Volume 14 Number 1 1986 Nucleic Acids Research

# Data structures for DNA sequence manipulation

Charles B.Lawrence

Department of Cell Biology, Baylor College of Medicine, One Baylor Plaza, Houston, TX 77030, USA

Received 6 June 1985

**ABSTRACT**

Two data structures designated Fragment and Construct are described. The Fragment data structure defines a continuous nucleic acid sequence from a unique genetic origin. The Construct defines a continuous sequence composed of sequences from multiple genetic origins. These data structures are manipulated by a set of software tools to simulate the construction of mosaic recombinant DNA molecules. They are also used as an interface between sequence data banks and analytical programs.

## INTRODUCTION

The recent availability of up-to-date compilations of nucleic acid sequences such as the GenBank (7) nucleic acid sequence data bank is of substantial benefit to the molecular biologist wishing to compare a gene sequence of interest with other known sequences and to the genetic engineer wishing to create chimeric recombinant DNA molecules in the laboratory having predictable properties. For the comparison or analyis of gene sequences, efficient methods are required to extract sequences of interest from the sequence data banks for use by analytical programs. For genetic engineering purposes, it is necessary to be able to locate a region of a sequence which will be used in a genetic construction (a restriction endonuclease fragment, for example) and join that region to other sequences used in the construction. At the same time it is useful to be able to keep track of the various genetic elements used in a complex genetic construction.

There have been several approaches taken to the problem of manipulating gene sequences in computer programs. One solution is the definition of a specific language which is used to specify the portion of a gene sequence to

* The software tools described in this paper will be available as part of a general purpose sequence analysis package as of July, 1986. The package will cost $600 (non-profit) or $1800 (commercial). Contact Charles Lawrence, Dept. of Cell Biology, Baylor College of Medicine, One Baylor Plaza, Houston, TX, 77030, (713) 799-6226.

be used or analyzed. Genglish (1) is an English-like language designed for use by genetic enginering researchers and serves as a user-interface to a knowledge base for molecular genetics. Delila (2) is a language designed for the manipulation of sequences in a specifically formatted data base called a library. In this case, the structure of the data base itself is the underlying data structure for the representation of sequences. DNA* (3) is another example of a language designed specifically for the manipulation of gene sequences. Another approach is the development of specialized programs for the simulation of the construction of recombinant DNA molecules. Programs such as KLONER (4), MATILDA (5) and a system based on SAS (statistical Analysis System) (6) can be used to simulate genetic construction by manipulating representations of restriction endonuclease fragments. However, these programs do not have a direct method of converting the recombinant constructions to the actual nucleotide sequence they represent.

This paper describes data structures which can serve as the basis for a flexible system to interface sequence data banks with analytical programs and for the efficient simulation of recombinant DNA constructions. The data structures are a representation of molecules of DNA as viewed by the molecular biologist and can be thought of as objects which can be manipulated in software in a manner analagous to operations performed on DNA molecules in the laboratory. These have been implemented with a series of software tools for sequence manipulation, but could be used as the underlying data structures for an interpreted language such as those described above.

## The Use of an Abstract Data Structure to Represent Nucleic Acid Sequences

To develop an efficient method to manipulate nucleic acid sequences, it was necessary to define an abstract data structure which represents the properties of a gene sequence which are of importance to the molecular biologist or genetic engineer. For example, information about the structure of the ends of a molecule of DNA is important in determining its properties in a ligation reaction. Appropriately organized information about a DNA molecule will allow programs to perform operations on the data structures which mimic procedures used by the scientist in the laboratory. Two data structures were defined to accomplish this. The **Fragment** data structure is used to precisely define a continuous nucleic acid sequence from a unique genetic origin. It can represent any subsection of a specific sequence in any of its possible orientations and retains the information which relates the sequence to its exact genetic origin through subsequent manipulations. A second data structure called

**Construct** consists of one or more Fragments linked together to represent a continuous nucleic acid sequence composed of sub-sequences of different genetic origins. The value of the Construct data structure is the ability to manipulate a complex mosaic sequence as a single molecule, yet retain the information which precisely defines the genetic origin of all of its sub-sequences (which are defined by the component Fragment data structures).

