature describes a gene, the conversion program tries to automatically pull out the name of the gene from the associated note lines for that feature so that it may serve as the primary identifier. Otherwise, a unique number is assigned to the feature. The locus entry and feature base sequence data are broken out and placed in separate clauses in another file for faster, more efficient access.

3 Detailed Format of the Prolog Database

At the present stage of IGD's development, a curator is expected to have a basic understanding of how data is represented in the Prolog database. Eventually, we hope to hide as much of this detail as possible. We begin with some examples and then discuss the general form.

3.1 Examples of Data in Prolog Representation

clones:

```
        format:
object(CloneId,Level,ListOfAttributes).

        example:
object('[101]9E4'(clone,'E.coli'),0,[length(17100)]).

        format:
precisely_bound(ObjectId,ListOfSubPiecesOfParent).

        example:
precisely_bound('[101]9E4'(clone,'E.coli'),
        [to('EC1'(chromosome,'E.coli'),387,17486)]).
```

(The value 'EC1' denotes the first—and in this case sole—chromosome of *E. coli*. The chromosome number is concatenated onto the genomic abbreviation 'EC' to form this identifier.)
restriction enzyme cutting sites (rsites):

```
object('EcoR5'(restriction_site),0,[site("GATATC"),cuts_at(3)]).
precisely_bound('EcoR5'(restriction_site),
[to('EC1'(chromosome,'E.coli'),6800,6805)]).
```

sequence fragments:

```
        Example derived from a GenBank entry:
        object('STYARALC'(sequence_fragment,'Salmonella'),0,
                [length(1286),nuc_seq_ref(94290)]).
        object('STYARALC'(sequence_fragment,'Salmonella'),1,
                [constituents([constituent('STYARALC',0,1285,direct,1286)])]).
        imprecisely_bound('STYARALC'(sequence_fragment,'Salmonella'),
                [to('SA1'(chromosome,'Salmonella'))]).

        Example of a meld:
object(meld13(sequence_fragment,'Salmonella'),0,
                        [length(5277),nuc_seq_ref(171681)]).
object(meld13(sequence_fragment,'Salmonella'),1,
                      [constituents([constituent('STYFLIG',0,2754,direct,2755),
                           constituent('STYFLIHIJ',132,2653,direct,2654)])]).
precisely_bound(meld13(sequence_fragment,'Salmonella'),
[to('SA1'(chromosome,'Salmonella'),2023079,2028355)]).
```

features:

```
object(aphC(cds,'Salmonella'),0,
               [note("alkyl hydroperoxide reductase C22 protein (aphC)"),
                codon_start(1)]).
precisely_bound(aphC(cds,'Salmonella'),
             [to('STYAHPCFA'(sequence_fragment,'Salmonella'),165,719,direct)]).
```

(Note that if the last argument been "complement" rather than "direct", it would have meant "take the subsequence of STYAHPCFA in the range 165 to 719, and then take the reverse complement of the result".)
mapped_genes:

```
        object(atbB(mapped_gene,'Salmonella'),0,
            [position(7.60),map('Sanderson')]).
        object(atbB(mapped_gene,'Salmonella'),1,
            [
            mnemonic("Attachment"),
            desc("attP27II; second attachment site for prophage P27"),
```

```
    refs("[410,412,418]")
    ]).
precisely_bound(atbB(mapped_gene,'Salmonella'),
     [to('SA1'(chromosome,'Salmonella'),384238,384338)]).
```

(Not all mapped genes need to be thought of as "bound to the chromosome at an approximate base pair position, but they should move to that state fairly rapidly.)

3.2 General Format

Facts describing objects in the Prolog database are of the form

```
object(Id,Level,[Attributes]).
```

These are referred to as 'object/3" clauses, since the name (in Prolog terminology, the "functor") of the clause is "object", and the clause has three arguments. There may be (and typically is) more than one object/3 clause associated with a given object. Such multiple clauses have the same Id, but differ in the Level and Attributes arguments. (The reasons for the parceling out of an object's data over more than one object/3 clause are described in Section 3.2.1.) The brackets ("[ ]") in the third argument above signify that we place a list of attributes (which consists of one or more members), rather than just one attribute, in that argument.

Facts constraining location of the object are placed in the form

```
precisely_bound(Id,[WhatItIsBoundTo])
```

or

```
imprecisely_bound(Id,[WhatItIsBoundTo]).
```

An object is precisely bound to a "containing" object (or set of objects) in which its relative position is known. The containing set would include more than one entry only for objects that are discontiguous pieces within a containing object (i.e., a gene with introns in a sequence fragment) or, in an even more general case, for objects that are composed of pieces from distinct "containing objects". (These will be rare.) In the case of precisely bound objects, the WhatItIsBoundTo entries will be of the form

```
to(Id,Beg,End) or to(Id,Beg,End,Direction)
```

where Beg and End are offsets (in base pairs, no matter how approximate) and Direction will be either 'direct' or 'complement'. Id will be the Id of the containing object (more on this below).

An object is imprecisely bound if the contained object occurs somewhere in the circumscribed interval. In the case of imprecisely bound objects, the WhatItIsBoundTo entries will be of the form

```
to(Id) or to(Interval)
```

where Interval is of the form

```
interval(Id,Beg,End) or interval(Point1,Point2)
```

where Beg and End are offsets (again, in bp), and Point1/Point2 would be points described by facts of the form

```
point(PointName,Id,Offset)
```

defining PointName as Offset into Id.

In the current database, objects of most types have either one precisely_bound/2 clause or one imprecisely_bound/2 clause. However, there is no theoretical factor barring us from placing the same object onto multiple larger objects or placing an object at multiple locations on the same object (which is exactly what we do for restriction sites).

Our framework allows us to have a hierarchy of objects, with each object being precisely (or imprecisely) bound to a location in a larger object. At present, we typically have a three-part hierarchy:

```
   feature or clone or rsite -- bound to -->
sequence fragment -- bound to-->
chromosome.
```

or a two-part hierarchy:

```
   mapped gene or rsite -- bound to --> chromosome
```

(Sequence fragments, which are objects themselves, of course, have just a two-part hierarchy: sequence fragment -¿ chromosome. The largest source of sequence fragments for our database at present are the locus entries in the GenBank and EMBL databases, or melds thereof. For the *E. coli* genome, substantial additional information has come from the ECOSEQ5 database compiled by Kenneth Rudd [10-11]. Genetic linkage data has been taken from the Bachmann *E. coli* map [12] and the Sanderson Salmonella map [13].)

