ELSEVIER

STRUCTURELAB: A heterogeneous bioinformatics system for RNA structure analysis

Bruce A. Shapiro* and Wojciech Kasprzak†

**Image Processing Section, Laboratory of Mathematical Biology, Division of Basic Sciences, National Cancer Institute, Frederick Cancer Research and Development Center, National Institutes of Health, Frederick, Maryland*
†*Frederick Biomedical Supercomputer Center, SAIC Frederick, National Cancer Institute/FCRDC, Frederick, Maryland*

STRUCTURELAB is a computational system that has been developed to permit the use of a broad array of approaches for the analysis of the structure of RNA. The goal of the development is to provide a large set of tools that can be well integrated with experimental biology to aid in the process of the determination of the underlying structure of RNA sequences. The approach taken views the structure determination problem as one of dealing with a database of many computationally generated structures and provides the capability to analyze this data set from different perspectives. Many algorithms are integrated into one system that also utilizes a heterogeneous computing approach permitting the use of several computer architectures to help solve the posed problems. These different computational platforms make it relatively easy to incorporate currently existing programs as well as newly developed algorithms and to best match these algorithms to the appropriate hardware. The system has been written in Common Lisp running on SUN or SGI Unix workstations, and it utilizes a network of participating machines defined in reconfigurable tables. A window-based interface makes this heterogeneous environment as transparent to the user as possible. © *1996 by Elsevier Science Inc.*

INTRODUCTION

The existence of computer systems, electron microscopy, X-ray crystallography, nuclear magnetic resonance (NMR), and biochemical techniques has made it possible to elucidate the structure of RNA molecules. Our research deals with the development of computer methodologies that may be used with external experimental biological information to determine the structural attributes of RNA and to make these attributes more interpretable by the individual researcher.

RNA is a nucleic acid, chemically quite similar to DNA, and it consists of a sequence of bases, the elements of which are composed of A (adenine), C (cytosine), G (guanine), and U (uracil). The RNA sequences are most often single stranded, as opposed to double stranded, which is the case for DNA. These single-stranded sequences, however, have a natural tendency to form double-stranded structures by folding onto themselves. G tends to pair with C and A with U, while less stable pairings, such as G with U, can also occur. The study of the RNA sequences and their structures is very important to the understanding of cell life cycles as well as disease pathways. The importance of RNA can be illustrated by its different cellular functions.[1] The messenger RNA (mRNA) serves as a key component in the synthesis of proteins. The ribosomal RNA (rRNA), behaves as a structural molecule that, as part of the ribosome, interacts with the mRNAs and the tRNAs (see below) in the synthesis of the protein chains on the ribosome. Transfer RNA (tRNA) facilitates attachment of the amino acids that form polypeptide protein chains. RNA molecules can also act as catalysts or enzymes. This characteristic has been used in genetically engineered drugs for its ability to cut and destroy viral RNA. In addition, RNA plays an important role in viruses. It is worthwhile remembering that HIV, the common cold, and polio are but a few viral infections caused by RNA molecules.

An RNA structure analysis system called STRUCTURELAB has been developed, (and is being constantly expanded to include more functionality) that allows a researcher to pursue interactively and in a coherent way a multiperspective analysis of RNA structural conformations. The process has involved the development of new algorithms to explore secondary structure motifs and RNA ter-

Color Plates for this article are on pages 222 through 224.

Address reprint requests to: Bruce A. Shapiro, Laboratory of Mathematical Biology, National Cancer Institute, FCRDC, Bldg. 469, Rm. 150A, National Institutes of Health, Frederick, Maryland 21702-1201.

Received 3 July 1996; accepted 9 August 1996.

0263-7855/96/$15.00
PII S0263-7855(96)00063-X

tiary structure, providing methods for the measurement and visualization of structural similarity from the global and contextual points of view. The system includes facilities allowing exploration in detail of both multiple as well as individual RNA structures. Databases containing thousands of structures can be analyzed. Functions also exist to permit the analysis of RNA tertiary (three-dimensional) substructure interactions. The system facilitates utilization of various software/hardware complexes available at the Frederick Cancer Research and Development Center (Frederick, MD) and elsewhere through computer networking to help determine relationships that exist in the RNA structure problem domain.

PROJECT RATIONALE

For historical reasons and because of architectural considerations many software applications for RNA structure prediction, analysis of the stability and significance of the predicted structures, classification of the families of proposed solutions, statistical and visual analysis and manipulation, etc., have been made available on a variety of computer platforms. These applications have also been written in a variety of languages. The software packages have been optimized to take advantage of specific architectures. Thus, it did not make sense to port them all onto a single platform.

Part of the idea of our ongoing project was to develop a computer workbench providing a uniform and user-friendly interface to the above-mentioned packages as well as new tools for visualization, manipulation, and analysis of the available data and data-processing results. These tools constitute the workbench Lisp code as well as the external applications. Since the field of bioinformatics has been rapidly expanding, it was only reasonable to expect the appearance of new software tools running on any or all of the available platforms of our local area network (LAN) or on some distant machines. Thus, a flexible, robust, and portable communications package linking a user-friendly interface with the remote and integrated analysis packages was required as a central part of the system, as is schematically illustrated in Fig. 1.

COMPUTING ENVIRONMENT AND COMMUNICATIONS REQUIREMENTS

Network characteristics

The previously mentioned RNA analysis software packages are available on a diverse range of computer platforms and operating systems. These currently include mainframes such as VAXes running VMS, a Cray Y-MP supercomputer, Convex minisupercomputers, and a MasPar SIMD massively parallel machine (16,384 processors), all running Unix (with idiosyncrasies: Ultrix, Unicos), as well as a number of SUN, SGI, and DEC workstations running Unix (SunOS, Ultrix). Some of the workstations are on the same Network File System (NFS), while others are not. The overall configuration of the network, i.e. topology, IP addresses, and NFS sharing, as well as specific platforms and their operating systems, are periodically upgraded/changed or totally removed. The programs are written in various languages (e.g., Lisp, C, Fortran, MPL). To facilitate additions of new architecturally dependent software, we have decided to make our system "portable," by which we mean the ability to expand and/or move the RNA Analysis. Workbench from one physical network onto another with comparable types of machines and operating systems, and not necessarily the ability to run it on any machine on a network. In those cases where necessary specific hardware is not available on the LAN, the system must be able to link easily to an appropriate architecture on the wide area network (WAN). Therefore the required communications package must encapsulate knowledge about the characteristics of the network. It must be modifiable as well as expandable in order to accommodate changes in the "environment." It should also make the underlying network complexity, architecture and operating system variety as transparent to the user as possible.

General characteristics of applications

Some of the RNA analysis routines are interactive, while others are run as batch jobs requiring up to several hours of CPU time on the fastest of machines. The entire HIV structure prediction, for example, takes in excess of 6 hr and 25 min on a massively parallel machine. Most of them require a mixture of the user (Q/A) and file input. Certain applications can be run in two modes, as a silent background process or as an interactive, response-driven process. The required communications package had to permit users of the workbench to interact with these programs, manipulate (transfer, read, write) files across the network, and monitor the state of the noninteractive processes (batch jobs, and other background processes).