The Fragment data structure is defined below as a structure in the 'C' programming language:

```
typedef struct {
     char frg_name[16];        /* name of Fragment */
     char const_name[16];      /* name of construct containing Fragment */
     char *seq;                /* points to string representing Fragment
                                  sequence */
     Specification specs;      /* information relating Fragment to its
                                  parent sequence */
     Frag_end l_end;           /* describes left end of Fragment */
     Frag_end r_end;           /* describes right end of Fragment */
     Bank_specs bspecs;        /* describes data bank from which Fragment
                                  sequence originated */
     Fragment *next;           /* used to form linked list of Fragments */
     } Fragment;
```

The structure contains members which specify a name for the Fragment, a name for the Construct of which it is a component (if any), a pointer to a string which represents the actual sequence, a pointer to another Fragment data structure which is used to form a linked list, and four other structures. These other structures contain information which specify the relationship of the Fragment sequence to its parent (Specification), the nature of the ends of the Fragment (Frag_end) and information about the data bank (if any) from which the Fragment sequence was obtained (Bank_specs).

The **Specification** structure relates the Fragment sequence to its parent sequence and is defined below:

```
typedef struct {
     long start;               /* position of first base in Fragment
                                  relative to parent sequence */
     long stop;                /* position of last base in Fragment
                                  relative to parent sequence */
     char dir;                 /* (0 or 1) if true, Fragment is in same
                                  orientation as parent sequence */
     char comp;                /* (0 or 1) if true, Fragment sequence is
                                  the complement of the parent sequence */
     long f_len;               /* number of bases in Fragment sequence */
     long p_len;               /* number of bases in parent sequence */
     char type[4];             /* "DNA" or "RNA" */
     } Specification;
```

Fragment sequences are often sub-sequences of a longer parent sequence (a sequence from the GenBank database, for example) or a transformation of a

parent sequence, such as its reverse-complement. The Specification structure members contain information that precisely defines the relationship of the sequence represented by the Fragment to its parent sequence. This information includes the position of the first and last bases in the Fragment relative to the parent (**start**, **stop**), its orientation (**dir**), and whether it is the sequence complement of the parent (**comp**).

The **Frag_end** structure describes one end (left or right) of a Fragment as seen by the genetic engineer:

```
typedef struct {
     int oh_len;              /* length and type of overhanging end (if
                                 any) positive = 5'-overhang; negative =
                                 3'-overhang */
     char oh_seq[8];          /* sequence of overhanging nucleotides */
     char enz[16];            /* name of endonuclease generating end */
     char lig;                /* (0 or 1) if true, Fragment end is ligated
                                 to next Fragment in linked list */
     } Frag_end;
```

The members of this structure specify the type and length of the overhanging end (**oh_len**), the actual sequence in the overhang (**oh_seq**), the enzyme that was used to generate the end (**enz**) and whether the end has been ligated to an adjacent Fragment to form a Construct (**lig**).

The **Bank_specs** structure members describe the sequence data bank (if any) from which the parent sequence was derived:

```
typedef struct {
     char entry[16];          /* code name of data bank entry for parent
                                 sequence */
     char acc[8];             /* accession # of bank entry */
     char b_name[12];         /* name of bank containing parent sequence
                                 (mammalian, viral, etc.) */
     int release;             /* release number of bank */
     int format;              /* indicates format of file containing par-
                                 ent sequence (i.e. 1 is GenBank format */
     } Bank_specs;
```

This structure includes the entry name of the sequence (**all**), its accession number (**acc**), the name of the bank (**b_name**; mammal, viral, etc.), the release number of the bank (**release**) and a code indicating the type of format of the sequence data bank file (**format**; for example, the integer value 1 indicates that the file is in GenBank format). A value of zero for the format member indicates that the sequence represented by the Fragment did not originate from a data bank file but an unformatted text file.

Fragment structures are chained together in a linked-list for use in programs by using the **next** member of the structure to point to next Fragment in the list. The last structure points to the first structure in the list to

make a circular list. Programs operate on a group of Fragments defined in the linked list.

The **seq** member of the Fragment structure points to a string containing the sequence represented by the Fragment. If the Fragment represents double-stranded DNA with overhanging ends, all nucleotides including those in the overhangs are included in the sequence. Memory to hold the sequence is always allocated dynamically using information in the Sepcification member to calculate the amount of space needed to hold the sequence. This allows the Fragment to specify a sequence of any length without wasting memory.

