Volume 10 Number 9 1982 Nucleic Acids Research

A design for computer nucleic-acid-sequence storage, retrieval, and manipulation

Thomas D.Schneider, Gary D.Stormo, Jeffrey S.Haemer and Larry Gold

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

Received 26 October 1981; Revised and Accepted 5 April 1982

ABSTRACT
We have designed and built a data-base system for the storage of nucleic-acid sequences. The system consists of a data base ("the library") and software that manages and provides access to that data base ("the Librarian").

1. INTRODUCTION

The recent proliferation of nucleic-acid-sequence data has provoked widespread interest in computer programs as tools for ferreting out patterns, palindromes, and other regularities in sequences that might not be easily perceived by eye or hand[1,2]. Currently, there is a growing awareness that the computer has another role in sequence studies: it provides a cheap, convenient way to store and retrieve sequence data[3]. Several computerized sequence libraries are now being built and plans have been made both in the U.S. and abroad to fund national and international centralized nucleic-acid-sequence libraries[4].

Such centralized facilities will free individual scientists from countless hours of entering, verifying, and rearranging sequences along with the other curatorial chores that would be necessary if every public and private institution with an interest in nucleic-acid sequences were to try to maintain an independent, up-to-date, comprehensive sequence library.

One point is often overlooked in discussions about such centralized facilities: A properly designed, central library does not replace the decentralized, locally controlled library. An analogy to a conventional library illustrates this point. If the Library of Congress did not exist, we would surely all agree that it should be built. Few of us, however, would sell our books, empty our reprint files or even sacrifice our departmental libraries merely because more comprehensive local and national libraries <u>do</u> exist. In an even closer analogy, the computerized bibliographic search facilities provided by many campus and public libraries are a valuable information-retrieval aid, but haven't persuaded many of us to throw away our filing cabinets or forget all the citations we carry around in our heads.

Here, we report our efforts to design and build software to fill the need for locally-controlled, small-scale sequence libraries. Our system has several important properties:

(1) The library has a structure that allows sequences to be stored in a way that parallels biological relationships.

(2) Each sequence is stored with its published coordinate system. Requests for sequence fragments may be made based on these coordinates.

(3) Sequence requests are made in a language (Delila: DEoxyribonucleic acid LIbrary LAnguage) that allows flexible manipulation and retrieval of arbitrary nucleic-acid fragments.

(4) Any set of sequences retrieved, including a set that encompasses the entire library, has the format of the entire library.

(5) The system includes modules that provide an interface between the librarian and other programs. These modules make auxiliary programs that interact with the system easy to build, maintain, and transport with the library.

(6) The entire Delila system is written in Pascal, a high-level language available on many computers.

TERMINOLOGY

The analogy between our data base and a library is one we use frequently. Accordingly, in this paper we will use the terms "library" and "nucleic-acid-sequence data base" interchangeably. By extension, we will use the term "librarian" to refer to the program that accepts requests from the user, then searches the library and returns the desired information. We will also occasionally use the term "library system" to refer to the library together with its librarian and assorted auxiliary software.

Where not otherwise specified, terminology follows that of Martin[5].

2. GENERAL DESIGN

The easiest way to describe a data-base system is to list the choices made in its design. Below, we enumerate and discuss a few of the choices that characterize ours.

2.1. DATA BASE CONTENT

Our data base is designed to hold wild-type sequences, together with the numbering schemes published for each sequence. Mutant sequences are stored as changes to the corresponding wild-type sequence. The data base also records the taxonomic, genetic, and (where known) physical relationships among the stored sequences. Sequences and sequence fragments can be retrieved by name, by position within the sequence, or by reference to biological features within the sequence (see section 3.1). Provision is made to store and also retrieve sites recognized by enzymes or chemical probes.

2.2. LOGICAL STRUCTURE

We have chosen a structured data-base design. This choice reflects our unwillingness to try to anticipate the keys on which we might want to search the data base (other than those listed in the previous section). It also allows biological information to be stored in the logical structure of the data base itself. Finally, the choice is consonant with our desire to relegate specialized search tasks to specialized programs (see below). Figure 1 is a schema that shows the logical structure of our library[5].

2.3. PHYSICAL STRUCTURE

We require that the output of successful library searches have a single, prescribed physical structure. This structure may be transparent (invisible) to the user, but is required to be formally identical to the physical structure of the library itself. There are two important consequences of this choice.

(1) Interfaces between the library and other sequence-manipulation tools can be built for a single data structure.

(2) All tools that work on output from the librarian can be applied to the entire library. As a special case of this, the librarian itself will work on the subsets it extracts.