SPECIFIC APPLICATION REQUIREMENTS

The following list of specific applications is meant to illustrate the problem domain. It is not an exhaustive list of functions available in STRUCTURELAB; rather, it attempts to present the major functional domains.

Sequence manipulation

Several real-time applications running on the central controller workstation accept, manipulate, and return nucleic

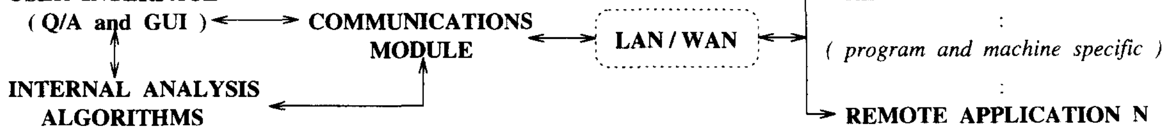


Figure 1. Top-level scheme illustrating the basic elements of STRUCTURELAB.

acid sequence strings (strings of characters). Functions available in this group perform customized sequence creation, manipulation via several types of mutations (single or many bases), and translation to amino acid sequences (protein building blocks). A typical sequence format, among several possible, is illustrated here:

(line format, name, sequence length, sequence)

```
 (60A1) test-file.zuk
117
GAAUUACCGAUAUCGAUACAUCAGGAAUAUUUGAUUCAGAUGAUAUGACUAUCAAGGCCG
CCUGAGUGCGGUUUUACCGCAUACCAAUAACGCUUCACUCGAGGCGUUUUUCGUUAU
```

Folding algorithms

Folding programs employ two different types of algorithms; the Dynamic Programming Algorithm (DPA)[2–7] and the Genetic Algorithm (GA).[8,9] While both attempt to predict secondary structures of an RNA sequence, they differ in basic concepts and rules used. In addition, the GA offers a pseudoknot prediction capability. The RNA folding algorithms accept sequence files (strings) as input, and output multiple region tables indicating which bases (nucleotides) pair with which ones in a folded structure (see Table 1). These region tables reflect energetically optimal and suboptimal solutions based on standardized energy rules. Figure 2 illustrates the fold of the entire HIVRF genome generated by the DPA folding program on a 16 384 processor MasPar MP-2 computer.

Structure representation

Structural representation for a set of RNA molecules can be created on the basis of the region files created by the folding programs. Secondary (2D) structures are represented as trees, utilizing Lisp's nested list notation, with symbols such as M (multibranch loop), B (bulge loop), I (internal loop), and H (hairpin loop) along with property lists. Optional R's present in some representations are no-op place holders indicating regions (hence R) or stems.[10]

Two of the tree representations used are

```
(N(H)(H)(BH)(H)(H)(H)(BBBIH)) —condensed
 (N(R(H))(R(H))(R(B(R(H))))     —expanded
             (R(H))(R (H))(R (H) )
   (R(B(R(B(R(B(R(I(R (H))))))))))))
```

Table 1. An example of a region table representing an RNA structure

Region number	Start (first 5′ base)	Stop (last 3′ base)	Region size (base pairs)	Energy (kcal/mol)
1	9	53	4	−4.3
2	13	48	2	−2.3
3	16	46	3	−2.0
. . .	. . .	. . .	. . .	. . .

Structure alignment

A pair-wise "Needleman–Wunsch" alignment function permits clustering of RNA secondary structures on the basis of similarity of substructures. It uses the parenthesized string form for representing trees, described earlier, and performs a multiple alignment clustering of such representations.[11,12]

Taxonomy operations

Taxonomy tree applications form a class of functions permitting calculation, display, search, and manipulation of structural taxonomy trees. One application does a fast pairwise tree comparison on the tree representations (mentioned above) of the RNA secondary structures and generates a taxonomy tree that clusters the structures on the basis of heuristic measures of similarity. This tree representation facilitates multiple levels of abstraction of the actual structure, allowing for structural comparisons of varying strictness.[10] Another function draws the tree and allows one to manipulate it (see Color Plate 2).

Structure matching

The structure-matching class of functions deals with motif analysis of a set of structures (possibly thousands). Functions available in this class can be divided into two subclasses: one dealing with tree matching and the other with linear features matching. The structure matching operates on the tree list representations and performs pattern searches for structural motif queries, which may include wildcards.

The linear pattern-matching functions utilize data structures used in RNA structure drawings, and they allow one to perform base-by-base searches on conjunctions of linear patterns including sequence, pairing, and structural element membership of the input sequence and its structural conformations.

RNA structure drawing and manipulation

RNA structure drawing and manipulation functions support the drawing of RNA structures and manipulation of the actual drawings for optimum visualization (rotation, resizing, bending/untangling, labeling, annotating). The drawings are based on a sequence file and related region tables.[13,14] In addition, drawings can be generated directly from stem histograms (composite drawings) or stem traces, both described below (see Color Plate 2). We have added a three-dimensional visualization and analysis tool utilizing H. Martinez's rna-2d3d software, which generates atomic coordinates from structures predicted by the various folding programs (H. Martinez, personal communication) and RasMol, which facilitates visualization of these three-dimensional representations[15] (see Color Plate 1).

Two-dimensional stem histogram

The 2-D stem histogram function produces a two-dimensional histogram of all base pairs that exist in a database of optimal and suboptimal structures produced by one of the RNA folding programs. In other words, a two-dimensional

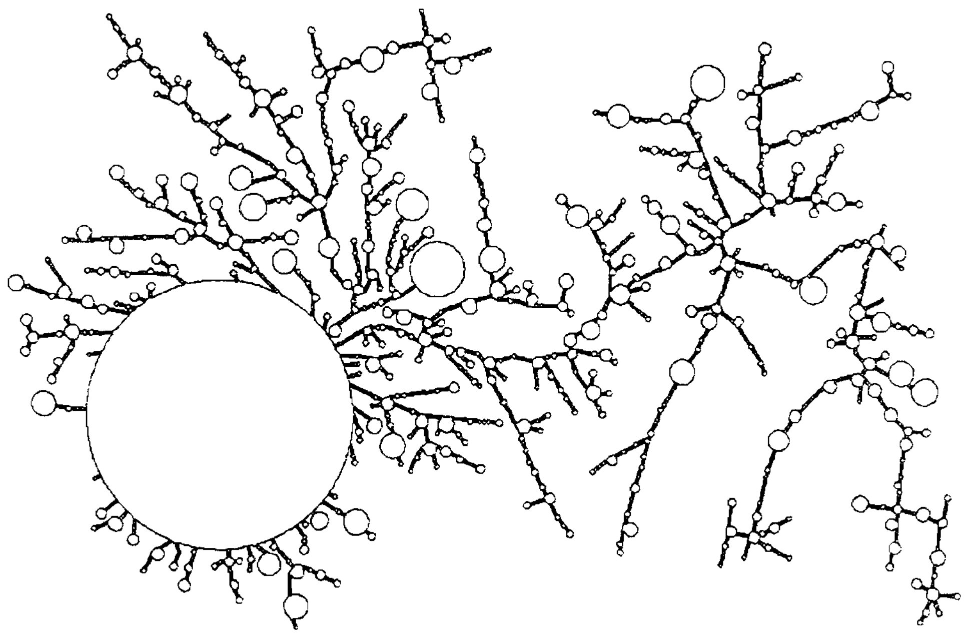


Figure 2. RNA secondary structure of HIV virus (HIVRF strain, 9,128 nucleotides long). Drawing was performed by our own visualization functions described in text.

dimensional histogram gives a picture of how prevalent certain structural motifs are among large numbers of energetically optimal and suboptimal structures.