A **Construct** is defined as one or more adjacent Fragments in a linked-list whose ends have been "ligated" by setting the **lig** member of the associated Frag_end structure to true. Fragment ends must be compatible for ligation in the biochemical sense in order to be ligated to form a Construct. Constructs allow defined genetic sequences from diverse origins to be joined and treated as a single molecule of DNA while retaining all the information pertaining to the origin of its components. Thus, Constructs can be used to represent complex recombinant DNA molecules, or the sequence of a mRNA molecule for a gene having several exons. In the latter case, individual Fragments are used to represent each exon from the complete sequence of the natural gene. A single linked-list can represent either a group of Constructs which specify a collection of sequences such as a group of actin mRNAs from different species or possibly a single recombinant molecule which can either be a linear or circular molecule (by ligating the left end of the first Fragment and right end of the last Fragment in the linked-list). Fragments can also be used to represent linkers, adapters and oligonucleotides which are used in genetic constructions. Thus, complex constructions can be precisely specified.

**Creation and Storage of Fragment Definitions**

Fragments can be created by two different methods. The simplest method is to "excise" a Fragment from a sequence data bank. In our implementation, an interactive program called EXC (excise) is used to create Fragment definitions from sequences in the GenBank sequence data bank. The user specifies the sequence entry name, the position of the first base in the Fragment relative to the first base in the data bank sequence, the position of the last base in the Fragment relative to the last base in the data bank sequence, and whether or not the reverse-complement of the data bank sequence is desired. EXC then locates the desired sequence in the sequence data bank, allocates space for the Fragment structure, fills in the specifications which

define the Fragment, writes the Fragment definition out to a file (see below) and prompts for the excision of another Fragment. It is important to note that the Fragment structure is not dependent at all on the format of information in a sequence data bank file and can be used to represent in a uniform manner sequences stored in databanks with different internal formats.

A second method to create Fragments is used for the conversion of sequences that are not included in a formatted data bank. A "raw" sequence can be entered into a file with a text editor in free format. A utility named RAW2FRG is then used to convert the raw sequence into a Fragment definition and to save it in a file.

Fragment definitions are saved as formatted text in a file that has an **frg** extension. A group of any number of Fragments can be saved in a single file. A library function, putfrgs(), is used to write all the Fragment definitions in a linked-list of Fragments to a frg file. The complementary function, loadfrgs(), loads Fragment definitions stored in a frg file into a circular linked-list of Fragments. Space is allocated dynamically for the necessary Fragment structures as they are loaded, so that the number of Fragments that can be used by a program is limited only by the memory available to that process.

**Manipulation of Constructs**

We have implemented several utilities for the manipulation of Constructs. These utilities operate on Constructs as a funcitonal unit rather than individual Fragments because the Construct is used to represent a single molecule of DNA (with multiple genetic components). The utilities listed in Table I mimic operations performed on DNA molecules in the laboratory and can therefore be used to simulate the construction of recombinant DNA molecules. All utilities modify Construct definitions to reflect the biochemical modifications which would result if the operations they represent were performed in the test tube. For example, TRM (trim) and FLN (fill in) alter the Frag_end, seq and Specification members of Constructs as if their overhang ends had been trimmed with an exonuclease or filled in with a polymerase to produce blunt ends. Combined wtih the ability to rapidly excise any portion of any sequence in the sequence data banks, these tools provide a convenient and powerful means of simulating genetic constructions and preparing nucleotide sequences for use by analytical programs.

Two key programs in this set of utilities are LIG and CUT. LIG is used to ligate two Fragment ends together to form a Construct. CUT is used to

**TABLE I**
**SOFTWARE TOOLS FOR MANIPULATING FRAGMENTS AND CONSTRUCTS**

| | |
|---|---|
| EXC | - Excise a Fragment from the nucleic acid sequence data bank |
| RAW2FRG | - Convert a sequence in a text file to a Fragment |
| LIG | - Ligate Fragment ends to make Constructs |
| RC | - Reverse-complement a Construct |
| REV | - Reverse a Construct |
| CMP | - Complement a Construct |
| TRM | - Trim overhanging ends of a Construct |
| FLN | - Fill in overhanging ends of a Construct |
| PUR | - Purify a Construct from a group of Constructs |
| NSRT | - Insert a Construct into a group of Constructs |
| DLT | - Delete a Construct from a group of Constructs |
| BRKFRG | - Break a Fragment at a specific point; creates new Fragments |
| LINKFRG | - Merge two Fragments to create a Construct (used for assembling sequences) |
| LSTFRG | - List Fragments and Constructs in a frg file |

generate from one Construct (consisting of one or more Fragments) the set of Constructs which would result if the DNA sequence represented by the original Construct was cut by one or more restriction endonucleases specified by the user. CUT recognizes all internal restriction endonuclease sites as well as the sites at all boundaries of ligated Fragments, thus it is able to correctly recognize all sites in a linked-list representing a covalent circle of DNA.