3.2.1 Efficiency Issues

When the built-in Prolog query-answering "engine" accesses a clause containing a string of characters, it must convert that string into another format for internal use. There is no way to shut off this automatic conversion. Therefore, to keep the response time of queries to the system as small as possible, we try to avoid accessing clauses containing long strings. One way to do this is to introduce levels in the definition of objects in order to segregate attributes into classes. The levels are numbered 0, 1, 2, ... The clauses corresponding to these levels occur in the appropriate sequence (i before j iff i ¡ j). Attributes are of the form

```
keyword(value)
```

Attributes at level 0 typically have fairly small values (and are few in number); they should be just the most commonly referenced attributes. Longer attributes (such as "constituents" of a sequence fragment or "descriptions") are relegated to high levels. Our system is written so that the level 0 clause is tried first. If the desired data is not present in the list of Attributes in that clause, we then access the level 1 clause for the object, then the level 2 clause, and so on, until the query is either satisfied or we have run out of levels (clauses) to try for that object. We store long strings (such as, say, the comment field from a GenBank entry describing a sequence fragment) at higher levels (typically level 2).

A second technique that we have implemented to increase efficiency involves indexing. The Id of an object may include many "fields". An Id is intended to be an unambiguous designator of the object. The natural way to specify such an id would be

```
type(K1,K2,K3...)  [e.g., prim_transcript(aphC,'E.coli')]
```

Unfortunately, this would incur an enormous performance penalty, due to the first-argument indexing scheme used in Prolog systems. (Since the type – prim_transcript in our example – could be expected to be duplicated many times, rather than being a unique case, response time will grow as the system searches through all the possibilities.) Hence, we try to use the most discriminating of the fields as the functor. For sequence fragments derived from GenBank, for example, this means using the GenBank locus id as the functor of an object's id. This does not offer perfect indexing, but is a good compromise (with the alternative of forming an atom made up of the relevant fields).

The intent of these representational conventions is to leave us with an extremely flexible, extensible system. We believe that we will be able to absorb the complexity of GDB and GenBank into a single framework using these conventions.

4 Use of the Prolog Database

The Prolog database offers access to integrated data across multiple genomes. It can be used to support curatorial activity, prepare maps for GenoGraphics, or to support comparative analysis of genomes. There are two basic levels of use: through a pidgin-English interface and with "raw Prolog." Since the most common use will be through the pidgin-English interface, we present a brief overview.

To gain access to the Prolog database, one simply types

```
IGD
```

and the system should respond (after some initialization messages) with

```
| ?-
```

which is the Prolog prompt. The user then types

```
query.
```

which causes the system to initialize and respond with the pidgin English prompt,

```
query:
```

At such a prompt, the user types in a command. Each command can spread over multiple lines, but it is terminated when a period is the last character of a line. The system will attempt to parse the command and then respond appropriately. For example, here is a record of a "session" composed of exactly two simple queries:

```
============================================================
query: print all sequence_fragments of Vaccinia.

print all sequence_fragments of Vaccinia

sequence_fragment VACCG of Vaccinia
    (location=chromosome VA1[0,191736]):
    accession_num: M35027
    constituents: VACCG[0,191736,direct]
    length: 191737

query: print occurrences of pattern in VACCG.

print occurrences of pattern in sequence_fragment VACCG of Vaccinia

pattern: p1=14...14 3...20 ~p1

--------------- 2 found ---------------
interval [16071,16112] of sequence_fragment VACCG of Vaccinia
GCCGGTGTAATAGA ATTATATATATCTA TCTATTACACCGGC

interval [130102,130134] of sequence_fragment VACCG of Vaccinia
AAAAAAACTATCTA TGCGG TAGATAGTTTTTTT

============================================================
```

The first query

```
print all sequence_fragments of Vaccinia.
```

caused the system to determine how many sequence fragments were included in the genome of Vaccinia. There was only one, VACCG (which covers the entire genome). The second query

```
print occurrences of pattern in VACCG.
```

caused the system to prompt for the particular pattern for which it should scan. Note that the end of the pattern is detected by an end-of-line (i.e., it is not delimited by a period). The pattern typed in by the user requested a search for hairpins with perfect stems of length 14 and a cap (loop) of 3 to 20 characters. The IGD found two matches to this pattern to report.

The general format for search patterns goes as follows. We think of a pattern as a sequence of pattern units, each of which can be (a) a string of DNA characters (including the codes to represent ambiguous characters), (b) a pattern unit that matches an arbitrary string of characters, where the length of the string varies between specified bounds, (c) a pattern unit that "matches" the reverse complement of a string matched by a previous pattern unit, or (d) a pattern unit that matches a string identical to a previously matched pattern unit. Both of the last two types of pattern units allow one to specify an allowable number of mismatches, insertions, and deletions (which gives an "approximate" matching capability).

For example, we would think of the pattern

```
p1=AYGG 3...5 ~p1 p1
```

as consisting of four subunits: p1, followed by three to five bases of any composition, followed by the reverse complement of p1, followed by p1 again.

Hence this pattern is capable of matching a sequence like ACGGTTCGCCGTACGG. (ACGG fits p1's AYGG pattern, TTCG falls within the allowed three to five arbitrary bases, CCGT fits p1, and ACGG fits the last p1 subunit.)

The IGD session is terminated with the command:

```
query: quit.
```

The set of possible queries/commands will eventually be far richer than the current set (it will certainly include all of the query classes implemented in our previous *E. coli* database system).

As with our previous system, we believe that the easiest way to describe the set of allowable queries is by showing examples. Our hope is that the user with a specific query in mind will be able to mimic one of the examples (if not, the required functionality probably is not yet in the pidgin-English interface). So, here are typical examples, along with short explanations of what is produced.