Combining the statistical base-pairing matrix analysis with the energy-oriented significance/stability information helps create a fuller picture of the reliability of molecule folding predictions (see Color Plate 2).

Stem trace

The stem trace function produces a two-dimensional plot of all unique regions (stems) that exist in a database of structure (region) files. It can be used in the analysis of GA (Genetic Algorithm) structure predictions and for visual exploration of the space of suboptimal solutions predicted by the DPA (Dynamic Programming Algorithm) folding programs. The horizontal axis shows generations (for GA traces) or suboptimal solutions (in the case of DPA), and the vertical axis shows unique regions. Persistence of structural elements thus can be viewed (see Color Plate 3).

Tertiary interaction prediction

The tertiary interaction prediction function generates lists of all potential three-dimensional (tertiary) interactions between the elements of the predicted two-dimensional (secondary) RNA structure (i.e., for a specified sequence and a related region table). Included in these are some valid pseudoknots.[16] The predicted interactions may be filtered on the basis of the user-selected criteria.[17]

Significance/stability plots and energy plots

The significance function computes and displays graphs of the stability/significance of structure folds based on the energy of the optimal fold compared to the energies of a randomly shuffled and folded sequences.[18,19] The energy function reformats the usually very large energy file created by the (DPA) folding program into an energy-only information file. Both of these applications have a remote batch job component, and independent interactive graphing of selected data running on the user's platform (see Color Plate 2).

Miscellaneous applications

A variety of miscellaneous applications is provided to let the user perform ''housekeeping'' and monitoring of tasks not necessarily strictly related to any specific application domain but available in many of them.

IMPLEMENTATION

Project overview

Originally, in the late 1980s, the first-generation system ran on a Symbolics 3675 platform interfaced to a network using TCP/IP.[10–12] This system grew out of work done in the early 1980s.[20] It was ported to a SUN workstation running Allegro Common Lisp with a mix of X-windows and Allegro's Common Windows, and it has been greatly expanded. A brand new communications module had to be written, as the Symbolics networking facilities were not available in other Lisp environments. Also, an entirely new system of windows, menus, and links to the operating system utilities was introduced since these were originally also Symbolics based. The system is currently available on SUN Sparc and SGI workstations. Any one of the host platforms mentioned above can be utilized as the system's central controller. The user can also treat them as front-end machines or utilize other remote stations (for example, a DEC 5000) as displays.

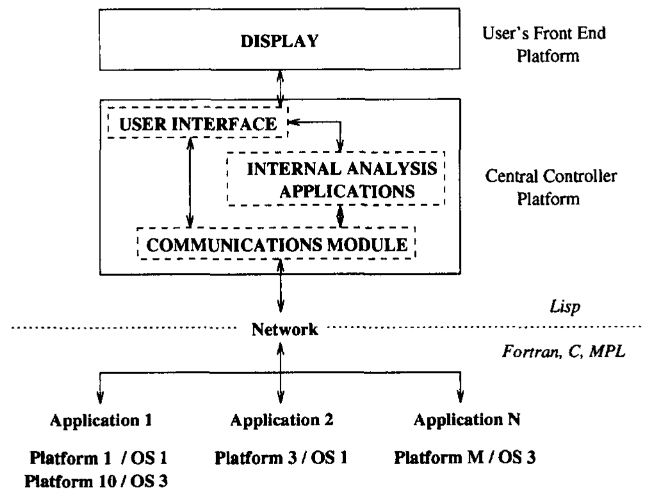


Figure 3. STRUCTURELAB: Layout of a system utilizing a network of heterogeneous computing platforms.

Functions invoked on a central controller workstation can run locally, or on other nodes of the network, such as SUNs, VAXes, Convexes, a Cray, and a MasPar (a massively parallel computer). The results are reported back to the central controller, where they are interpreted and displayed on the user's front-end machine. In such a paradigm algorithms developed and optimized for different architectures may be incorporated and run under an integrated environment. The power and flexibility of Common Lisp[21] makes this programming language ideally suited for the development of a unifying environment on a heterogeneous network, illustrated in Figure 3. On the other hand, the actual programming language used is irrelevant from the user's perspective, since the user does not have to deal with its elements at the interaction level.

A mouse-driven hierarchy of menus allows for a smooth human interface to the available functions. The user can specify any source and target machines within a defined network, and the system will handle necessary connections, file transfers, batch job submissions, or real-time interactions. On-line help information is also available to the user.

Distributed error trapping attempts to allow the user as many run time corrections of errors as possible, or at the very least it signals the nature of fatal errors before aborting the impossible function execution and returning to the last consistent state or the initial menu level, depending on the context of the failure.

In summary, the knowledge about what kind of communications is required to run an application and gather its results is encapsulated within the user interface module of the RNA analysis workbench. The communications module facilitates connections and data transfers. Figure 3 illustrates the system modules.

Communications requirements

The existing computational environment, while essentially having a lot of commonalities, was diverse enough to require a smart and flexible communications module with the following characteristics:

- Encapsulation of knowledge about the network (but not the applications)
- Flexibility, reconfigurability, and portability
- Uniform appearance of the network to the user interface despite heterogeneous nature of the machines employed
- Independence of the user's interface while providing services required by it

The communications module provides support for the following basic communication modes with all platforms on the network:

1. File reading from any machine
2. File writing to any machine
3. Simultaneous combinations of (1) and (2)
4. Independent communication channels for invocation of batch jobs, synchronization of coupled processes, and monitoring of any job's status.
5. Interaction with remote applications

Communications module implementation issues

General communications Since the network with which we are dealing includes machines running multiple operating systems, and physically located in the NCI Frederick, Maryland campus as well as the NIH campus in Bethesda, Maryland, roughly 40 miles apart, we have decided to keep the communications on the high, generic level of TCP/IP streams.

