J. Functional Programming 5 (3): 443–460, July 1995

Exploring the conformations of nucleic acids

MARCEL TURCOTTE, GUY LAPALME AND FRANÇOIS MAJOR

Laboratoire de biologie informatique et théorique,
Département d'informatique et de Recherche Opérationnelle, Université de Montréal,
C.P. 6128, succursale Centre-ville, Montréal, Québec H3C 3J7, Canada
(e-mail: {turcotte,lapalme,major}@IRO.UMontreal.CA)

Abstract

This paper presents an application of functional programming in the field of molecular biology: exploring the conformations of nucleic acids. The *Nucleic Acid three-dimensional structure determination problem* (NA3D) and a constraint satisfaction algorithm are formally described. Prototyping and experimental development using the Miranda functional programming language, over the last four years, are discussed. A Prolog implementation has been developed to evaluate software engineering and performance criteria between functional and logic programming. A C++ implementation has been developed for distribution purpose and to solve large practical problems. This system, called MC-SYM for 'Macromolecular Conformation by SYMbolic generation', is used in more than 50 laboratories, including academic and government research centres and pharmaceutical companies.

Capsule Review

Determining the three-dimensional structure of nucleic acids and proteins is one of the great challenges in molecular biology. It requires extensive combinatorial and numeric computation, and a great deal of explorative programming. This paper demonstrates how the virtues of FP can contribute to this fascinating research area.

1 Introduction

The work described here is part of a joint collaborative project on the structure/function relationship of nucleic acids, involving the Biochemistry and the Computer Science Departments at the Université de Montréal. Interest in nucleic acids is in part due to recent discoveries of their enzymatic activity and the role they play in molecular evolution (Gray and Cedergren, 1993). Detailed knowledge of their structure is essential for the comprehension of their function. For the majority of nucleic acids, only the *sequence* of nucleotides (from which the complete chemical composition can be deduced) is known, and not the three-dimensional structure. This is attributable in part to the progress in sequencing techniques and, in a related way, to mega-sequencing projects, such as the Human Genome Project (Frenkel, 1991). The ability of the biological research community to analyse these sequences

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

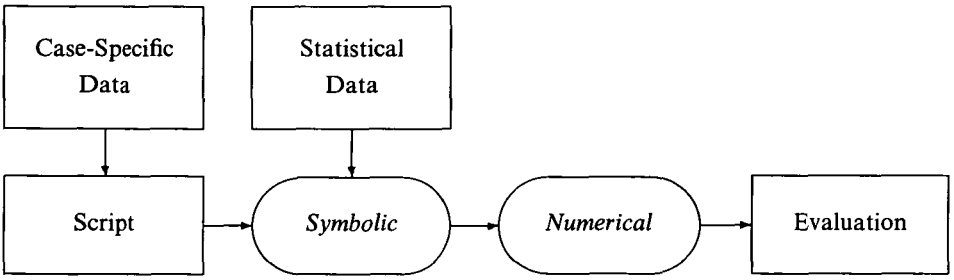


Fig. 1. Flow of information and processes. This exploration process is repeated with slightly different parameters based on the results of previous runs

(primarily through determining their 3D structure) greatly lagging behind the generation of the primary data. There is thus a great need for sequence analysis tools to infer 3D structure from the primary sequence.

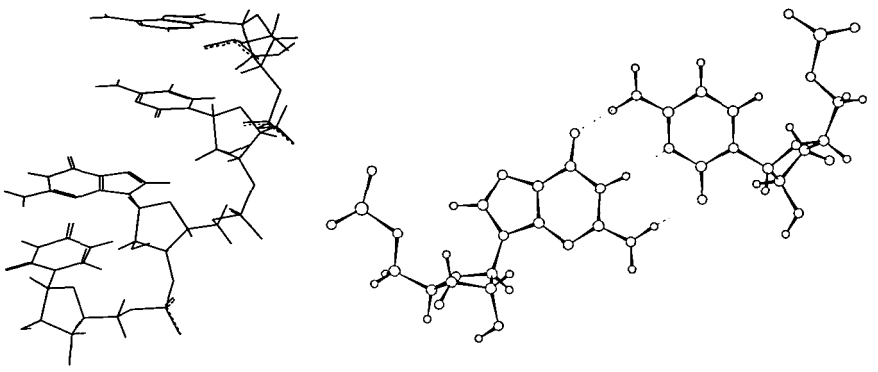
Most successful computational approaches to the structure determination problem rely on homology and computer graphics modelling. These approaches are based on the observation that natural selection has produced families of molecules in which the sequences of nucleotides have diverged widely, but the three-dimensional structures and functions have remained the same. By selecting parts of experimentally determined structures which have significant sequence homology to fragments of the target nucleic acid sequence, one can 'manually' construct three-dimensional structures with the aid of a molecular display program. However, the small number of experimentally determined nucleic acid structures limits the application of this approach. A review of RNA modelling techniques is presented elsewhere (Major, 1995).

