Volume 12 Number 1 1984 Nucleic Acids Research

Delila system tools

Thomas D.Schneider, Gary D.Stormo, Matthew A.Yarus and Larry Gold

Department of Molecular, Cellular, and Developmental Biology, University of Colorado, Boulder, CO 80309, USA

Received 12 August 1983

ABSTRACT

We introduce three new computer programs and associated tools of the Delila nucleic-acid sequence analysis system. The first program, Module, allows rapid transportation of new sequence analysis tools between scientists using different computers. The second program, DBpull, allows efficient access to the large nucleic-acid sequence databases being collected in the United States and Europe. The third program, Encode, provides a flexible way to process sequence data for analysis by other programs.

1. INTRODUCTION

In a previous paper (1) we described a system of programs useful for the analysis of nucleic-acid sequences. The central program is called Delila (DEoxyribonucleic-acid LIbrary LAnguage). Delila is connected to a collection of sequences called a Library, and can be instructed (in the language Delila) to extract a subset of sequences for analysis by a number of auxiliary programs.

Some of the auxiliary programs have been described previously (2,3). In this paper, we present several new auxiliary programs that are useful tools for transportation of programs to other groups of people (Module), for access to the large international databases (DBpull), and for analysis of sequences (Encode).

2. THE MODULE SYSTEM

A. Program Transportation

The Delila system is designed to be transported. Transportation from one computer to another is frequently a difficult, if not impossible, task. After reading the tape (or other media), the programs must be modified so they function on the new computer. Unfortunately many programs are poorly designed, so hundreds of changes may be necessary. The effort involved also discourages transportation of improved versions.

The author of a program can, however, write the program with these problems in mind. A first step is picking a language that is high level and likely to be portable. One must then stick to the language standards. This means avoiding special features specific to ones own computer. When one cannot avoid using a non-standard feature, one can collect all occurrences of non-standard code into a few procedures or subroutines. This reduces the number of repetitious changes that have to be made by the recipient.

In our attempts to construct the large set of Delila auxiliary programs, we faced a problem related to those of transportation. Many auxiliary programs must be able to read a Library. If there were 50 programs, how could we change the form of the Library without changing each of the 50 programs by hand? As with the transportation problem, we could collect a set of procedures that read the Library's format. We would then distribute these procedures to all of the 50 programs.

One obvious way to distribute procedures cannot be done with the programming language in which the Delila System is written. Pascal has no provision for separate compilation of program parts (4). In other words, we cannot simply collect all the procedures, compile them and then distribute the compiled code to the 50 programs with a linking program.

Many Pascal compilers do provide a mechanism for inserting sections of source code into a program at the time the program is compiled. This is also an unacceptable solution for us because it is not standard Pascal, and would hinder transportation. Instead, an insertion tool can be written (in standard Pascal) to insert the procedures into the source code (5).

Many insertion mechanisms function by searching the source code for a signal such as

```
(*$I"name-of-procedure"*)
```

The signal is then replaced by the appropriate procedure. Although functional, this forces one to keep two copies of the program because the insertion symbol is destroyed upon insertion. Alternatively, one could perform the insertion every time the program is to be compiled, but this would be quite tedious during program development. One could instead modify the program after insertion. Unfortunately, if one of the inserted procedures (in the collection) is changed later, then it is impossible to insert the procedure into the program again using the original mechanism. To avoid these problems, we have designed a new kind of insertion tool.

B. The Module Program

The program Module uses two insertion signals contained inside Pascal comments:

```
(* BEGIN MODULE name.of.procedure *)
                    .
                    .
                    .
        code of the procedure
                    .
                    .
                    .
(* END MODULE name.of.procedure *)
```

The entire procedure and its two signals is called a module. A collection of modules is called a module library (MODLIB). Since each module has two signals, the transfer mechanism need not be an insertion, but rather a replacement of the previous module contents.

For example, suppose a source program contained the following module:

```
(* BEGIN MODULE A.PROCEDURE *)
      THIS WILL BE REPLACED
(* END MODULE A.PROCEDURE *)
```

and the module library contained in it:

```
(* BEGIN MODULE A.PROCEDURE *)
      REAL CODE GOES HERE
(* END MODULE A.PROCEDURE *)
```

then after running the Module program, the source program would contain:

```
(* BEGIN MODULE A.PROCEDURE *)
      REAL CODE GOES HERE
(* END MODULE A. PROCEDURE *)
```

To perform this operation, the Module program copies the source input (SIN) to the source output (SOUT). During the copy, the text is searched for modules. When a module is found in SIN, the contents of the module by the same name in the MODLIB are copied to SOUT, and the contents of the module in the SIN are skipped. This effectively replaces the module in the source program. Further details of the transfer mechanism are in the document used to define the module system, MODDEF (available upon request).