Alternatives considered included sockets or remote procedure cells (rpc), higher level socket packages, and the **rsh/rpc** combination. The most important problem posed by these shell utilities is that they are too specialized and cannot be used for multiple purposes such as batch job submission or interaction, or reused later with other utilities on the same machine. In addition, use of sockets would force us to maintain multiple programs on all the relevant machines of our network. Such spreading of software packages comprising the system would be more cumbersome to maintain. Finally, at the time of the port from Symbolics to SUN workstations our VAX mainframe, an important applications provider on the network, did not offer sockets or rsh daemons. It is only recently that they have become a part of the standard VMS environment.

Network configuration, i.e., names, aliases, and internet addresses of all utilized machines, is read in at the system initialization time from user-defined tables. On the basis of the accessibility of the host relative to the central controller machine we define three classes of platforms; NFS-accessible Unix machines (unix-machine), remote Unix platforms accessible only via TCP/IP streams (non-nfs-unix), and remote VMS machines (vms-machine). This information is stored internally using the Lisp property list, as the following example illustrates:

```
(setf (get 'helix.nih.gov 'net-type)
'non-nfs-unix)
 (setf (get 'fcsparcl.ncifcrf.gov 'net-
 type) 'unix-machine)
 (setf (get 'fcrfvl.ncifcrf.gov 'net-
 type) 'vms-machine)
```

The association of applications with hosts on which they can run is also read in from a user-defined file at startup time.

Our solution utilizes Lisp streams to a shell from which a telnet/ftp connection is opened to the specified remote host. Thus, regardless of the machine with which the front-end applications are communicating, they all talk to standard Lisp streams.

A connection and stream are established quite simply as follows:

```
(setq stream (ERROR-HANDLING-MACRO
(OPEN-STREAM ''telnet host-name'')))
```

Our OPEN-STREAM function essentially executes Allegro CL's command, which starts a Unix child process running telnet.

The ERROR-HANDLING-MACRO intercepts error conditions and extracts error messages for automatic keyword search and response as well as display. Error handling is based on error message contents, and it is performed by the calling environment. In the above case, a failure to open a connection does not break the program, but displays the message and prompts for correction.

At the heart of the communications module are two functions: SEND-MESSAGE-TO-HOST and RECEIVE-DATA. Utilizing the two, STRUCTURELAB uses a form of message passing between processes to synchronize their actions, taking advantage of the naturally interactive nature of the telnet connections between the central controller and remote hosts.

The SEND-MESSAGE-TO-HOST function accepts input strings and sends them out to the appropriate remote host. This function can be safely used to send strings, both commands and output file data, up to 80 characters long. To write longer lines to an output file we provide several higher-level functions building on SEND-MESSAGE-TO-HOST and adding logic to prevent I/O buffer overflows, preserve the original string's format, etc. The SEND-MESSAGE-TO-HOST does not perform any synchronization on its own. It is the caller's responsibility to invoke RECEIVE-DATA for that purpose.

The function RECEIVE-DATA is responsible for synchronization of the central controller actions. It listens, using the Lisp function (**listen** stream), to the responses coming from the target machine and returns received strings up to (and including) synchronization strings passed to it as arguments however many are anticipated in the specific context and whichever matches first. In this way we can easily detect error messages as well as prompts for next input. This function returns the raw string received and a timeout flag, which signals unusual, although not necessarily fatal, problems. The timeout flag raising is based on a real-time limit set by the calling functions. It is left up to the calling environment to handle the context-specific information returned by RECEIVE-DATA.

The majority of problems encountered in this communication scheme dealt with the need to anticipate and handle login time obstacles. The login functions utilize knowledge of the targeted operating system. In case of fatal errors, such as incorrect login name, we close the telnet stream and restart the connection automatically, allowing several trials before quitting. Nonfatal problems are also handled automatically by issuing appropriate responses to queries or issuing break sequences to bypass them. Problems related to customized user shells are handled in this way. Unanticipated shell prompts, for example, are handled via timeout in the RECEIVE-DATA function. In general, as soon as we login safely, we switch to a plain Unix shell (csh -f) and reset the prompt to a unique string for synchronization purposes at which point the process of logging into a remote host is finished.

To minimize connection setup time during the user session, we maintain live streams for reuse by any application running on the same remote host. They are stored in a look-up table (property list), which associates up to two different streams (could be expanded) with every one of the defined hosts as shown here:

```
(setf (get 'host-name 'stream)
'(stream1 · stream2))
```

Two unique bidirectional connections are opened and maintained only when needed, as, for example, in cases in which simultaneous pointing to different directories, processes, or files becomes necessary. In the majority of cases the sequence of operations related to an application startup permits one stream to be used sequentially for both the input and output purposes.

Before attempted use, the streams are tested (live or dead) and connections are automatically reestablished where necessary.

File I/O The high-level communications scheme allows us to tap the system (shell)-level utilities for input file specification expansion by the directory calls such as **ls** in Unix or **dir** in VMS. We also rely on the same calls (issued via the Allegro's excl:run-shell command) for obtaining lists of input files from locally available file systems. However, since we may deal with thousands of input files, limitations of shell expansions of wildcards prompted us to provide backup directory-handling routines. These act on NFS-mounted directories utilizing the Lisp function (**directory** file-specs).

We have also found it necessary to build our own I/O path specification parsers, since the standard Lisp functions perform local machine (central controller) expansion of the path specifications. This is sufficient in cases of local I/O but undesirable in cases where the remote machine is the target. Thus, any wildcard characters are preserved, and the interpretation is left to the target host. The generic path-parsing functions also proved to be quite useful in automatic default path building based on the previous input, which we found to be desirable from the user's point of view.

File reading and writing on remote machines is based on system (shell) commands (**cat** on Unix, **create/type** on VMS, with ftp used for file transfers in both cases). Returned strings are then processed to extract plain file information. Functions used for sending long strings of data to files opened on remote machines keep track of buffer sizes accepted in communications over telnet, and assure preservation of the original format when it is desired.

In case of local (NFS mounted) machines, simple file I/O is utilized by the communication functions. In either case, the same routine is called, and the decision about which streams to utilize is based on the specified host's net-type (explained earlier). The calling routine always receives a Lisp stream.

Communications and I/O setup On the basis of the practical needs of the applications, the routines comprising the communications module are organized into a hierarchy of functions that can be tapped, depending on the specific needs, at the most convenient point. The top function, SET-UP-I/O, opens input and output streams, reads a list of files from the input directory, and opens an output file, at every step performing all the specification correctness checks.

Since the external code utilities accept only the locally accessible input file path specifications, it was necessary to facilitate file transfer to target machines. Initially we tried a simple copy via **cat/type** commands over the telnet connection. This proved to be too slow. Therefore for file transfer purposes we invoke the ftp utility. In our paradigm it is always the target machine that performs the **get/mget** operation on its input files coming from the external source machine. The same stream that is then used to invoke/submit a batch job on the remote host is first used to invoke ftp on that machine and to synchronize input file transfer.