(1) Retrieving the data for the gene fliC

A user would type the following:

```
query: print fliC.
```

The response is:

```
 4 possible interpretations

    1: print cds fliC of E.coli
    2: print mapped_gene fliC of E.coli
    3: print cds fliC of Salmonella
    4: print mapped_gene fliC of Salmonella

Which would you like?  [1]: 3
```

The system understands that there is more than one interpretation of the request, and allows the user to choose which is correct. Once a choice is made (the third interpretation is chosen in the above example), the following appears:

```
print cds fliC of Salmonella

cds fliC of Salmonella
    (location= *** multiple locs:
sequence_fragment STYFLICA[0,851] complement;
        sequence_fragment STYFLID[0,63] complement):
    mnemonic: Flagella
    desc: H1; Flagellar synthesis; flagellin (filament structural protein);
              phase 1 flagellin gene
    refs: [143,199,205,222,243,247,302,410,412,418,507,508]
```

Note that the query system first prints out the query, as it understands it, before executing it. This is useful as a check on what the system is actually doing, as compared to what the user might think it is doing.

(2) Finding data for a specific coding region (cds)

The following interaction requests all information about the cds "A13L". Since there is only one database entry for A13L, there is no interpretation conflict to be resolved.

```
query: print the sequence of A13L.
```

The response is:

```
print the sequence of cds A13L of Vaccinia

cds A13L of Vaccinia
    (location=sequence_fragment VACCG[126535,126747] complement):
    note: A13L
    codon_start: 1
                                                    sequence

     0          ATGATTGGTA TTCTTTTGTT GATCGGTATT TGCGTAGCAG TTACCGTCGC
     50         CATCCTATAC TCGATGTATA ATAAGATCAA GAACTCACAA AATCCGAATC
     100        CAAGTCCGAA TTTAAATTCG CCTCCTCCAG AACCAAAAAA TACCAAGTTT
     150        GTAAATAATC TGGAAAAGGA TCATATTAGT TCATTGTATA ATCTAGTTAA
     200        ATCTTCTGTA TAA
```

(3) Retrieving the sequence around a specific point

To get the sequence that includes 20 bases on either side of the start for cds A13L, the following is entered:

```
query: print the sequence of the interval from the start of A13L - 20
query: to the end of A13L + 20.
```

The response is:

```
print the sequence of the interval from the start of
cds A13L of Vaccinia - 20 to the end of cds A13L of Vaccinia + 20

interval [126515,126767,complement] of sequence_fragment VACCG of Vaccinia
                                                    sequence

     0          AGTGAAGTTA TTGTCAATAA ATGAT     ; TTCTTTTGTT GATCGGTATT
     50         TGCGTAGCAG TTACCGTCGC CATCCTATAC TCGATGTATA ATAAGATCAA
     100        GAACTCACAA AATCCGAATC CAAGTCCGAA TTTAAATTCG CCTCCTCCAG
     150        AACCAAAAAA TACCAAGTTT GTAAATAATC TGGAAAAGGA TCATATTAGT
     200        TCATTGTATA ATCTAGTTAA ATCTTCTGTA TAAATAAAAA TATTTTTAGC
     250        TTC
```

(4) Finding upstream and downstream regions relative to chromosome features
The following is an example of such a regional description:

```
query: print the sequence of upstream(A13L,10).
```

The response is:

```
print the sequence of the interval from the start of
cds A13L of Vaccinia - 10 to the start of cds A13L of Vaccinia - 1

interval [126748,126757,complement] of sequence_fragment VACCG of Vaccinia
                                                    sequence

     0          TTGTCAATAA
```

Alternatively, one can ask to see the downstream region.

```
query: print the sequence of downstream(A13L,10).
```

The response is:

```
print the sequence of the interval from the end of
cds A13L of Vaccinia + 1 to the end of cds A13L of Vaccinia + 10

interval [126525,126534,complement] of sequence_fragment VACCG of Vaccinia
                                                    sequence

     0          ATAAAAATAT
```

(5) Finding common substrings

One of the earliest queries often made to compare similar regions between species involves looking for sequences in common for specific gene regions that are believed to have similar functions. The following is an example of such a request.

```
query: print all sequences of length at least 10 common to {aceA,aceE}.
```

The response is:

```
 9 possible interpretations

    1: print all sequences of length at least 10 common to
       {cds aceA of E.coli, cds aceE of E.coli}
    2: print all sequences of length at least 10 common to
       {cds aceA of E.coli, mapped_gene aceE of E.coli}
.
.  (rest of output omitted)
.

Which would you like?  [1]: 1
```

After the user makes her choice above, the following is displayed:

```
print all sequences of length at least 10 common to
{cds aceA of E.coli, cds aceE of E.coli}

14 sequences in common in the following objects:
====
----- 1 -----
cds aceA of E.coli
    (location=sequence_fragment hydGecoM[14191,15495] direct):
    accession: EG10022
    SwissProt: P05313

----- 2 -----
cds aceE of E.coli
    (location=sequence_fragment ampDecoM[4713,7373] direct):
    accession: EG10024
    SwissProt: P06958

====
ATCTGGAACTGG length=12
    positions in object 1 are [880]
    positions in object 2 are [223]
.
.
. (omitted)
.
```

```
TCGAAAAAAG length=10
    positions in object 1 are [180]
    positions in object 2 are [291]
```

(6) Searching for patterns

Structural patterns such as hairpins can be searched for inside specific regions. This capability is demonstrated in the following interaction. Here the user is looking for all hairpin structures with a fully base paired stem of 7–8 bases in length and a loop of 3–8 bases in the DNA fragment rrnDecoM.

```
query: print occurrences of pattern in rrnDecoM.
```

The response is:

```
print occurrences of pattern in sequence_fragment rrnDecoM of E.coli

pattern: p1=7...8 3...8 ~p1
--------------- 8 found ---------------
interval [3290,3307] of sequence_fragment rrnDecoM of E.coli
AGTGTGT CGAA ACACACT
.
.
.
.

interval [4088,4109] of sequence_fragment rrnDecoM of E.coli
TGGAGTC GAGTTGCA GACTCCA
```