2.4. INTERACTIONS OF THE LIBRARIAN WITH OTHER PROGRAMS

We deliberately designed the librarian as a tool with limited capabilities: it takes one or more libraries and a set of instructions as input; it produces a new library that contains all or part of the input. We have restricted ourselves to the job of making it do this task well. All other tasks (fancy formatting, sophisticated searches, etc.) have been left to other, independently-constructed tools. The discussion in the literature on the advantages of limited tools is extensive and need not be repeated here[6].

The librarian communicates with other tools through two sets of

```
                          Schema

  Below is an overview of the structure of the library.
       a-->>--b  means a has one or more of b.
       c--->--d  means c has exactly one of d.

                           library
                             :   :
                             v   v
                             v   v
                             :   :
                 ............:   :.............
                 :                            :
            organism                 recognition-class
                 :                            :
                 v                            v
                 v                            v
                 :                            :
            chromosome                        :
              : : : :                         :
              v v v v                         :
              v v v v                         :
              : : : :                         :
  ............: : : :.........                :
  :       ......: :....      :                :
  :       :           :      :                :
 marker  transcript   gene   piece....       enzyme
  : :     :           :     : : :   :         :
  v v     v           v     : : :   v         v
  : :     :           :     : : :   :         v
  : :     :           :.....: : :   :         :
  : :     :...................: :   :         :
  : :...........................:   :         :
  :                                 :         :
 sequence                       sequence  sequence
```

Figure 1: Each node of a library contains several attributes. Library nodes hold the library title, the date and time of library creation, and the date and time of the parent library. Organism nodes hold the name and map units of the organism they represent. Chromosome nodes hold the range of the chromosome's genetic map. Piece nodes contain both a sequence and the appropriate numbering scheme for that sequence. (As adjacent sequences become available, they are joined to already stored sequences until a chromosome becomes represented as a single piece.) Marker, gene and transcript nodes point to specific parts of pieces. A marker records changes (insertions, substitutions, and deletions) in its piece. An enzyme node holds a sequence that is recognized by a biological or chemical process.

routines. One set reads library-structured data and interprets it as needed by any particular program. The second set allows programs to write instructions that can be used as requests to the librarian. These two sets are separable from the librarian itself.

2.5. IMPLEMENTATION LANGUAGE

Our system is written in a portable, high-level language.

Portable code in computer science serves the same purpose as published methods and standard materials serve in other areas of science. It permits others to use, improve on, and ultimately supplant each published piece of work. We assume that our data-base system will be far from the last word on the subject. So, we have set out to make it as portable, modifiable, and intelligible as possible. This requires using a widely available, easily readable, higher-level language[7].

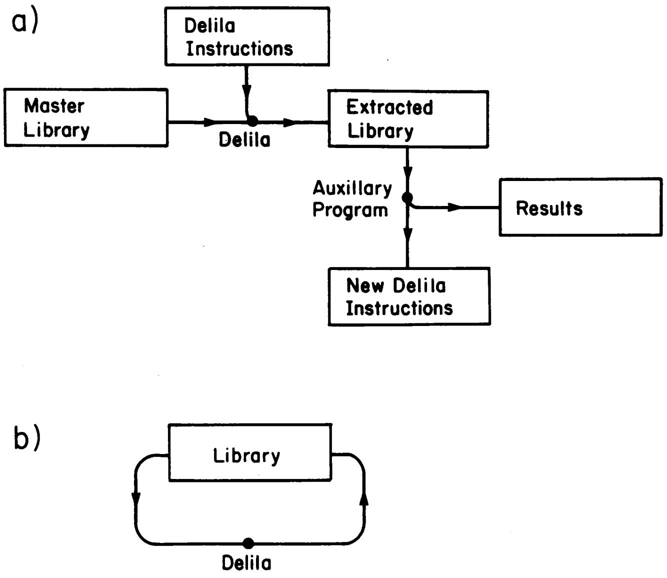

Figure 2: Parts of the Delila system can be joined like tinkertoys. In this figure bubbles represent data, arrows show directions of data flow, and dots represent the programs that control the flow. A typical data flow construction is shown in figure 2a. A user chooses a Library for study (in this case, a Master Library), creates a set of Delila Instructions (see figure 3a), and gives these to the librarian. The librarian performs the desired extractions (see figure 3b). The Extracted Library can be passed through an Auxiliary Program to obtain further Results (see figure 3c). The fundamental distinction between auxiliary programs and the librarian is that their output need not be a library. Auxiliary programs can communicate with the librarian by producing Delila instructions. For example, a search program may direct Delila to extract regions near interesting sites.