Classification of the applications

The major utilities provided in STRUCTURELAB can be divided into batch and interactive classes on the basis of the nature of their communications needs. Composite applications, such as the three-dimensional drawing or the Genetic Algorithm coupled with two-dimensional structure visualization, utilize both of the approaches in a single run.

Please note that in the following classifications, the domains of Taxonomy Operations, Significance/Stability, and Energy Plotting functions were divided into both batch and interactive parts.

Interactive applications The interactive applications require a communications and file I/O capability across the network as well as user-entered input and mouse interactions handling. This class of applications includes analysis and visualization functions. In general, exiting the chosen application domain ends interacting with the data.

Applications belonging to this category are as follows:

- Pattern matching (Pairing, Linear, and Subtree)
- Sequence manipulations
- Drawing applications, such as two-dimensional and three-dimensional structure drawing, two-dimensional stem histogram, stem trace, energy and significance/stability plots, and structure taxonomy tree
- Generation of structure tree representations
- Tertiary interactions
- Miscellaneous user interface functions (discussed in the section User Interface, below)

Instead of presenting a detailed discussion of every application on the list, we describe representative functions from the interactive applications domain.

Interactive applications: Drawing applications Let's discuss the three related drawing applications: the structure drawing, the two-dimensional stem histogram, and the stem trace displays. They all share the same input requirements: sequence and related region table file or files. The input files can be read from any host on the network. Once the file information is read and processed, many of the internal structure representations are shared by the three functions. Therefore it is quite natural that they can be combined to work in step on the elements of the same RNA structure.

The RNA Structure Drawing facilitates visualization of multiple structures in a single run, e.g., multiple sequence files and their related suboptimal region files are read, processed, and stored in an internal directory of structures. All the supporting information, such as nucleotide sequence associated with the current drawing and internal representations needed to calculate the energy of the RNA structure, are part of the state updates performed any time a selection of a drawing from this menu is made.

The 2-D Stem Histogram visualizes the prevalence of certain structural motifs through the use of color coding based on the frequency of stem (region) occurrences in the analyzed pool of structural conformations. Already drawn structures can be used in conjunction with the 2-D stem histograms or stem traces (discussed below), in which case the correspondence between a particular stem in a histogram or trace and the region of the current structure drawing is signaled by labeling portions of the structure equivalent to the symbolic stem representation.

The Stem Trace provides a view orthogonal to that of the 2-D Stem Histogram. On the vertical axis we are plotting all the unique stems present in all the structures analyzed. On the horizontal axis we are plotting the generation number for the GA generated files, or suboptimal solution space ranking, in case of the DPA results. The stem traces are color coded, based on the user's choice of frequency or energy as the coding criterion. A vertical "slice" cutting through the stem traces at any given point contains information equivalent to the region table for an entire structure. On the basis of this information, the structure can be easily drawn (without the need for any file I/O). What is more, a series of structural drawings based on the data extracted from a selected stem trace range can be displayed in a slow-motion movie-like sequence. This is particularly useful in the analysis of the GA folding pathways without the need to run the program itself. A group of functions related to the Stem Trace plots facilitates structure drawing, stem information display, stem and structure energy calculations, and the previously mentioned labeling of the related structure drawings.

Exiting the structure drawing application domain ends the interactions. On the other hand, stem traces and histograms are based on a different, object-oriented windowing system (Common Windows), and they remain active, as a side effect of multiprocessing, unless explicitly disabled by the user. All the created windows and drawings are left intact until the user explicitly deletes them. The GUI aspects of these functions are discussed in the User Interface section.

Batch applications Our definition of a "batch" application classifies as such all processes submitted to standard batch queues, as well as jobs running without any need for synchronization after startup. The batch-oriented applications require user I/O gathered as input data and used in creation of a shell script, or parameters file. They also perform all input file transfers between source and target hosts prior to the actual batch job invocation or its submission to a batch queue manager. After the remote job submission, the system returns to the application domain menu and is ready for further use. The application runs independently and outputs data to the remote machine's file system. We do not automatically transfer the results back to the central controller platform, since they can be read directly from the remote machine by the next application used. If the user chooses to do so, they can be transferred by one of the auxiliary functions grouped with the batch applications, which also facilitates monitoring of their execution progress via batch queue lookups on demand. Where the queue managers allow it, e-mail is sent to the user upon the completion of a job.

Applications belonging to this category are as follows:

- Folding programs: Dynamic Programming Algorithms and Genetic Algorithms
- Structures alignment
- Taxonomy calculations
- Energy data extraction (for plotting)
- Significance/Stability calculations (for plotting)
- Three-dimensional coordinates calculations (for 3D drawing)

Batch applications: Dynamic programming algorithm The RNA structure prediction programs based on the Dynamic Programming Algorithms are a good example of purely batch-oriented applications. The DPA-based folding programs can handle multiple input sequences. The system permits easy activation of DPA fold on a VAX 6650, a Cray YMP, a MasPar MP-2 2216, and two Convexs for folding of up to thousands of sequences. The user can choose between a smaller and faster algorithm producing a single structure output file (the energetically optimal one) and the more recent version utilizing newer energy rules and outputting a database of multiple suboptimal solutions for every input file. Since the suboptimal folding programs are modular, several combinations of modules, as well as chaining of some or all of them, is facilitated by STRUCTURELAB. The user can select to create several optional output files for more detailed information and potential retracing of the solution matrix calculated in the first phase of the program. All the information provided by the user is gathered and stored in the input data structures. At the end of the run setup stage, the user selects a remote host from a menu of the hosts associated with the application and a queue to which the job is to be submitted. On the basis of the input path specifications and the choice of a remote host on which the job is to be run, the communications streams are opened or tested (if they had been opened previously), and the input files are copied from the source machines, unless they are accessible on the target machine via its file system. Since the DPA folding applications (Fortran/MPL code) gather their input parameters interactively, we create script files with all the parameters entered earlier by the user. Finally, we submit batch job script files on the machines with queue managers. In other cases, jobs are run as shell processes.

Composite applications A good example of a composite application is the three-dimensional structure presentation that first translates the output of a folding program into a representation acceptable to **rna-2d3d** running on an SGI. This program calculates the atomic coordinates of the folded molecule and outputs them into a PDB-formatted file. Once the PDB is available, STRUCTURELAB automatically invokes RasMol, which facilitates interactive molecule visualization and manipulation.

An even better example of an application combining the two modes of operation discussed above and utilizing the flexibility of the communications module is a remote GA fold run controlled from the central workstation. There are several different versions of the GA (it's an ongoing project) that differ in the number of windows created and displayed by the program, as well as inclusion of a GUI. One of the latest versions of GA puts up its own dialog box. In this case our own interactive part of the application is reduced to the minimum number of questions needed to start the run.