CUT makes use of a data structure called **Endo** to represent the recognition and cutting sites for specific endonucleases:

```
typedef struct {
     char name[24];          /* name of endonuclease */
     char pattern[24];       /* recognition sequence for enzyme */
     char sym;               /* (0 or 1) if true, recognition site is
                                symmetrical */
     int oh_size;            /* length and type of overhang left by
                                endonuclease cut */
```

```
int to_end;              /* number of bases past start of recognition
                            site to the end of the fragment on the
                            5'-side of the cut */
int to_next;             /* number of bases past start of recognition
                            site to the end of the fragment on the
                            3'-side of the cut */
Endo *next;              /* used to form linked list of Endo
                            structures */
} Endo;
```

The **pattern** member of the structure is used by a general purpose pattern finding function to find all occurences of the pattern in a Construct sequence. The **to_end** and **to_next** members specifiy exactly where the cuts occur relative to the start of the pattern and this information is used to generate the new Fragment structures representing the cut Construct (Figure 1). The **sym** member is used to determine if the reverse-complement of the sequence should be searched for recognition sequences which aren't symmetrical.

## Display of Fragments and Constructs

Fragment definitions contain a great deal of information related to a sequence, much of which is housekeeping data for the manipulation utilities. To make the sequence represented by a Fragment or Construct comprehensible to the user, a simple text representation was devised. This is used to display the contents of a linked-list of Fragments being used in a program or to show the contents of a frg file. The output of a utility called LSTFRG (list

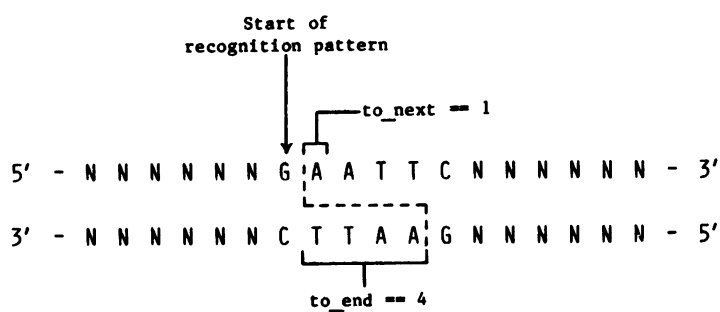


**Figure 1.** Relationship of **Endo** structure members to the recognition site for the restriction endonucleases EcoR1. The **to_end** member is the number of bases from the start of the recognition site pattern to the last base (including overhang) of the DNA on the 5'-side of the cut (represented by the dashed line). The **to_next** member is the number of bases from the start of the recognition pattern to the first base of the DNA on the 3'-side of the cut.

fragments) which displays the contents of a frg file as Constructs is shown here:

```
Construct #1:   [655 bases]
5'-AVAII-(#2014)+++++SV40+++++(#2668)-HPAI-3'      [655 bases]

Construct #2:   [655 bases]
5'-HPAI-(#2668)-----SV40-----(#2014)-AVAII-3'      [655 bases]

Construct #3:   [626 bases]
5'-AVAII-(#5119)+++++SV40+++++(#5243)-|-3'~~       [125 bases]
~~5'-|-(#1)+++++SV40+++++(#501)-HPAI-3'            [501 bases]
```

Illustrated are three constructs derived from cutting (with the program CUT) a construct representing the SV40 genome with the enzymes AvaII and HpaI. The first Construct has a left end generated by AvaII (the 5'-nucleotide of the overhanging end is position 2014 in the parent sequence - the SV40 sequence from GenBank), and a right end generated by HpaI (the 3'-nucleotide is position 2668 in the parent). The "+++++" strings indicate that the sequence positions represented by the Fragment increases from 2014 to 2668. Construct 2 is the reverse-complement of Construct 1. Note that the "-----" string idicates that positions decrease from 2668 to 2014 in this construct. Construct 3 represents two ligated Fragments. Ligation is indicated by the two tildes on the right end of the first Fragment and left end of the second. This construct spans the ends of the SV40 sequence (positions 5243 and 1) as it is provided in GenBank illustrating the use of a construct to represent continuous sequences composed of multiple discrete genetic elements.