The collection of Library reading procedures is called Delmods. We made Delmods into a program that tests the modules. This has the advantage that when the Delila System is received, one only needs to compile and run Delmods and the other module libraries in order to assure that ALL the auxiliary programs will function. For example, nonstandard access to the computer's clock is buried within a procedure inside Delmods. After transportation, this procedure must be changed to fit the new computer. Once this is done, and the module containing the procedure is transferred, all auxiliary programs that

use the clock will function correctly.

A collection of nonstandard modules for interactive input/output is kept in the program Prgmods. To allow these modules to be used along with those in Delmods, the Module program can successively pick up modules from several module libraries. When a module is not found in a module library, the contents of the SIN module are copied to the SOUT.

The Module program also has a recursive mechanism that allows one to pick up many modules in one package. A module found inside another module in the module library will be inserted into the SOUT. For example, the module library:

```
(* BEGIN MODULE PACKAGE *)
                -
(* BEGIN MODULE PROCEDURE.1 *)
(* END MODULE PROCEDURE.1 *)
                -
(* BEGIN MODULE PROCEDURE.2 *)
(* END MODULE PROCEDURE.2 *)
                -
(* END MODULE PACKAGE *)

(* BEGIN MODULE PROCEDURE.1 *)
      CODE OF PROCEDURE 1
(* END MODULE PROCEDURE.1 *)

(* BEGIN MODULE PROCEDURE.2 *)
      CODE OF PROCEDURE 2
(* END MODULE PROCEDURE.2 *)
```

allows one to pick up either procedure alone, or both together in one package. (The dashes are used in this example only to separate parts of the package module.)

We can define another feature to reduce the amount of source code sent on a tape. If the file MODLIB is empty, then during the copying of SIN to SOUT, the contents of the modules are skipped. This effectively "strips" a program in preparation for transport or storage. It also allows one to look at the structure of the modules in a program.

C. The Break Program

Every page of the manual for the Delila system is a module. This allows one to extract pages of the manual. In addition, the Break program breaks the manual (or a program) into pages, numbers each page, and adds an index to the end. New pages can easily be added to the manual, since pagination is automatic.

D. The Show Program

Every page of the Delila manual has a multi-part name. The parts of the

names are separated by periods, as are file names on many computer systems. The Show program allows one to interactively explore the manual, since the multi-part names make the manual pages have a tree-like structure. In other words, the Show program provides a mechanism for looking at pages of a large document on-line.

One can also explore program source code. Some of our most recent programs (see the DB System, below) are entirely modular to facilitate this.

Like geneticists who exchange organisms by sending them through the mail, we envision widespread exchange of sequence analysis programs. This can be greatly facilitated by the isolation of non-standard procedures into modules.

3. THE DB SYSTEM

There are currently at least two major international collections for nucleic-acid sequence data. One is the Genetic Sequence Data Bank, or simply GenBank (TM), which is based at Bolt Beranek and Newman Inc. in Cambridge, Massachusetts and at the Los Alamos Scientific Laboratory in New Mexico. The other is the European Molecular Biology Laboratory collection (EMBL), which is based in Heidelberg, West Germany. These two collections are similar enough in format that we have been able to write a small set of programs that handles them both.

There has been a virtual explosion in the amount of nucleic-acid sequencing being done around the world. GenBank, for instance, has more than two million base pairs in its entries. Since that is too much information for most computer programs to handle quickly, and because most people are only interested in looking at particular sequences at a given time, some way of easily extracting subsets of the data bases had to be found.

That task is made possible by the structuring of GenBank and EMBL. Both collections are divided into series of entries. Each entry contains a unique name, several lines of relevant information, and a section of nucleic-acid sequence. The division of the data into subsets is further aided by the fact that within each entry, each line or section begins with a code or word indicating what kind of information it contains.

The first of our programs is called DBcat, short for "Data Base cataloguer". It reads through EMBL and GenBank and makes a record of the line number of the beginning of each entry, the file the entry is found in, and the name of the entry. Since DBcat produces a file of these records for output, it is not intended for human use as much as it is to increase the speed of our second program, DBpull.