The figure has been simplified in several ways. First, some auxiliary programs use input other than libraries (such as previous results) and may produce other kinds of output (such as libraries). Second, the two library bubbles could be united (as shown in figure 2b) to indicate that Delila's input has the same structure as its output. The instructions can also be united, as can the inputs and results of auxiliary programs. Finally, auxiliary programs can be thought of as users of the system, and vice-versa.

```
(a)
title 'example';

(* specification *)
organism t4;
   chromosome t4;
      piece rii;

(* request *)
get from 12 to 41;
```

```
(b)
* 81/06/18 11:05:33, 81/06/18 11:04:22, example
organism
* t4
* bacteriophage t4
note
* # 1
* host: ecoli
* bact. rev. 40(4): 847-868 dec 1976
* the genome of bacteriophage t4
* wood and revel
note
* kilobase pairs (kb)
chromosome
* t4
* t4 chromosome (dna)
note
* # 2
* chromosome is circular
note
*  0.00
*  166.00
piece
* rii
* riia end to riib begin
note
* # 3
* from david pribnow (base 90 corrected to a g)
note
*  0.00
* linear
* -
* 1
* 873
* linear
* +
* 12
* 41
dna
* gatttatatgtagatgcttttgatgatgta
dna
piece
chromosome
organism
```

```
(c)
book lister 3.05.  a "/" means end of coordinates.
* 81/06/18 11:05:33, 81/06/18 11:04:22, example

organism t4; chromosome t4;
piece rii  # 3  config: linear  direction: +  begin: 12,  end: 41

     *       *20       *        *30       *        *40
g a t t t a t a t g t a g a t g c t t t t g a t g a t g t a
 asp - leu - tyr - val - asp - ala - phe - asp - asp - val -
   ile - tyr - met -amber-fmet - leu - leu - met - met -
     phe - ile - cys - arg - cys - phe -opal -opal -     -


index to the locations of pieces in this book

organism t4; chromosome t4;
page    1, piece rii                # 3
```

Figure 3: An example of the use of the library system. (a) Delila instructions for getting bases 12 through 41 of the rII sequence of Pribnow et al[16]. Instructions are free-format, and indentations and blank lines are used to enhance legibility. The symbols "(*" and "*)" delimit comments. (b) Output from the instructions shown in part a. This illustrates the current

physical structure of the library. An asterisk ("*") in the left-most column signals that the line holds an attribute. Other characters ("o", "c", etc.) delimit structures seen in the schema in figure 1. A formal definition of the library structure is available on request. (c) The result of running the output seen in part b through an auxiliary program -- lister -- that produces a numbered listing.

2.6. BATCH USE

The librarian is batch-oriented, not interactive. This choice stemmed primarily from our desire to make large requests to the librarian[8,9]. With a batch design, large requests can be prepared or modified with a text-editor and submitted. We believe it would be a mistake to try to match the ease of entry, manipulation, and modification provided by good text-editors in a program whose primary purpose is data-base management. Another influence is the choice of languages available to us. In most widely available programming languages, interactive programming is implementation-dependent (the conspicuous exceptions being BASIC and APL). Consequently, although we typically run our programs in interactive environments, the source code itself is designed as non-interactive code.

3. STATUS

Having outlined the major decisions that characterize our library system, we can talk in more detail about what we have available.

3.1. A PORTABLE IMPLEMENTATION

We have constructed a prokaryotic-sequence library together with a librarian that extracts data from this library. The design of the library system follows the principles set forth in the previous section. The library currently holds about 100,000 bases of sequence data. About 150,000 characters are required to store the sequences and associated information. The librarian accepts requests in a language, Delila (DEoxyribonucleic acid LIbrary LAnguage). In effect, our librarian is a special-purpose compiler that accepts programs in Delila, parses them, produces a listing, and performs the requested extraction. The interaction of the library, librarian, and auxiliary programs is diagramed in figure 2. Figures 3 and 4 show example requests.