(7) Searching sets of objects

The following interaction demonstrates how to use the previously defined haaripin pattern to search through the 20 base region surrounding the start of every cds feature defined for the Vaccinia genome.

```
query: for each cds in Vaccinia $X, print occs of pat in the interval
query: from the start of $X - 20 to the start of $X + 20.
```

The response is:

```
for all cds of Vaccinia $X, print occurrences of pattern in the interval
from the start of $X - 20 to the start of $X + 20

pattern: p1=7...7 3...8 ~p1
====<< context >>====
----- 1 -----
cds A10L of Vaccinia
```

```
    (location=sequence_fragment VACCG[122285,124960] complement):
    note: A10L
    codon_start: 1

====
interval [2,23] of interval(VACCG(sequence_fragment,Vaccinia),124940,124980,
complement)
TCATTTA ATACTAAA TAAATGA

.
.
.
.

======<< context >>====
----- 1 -----
cds I8R of Vaccinia
    (location=sequence_fragment VACCG[70073,72103] direct):
    note: I8R
    codon_start: 1

interval [8,26] of interval(VACCG(sequence_fragment,Vaccinia),70053,70093,
direct)
TTTCCAT TTATC ATGGAAA
```

(8) fastA comparison of two sequence fragments

Access is also provided to established sequence comparison programs. For example, the following shows how to use the fastA algorithm [14] to compare two sequenced regions from different genomes.

```
query: fastA comparison of STYADA and rrnDecoM showing best 2.
```

The response is:

```
fastA comparison of sequence fragment STYADA of Salmonella and
rrnDecoM of E.coli showing best 2

comparing STYADA/Salmonella and rrnDecoM/E.coli:
          STYADA/Salmonella will show as tmp_fasta
  78.6% identity in 28 nt overlap

          860       870       880       890       900       910
tmp_fa CGCCGTTGCTGCGCGCGCGCATTATCCGGCTGACAGCGCTTGCAGGGGCGAAAACCGGCG
                                        X::: ::::  ::::: :::::: : :X
rrnDec CAATGGCATAAGCCAGCTTGACTGCGAGCGTGACGGCGCGAGCAGGTGCGAAAGCAGGTC
     4730      4740      4750      4760      4770      4780

          920       930       940       950       960       970
tmp_fa TCCAGCGCCTGCTGCGCGTTGGCAAAAAAGCGAACATTTTTACGTAACGCCCGCTTCGAG
```

```
rrnDec ATAGTGATCCGGTGGTTCTGAATGGAAGGGCCATCGCTCAACGGATAAAAGGTACTCCGG
     4790      4800      4810      4820      4830      4840

rrnDecoM-reversed                                          66   66   68
  68.4% identity in 38 nt overlap

          290       300       310       320       330       340
tmp_fa CTGTTGAAACGCGGTTCCCTGGATATCCAGCGGCAACGAGAGCAGCACATCGCGTGTGTT
                                    : X::::::: ::: : : :  : : ::::
rrnDec AGCTCGTGTTGTGAAATGTTGGGTTAAGTCCCGCAACGAGCGCAACCCTTATCCTTTGTT
             1500      1510      1520      1530      1540      1550

          350       360       370       380       390       400
tmp_fa GATAGCCGCGACAACCTGCCGCACCCGTTGCTGAAAAAGCGCGTCAGCAGGTTCGTGGCG
       :  ::X :
rrnDec GCCAGCGGTCCGGCCGGGAACTCAAAGGAGACTGCCAGTGATAAACTGGAGGAAGGTGGG
             1560      1570      1580      1590      1600      1610

Library scan:  0:00:01  total CPU time:  0:00:03
```

(9) Getting codon usage and n-mer decompositions

It is also possible to evaluate the simple statistics for nucleotide composition for a genome. The following interaction shows how to get both codon usage for the defined cds features of a genome and the full trinucleotide statistics.

```
query: print the codon usage of cds in Vaccinia.
```

The response is:

```
print the codon usage of all cds of Vaccinia

number valid codons = 60974
number invalid codons = 0

    alanine: 2310        3.79%
        GCA: 690         1.13%
        GCC: 409         0.67%
        GCG: 440         0.72%
        GCT: 771         1.26%
.
.
.
.

    valine: 3753         6.16%
        GTA: 1425        2.34%
        GTC: 535         0.88%
```

```
        GTG: 558          0.92%
        GTT: 1235         2.03%

query: print the 3-mer decomposition of cds in Vaccinia.
```

The response is:

```
print the 3-mer decomposition of all cds of Vaccinia

 AAA: 7572              4.15%
 AAC: 3493              1.92%
 AAG: 3180              1.74%
 AAT: 6247              3.42%
.
.
.
.
 TTC: 3031              1.66%
 TTG: 2889              1.58%
 TTT: 5086              2.79%
```

(10) Directing output to files

For further statistical analysis, data files can be generated that could be used by some external statistical package. The following shows how to create such output files.

```
query: to file.
Where should the output go? tmp
```

The response is:

```
to file [tmp]

\begin{verbatim}

query: print the 4-mer decomposition of cds of Vaccinia.
```

The response is:

```
print the 4-mer decomposition of all cds of Vaccinia

*** This writes suitably formatted output to the file "tmp". Here are the first
*** few lines:

2757 AAAA
1394 AAAC
1237 AAAG
2184 AAAT
```

```
1274 AACA
 431 AACC
    .
    .
    .
```

Redirection back to the terminal is done by the following:

```
query: to person.
```

The response is:

```
to person
```

(11) Conducting basic curating functions with the "undo" facility

The assembly of a genome requires a domain expert who has sufficient knowledge of the genetics and an understanding of how the data was derived. That curator would begin by gathering genetic and physical mapping data to assemble a view of the genome. This task requires tools that allow consistent mapping of features to the constituent fragments that will be melded. A melding tool is needed to generate contigs that accurately reflect the features assigned to the constituents. We also need a means of backing out of changes to the genome view (i.e., of "rolling back") as new information becomes available. The IGD system accomplishes all of these tasks through the use of an update log that records the necessary manipulations of sequence assignments and assembly in memory. This allows nondestructive manipulations that can be reversed at any time. The following curator's interaction demonstrates many of these functionalities.

```
query: print features on STYGLGC.
```