Our approach to the structure determination problem is two-fold (figure 1). In the first step, the 'symbolic generation', a preliminary pool of coarse structures is generated using a *Constraint Satisfaction Problem* (CSP) algorithm described here. In the second step, the 'numerical refinement', off-the-shelf energy minimization and molecular dynamics packages are used to refine the structure. This two step approach has the advantage of reducing the size of the search space explored by the energy minimization method. The precision lost in the symbolic generation model is recovered in the numerical step.

The next section provides some background theory on nucleic acid structures. Section 3 leads to the description and resolution of this problem within the CSP paradigm. Section 4 presents the early development of this project with the Miranda system. Finally, Section 5 discusses the latest development of this system, namely its translation to other programming paradigms.

2 Nucleic acid structure

Nucleic acid molecules exist in two forms (see Saenger, 1984, for a review). The first, deoxyribonucleic acid (DNA), serves as the main storage medium for genetic information. The second, ribonucleic acid (RNA), usually participates to the intermediary steps for protein synthesis, such as genetic information transportation. In



(a) Nucleotide units (b) Watson-Crick type base-pair

Fig. 2. Nucleotide units. (a) shows the four types of ribonucleotides; from top to bottom A,C,G and U. "... *nucleotides contain a phosphate group linked to a five-carbon-atom sugar* group, which, in turn, is joined to a flat aromatic molecule that can be either a double-ringed purine or a single-ringed pyrimidine. Since they contain the sugar deoxyribose, the nucleotides of DNA are called deoxy-ribonucleotides, while those of RNA, which contain the sugar ribose, are known as ribonucleotides" (Watson *et al.*, 1987). The flat aromatic molecule, commonly called the nitrogen base or simply base, are stacked on the top of each other on this picture. (b) shows a Watson–Crick type base-pair, nucleotide G to the left and C to the right, dotted lines indicate hydrogen bonds

addition, several RNAs are also known to have enzymatic activities. Nucleic acids are chains of smaller molecules, the nucleotides (figure 2). Nucleotides start by a phosphorus atom and end by an oxygen atom labeled, $O_{3'}$. The $O_{3'}$ atom of the nucleotide i is connected to phosphorus atom of nucleotide $i+1$ in the chain. There are four types of nucleotides, notated A, C, G and T for DNA and A, C, G and U for RNA. Nucleic acids mostly consist of two complementary strands, running in opposite directions, bonded together by forming hydrogen bonds. The canonical complementarities are A with U and G with C (Watson–Crick type base-pairing pattern). DNA usually forms double helix, that is, the two strands have a spiral shape. The length of DNA chains is often measured in thousands of nucleotides. While RNA chains range from small nuclear RNAs of less than 30 nucleotides in length to large ribosomal RNAs which contain over 3000 nucleotides. RNAs are characterized by three levels of organization (see figure 3). The sequence of nucleotides is called the *primary structure*. The second level of organization, the *secondary structure*, arises from the fact that the complementary bases interact and form double-helical domains. Finally, bases from single-stranded regions can interact in space and further fold the molecule defining the relative placement of double-helical domains and exact 3D coordinates of all atoms, the *tertiary structure*.

Amongst the biologically active RNAs, the $tRNA^{Phe}$ and the $tRNA^{Asp}$ from yeast organism are the only ones for which the tertiary structure has been so far determined, see Table 1. Transfer RNAs are involved in the translation of the genetic code, from an RNA molecule termed messenger RNA, to proteins, and are present