The entire data-base system is written in Pascal, a well-structured language that provides a wide range of data types as an integral part of the language[10]. Pascal compilers are available under many operating systems for many large and small computers. Our system is almost dialect-independent, and we have run major pieces of it on the CDC 6400, Cyber 720, and VAX 11/780,

```
title 'further example delila commands';

organism ms2;
   chromosome ms2;
      piece ms2;
get all piece;

organism g4;
   chromosome g4;
      transcript iii;
get all transcript;

organism ecoli;
   chromosome ecoli;
      gene laci;
get from gene beginning - 5
    to gene beginning + 20;

   chromosome tn9;
    (* notice that the organism need not be re-specified *)
      piece tn9;

get from piece ending
    to piece beginning;
    (* by design, this retrieves the complement: see figure legend *)

default out-of-range reduce-range;

   chromosome pbr322;
      piece pbr322;
get from coordinate beginning - 10
    to coordinate beginning + 10
    direction +;
(* bases 4352, 4353, ..., 4362, 1, ..., 11 *)

get from coordinate beginning + 10
    to coordinate beginning - 10
    direction -;
(* the complement of bases 11, 10, ..., 1, 4362, 4361, ... 4352 *)

get from coordinate beginning - 10
    to coordinate beginning + 10
    direction -;
(* the complement of bases 4352, 4351, ..., 11 *)

get from coordinate beginning + 10
    to coordinate beginning - 10
    direction +;
(* bases 11, 12, ..., 4352 *)
```

Figure 4: This figure illustrates further types of commands available in Delila. Of particular note are the following points: (a) All sequences are provided 5' to 3'. This means, for example, that a request for bases 20 to 10 of a sequence will produce the complement of bases 10 to 20. (b) Several switches that control the performance of the librarian may be set by the user by the 'default' command. In this example, the user explicitly asks that the librarian respond to requests for more of a sequence than is contained in the library being searched by providing as much of the sequence as is available. (c) Requests for segments from circular sequences must be more specific than requests for segments from linear sequences. The bottom four requests extract the four different segments of the sequence of pBR322 defined by the bases 4352 and 11.

under KRONOS, VMS, and UNIX operating systems. The library system has also been transported to a SINTRAN III operating system on a NORD-10S at the European Molecular Biology Laboratories, Heidelberg (G. Hamm, personal communication). Example costs of the current implementation are described in Appendix I.

3.2. INTERFACE PACKAGES

Like most groups interested in sequence data, we have a wide range of programs that analyze and manipulate sequence data. (For a detailed discussion of several such tools see [8,9]). To use our library effectively, we have built plug-in modules that let such "auxiliary programs" read output from, and write instructions to the librarian.

3.3. OTHER TOOLS

For our convenience, we have constructed a few other tools that do not fit easily into the categories described above. These include programs that help us enter sequence data into the library, and a program that constructs catalogs of existing libraries to aid the librarian in its tasks.

3.4. AVAILABILITY

We will supply any of the software, data, and documentation described above as machine-readable source code on request. We will also provide the document LIBDEF, which includes the formal specification of the library structure and defines the syntax of Delila in Backus-Naur Form [15].

4. RETROSPECTIVE

Foresight helps software projects succeed. Hindsight helps replace them. In this section, we summarize some of the lessons that we learned about design and implementation by using our system after it was built.

4.1. BATCH DESIGN

Because the librarian was built as a special-purpose compiler, lengthy library requests can be stored as programs, and later modified and rerun without much additional effort. In practice, such re-use is common. We now feel that the ability to accept batch input is an essential part of the design of any future librarian. By analogy to other compilers, it may prove useful to have future versions of the librarian produce intermediate "object code" that would let frequently used requests be rerun without having to be reparsed.

If it becomes important to create an interactive version of the librarian then creating a set of routines that manipulate Delila instructions will become a high-priority item. Such routines would provide a simple editor to read, modify, and write out instruction sets interactively. A Delila-instruction editor independent of the librarian would provide a first step toward this goal and be useful in its own right.

4.2. PORTABILITY

The rate of hardware and operating-system improvement nearly matches that of new sequence acquisition[11,12]. Thus, the typical library must prepare to face either frequent transportation or rapid obsolescence. Achieving portability not only allows others to use and improve your code, but also provides substantial immunity to hardware and operating-system changes[7].

Midway through our project, we were given the opportunity to use a substantially better computing environment than the one in which we started. Because we had tried to write system-independent code, we were able to make the transition easily. Worth noting, however, is the fact that we were fooled more than once into learning coding practices that we thought were standard but were not. In several cases, even tools designed to pick out non-portable constructions were unable to catch our mistakes. Only real transportation points out real transportation problems.

4.3. INTERFACE PACKAGES

We wanted to create an all-purpose module that would let any program read output produced by the librarian. In our zeal, we made the module so large and all-purpose that its inclusion is often the single biggest cost in program development. (Paradoxically, the problem is most severe for large programs. Such programs are more likely to have many run-time errors, which require frequent re-compilation to eliminate.)