The response is:

```
print features on sequence fragment STYGLGC of Salmonella

--------------- 2 found ---------------
cds glgA of Salmonella
    (location=sequence_fragment STYGLGC[0,87] complement):
    mnemonic: Glycogen
    desc: Starch (bacterial glycogen) synthase (ECf 2.4.1.21)
    refs: [290,291,412,418]

cds glgC of Salmonella
    (location= *** multiple locs: sequence_fragment STYGLGC[87,1382] complement;
                                  sequence_fragment STYGLGCG[0,2] complement):
    mnemonic: Glycogen
    desc: Glucose-1-phosphate adenylyltransferase (EC 2.7.7.27)
```

```
    refs: [290,291,412,418]
```

```
query: delete STYGLGCG.
```

The response is:

```
delete sequence_fragment STYGLGCG of Salmonella
```

```
query: show update history.
```

The response is:

```
show update history

**********
1. delete sequence_fragment STYGLGCG of Salmonella
**********
```

```
query: print cds glgC of Salmonella.
```

The response is:

```
print cds glgC of Salmonella

cds glgC of Salmonella
    (location=sequence_fragment STYGLGC[87,1382] complement):
    mnemonic: Glycogen
    desc: Glucose-1-phosphate adenylyltransferase (EC 2.7.7.27)
    refs: [290,291,412,418]
```

```
query: undo.
```

The response is:

```
undo
```

```
query: print cds glgC of Salmonella.
```

The response is:

```
print cds glgC of Salmonella

cds glgC of Salmonella
    (location= *** multiple locs: sequence_fragment STYGLGC[87,1382] complement;
                                  sequence_fragment STYGLGCG[0,2] complement):
    mnemonic: Glycogen
    desc: Glucose-1-phosphate adenylyltransferase (EC 2.7.7.27)
    refs: [290,291,412,418]


query: fastA comparison of STYGLGC and STYGLGCG.
```

The response is:

```
fastA comparison of sequence fragment STYGLGC of Salmonella and STYGLGCG of
Salmonella showing best 1

comparing STYGLGC/Salmonella and STYGLGCG/Salmonella: STYGLGC/Salmonella
will show as tmp_fasta 88.9% identity in 27 nt overlap

             1360      1370      1380      1390      1400
tmp_fa CATTACACGATCGTTCTTCTCTAAACTCACCATAGTTAACTCCTTTTTTA-CCTCTCAAG
                                    X::  :::::::::::::: v^::::
STYGLG                              CATGATTAACTCCTTTTTTATCCTCTCTGG
                                           10        20        30

    1410
tmp_fa G

STYGLG AACACACACAGTCCATGCGCAGGTCCCTGCCAGAGCGCCGTTATCACCGGATTATCCTCT
               40        50        60        70        80        90

Library scan:  0:00:00  total CPU time:  0:00:00


query: print the sequence of STYGLGC.
```

The response is:

```
print the sequence of sequence_fragment STYGLGC of Salmonella

sequence_fragment STYGLGC of Salmonella
    (location=chromosome SA1[1226202,1227611]):
    length: 1410
                                                  sequence

     0          GTCGCGCAGG CAACGCGCCT ATCACATCCG CCAGCCCCCC AGTCTTCAGC
     50         AGGGGGAACA TCTCTGAACA TACATGTAAA ACCTGCATTA TCGCTCCTGT
```

```
     1350       CATTACACGA TCGTTCTTCT CTAAACTCAC CATAGTTAAC TCCTTTTTTA
     1400       CCTCTCAAGG
```

```
query: print the sequence of STYGLGCG.
```

The response is:

```
print the sequence of sequence_fragment STYGLGCG of Salmonella

sequence_fragment STYGLGCG of Salmonella
    (location=chromosome SA1[3589324,3589781]):
    length: 458
                                                  sequence

     0          CATGATTAAC TCCTTTTTTA TCCTCTCTGG AACACACACA GTCCATGCGC
     50         AGGTCCCTGC CAGAGCGCCG TTATCACCGG ATTATCCTCT CCAGCGAATG
     100        GGGGAACGGC GCGCCATTCC CCTTCGGGTA AAACTATATC TGTCACCTCA
     150        AGCGTGGCAT TTATCGCAAT CAGAAAACGG TCCGACAACA GAATTTGCAT
     200        CNAGCTTAGG CCCGTTTTGC CACTCANTCC GCACTCAAGG GTTGCGCGTT
     250        TTTATTCAGC CAACGCACGT TGCCATCGCC TTCTTCCCAC CAGCTATTGC
     300        CGGTTAAAGC CGGTATCTGC TGACGCAAAC GAATCAGCGC GGCGGTAAAC
     350        GTGGTTAACC CACGATTTGC CTGCTGCCAG TCCAGCCAGG TTAAGGCATT
     400        ATCCTGACAG TAGGCGTTAT TGTTGCCATG CTGGCTATGG CCGTGTTCAT
     450        CGCCTGCC
```

```
query: meld STYGLGC positions 0 to 1399 with STYGLGCG positions 20 to 457.
```

The user is prompted for the name of the resulting meld:

```
giving? glgC_meld
```

The response is:

```
meld sequence fragment STYGLGC of Salmonella positions 0 to 1399 with
     sequence fragment STYGLGCG positions 20 to 457 [giving glgC_meld]
```

```
query: print cds glgC of Salmonella.
```

The response is:

```
print cds glgC of Salmonella

cds glgC of Salmonella
    (location=sequence_fragment glgC_meld[87,1382] complement):
    mnemonic: Glycogen
    desc: Glucose-1-phosphate adenylyltransferase (EC 2.7.7.27)
    refs: [290,291,412,418]
```

Note that the cds "glgC" now appears on the newly created meld.

```
query: relocate glgC_meld to SA1+45670.
```

The response is:

```
relocate sequence_fragment glgC_meld of Salmonella to chromosome SA1
of Salmonella+45670
```

```
query: show update history.
```

The response is:

```
show update history

**********
1. meld sequence fragment STYGLGC of Salmonella positions 0 to 1399 with
     sequence fragment STYGLGCG positions 20 to 457 [giving glgC_meld]
2. relocate sequence_fragment glgC_meld of Salmonella to chromosome SA1
of Salmonella+45670
**********
```

```
query: undo.
```