Let's discuss the following scenario. We invoke the synchronized GA folding algorithm running on the MasPar. The input sequence file needs to come from a VAX. The structure-drawing program displays its results on a DEC 5000 workstation while a SUN workstation is the central controller platform. The drawing function requires the same sequence file as part of the input, while structure region table files, computed in real time, come from the GA folding algorithm. In this example the communications package must facilitate invocation of the GA on the MasPar and synchronization between the MasPar and the central controller workstation (the SUN) on which the drawing program is run (Color Plate 4).

The following sequence of actions takes place:

1. As a preliminary step, the user sitting in front of the DEC 5000 station opens a window to the SUN workstation and invokes STRUCTURELAB with the DED platform specified as display. Via a menu entry the user starts the GA fold application.
2. All the parameters needed to start the GA are collected in a questions/answers session.
3. Two separate connections (telnet streams) are opened between the SUN workstation (central controller) and the MasPar.
4. The ftp transfer of the input file from VAX to MasPar is performed. Its new location path is passed to the parameter file.
5. The input parameter file is written out and the GA is invoked as a shell process.
6. One stream is used to start the GA run, to synchronize stepping through the GA generations and structure drawing, as well as to read statistical data outputted by the GA after every generation for local display in an auxiliary window. Finally, it is used to terminate the execution. Our functions SEND-MESSAGE-TO-HOST and RECEIVE-DATA are at the core of the synchronization. The other stream is used for reading the input sequence file from the MasPar, and for reading of a region table file created at every generation. After a drawing is finished, a "go" signal is sent to the MasPar to compute the next generation. The synchronization can be performed automatically or manually by the user.
7. Execution pauses after the last GA generation and the user decides when to quit via a small pop-up "quit menu," at which point all the displayed windows (related to the run) are closed.

User interface

The main goal of the user interface is to provide a user-friendly and uniform front-end to the many applications distributed on different machines on the network.

The Graphical User Interface (GUI) is based on menus and windows. The variety of solutions in actual use reflects the changes in tools that have been made available in Lisp development environments since 1990. During the initial phase of the move from the Symbolics to SUN platforms we were using Lucid products. Then we switched to Franz's Allegro Common Lisp. Our menus, which constitute the first layer of user interface, are built on plain CLX library calls. We utilize X-Windows, as well as Allegro's powerful and easy to use object-oriented Common Windows, which were available before the Common Lisp Interface Manager (CLIM) tools. Once CLIM became the industry standard, we have been experimenting with it, and we have some prototypes waiting to be integrated into the system. The dialog boxes for batch utilities are our first candidates. With some assistance from Franz we have tapped into the object-oriented GRAPHER package for graphing and node manipulation of taxonomy tree structures (see Color Plate 2). The GRAPHER package is an integral part of the Allegro Composer development environment, but not a generally available utility. The package facilitates creation of graph/tree structure layouts and display of them in Common Windows. In addition the GRAPHER is used to display the compressed and selectively decompressed terminal nodes, which is helpful in the managing of large graphs with thousands of terminal nodes representing multiple RNA structure conformations. Wherever it was naturally helpful we made objects depicted in windows mouse sensitive for the ease of related data extraction or drawing manipulation, such as in the case of interactive untangling and annotating of RNA structure drawings. In the case of 2D Stem Histograms and Stem Traces, graphical representations of the paired regions (stems) of an RNA structure are mouse sensitive, and these utilities can be coupled with structure drawings so that touching a displayed stem, labels the corresponding area on the structure drawing. In the case of the Stem Traces, an entire structure can be drawn on the basis of the data extracted from the plot with a mouse click. We also make extensive use of color coding of statistical data depicting frequency of element occurrence in any related structural displays. Extraction of numeric data from the energy and significance/stability plots is facilitated via stretchable windows that the user can draw with a mouse over the area of interest. Similar stretchable windows are used to let the user indicate areas of RNA structure drawings to be zoomed (redrawn and enlarged).

The STRUCTURELAB menus are organized into a tree of submenus grouping functions related to one application domain. For example, sequence folding functions or structure visualization functions are accessible from their respective submenus. In the case of the majority of functions some interactive input gathering is necessary. The number of on-line queries is kept to a minimum. Wherever the use of the system has revealed typical input patterns, we have provided pop-up submenus in which certain choices result in an automatic input path and file name building in accordance with accepted conventions. On the other hand, we have left the maximum flexibility for manual customization for unconventional needs to the system user. A good example of this general philosophy can be found in the structure-drawing utility in which, after entering the input path to a sequence file, the user can select items such as OPTIMAL-REGION-FILE-FORMAT or SUBOPTIMAL-REGION-FILE-FORMAT, which automatically accesses related region table files in the same directory from which the sequence file is coming. Choosing the USER-DEFINED-LIST-OF-REGION-FILES gives the user full freedom of specifying input paths to related region files, which in such a case may be read from many different hosts on the network.

In general we practice maximum anticipation of typical query answers, and we provide defaults that can be accepted by entering carriage returns or overwritten at will. In cases where it's impossible to guess the potential answers, we store the user-entered values and provide them as defaults during the next invocation of the same utility. This approach turned out to be helpful in the interfaces to batch utilities. These will be targeted first in our phasing in of the dialog boxes.

Uniformity of the I/O format specifications makes it easy for the user to encode networking elements within the source/target path specifications. Our format appears as follows:

[hostname::]/directory/subdir/filename for the Unix platforms
hostname::disk$name:[dir.name]filename for the VMS platforms

All the acceptable wildcard characters and regular expressions are permitted. The host name is optional for files coming from the NFS directories. The default host is established at the system start-up time, and it is the central controller machine on which the system is running. This example also illustrates the limits of uniformity. It is left up to the user to know the proper path formats on the machines that are on the utilized network.

We provide on-line help information using the same menus to the system utilities. By first selecting the HELP entry and then another menu item, the user can read a short description of the application/function and an overview of its I/O requirements. Examples of runs and results illustrate the usage. In addition to this help information, hints on the use of the mouse in the mouse-driven applications (such as Stem Histogram, Stem Trace, or structure drawings) are displayed in a small information window put on the display for the duration of an application's use.

The majority of the miscellaneous functions in the STRUCTURELAB system are provided so that the user can easily tap the system/shell resources on any of the machines without leaving the workbench environment. These can be used for directory listing, or batch queue status display. Another group of functions allows the user to write text information or pictures to local files or print them out on any of the several accessible printers. Basic sequence retrieval from a selected database (GenBank, or other) via an interface to the GCG package, and format conversions from one standard to another, also belong to this category. A useful utility taking advantage of Allegro's fasl-write and fasl-read functions facilitates binary state dumps and retrievals of the results of interactive applications such as structure visualizations, stem traces, and stem histograms. These allow one to store the state of data structures compactly, the rebuilding of which would require a tedious and/or time-consuming sequence of steps.

Error handling

The error-handling routines are decentralized, with localized, context-specific knowledge. These functions are meant to deal with problems at the levels at which errors can be detected and attempt to let the user correct mistakes and continue execution of the code. As it was mentioned in the discussion of the communications module, we intercept error conditions that normally throw an execution sequence into the Lisp interpreter level and let the user see the messages and take action whenever it is appropriate.