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

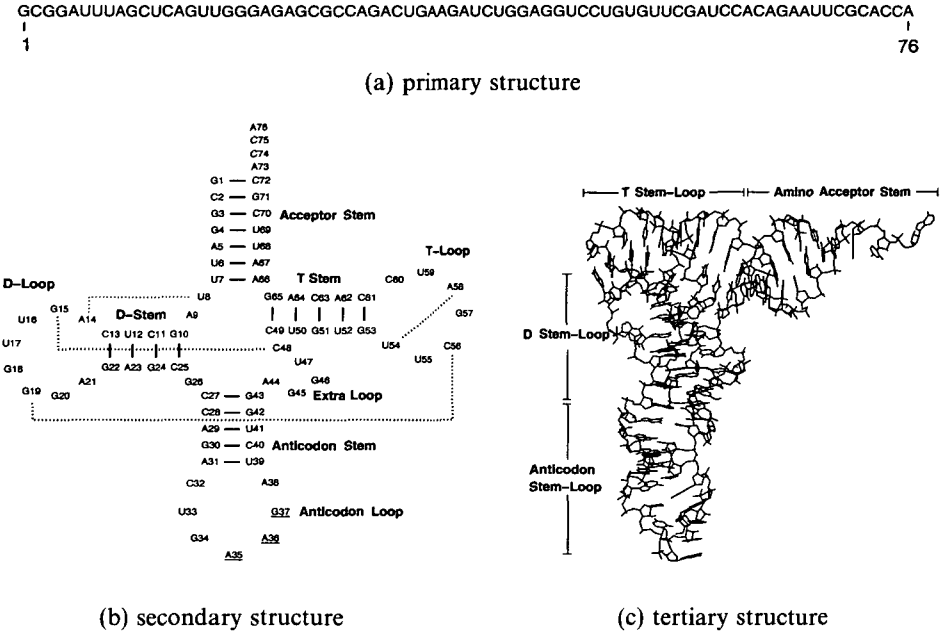


Fig. 3. Example of the three levels of organization of RNAs. (a) Linear sequence of nucleotides; (b) two-dimensional folding of the molecule: it shows the juxtaposition in space of distant nucleotides in the sequence (solid lines represent regular base-pairings while dotted lines show long-range tertiary interactions); (c) three-dimensional structure. Anticodon, acceptor, D, T and extra loop are the main regions of tRNA. A stem is a double helical domain, while a loop is a single-stranded region. The molecule shown is the yeast Phenylalanine tRNA, entry number 1TRA of the Protein Data Bank (Bernstein et al., 1977), or entry number TRNA06 of the Nucleic Acid Database (Berman *et al.*, 1992)

in multiple copies in every living organism. A compilation of all known tRNA sequences (Steinberg *et al.*, 1993), containing 2011 entries has been assembled. The yeast $tRNA^{Phe}$ molecule, shown in figure 3, is 76 nucleotides long and is composed of 1652 non-hydrogen atoms. It serves as a benchmark for modelling techniques and will be used as an example in the remainder of this paper.

Molecular biologists have developed reliable and rapid methods for determining the primary structure of proteins and nucleic acids. The resulting data are collected and made available by research organizations such as the National Center for Biotechnology Information; the October 1994 version of the GenBank (release 85.0) contained over 200 millions bases from over 215,000 sequences. On the other hand, determining the 3D structure by purely experimental means is still a time consuming task. This explains why the October 1994 Protein Data Bank release contained only 2,921 three-dimensional structures from proteins, DNAs and RNAs (see figure 4).

The most accurate method for determining the 3D structure of nucleic acids is X-ray crystallography (Saenger, 1984). It is a costly process that can not be applied to all nucleic acids. X-ray crystallography has been used successfully to determine the 3D structure of only about two hundreds nucleic acids. This technique permits

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

Table 1. *List of all published tRNA structures*

NDB[a]	PDB[b]	Citation	Organism	Amino acid
TRNA03	1TN2	(Brown et al., 1985)	*yeast*	Phenylalanine
TRNA04	6TNA	(Sussman et al., 1978)	*yeast*	Phenylalanine
TRNA05	—	(Comarmond et al., 1986)	*yeast*	Aspartic
TRNA06	1TRA	(Westhof and Sundaralingam, 1986)	*yeast*	Phenylalanine
TRNA07	2TRA	(Westhof et al., 1988)	*yeast*	Aspartic
TRNA08	3TRA	(Westhof et al., 1988)	*yeast*	Aspartic
TRNA09	4TRA	(Westhof et al., 1988)	*yeast*	Phenylalanine
TRNA10	4TNA	(Hingerty et al., 1978)	*yeast*	Phenylalanine

[a] Unique accession number to the Nucleic Acid Database (Berman *et al.*, 1992)
[b] Unique accession number to the Protein Data Bank (Bernstein *et al.*, 1977)

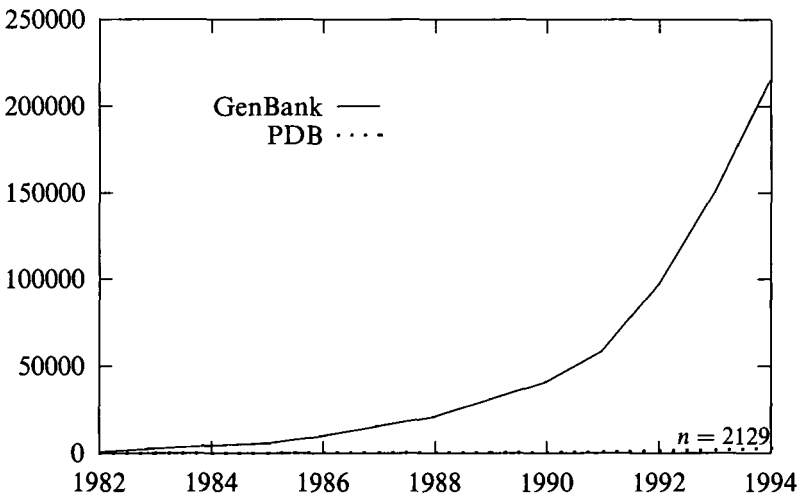


Fig. 4. Number of entries in the sequence and structure databases. GenBank is the NIH genetic sequence database, a collection of all known DNA sequences and PDB is the Brookhaven National Laboratory protein structure data bank

determination of the structure of large molecules, but as if requires formation of crystals *in vitro*, it may not accurately describe the structure of the molecule in the cell, *in vivo*. Another method, nuclear magnetic resonance (NMR), alleviates some of the problems with X-ray crystallography as it allows one to study molecules in solution closer to their normal environment. Under some conditions, NMR experiments can produce sufficient proton-proton distances (Nuclear Overhausser Effects, NOE, distance restraints) so that distance geometry algorithms (Blumenthal, 1970) can be applied to produce accurate 3D models. So far, NMR has only been applied successfully on small nucleic acids. However, like X-ray crystallography, NMR cannot be applied to all nucleic acids and for the two methods, the conditions of experiments, state, ionic concentration, pH, context, etc. differ from the ones in the living cell.