Such a display shows very concisely the contents of a group of Constructs and emphasizes the notion that Constructs represent discrete objects which can be manipulated. Our experience has been that users become comfortable with this concept quite readily because of its analogy to the work they perform with DNA molecules in the laboratory.

## The Use of Constructs in a Program

The use of Constructs to represent complex sequences provides a conventional method for interfacing sequences in a sequence data bank with analytical programs. A typical analysis would involve excising desired Fragments from the data bank with EXC followed by any necessary modification with the utilities in Table I. Analytical programs can then use frg files directly as input to specify the sequence to be analyzed.

The following listing illustrates the use of Constructs in a program written in 'C':

```
/* program produces formatted output of the sequence represented by
Constructs in the "frg" file specified on the command line */

#include <stdio.h>

#include <fragment.h>          /* defines data structures */

main(argc,argv)
int argc;
char **argv;
{
     char fname[32];;          /* file name */
     char *seq;                /* pointer to sequence */
     Fragment *first;          /* pointer to first structure in list */
     Fragment *cstp;           /* pointer to first Fragment in various
                                                    Constructs */
     char *getcstseq();
     Fragment *loadfrgs();

     strcpy(fname,argv[1]);
     /* get "frg" file name from command line */

     first = cstp = loadfrgs(fname);
     /* loadfrgs() returns pointer to first Fragment in circular lnked-
     list of Fragments stored in file "fname" */

     do {                      /* loop through Constructs */

          seq = getcstseq(cstp);
          /* getcstseq() allocates space for and returns a pointer to the
          sequence defined by the construct pointed to by "cstp" */

          format(seq);
          /* produce formatted display of sequence pointed to by "seq" */

          free(seq);           /* free space allocated by getcstseq() */
          cstp = nxtcst(sctp);
          /* nxtcst() returns a pointer to the first Fragment in the next
          construct in the linked-list */

          }                    /* end do */

     while (cstp != first);
     /* test to see if all have been formatted */

     }                         /* end main */
```

This example shows the ease with which Constructs are used to represent sequences in analytical programs. The function loadfrgs() loads the Fragment definitions stored in a frg file into a linked-list. The function getcstseq() (get construct sequence) takes care of the details of putting together the

Construct sequence composed of an arbitrarily complex collection of ligated Fragments. The function format() is a general purpose routine for producing formatted output of Construct sequences. The function nxtcst() (next construct) finds the next construct in the linked-list. These functions are stored in a common library for use by a number of different programs.

**Summary**

Two data structures have been developed to represent DNA sequences that can be manipulated by computer programs in a manner similar to the way that DNA molecules are manipulated in the laboratory for genetic engineering experiments. This has proven to be an effective means of simulating the construction of recombinant DNA molecules as well as providing a convenient interface between nucleic acid sequence data banks and analytical programs. The underlying data structures and functions used to operate on them are quite flexible and could serve as a basis for developing efficient user interfaces for the manipulation and analysis of sequences. For example, they could easily be integrated with a system which provides a bit-mapped graphics display and a mouse pointing device to create an interface where the user controls the analysis of sequences simply by pointing to various items and menus on the terminal display: or they could be used as the underlying data structure for an interpreted language for sequence manipulation such as DNA*(3).

The software tools described here are part of a general purpose sequence analysis package which is written in 'C' for processors running the UNIX 4.2bsd operating system.

**ACKNOWLEDGEMENTS**

The author would like to thank Michelle Browner for helpful feedback during the implementation of the data structures and software tools and for constructive criticism of the manuscript; and Steve Delaune for typing the manuscript. This work was supported by NIH grant AI 19578 to the author.

**REFERENCES**

1. Friedland, P., Kedes, L., Brutlag, D., Iwasaki, Y. and Bach, R. 1982. Nucl. Acids Res. 10,323-340.
2. Schneider, T.D., Stormo, G.D., Haemer, J.S. and Gold, L. (1982). Nucl. Acids Res. 10,3013-3024.
3. Schroeder, J.L. and Blattner, F.R. 1982. Nucl. Acids Res. 10,69-84.

4. Caron, P.R. 1984. Nucl. Acids Res. 12,731-737.
5. Shalloway, D. and Deering, N.R. 1984. Nucl. Acids Res. 12,739-750.
6. Engel, L.W. 1985. Bio/Technology 3,329-335.
7. GenBank is a registered trademark of the Department of Health and Human Services.