DBpull takes input in the form of a few lines of simple instructions, and gives back part or all of one or more entries. Below are two examples of input instruction sets.

```
Example 1:  GENBANK          Example 2:  EMBL
            M13 ALL                      EVERY ID DE
            T7 RAW                       GENBANK
                                         *RNA* LOCUS ORIGIN
```

The lines which have "EMBL" or "GENBANK" on them are used to indicate which data base format is requested on the following lines. In example 1, "M13" and "T7" are names of entries. The word "ALL" tells DBpull to extract all of the "M13" entry while the word "RAW" results in only the sequence data of "T7" being pulled. In example 2, "EVERY" is a command that pulls every entry of a data base. "*RNA*" is a wildcard request where "*" represents any number of unspecified characters. This results in any GenBank entry whose name has "RNA" anywhere within it being extracted. The phrases "ID", "DE", "LOCUS", and "ORIGIN" are line codes. These are the codes that begin each entry line. DBpull is instructed in example 2 to pull only the lines of EMBL entries that begin with "ID" or "DE" and only the lines of GenBank that begin with "LOCUS" or "ORIGIN".

Once one has extracted EMBL and GenBank entries one may want to analyze their sequences using a Delila auxiliary program. The program DBbk, short for "Data Base into Delila book", converts the other formats into entries that Delila programs can read.

Like the Delila program (1), DBpull can act on its own output. For example, this allows one to construct a small database of human DNA sequences on which further extractions or analyses can be performed.

These three "DB" programs are a good foundation for any plans one has to use the rapidly expanding international sequence data collections.

4. ENCODING OF SEQUENCES

It is often convenient to replace raw sequence data with a numerical representation of that data. The four numbers that represent the mononucleotide composition of a sequence is an example. Another common example is to represent a coding sequence by the number of times each codon is used. Such an encoding has been shown to be useful in distinguishing the phyla from which the sequence originated (6). We have encoded ribosome binding site sequences into strings of 1's and 0's in order to apply pattern learning techniques to the problem of distinguishing ribosome binding sites from other sites in messenger RNA (3). We are now using encoded sequences and

a. rIIB gene start

```
                -10                0                10
                 *                 *                *
          5'  T A A G G A A A A T T A T G T A C A A T A T T  3'
                                    fMET - TYR - ASN - ILE
```

Perceptron encoding:

```
A   0 1 1 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0
C   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
G   0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
T   1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 1 1
```

b. Encode program parameters for the perceptron encoding above:

```
-11 to +11 region
     1  window size
     1  window shift
     1  coding level
     1  coding shift
```

c. Mononucleotide composition:

```
-11 to +11                    A 11
    23                        C  1
    23                        G  3
     1      --->              T  8
     1
```

d. Mononucleotide composition of codons:

```
0 to 11                  0   3   6   9
   3                 A   1   1   2   1
   3                 C   0   1   0   0
   1      --->       G   1   0   0   0
   1                 T   1   1   1   2
```

e. Dinucleotide composition of the Shine and Dalgarno region:

```
-11 to -6          AA 2
     6             AC 0
     6      --->   AG 1
     2:0           AT 0
     1             CA 0
                   CC 0
                   CG 0
                   CT 0
                   GA 1
                   GC 0
                   GG 1
                   GT 0
                   TA 1
                   TC 0
                   TG 0
                   TT 0
```

f. Dinucleotide composition of the first two bases of codons:

```
0 to 11            AA 1
  12               AC 0
  12      --->     AG 0
    2:0            AT 2
    3              CA 0
                   CC 0
                   CG 0
                   CT 0
                   GA 0
                   GC 0
                   GG 0
                   GT 0
                   TA 1
                   TC 0
                   TG 0
                   TT 0
```

Figure 1. Encoding of the rIIB gene from -11 to +11

multiple regression analysis to find functions which will relate sequence features with their quantitative biological activities.

In each of the above examples, sequence data are replaced by numerical data which can then be processed by a variety of vector analysis techniques. We have designed an algorithm that allows the user to specify a few parameters and encode a sequence in a large variety of ways. Five parameters are essential to maximize the flexibility of the algorithm, and they are each

discussed in detail below. As a sequence example we will use the region of bacteriophage T4 from 11 bases before the rIIB initiation codon through the fourth codon (7). (This is -11 to +11 from the first nucleotide of the initiating ATG, which is numbered 0.) In our "perceptron" paper (3), we encoded this sequence as in Figure 1a. Many more encodings are possible, and by specifying the following parameters the user may choose among a wide variety of alternatives.