Hence, one of the most important unsolved problems in molecular biology is still the *structure determination problem*: 'given a sequence of nucleotides, determine the three-dimensional structure of the biologically active molecule'. But at least in the short term, it may not be necessary to solve this problem without additional information.

Additional three-dimensional structure data come from various experimental and theoretical methods, such as chemical and enzymatic probing, and comparative sequence analysis (Gautheret and Cedergren, 1993). The use of specific enzymes, e.g. enzymes that cut single-stranded regions, and sequencing techniques can provide additional information about the secondary structure of a molecule. Some chemical agents that are specific to the nucleotide bases can be used to detect paired and non-paired nucleotides. Comparative sequence analysis is based on the observation that corresponding RNA molecules from different organisms adopt a similar set of base-pairs, i.e. the molecules have a common secondary structure. By comparing the nucleotide sequences of RNA molecules one can infer almost all secondary structure interactions and some tertiary interactions (Levitt, 1969).

Thus, the nucleic acid structure determination problem (NA3D) can be reformulated as: 'given the sequence of nucleotides determine the structures that are consistent with inferred secondary and tertiary interactions'. Since the information from the comparative sequence analysis and the experimental data is easily expressed in terms of constraints we have encoded the problem within the CSP paradigm.

3 Solving the problem

In Major *et al.* (1991b), we defined the NA3D problem as a discrete CSP. The CSP consists of finding assignments for the variables $x_1, .., x_n$ so that a set of constraints is satisfied. To use the standard resolution techniques, such as backtracking, each variable has to be restricted to a discrete domain, $x_i \in D_i$.

3.1 Backtracking with help

A backtracking algorithm is used to solve the CSP. This algorithm enumerates systematically all solutions by recursively appending values for the next variable that satisfies the constraints. Complete assignments are stored in a list of solutions. The Miranda (Mir, 1989)† implementation presented here utilizes problem-specific information to narrow the search space dynamically. This concept, Domain Generating Functions (DGF) (Major *et al.*, 1991b), is elegantly implemented through higher order functions. Given the following data types:

```
solution * == [*]
domain * == [*]
dgf * == solution * ->  domain *
constraint * == * -> solution * -> bool
```

† Miranda is a registered trademark of Research Software Ltd.

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

where solutions and domains are lists of values, a DGF is a function that, given a partial solution in input, calculates the domain of values for the next variable assignment, a constraint verifies that a value is compatible with the list of previously assigned variables. The backtracking function is defined as follows. The `backtracking` function has three parameters: the list of instantiated variables (the path from the root up to that node), the list of curried DGFs and the constraint function.

```
backtrack :: solution * -> [dgf *] -> constraint * -> [solution *]

backtrack p [] c = [p]
backtrack p (f:fs) c
  = concat [backtrack (v:p) fs c | v <- f p; c v p]
```

Equation (1) stops the recursion when it encounters an empty DGF list, meaning that a complete solution, `p`, has been found. Equation (2), the curried DGF `f` is removed from the list and applied to the partial solution, `p`, and its evaluation generates the values for the current variable. Those values that satisfy the constraints, the qualifier `c v p`, are explored through the recursive call, `backtrack (v:p) fs c`. Finally, the resulting solution lists are concatenated.

The manipulation of functions as first class objects in higher-order FP languages simplifies the abstraction and implementation of common behaviors. Here the DGF list, `f:fs`, expresses backward dependencies, that is the domain of values, D_i, for the variable x_i, is a function of the previously assigned variables $x_1, .., x_{i-1}$.

3.2 Mapping NA3D into data types

In the nucleic acid application there is one CSP variable per nucleotide. The values specify 3D positions, orientations and conformations of each CSP variable. The conformation corresponds to internal structure, as external factors can be simulated with the use of a set of rigid nucleotides. The specification of the position and orientation is somewhat problematic since it needs to be discrete. Other approaches are based on lattices, but to attain a useful precision these lattices need to be dense. The strategy we adopted here is to introduce problem-specific information to dynamically reduce the degrees of freedom of the domains. The justification is that the secondary and tertiary interactions restrict the relative placement of nucleotides in space (see section 2 and figure 3).

In this application, the nucleotides are modelled as rigid objects, a set of points `nuc`, and a transformation matrix, `tfo`.

```
pt  ::= Pt num num num
tfo ::= Tfo num num num num num num num num num num num num
var ::= Var num tfo nuc
nuc ::= Nuc tfo tfo tfo tfo
        pt pt pt pt pt pt pt pt pt pt pt pt pt pt pt pt pt
        pt pt pt pt pt pt pt pt
        nuc_specific
```

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

```
nuc_specific
    ::=   A pt pt pt pt pt pt pt pt
        | C pt pt pt pt pt pt
        | G pt pt pt pt pt pt pt pt pt
        | U pt pt pt pt pt

na3d :: solution var -> [dgf var] -> constraint var -> [solution var]
na3d = backtrack
```

One can define a specialized version of `backtrack` applied to this problem.