The first layer of error checking is the input type correctness check. It is a part of our function for gathering user input. In cases where we know what kind of input to expect, the appropriate test function name is passed as a key argument. Input is not accepted until the correct type or a break string ``$'' is detected. Our test functions check for string, list, numeric, real, and integer input types, thus providing functionality now available in CLIM I/O.

In addition to strict input type tests, we also provide warnings and assurance checks on input files with unusually named extensions, which is quite useful since the majority of files do have standard extensions, and it is an easy way of preventing potentially fatal errors caused by the mismatching of an input file with a utility. The user, however, can override any system warnings of this kind.

The I/O error interception permits the user to correct entries found to be erroneous, such as wrong input path specification, or inaccessible, such as protected directories, without the need to restart the entire login sequence from the very beginning. In case of fatal communication problems, such as connection closing by foreign host in the middle of data transfer, we provide cleanup procedures for the critical pieces of code enclosed in the unwind-protect special Lisp forms that facilitate controlled recovery from error conditions. In general, rules and actions taken are similar in nature to database transaction handling with rollback to the last consistent state. For example, if a fatal I/O error happens during drawing of multiple RNA structures before all data structures related to the current drawing are safely stored in the system's internal drawing directory, the last drawing window is deleted, and the latest fully finished (committed) drawing and its internal structures are restored as the current state. The user receives a message explaining the actions taken and the state of the system.

PORTABILITY

We have two run-time images of our system, one of which runs on a SUN and the other on an SGI platform, sharing the same source code with only a few code extensions that are different. Use of the time and space profiler in the Allegro Composer development environment has allowed us to optimize the crucial parts of the code and minimize memory usage on both platforms. We have introduced data type declarations (optional in Lisp) and incorporated compiler directives for maximum speed and minimum debugging entry points in the most speed-sensitive, often utilized, and fully debugged functions.

In preparation for the SUPERCOMPUTING '94 conference in Washington, D.C. we created a runtime system image independent of the site libraries to be transferred and run on a remote machine. We have also conducted a series of test runs utilizing the MasPar facilities in California. Cross-country tests of the Genetic Algorithm for RNA structure folding,running on a machine in California, tightly coupled with STRUCTURELAB running on the central controller workstation in our laboratory (Frederick, MD), proved quite successful. All that was needed was to add the remote MasPar address to our network tables (and an account on the remote machine, of course). In addition, we moved the central controller Lisp code to a remote workstation, and let the MasPar people test run it locally. Finally, we have successfully demonstrated the system portability onto a temporary network set up for the SUPERCOMPUTING '94 exhibit, utilizing MasPar's floor machine as well as our own MP-2 in Frederick, Maryland (50 miles apart). Names and IP addresses of the machines employed were defined on the show floor.

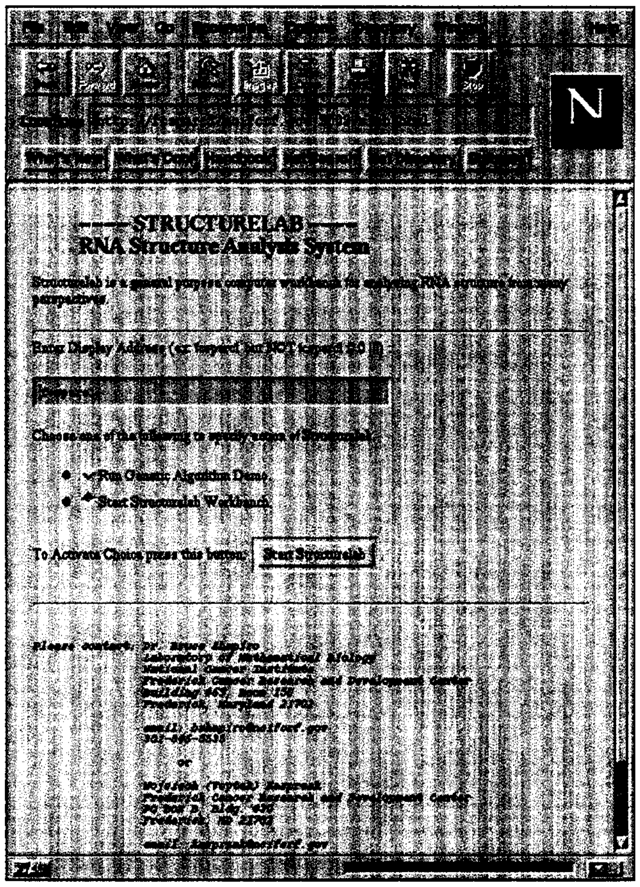
WWW INTERFACE

We have written a WWW/http home page for access to the system as well as an automatic demonstration run of the Genetic Algorithm. The home page (html code) gathers initial arguments (see Figure 4) and invokes a WWW server (C code) which, in turn, starts the system. The user must specify the name of the display, i.e., the host name and the domain name of the user's machine (such as fcsparc1.ncifcrf.gov or fcsparc1, as shown in Figure 4) and to choose starting the entire system or the ``canned'' Genetic Algorithm demonstration. An X-terminal window brought up at the indicated user's host display is utilized as the main STRUCTURELAB I/O window, with the regular menus and graphics windows providing the graphical interface.

The Genetic Algorithm demonstration requires the absolute minimum of interaction from the user while it illustrates a live run of the GA on the MasPar coupled with the STRUCTURELAB drawing program running on the central controller machine. The STRUCTURELAB Workbench choice allows the user to explore all the tools of the system within limits of a predefined network of machines. At the moment access to these demonstrations is limited to password holders.

CONCLUSION

STRUCTURELAB has proved to be a useful tool in several collaborations. Some examples include the study of structural influence on translation efficiency of the *LamB* gene and the L11 operon of *Escherichia coli*.[12] Work has also been done on refining the secondary structure of the Rev response element sequence of HIV-1.[22] STRUCTURELAB has been used to explore RNA termination sites[23] and is currently being used in exploration of the structural aspects



——— STRUCTURELAB ———
RNA Structure Analysis System

Structurelab is a general purpose computer workbench for analyzing RNA structure from many perspectives.

Enter Display Address (ex: fcsparc1 but NOT fcsparc1:0.0):

Choose one of the following to specify usage of Structurelab:

- Run Genetic Algorithm Demo.
- Start Structurelab Workbench.