A. Region

In the perceptron example, all the nucleotides were encoded identically at all positions. We might, however, want to encode different regions differently. For example, from the ATG rightward we may only be interested in the codons, and not the mononucleotides. In the Shine and Dalgarno region 5' to the ATG (8), we may care about the dinucleotides and we may not care at all about the nucleotides between that region and the ATG. Therefore the algorithm allows the user to specify which region of a sequence is to be encoded by the remaining parameters (sections B to E), and the entire sequence may be encoded as a combination of those independently coded regions.

B. Window Size

Within a region of encoding we can specify the "window" size for the nucleotides being counted. In the perceptron encoding the window was 1 base wide because each nucleotide was assigned to a particular position. But we may not care exactly where a nucleotide or oligonucleotide is, only that it occur in some window of possible positions. In the extreme, where the window is the whole region, we end up just counting the composition with total disregard for position within the region (figure 1c).

C. Window Shift

In the case where many windows exist within the region, we can also specify the movement from one window to the next. For instance, we may not care about the codons *per se*, but do care about the composition within each codon. Then our window would be 3 bases wide and the next window would be 3 bases over (figure 1d).

D. Coding Level

A fundamental parameter to specify is the coding unit; are we encoding as mono-, di-, tri-, or longer oligonucleotides? For oligos longer than monos we can also decide if we want to skip bases between the ones we count. For instance, if we specify a coding parameter of 2:0, then we are counting dinucleotides as they occur in the sequence with zero skips in between (figure

1e). But we might want to know the number of times each dinucleotide occurs, separated by one unspecified base (i.e., AXA, AXC ... TXT). This encoding parameter we would specify by 2:1 (dinucleotides with one base skipped). We might want trinucleotides from the sequence, so we would specify 3:0 0 (trinucleotides with no skips between any of the bases). If we wanted the trinucleotide count for the corresponding positions in three adjacent codons we would specify 3:2 2 (i.e., we are counting the number of AXXAXXA, AXXAXXC ... TXXTXXT). Skips are specified independently between each of the encoded bases (3:1 2 counts AXAXXA, AXAXXC ... TXTXXT).

E. Coding Shift

Lastly, we can also specify the shift to the new coding site. For instance, if we want to keep track of frames we would shift the coding site by three each time (figure 1f).

Figure 1b shows how the perceptron encoding of figure 1a is specified by these parameters. Figure 1c to f shows further example parameters and their corresponding encodings. (In 1d, 0 to 9 are the first bases of each codon.)

We have recently been using encoded sequences to determine whether a region is translated, and if so, in which frame (in preparation). In bacteriophage T4, which is 63% A+T, coding sequences have a very non-random distribution of mononucleotides. For instance, 57% of the G's occur in the first position of codons and 50% of T's occur in the third position. We can take the collection of known T4 coding sequences (there now exist sequences from 21 genes) and make an encoded vector of the mononucleotides at each codon position. This standard can then be used to evaluate, via a simple correlation coefficient, other sequences of unknown function. In conjunction with other methods, this has allowed us to predict several new protein products from the T4 genome.

5. CONCLUSION

In this paper we have described three kinds of tools: those that help transportation of other tools, those that gain one access to sequence data and those that facilitate analysis of the sequences. Since the Delila System contains more than 50 programs at present, we have presented only the key ones. One line descriptions of each program are given in Table I. The Delila Manual, containing page-long descriptions of each program, is available upon request.

Table I. Delila System Programs
(Note: A Book is a subset of a Library)

MODULE LIBRARIES

AuxMods	Auxiliary Program Modules
DelMods	Delila Modules
MatMods	Mathematics Modules
PrgMods	Programming Modules
VaxMods	Modules for Transportation to the VAX Computer

TOOLS

BigLet	Text Enlargement
Break	Breaks Up Files into Pages
Calico	Character And LIne COunts of a File
Chacha	Change Characters
Code	COmment DEnsity of Pascal Programs
Concat	Concatinate Files Together
Copy	Copy Files
Flag	Flags Long Lines in a File
Merge	Merge Raw Sequences, Compare Files
Module	Module Transfers
Rembla	Remove blanks from Ends of Lines in a File
Repro	Reproduce Copies of a File
Shift	Shift Files to the Right
Show	On-Line Access to the Delila Manual and Module Libraries
Split	Splits Files Vertically
Ver	Look at the Version of a Program
Verbop	Increment the Version Number of a Program
Whatch	What Characters are in a File?
Worcha	Change Words in Pascal Programs