3.3 Nucleic acids specific domain generating functions

DGFs are used to introduce structural information. The MC-SYM system has two kinds of users, one would like to study various parameterizations of nucleic acid structure thus defining new DGFs. The second kind of user would employ a specific set of DGFs to build 3D models of some experimental molecule and explore a particular simulation.

A DGF is a function of `solution *` to `domain *`. When fed with a partial solution, it computes the candidates for the next variable assignment. As an example, the `wc` function calculates a transformation matrix so that each `nuc` taken from the list `nucs` will form a Watson-Crick type base-pair with the complementary nucleotide `j`:

```
wc :: [nuc] -> num -> num -> solution var -> domain var

wc nucs i j partial_inst
  = [ Var i (t nuc) nuc | nuc <- nucs ]
    where
      t      = dgf_base wc_tfo (get_var j partial_inst)
      wc_tfo = Tfo
               (-1.0000)  (0.0028) (-0.0019)
                (0.0028)  (0.3468) (-0.9379)
               (-0.0019) (-0.9379) (-0.3468)
               (-0.0080)  (6.0730)  (8.7208)
```

More precisely, `get_var j partial_inst` fetches the nucleotide labeled `j` in `partial_inst`. This nucleotide serves as a reference point for `dgf_base`, which computes the coordinate frame of nucleotide `j`, effectively a transformation matrix. Then, `dgf_base` combines it with the `wc_tfo` matrix given in input. The result is the list of `var` and `nuc` is drawn from the list of rigid nucleotides `nucs`, which mimic internal deformations.

This pre-calculation of DGFs constitutes *a priori* pruning of the domains. The transformation matrices can only be computed when all residues involved in a relation are appended to the structure. DGFs reflect this dynamic behaviour well.

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

3.4 Case study: transfer RNA anticodon loop

To illustrate the use of the control structure in the context of nucleic acid modelling, we have considered the modelling of the anticodon loop region of the tRNA molecule. This region comprises the nucleotides 31 to 39 (see figure 3).

Before the tertiary structure of tRNA was established, Levitt compared fourteen different tRNA sequences (Levitt, 1969) and proposed that nucleotide A31 formed a base-pair with the complementary nucleotide U39, in the anticodon stem. He also suggested that the bases of nucleotides 34 to 38, in the anticodon loop, were stacked on the top of each other.

To program the search, the user has to choose an order in which the nucleotides will be appended to the structure. This order is defined by the list `anticodon_domains`. Note that this order may differ from the sequence order. For instance, the loop region could be grown from both ends, and a distance constraint is used to ensure that the chain will be closed. The user supplies the constraints of the problem, `anticodon_constraint`, here a phosphorus-oxygen distance of 3 angström (Å) or less between U33 and G34, that are not explicitly connected by a DGF:

```
anticodon_domains
  = [ reference rA  31,
      wc        rU  39 31,
      stacked3' rA  38 39,
      stacked3' rG  37 38,
      stacked3' rA  36 37,
      stacked3' rA  35 36,
      stacked3' rG  34 35,
      p_o3'     rCs 32 31,
      p_o3'     rUs 33 32]

anticodon_constraint v partial_inst
  = (dist 34) <= 3.0, if i = 33
  = True, otherwise
    where Var i t n = v
          dist j = pt_dist p o3'
                   where p = atom_pos nuc_P (get_var j partial_inst)
                         o3' = atom_pos nuc_03' v
```

The following call produces the list of all solutions:

```
na3d [] anticodon-domains  anticodon-constraint
```

The evaluation of the expression `reference rC 31 partial_inst` produces a single placement for the first nucleotide (an arbitrary reference point); `rC` is one rigid nucleotide conformation for the C ribonucleotide, 31 is its label, and `partial_inst` is an empty list. To reduce the expression `wc rU 39 31 partial_inst`, nucleotide 31, which is used as a reference for the transformation of U39, is sought in the list `partial_inst`. A 3-D transformation matrix is calculated so that when applied to

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

the rigid conformation of U39, a Watson-Crick type base-pairing pattern, involving A31 and U39, is formed.

The DGF `wc` returns single-value domain, and therefore only one partial structure is built. The anticodon loop region is composed of nucleotides 32 to 38. Placement flexibility must be introduced for these nucleotides to close the chain. The function call `stacked3' rA 38 39 partial_inst` generates two possible placements for the rigid nucleotide A38 so that its base is stacked under the base of A39. A similar computation is applied to nucleotides 37, 36, 35 and 34. Since there is no constraint for these nucleotides, 32 (2^5) partial structures are generated.

The `p_o3'` DGF takes a set of rigid conformations (for instance `rCs` and `rUs` contains 10 different conformations), and tries three different placements for each of them. In this way, 30×30 values for C32 and U33 are generated. The number of leaves of the complete search tree for this problem is thus 28,000, a leaf corresponding to a complete anticodon model.