The response is:

```
undo
```

```
query: show update history.
```

The response is:

```
show update history

**********
```

```
1. meld sequence fragment STYGLGC of Salmonella positions 0 to 1399 with
     sequence fragment STYGLGCG positions 20 to 457 [giving glgC_meld]
********************
```

(12) Handling related objects (a meager start)
Two sample queries in this area are shown below.

```
query: print all objects related to cds fliC of Salmonella.
```

The response is:

```
print all objects related to cds fliC of Salmonella

cds fliC of E.coli
    (location=sequence_fragment fliCecoM[101,1597] complement):
    accession: EG10321
    SwissProt: P04949


query: print all objects related as same_function to fliC of Salmonella.
 2 possible interpretations

    1: print  all objects related as same_function to cds fliC of Salmonella
    2: print  all objects related as same_function to mapped_gene fliC
       of Salmonella

Which would you like?  [1]: 2
```

The response is:

```
print all objects related as same_function to mapped_gene fliC of Salmonella

mapped_gene fliC of E.coli
    (location=chromosome EC1[2017641,2017741]):
    position: 4.252E+01
    map: Bachmann
```

(13) Checking constituents
Melded sequences are built from constituent sequences. One of the curator checks to validate the assembly process is demonstrated below.

```
query: check constituents of alkBecoM.
```

The response is:

```
check constituents of sequence_fragment alkBecoM of E.coli

alkBecoM is ok

query: check constituents of appAecoM.
```

The response is:

```
check constituents of sequence_fragment appAecoM of E.coli

appAecoM can not be constructed from constituents (see position 577)
  99.9% identity in 2076 nt overlap

               10        20        30        40        50        60
tmp_fa GATCTCCAGCCTGACGTTGTGGGACAGTACTTCCAGTCAGCTGACGCTGAGCATTATGTT
       X:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
appAec GATCTCCAGCCTGACGTTGTGGGACAGTACTTCCAGTCAGCTGACGCTGAGCATTATGTT
               10        20        30        40        50        60

.
.
.

              550       560       570       580       590       600
tmp_fa CAACCTGGCCGGTAAAACTGGGTTGGCTGACACCGCGNGGTGGTGAGCTAATCGCCTATC
       ::::::::::::::::::::::::::::::::::::::::.:::::::::::::::::::::
appAec CAACCTGGCCGGTAAAACTGGGTTGGCTGACACCGCGTGGTGGTGAGCTAATCGCCTATC
              550       560       570       580       590       600
.
.
.


             2050      2060      2070
tmp_fa TCGTTCGCCCATTCCAGTAATTGACGCATCCGATCG
       ::::::::::::::::::::::::::::::::::X
appAec TCGTTCGCCCATTCCAGTAATTGACGCATCCGATCG
             2050      2060      2070

Library scan:  0:00:00  total CPU time:  0:00:01
```

This concludes our set of sample pidgin-English queries.

5 GenoGraphics and Its Use

The basic functionality of the GenoGraphics system is described in detail in the GenoGraphics users' manual. In this section, we describe how a curator may visualize his work in the IGD by means of GenoGraphics.

GenoGraphics is used to graphically peruse the results of a series of map construction operations. An example of the IGD interaction for making a collection of computed map data is demonstrated below.

```
query: make maps for E.coli.
```

The response is:

```
make maps for E.coli.

completed making map EC01SF
completed making map EC01SG
completed making map EC01GC
.
.
.
```

```
query: make maps for Salmonella.
```

The response is:

```
make maps for Salmonella.

completed making map SA01SF
completed making map SA01SG
completed making map SA01GC
completed making map SA01MG
completed making map SA00SF
completed making map SA00SG
completed making map SA00GC

query: compose {EC01CL,EC01RS,EC00GC,EC00SF,
query:          EC00SG,EC01GC,EC01SF,EC01SG,
query:          EC01MG,SA01MG,SA01SG,SA01SF,
query:          SA01GC,SA00GC,SA00SF,SA00SG}.

Where should the output go? Maps/EC_SA
```

In this session, the composite map information for *E. coli* and Salmonella is computed separately for each map. Then aligned maps are "composed" for viewing with GenoGraphics. The IGD places the selected composition of maps in the file "EC_SA" in the directory "Maps." To view the maps, the user starts GenoGraphics as a separate process. Once inside GenoGraphics, the EC_SA file is accessed as a normal GenoGraphics data file. Figure 1 shows a subset (seven of the *E. coli* maps) of the set of aligned maps created above. Figure 2 shows a zoomed window of the map data revealing the details of the several of the *E. coli* and Salmonella aligned maps. The connecting lines between objects on separate maps indicate sequences of high similarity between *E. coli* and *S. typhimurium* (strain LT2).

6 Conclusion

We set out to create a flexible database system that would allow the integration of several different types of genomic data. The IGD prototype presented here has proven to be adaptable enough to allow the incorporation of data from several archival database systems as well as laboratory data supplied directly. In this paper we have provided an overview of the IGD system. We have described how one interacts with the system, using the incorporated data from several model organisms. We have already used this system to incorporate the cosmid/YAC hybridization data for all three chromosomes of *S. pombe*, the complete DNA sequence for chromosome 3 of yeast, and clone restriction fragment mapping data.

We intend to continue expanding the breadth of chromosome analysis tools that can take advantage of this database technology. We have already implemented portions of David Searls's GenLang grammar/parser for DNA sequence patterns [15, 16]. The tRNA subgrammar has been used successfully to identify all of the sequenced tRNAs for *E. coli* found in the integrated data. The results of this analysis will be presented in another paper.

We also intend to expand the Prolog database "upward" so that it contains information on proteins (alignments, evolutionary relationships, and other information of the sort stored in the Swiss Protein Database, which we are currently putting into Prolog format) and on metabolic pathways. In addition, we will be adding interfaces for other existing genome database management tools, including "acedb" and other commercial DBMS systems. The goal is to provide an integrated database and analysis environment that is capable of dealing with the ever more complex questions concerning the comparisons of chromosome organization between species.