One solution is to switch to an implementation language that allows separate compilation of subroutines. Unfortunately, most widely available languages that allow this have other important drawbacks, so that a language switch at this stage would change our problems but not eliminate them. In the next decade, however, at least two languages (Ada and C) promise to become widely available that provide data structuring, modern control-flow, and separate compilation[13,14]. It should prove useful to write a version of the data-base system in one of those languages at some point.

A more immediate solution is to shrink the interface module. Although this could be done by limiting its abilities, we are redesigning it so that the user can include only the parts his or her program needs.

5. ACKNOWLEDGEMENTS

We thank Mike Aden and Paul Morrisett for help with the code; Greg Hamm for bringing up the system overseas; Andrzej Ehrenfeucht, Evi Nemeth, and Paul Zeiger for helpful suggestions; Lloyd Fosdick and Lewis Sawin for access to computers; and all those who helped us type in sequences. Funds for this project were provided through grants NIH 1 R01 GM28755, NIH 5 R01 GM19963, ACS NP-178D to LG, DRG 195-FT to JSH, and NIH 5 T32 GM07385 to TDS.

REFERENCES

1. Queen, C.L. and Korn L.J. (1980) in Methods in Enzymology, K. Moldave (ed.) pp. 595-609, Academic Press, New York
2. Gingeras, T.R. and Roberts R.J. (1980) Science 209, 1322-1328.
3. Anon., "Banking DNA Sequences," (1980) Nature 285, p. 59
4. Jordan, E. (1980) Science 210, p. 1074
5. Martin J. (1977) Computer Data-Base Organization Prentice-Hall, Englewood Cliffs, New Jersey
6. Kernighan, B.W. and Plauger P. (1976) Software Tools Addison-Wesley, Reading, Mass.
7. Brown, P.J. (1979) Software Portablility Cambridge University Press, Cambridge
8. Stormo, G.D., Schneider, T.D., and Gold, L. (1982) Nucleic Acids Research, this issue.
9. Stormo, G.D., Schneider, T.D., Gold, L. and Ehrenfeucht, A. (1982) Nucleic Acids Research, this issue.
10. Jensen, K., and Wirth, N. (1974) Pascal User Manual and Report, Second Edition Springer-Verlag, New York
11. Abelson, P.H. (1982) Science 215, 751.
12. Dayhoff, M., Schwartz, R.M., Chen, H.R., Hunt, L.T., Barker, W.C. and Orcutt, B.C. (1980) Science 209, 1182
13. Kernighan, B.W. and Ritchie, D.M. (1978) The C Programming Language Prentice-Hall, Englewood Cliffs, New Jersey
14. Ledgard, H. (1981) Ada: An Introduction / Ada Reference Manual (July 1980) Springer-Verlag, New York
15. Naur, P. (ed.), (1963) Comm. ACM, 6 (1) p. 1
16. 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 (3) pp. 337-376

APPENDIX I: EXAMPLE COSTS OF THE CURRENT IMPLEMENTATION

Our library (dated 81/01/18 22:29:26) contains 95975 bases of prokaryotic sequences stored in 149104 characters. The current library is implemented in two files. (This is normally invisible to the user.) The librarian (version 1.20) contains 147656 characters, 4549 lines of code, and roughly 2000 Pascal statements. About thirty percent of the characters are in comments.

To test how access time depends on library size, ten pieces were chosen, scattered across the second library file. The first ten bases of each piece were retrieved. On a Cyber 720, the access time for the ten independent requests was:

cp sec = 0.53 sec + (0.50 sec/2237 lines) x
(line number of sequence within the file).

The fit to a straight line was good (r = 0.992) with 0.53 seconds devoted to overhead generated by the system and the librarian. There is no qualitative change if depth is measured in characters instead of lines, since the two measures are highly correlated. Removal of the lower half of the library did not affect access time to the remaining half. Access time is only a function of the depth of the sequence in a library file, not of the size of the file itself.

As we expected from the design of this librarian, when all ten requests were made in one instruction set, the order was found to be important. When _pieces_ are requested in the same order that they are stored in the library, they are retrieved faster than if they are requested in reverse order.

The time to pull out 10-base fragments from PHIX174 was linearly dependent on the depth within that _piece_ (n = 14, r = 0.996):

cp sec = 0.51 sec + 0.00011(sec/base) x (# bases deep in _piece_)

The length of fragments obtained also affects retrieval time linearly (n = 11, r=0.9999):

cp sec = 0.62 sec + 0.00052(sec/base) x (length obtained, bases).

The overhead times in these last two cases differ because different instructions were used.