LIBRARIAN

Delila	Sequence Manipulation

AUXILIARY PROGRAMS FOR DATA BASE CONSTRUCTION

Reform	Invert, Complement or Reformat a Raw Sequence
Rawbk	Convert Raw Sequences to a Book (for Rapid Analysis)
Makebk	Make a Book from Raw Sequence (for Library Insertion)

AUXILIARY PROGRAMS FOR THE CATALOGUE

Catal	Cataloguer of Delila Libraries
Loocat	Look at a Catalogue (for Debugging)

AUXILIARY PROGRAMS FOR SEQUENCE LISTING

Lister	List Sequence Content of a Book with Translation
Parse	List Parts of a Book

AUXILIARY PROGRAMS FOR ALIGNED SEQUENCES

Alist	Aligned Listing of a Book
Hist	Histogram of Aligned Book
HistAn	Histogram Analysis

AUXILIARY PROGRAMS FOR ANALYSIS

Comp	Composition of a Book
CompAn	Analysis of a Book Composition
Count	Count the Bases in a Book
DotMat	Dot Matrices between Two Books
Helix	List Helicies between Two Books
Index	Alphabetical Listed Oligonucleotides in a Book
IndAna	Analysis of an Index
Matrix	Print Helicies from HELIX in Dot Matrix Form
Pemowe	PEptide MOlecular WEights
Search	Search a Book for Strings

AUXILIARY PROGRAMS FOR PATTERN LEARNING

PatLrn	Pattern Learning (The Perceptron)
PatLst	Pattern Lister - Prints out the Pattern
PatAna	Analysis of Each Column in a Perceptron Pattern
PatSer	Pattern Search in a Book
PatVal	Evaluates Sequences in an Aligned Book Using a Pattern

AUXILIARY PROGRAMS FOR ENCODED SEQUENCES

Encode	Encode a Book of Sequences into Strings of Integers
EncFrq	Encoded Sequence Frequency Analysis
EncSum	Sum of the Vectors of Encoded Sequences
Frame	Correlations between Coding Frames

AUXILIARY PROGRAMS FOR OTHER USES

Refer	Lists References in a Library
Sepa	Separates Delila Instruction Sets

ACCESS TO INTERNATIONAL DATA BASES

DBpull	Extracts Data Base Entries
DBcat	Generates Catalogues for DBpull
DBbk	Converts Entries to Delila Format
DBlook	Allows one to Look at DBcat Catalogues (for Debugging)

ACKNOWLEDGEMENTS

This work is supported by grant number GM28755 from NIH. We are grateful for the continuing support of the University of Colorado Academic Computing Services. We also thank Patrick Roche for writing Break; Billie Lemmon for suggesting Show; Kathie Piekarski for typing the manuscript; and John Hoffhines, Melissa Mockensturm and John Childs for reading the manuscript.

REFERENCES

1. Schneider, T.D., Stormo, G.D., Haemer, J.S. and Gold, L. (1982). Nucl. Acids Res. 10, 3013-3024.
2. Stormo, G.D., Schneider, T.D. and Gold, L.M. (1982). Nucl. Acid Res. 10: 2971-2996.
3. Stormo, G.D., Schneider, T.D., Gold, L. and Ehrenfeucht, A. (1982). Nucl. Acid Res. 10: 2997-3011.
4. Jensen, K. and Wirth, N. (1974). Pascal User Manual and Report, Second Edition, Springer-Verlag, New York.
5. Kernighan, B.W. and Plauger, P.J. (1981). Software Tools in Pascal, Addison-Wesley, Reading, Mass.
6. Grantham, R., Gautier, C., Gouy, M., Jacobzone, M. and Mercier, R. (1981). Nucl. Acid Res. 9, r43-r74.
7. Pribnow, D., Sigurdson, D.C., Gold, L., Singer, B.S., Napoli, C., Brosius, J., Dull, T.J. and Noller, H.F. (1981). J. Mol. Biol. 149, 337-376.
8. Shine, J. and Dalgarno, L. (1974). Proc. Natl. Acad. Sci. USA 71, 1342-1346.