References

[1] Baehr, A.; Dunham, G.; Ginsburg, A.; Hagstrom, R.; Joerg, D.; Kazic, T.; Matsuda, H.; Michaels, G.; Overbeek, R.; Rudd, K.; Smith, C.; Taylor, R.; Yoshida, K.; and Zawada, D., *An Integrated Database to Support Research on Escherichia coli*, Argonne National Laboratory report ANL-92/1 (1992)

[2] Michaels, G.; Kazic, T.; Overbeek, R.; Zawada, D.; Dunham, G.; Rudd, K.; and Smith, C.; *Logic programming-based system for querying E. coli chromosomal information*, Cold Spring Harbor Genomic Mapping and Sequencing meeting, May 8–12, 1991

[3] Kazic, T.; Michaels, G.; Overbeek, R.; Zawada, D.; Dunham, G.; and Rudd, K., *An integrated database of E. coli chromosomal information to support queries and rapid prototyping*, AAAI Workshop on Approaches to Classification and Pattern Recognition in Molecular Biology, Anaheim, Calif., July 12, 1991

[4] Jaworski, M.; and Edwards, E., *Integrated genetic databases in the study of neuropsychiatric diseases: inborn errors of cerebral metabolic pathways*, Prog. Neuropsychopharmacol. Biol. Psychiatry 15: 171–81 (1991)

[5] Burks, C.; et al., *GenBank: Current Status and Future Directions*, Methods in Enzymology 183: 3–22 (1990)

[6] Kahn, P.; and Cameron, G., *EMBL Data Library*, Methods in Enzymology 183: 23–31 (1990)

[7] Sidman, K. E.; George, D. G.; Barker, W. C.; and Hunt, L. T., *The protein identification resource (PIR)*, Nucleic Acids Res. 16: 1869–71 (1988)

[8] Hagstrom, R.; Michaels, G. S.; Overbeek, R.; Price, M.; Taylor, R.; Yoshida, K.; and Zawada, D., *GenoGraphics for OpenWindows*, Argonne National Laboratory report ANL-92/11 (April 1992)

[9] Sulston, J.; Mallett, F.; Staden, R.; Durbin, R.; Horsnell, T.; and Coulson, A., *Software for genome mapping by fingerprinting techniques*, Comput. Appl. Biosci. 4: 125–32 (1988)

[10] Rudd, K. E.; Miller, W.; Ostell, J.; and Benson, D. A., *Alignment of Escherichia coli K12 DNA sequences to a genomic restriction map*, Nucleic Acids Res. 18: 313–21 (1990)

[11] Rudd, K. E.; Miller, W.; Werner, C.; Ostell, J.; Tolstoshev, C.; and Satterfield, S. G., *Mapping sequenced E.coli genes by computer: software, strategies and examples*, Nucleic Acids Res. 19: 637–47 (1991)

[12] Bachmann, B. J., *Linkage map of Escherichia coli K-12, edition 8*, Microbiological Reviews 54: 130–97 (1990)

[13] Sanderson, K. E.; and Roth, J. R., *Linkage map of Salmonella typhimurium, edition VI*, Microbiological Reviews 52:485–532, (1988)

[14] Pearson, W., *Rapid and sensitive sequence comparison with FASTP and FASTA*, Methods in Enzymology 183: 63–98 (1990)

[15] Searls, D. B., *Investigating the Linguistics of DNA with Definite Clause Grammars*, Proceedings of the North American Conference on Logic Programming, vol. 1, pp. 189–208 (1989)

[16] Searls, D. B., *The Computational Linguistics of Biological Sequences*, Unisys Paoli Research Center report CAIT-KSA-9010 (1990)

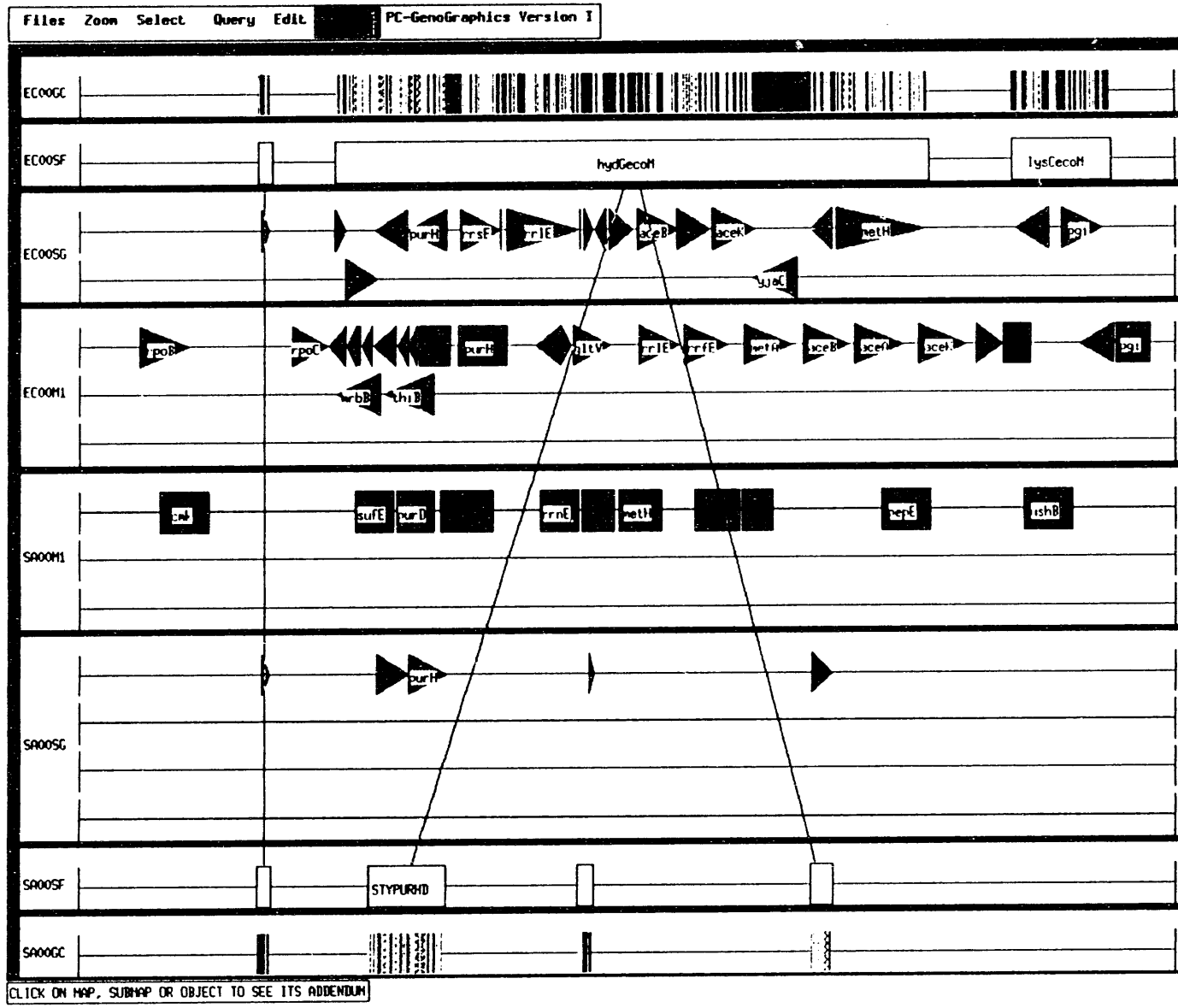
Files Zoom Select Query Edit
PC-GenoGraphics Version I
ECOOGC
ECOOSF
hydGecoM
lysCecoM
ECOOSG
purH
rrsF
rrlE
aceB
aceK
metH
pgl
yjaC
ECOOM1
rpoB
rpoC
purB
gltV
rrlE
rrfE
metH
aceB
aceK
aceI
pgl
thiB
SA00M1
sufE
purD
rrnE
metH
ushB
SA00SG
purH
SA00SF
STYPURHD
SA00GC
CLICK ON MAP, SUBMAP OR OBJECT TO SEE ITS ADDENDUM