Only 179 solutions satisfy the polynucleotide chain closure constraint. Figure 5 illustrates the tree-like computation. The accuracy of the structures generated by MC-SYM could then be evaluated by their agreement with the known structure. The whole tRNA molecule has been modelled with this system. For a molecule whose farthest two atoms are 90 Å apart, the average error is 3.5 Å (Major *et al.*, 1993). This system has also been applied to the discovery of new molecular structures such as the HIV-1 Rev-binding element (Leclerc *et al.*, 1994), the leadzyme (Foucrault *et al.*, 1995) and the orientation of tRNA in the ribosomal A and P sites (Easterwood *et al.*, 1994).

4 Program development

The project was carried out by a multi-disciplinary team; two computer science graduate students and one molecular biology graduate student supervised by one professor from the computer science department and one professor from the biochemistry department. During the four years of experimental development, four generations of the MC-SYM system were built in Miranda. The size of the last Miranda version is 7865 lines of code (the database accounts for 1700 lines).

4.1 Choosing the programming language

The major dilemma in constraint programming (CP) is to choose between expressiveness and efficiency. On the one hand, a general implementation of solving techniques liberates the programmer of tree search programming but complicates the introduction of problem specific information that could 'help' the search. On the other hand, a very specific program might complicate its modification and disables reusability. Mainly because of its relational form, logic programming (LP) has been considered an excellent vehicle for constraint programming (Dincbas, 1986). Efficient resolution procedures based on forward checking and looking ahead have been embedded in Prolog using the concept of domains (Van Hentenryck and Dincbas, 1986; Hentenryck, 1989).

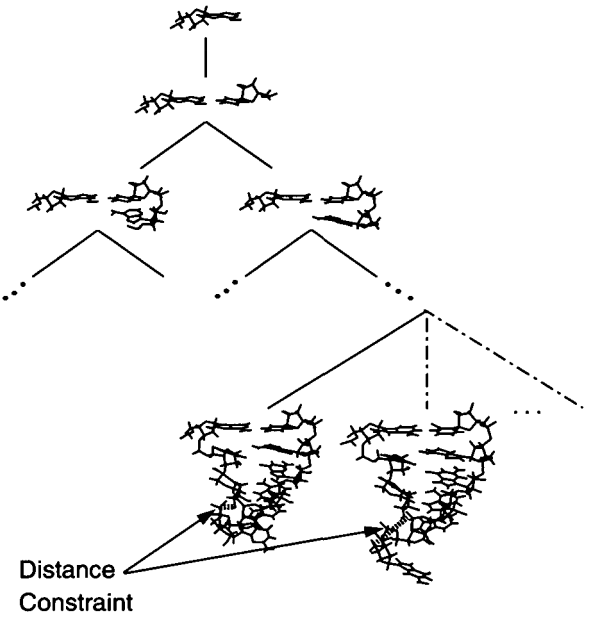


Fig. 5. Tree-like computation. First node corresponds to line 1 of the anticodon script, second node to line 3, third level corresponds to line 4, and the last level corresponds to line 9. The consistency check of line 12 is shown with arrows on the last stage, the structure to the left satisfies the 3 Å requirement for the chain to be closed while the structure to the right does not

loosing algorithmic expressiveness. To develop the DGF concept, polymorphism and higher-order functions were used to abstract the backtracking control structure (Major *et al.*, 1991a). Lisp was originally chosen for its rich development environment and expressiveness. After some investigations on ways to implement lazy evaluation for the large and dynamic domains and to implement a terse syntax for list expressions, Miranda became the language of choice.

5 Recent development

The modelling experiment with MC-SYM is an iterative process. The user has to elaborate many scripts with different hypotheses, examine the results, and restart the procedure. Thus, turnover time is very important.

As the size of the NA3D problems processed by MC-SYM increased (several days of computation on currently available workstations), it became clear that the runtime had to be reduced. It was a limiting factor for the modelling of larger molecules, for the implementation of larger DGF sets and to check for van der Waals overlaps, which occur when the distance between any two atoms is less than the sum of their van der Waals radii.

Finally, to make the modelling tool useful for a large number of molecular biologists, a portable stand alone application was required.

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

5.1 Translation to C++

The Miranda code is organized into a series of abstract data types and their operations. In C++, each data type is implemented as a class object. An important class object is the *CSP*. A *CSP* instance contains a set of variables, domains and constraints. To each variable is associated a pointer to its domain and an other to its current value. In the original Miranda program, the user expresses a modelling problem by defining the list of DGFs, such as the `anticodon-domains` in the above example. In the C++ implementation, this function is replaced by an RNA structure description syntax. This has required the development of a parser to read the syntax and create a *CSP* instance. Memory is allocated dynamically during parsing, so that a fixed amount of memory is used during the simulation. The nucleotide conformations and transformation matrices, which are part of the Miranda code, have been put in an external database. Additional syntaxes for the description of nucleotide conformations, spatial transformations and their respective sets have also been developed.