To Activate Choice press this button: Start Structurelab

```
Please contact: Dr. Bruce Shapiro
                Laboratory of Mathematical Biology
                National Cancer Institute
                Frederick Cancer Research and Development Center
                Building 469, Room 150
                Frederick, Maryland 21702

                email: bshapiro@ncifcrf.gov
                [illegible]

                      or

                Wojciech (Wojtek) Kasprzak
                Frederick Cancer Research and Development Center
                PO Box B, Bldg. 430
                Frederick, MD 21702

                email: kasprzak@ncifcrf.gov
```

Figure 4. WWW home page (under Netscape) with options to run a GA demonstration or the entire system.

of the HIV dimerization site.[24] Finally, it has also been used for studying the structural aspects of the coxsackie virus[25] and polio virus.[26]

We also view STRUCTURELAB as a good demonstration of the flexibility of Common Lisp. The language turned out to be useful in building the system from the low-level routines, such as those utilized in our Communications module, up to high-level functions and data representations. The memory requirements are perfectly reasonable for today's workstations, and the execution speed is more than satisfactory.

Tightly coupled heterogeneous computations are already utilized, and expansion of the idea to even more applications is envisioned. Computations using HIPPI or ATM connections between the fastest of the machines on the network are being considered, and facilitating some sort of tight scheduling and chaining of multiple processes on several machines is envisioned.

We are also in the process of expansion of the newly added three-dimensional structure visualization and analysis capabilities and more extensive integration of them with other tools provided by the system.

ACKNOWLEDGMENTS

The authors thank Dr. Kathleen Currey for patient testing of the system and for feedback on bugs and the need for new utilities and improvements in the user interface. We also extend our gratitude to the people of Franz, Inc. for their help and support. In addition, we thank Dr. Jacob Maizel, Jr. for his support.

REFERENCES

1 Watson, J.D., Hopkins, N.H., Roberts, J.W., Argetsinger Steitz, J., and Weiner, A.M. *Molecular Biology of the Gene,* Benjamin/Cummings, 1987

2 Jaeger, J.A., Turner, D.H., and Zuker, M. Improved predictions of secondary structures for RNA. *Proc. Natl. Acad. Sci. U.S.A.* 1989, **86,** 7706–7710

3 Jaeger, J.A., Turner, D.H., and Zuker, M. Predicting optimal and suboptimal secondary structure for RNA. *Methods Enzymol.* 1989, **183,** 281–306

4 Turner, D.H., Sugimoto, N., and Freier, S.M. RNA structure predictions. *Annu. Rev. Biophys. Biophys. Chem.* 1988, **17,** 167–192

5 Zuker, M. On finding all suboptimal foldings of an RNA molecule. *Science* 1989, **244,** 48–52

6 Zuker, M. and Stiegler, P. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Res.* 1981, **9,** 133–148

7 Shapiro, B.A., Chen, J., Busse, T., Navetta, J., Kasprzak, W., and Maizel, J., Jr. Optimization and performance analysis of a massively parallel dynamic programming algorithm for RNA secondary structure prediction. *Int. J. Supercomput. Appl.* 1995, **9**(1), 29–39

8 Shapiro, B.A. and Navetta, J. A massively parallel genetic algorithm for RNA secondary structure prediction. *J. Supercomput.* 1994, **8,** 195–207

9 Shapiro, B.A. and Wu, J.C. An annealing mutation operator in the genetic algorithms for RNA folding. *Comput. Appl. Biol. Sci.* 1996, (in press)

10 Shapiro, B.A. and Zhang, K. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biol. Sci.* 1990, **6**(4), 309–318

11 Shapiro, B.A. An algorithm for comparing multiple RNA secondary structures. *Comput. Appl. Biol. Sci.* 1988, **4,** 378–393

12 Margalit, H., Shapiro, B.A., Oppenheim, A., and Maizel, J., Jr. Detection of common motifs in RNA secondary structure. *Nucleic Acids Res.* 1989, **17,** 4829–4845

13 Shapiro, B.A., Maizel, J., Jr., Lipkin, L., Currey, K., and Whitney, C. Generating nonoverlapping displays of nucleic acid secondary structure. *Nucleic Acids Res.* 1984, **12,** 75–88

14 Shapiro, B.A., Lipkin, L., and Maizel, J., Jr. An interactive technique for the display of nucleic acid secondary structure. *Nucleic Acids Res.* 1982, **10,** 7041–7052

15 Sayle, R. *RasMol v2.5: A Molecular Visualization Program.* Glaxo Research and Development, Greenford, Middlesex, UK, 1994

16 ten Dam, E., Pleij, K., and Draper, D. Structural and functional aspects of RNA pseudoknots. *Biochemistry* 1992, **31**(47), 11665–11676

17 Le, S., Shapiro, B.A., Chen, J., Nussinov, R., and Maizel, J.V., Jr. RNA pseudoknots downstream of the frameshift sites of retroviruses. *Genet. Anal. Tech. Appl.* 1991, **8**(7), 191–205

18 Le, S., Chen, J., Currey, K., and Maizel, J.V., Jr. A program for predicting significant RNA secondary structures. *Comput. Appl. Biol. Sci.* 1988, **4,** 153

19 Chen, J., Le, S., Shapiro, B.A., Currey, K.M., and Maizel, J.V., Jr. A computational procedure for assessing the significance of RNA secondary structure. *Comput. Appl. Biol. Sci.* 1990, **6**(1), 7–18

20 Shapiro, B.A. and Lipkin, L.E. Nucleic acid morphology: Analysis and synthesis. In *Computing in Biological Science* (. Geisow and Barrett, Eds.). Elsevier Biomedical Press, Amsterdam, 1983, pp 233–271

21 Steele, G.L. *Common Lisp—The Language,* 2nd ed. Digital Equipment Corporation, 1990

22 Dayton, E.T., Konings, D.A.M., Powell, D.M., Shapiro, B.A., Butini, L., Maizel, J.V., Jr., and Dayton, A.I. Extensive sequence-specific information throughout the CAR/RRE, the target sequence of the human immunodeficiency virus type 1 Rev protein. *J. Virol.* 1992, **66**(2), 1139–1151

23 Cheng, S.C., Lynch, E.C., Leason, K.R., Court, D.L., Shapiro, B.A., and Friedman, D.I. Functional importance of a sequence in the stem–loop of a transcription terminator. *Science* 1991, **254,** 1205–1207

24 Sakaguchi, K., Zambrano, N., Baldwin, E.T., Shapiro, B.A., Erickson, J.W., Omichinski, J.G., Clore, G.M., Gronenborn, A.M., and Appella, E. Identification of a binding site for the human immunodeficiency virus type 1 nucleocapsid protein. *Proc. Natl. Acad. Sci. U.S.A.* 1993, **90,** 5219–5223

25 Tu, Z., Chapman, N.M., Hufnagel, G., Tracy, S., Romero, J.R., Barry, W.H., Zhao, L., Currey, K., and Shapiro, B.A. The cardiovirulent phenotype of coxsackievirus b3 is determined at a single site in the genomic 5′ untranslated region. *J. Virol.* 1995, **69,** 4607–4618

26 Currey, K. and Shapiro, B.A. Secondary structure computer prediction of the polio virus 5′ non-coding region is improved with a genetic algorithm. *Comput. Appl. Biol. Sci.* 1996 (in press)