Figure 1

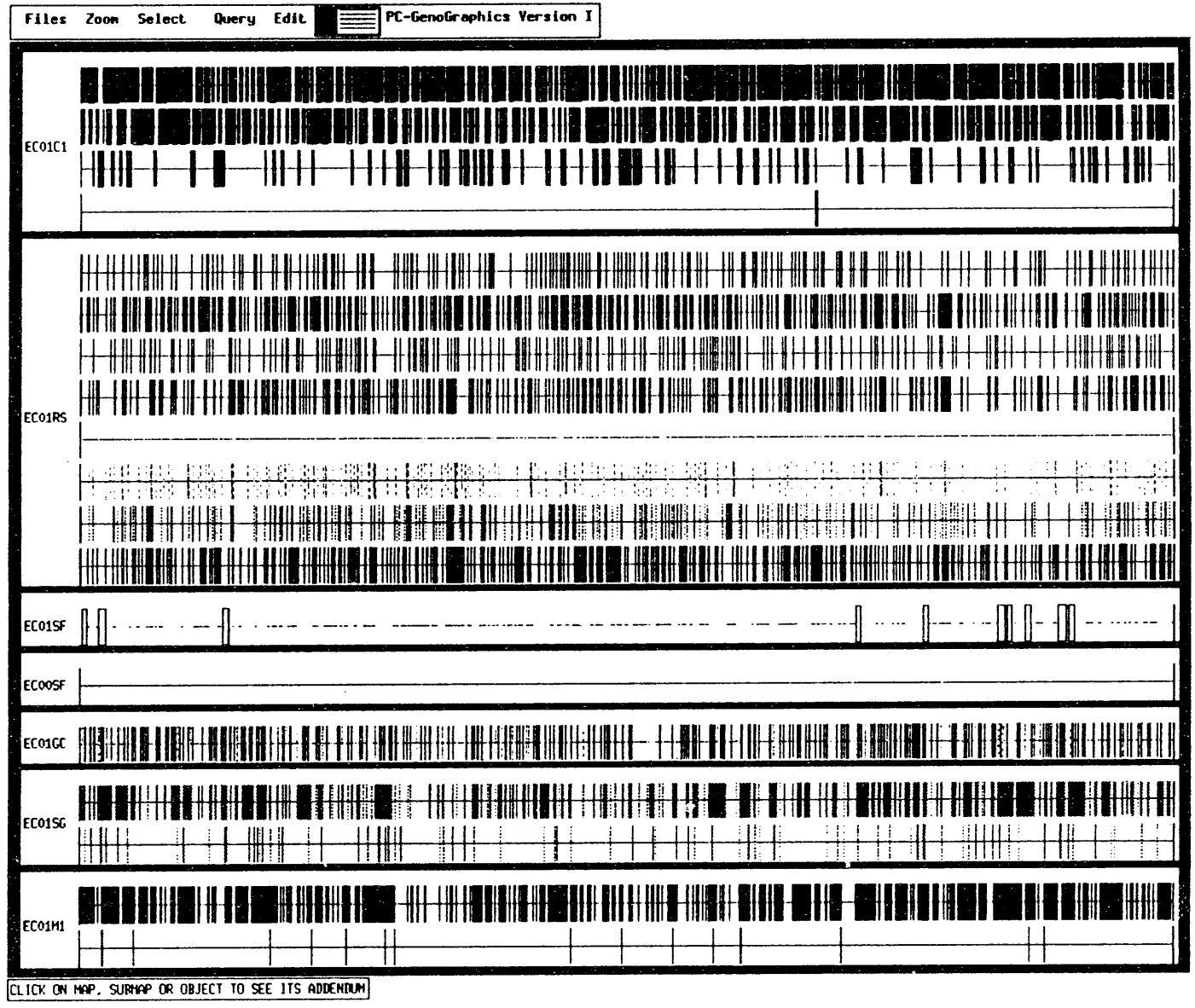
Files Zoom Select Query Edit
PC-GenoGraphics Version I
EC01C1
EC01RS
EC01SF
EC00SF
EC01GC
EC01SG
EC01M1
CLICK ON MAP, SUBMAP OR OBJECT TO SEE ITS ADDENDUM

Figure 2

END

DATE
FILMED
2 / 26 / 93