The backtracking algorithm in the C++ system is iterative, not recursive such as the backtracking control structure in Miranda. The algorithm operates using an integer array and indexes indicating the current status of the search. Domains are represented by two arrays: one for the conformation and one for the transformation sets. The domains are maintained using two indexes: one to indicate the current conformation and one to indicate the current transformation. For each variable assignment, the indexes are incremented, a message is sent to the *CSP* object so that the current value is calculated and assigned to the nucleotide, and a constraint function is applied for the partial structure. The message sent to the *CSP* object mimics the application of a DGF in the Miranda system. The constraint function, such as the backtracking procedure, is independent of the *CSP* object. It sends messages to the *CSP* object to get a list of constraints to be checked as well as their parameter values.

Power of expression for the users has been lost with the RNA structure description syntax. For instance, it is not possible to introduce new functions that can be applied in the description of a molecule. Furthermore, all the system functions that were available to the users are now hidden in objects and are not recognized by the parser. A much more sophisticated parser or an embedded command language such as TCL (Ousterhout, 1990) would be necessary to keep this feature without a compilation step. For developers, code conciseness, clarity and ease of use have all decreased with the C++ implementation. The implementation of functions in C++ does not follow the Miranda 'philosophy'. For instance, curried functions which were largely used in the Miranda prototype are absent in the C++ code. Currying functions allows one to develop flexible control structures which can be curried into their variations, such as the backtracking into forward checking. This flexibility to fundamentally modify an algorithm has been lost using C++. Code modifications have repercussions in many other parts of the program, where in Miranda they were often local to a function. The current C++ version is composed of nearly 9000 lines of source code and an enhanced database of near 100,000 lines.

In spite of these limitations, the goals of the MC-SYM C++ translation have been reached. The new system is much faster than the Miranda prototype. The

speed of the very first C++ working implementation had increased by nearly two orders of magnitude. This speed factor has also allowed for the implementation of additional useful features such as the detection of van der Waals overlaps and the control of execution by generating traces. The traces allow for incremental runs and generation of qualitative reports of the output. The trace also allows the user of MC-SYM to kill a running search so that it can be restarted where it was stopped. Once a script has been run, traces indicate which values were used in the solutions so that the rediscovery of the same inconsistencies is reduced when the script is run again ('trashing' effect). In this way it allows the user to split a problem in several subproblems, and incorporate previously developed scripts. MC-SYM also generates specific information for each solution. This information is used when individual solutions are requested. This information can also be parsed to generate a qualitative report on the conformational properties of the nucleotides of a particular model. In addition to the basic functions of the system, global variables have simplified error handling and search monitoring. For instance, it was simple to add global variables that count the number of nodes explored, the number of constraint checks, etc.

As much lazy evaluation as possible has been retained in the C++ version. For instance, atomic coordinates are calculated on demand when a constraint is checked, the nucleotide conformations are loaded only if required, etc. We found that the backtracking function when implemented as lists of success and DGF has simplified the translation to this applicative language. We believe that this would not have been the case if we had used logic programming languages extented with consistency techniques.

The anticodon problem with the C++ version generates 35 solutions, with van der Waals overlaps detection, in approximatively 3 seconds using a R4400/150 MHz mips CPU. Twenty-six complete tRNA-Phe models are generated in approximatively five minutes.

The statement of the anticodon problem using the structure syntax developed for the C++ system is as follows:

```
SEQUENCE
   A    rA 31 reference          typeA
   A    rU 39 wc              31 typeA
   A    rA 38 stack           39 typeA
   A    rG 37 stack           38 typeA
   A    rA 36 stack           37 typeA
   A    rA 35 stack           36 typeA
   A    rG 34 stack           35 typeA
   A    rC 32 connect         31 typeA
   A    rU 33 connect         32 typeA
GLOBAL
  P    P      3.0
  C1'  C1'    3.0
  PSE  PSE    3.0
```

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

```
  P     C1'     2.5
  P     PSE     2.5
  C1'   PSE     2.5
  C4    C4      2.0
  N1    N1      2.0
  N1    C1'     2.0
  N1    C4      2.0
  O2'   O4'     2.0
  O3'   C5'     2.0
  P     C3'     2.0
ADJACENCY
   0  3
FORMAT
 allbh
```

5.2 *Parallelization*

More recently, we investigated the parallelization of the system using the Multilisp language. In Feeley *et al.* (1994), we found that the Miranda prototype was relatively easy to parallelize using the Multilisp language. On 64 processors, the parallel functional program is up to 27 times faster than a sequential version of the same program rewritten in C. We also found that the major speed up came from tuning the sequential application, especially with regard to floating-point operations and memory allocation. A factor within 2 has been obtained with the sequential Scheme version, compared to the optimized C code.

This subset system has been the subject of a benchmark experiment for many FP languages, including Caml, Clean, Common Lisp, Erlang, Gofer, Haskell, Id, Miranda, Opal, RUFL, SML, Scheme, Stoffel, and Trafola (Hartel *et al.*, 1994). This experiment showed that, for this application, the speed of C can be approached by some implementations, but not without special measures such as strictness annotations.

5.3 *Translation to Prolog*

Mainly because of its relational form, logic programming has been considered the language of choice for implementing constraint programming techniques (Dincbas, 1986). To evaluate the costs of choosing FP as the implementation language, we have implemented a Prolog version. Quintus Prolog was chosen for its rich development environment featuring a debugger and a native-code compiler (Qui, 1991).

The algorithm and overall structure of the Miranda program were preserved. The implementation strategy was to replace each Miranda function by a rule having one more parameter through which it returns the result.

There is no explicit search procedure since this program relies on the implicit backtracking mechanism of Prolog. The DGFs still play the central role. Definite Clause Grammars (Pereira and Warren, 1980) formalism is used for problem state-

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

ment. Thus, Prolog automatically adds two parameters to each predicate, those parameters chain each consecutive predicate. The first parameter is the list of nucleotide variables as calculated by the previous predicate. The second parameter is the result of the consing operation of a newly calculated value to the list, or the same list if the predicate is a constraint, in which case it might have access to the list but it does no action on it:

```
anticodon -->
        reference(rA, 31),
        wc(       rU, 39, 31),
        stacked3_(rA, 38, 39),
        stacked3_(rG, 37, 38),
        stacked3_(rA, 36, 37),
        stacked3_(rA, 35, 36),
        stacked3_(rG, 34, 35),
        p_o3_(    rCs,32, 31),
        p_o3_(    rUs,33, 32),
        dist_lt(nuc_O3_, 33, nuc_P, 34, 3.0).
```

The following call produces the solutions:

```
anticodon( [], ListOut).
```

`ListOut` holds one solution, `bag_of` predicate should be used to accumulate the solutions if needed. The DGF `wc` is implemented as follows:

```
wc(Nucs, I, J, Partial_inst, [var(I, Tfo, X)|Partial_inst]) :-
           get_var(J, Partial_inst, Nuc_J),
           wc_tfo(T),
           Term =.. [Nucs, X],
           call(Term),
           dgf_base(T, Nuc_J, X, Tfo).

wc_tfo(tfo(-1.0000, 0.0028, -0.0019,
            0.0028, 0.3468, -0.9379,
           -0.0019,-0.9379, -0.3468,
           -0.0080, 6.0730,  8.7208)).
```

`get_var` fetches the nucleotide labeled `j` in `partial_inst`. This nucleotide serves as a reference point for `dgf_base`, which computes the coordinate frame of nucleotide `j`, effectively a transformation matrix. Then, it combines it with matrix `T`, as returned by `wc_tfo( T )`. `Univ` operator is used to construct a term of the form `nucs(X)`, its evaluation, `call(Term)` binds `X` to nucleotide conformations. The backtracking mechanism constructs of all successive alternatives in `var(I, Tfo, X)`.

In the Miranda program, atom access functions were manipulated as first-class objects, in Prolog we manipulate functors and function application is replaced by the construction of a term and its evaluation, `Term =.. nuc_p( Nuc, X), call(Term)`, the evaluation of `Term` binds `X` to the coordinates of atom P in `Nuc`.

https://doi.org/10.1017/S0956796800001428 Published online by Cambridge University Press

Table 2. *Timing results for the Miranda, Prolog and C implementations. C benchmarks have been considered the lower bound for speed. Run on a Sun Sparc 10/512 machine, 4 processors SuperSparc 50 Mhz, 224 Mbytes RAM, running Solaris 2.2*

Problem	Miranda		Prolog		C	
	Compile	Run	Compile	Run	Compile	Run
anticodon	5.0	298.8	11.5	37.8	55.2	0.9
pseudoknot		548.2		56.3		1.7

One of the drawbacks of the Prolog version is the implicit backtracking mechanism. When this mechanism is not needed, it has to be avoided by using explicit cut operations or -> (if-then-else) constructs. In fact, the backtracking is only required for the DGFs.

The translation to Prolog took 2 days of work. Table 2 presents the timing results for the Miranda, Prolog and C versions for the same subset system. The Prolog implementation is 7.9 times and 9.7 times faster than Miranda for the anticodon and pseudoknot problems respectively.

6 Conclusions

Higher-order and currying functions combined with polymorphism simplified the development and experimentation of the DGF concept. The terse syntax of list expressions and lazy evaluation influenced the choice of Miranda over other FP languages for the prototyping of MC-SYM. However, mainly due to performance and portability issues, we had to translate the system to C++ and explore its parallelization.

The abstraction of the backtracking control structure and the identification of its functional parameters have allowed us to emphasize their implementation in other programming paradigms. In particular, the translation of DGFs to logic and imperative languages was relatively easy.

This 'real-world' application is in use in more than 50 laboratories all around the world, although using it requires some expertise in defining the order of the variable and choosing the right set DGFs. It would be interesting to develop a way to deduce this information from a higher order description of the problem.

The MC-SYM system is available by anonymous FTP from
<ftp://ftp.IRO.UMontreal.CA/pub/lbit>.

Acknowledgments

Daniel Gautheret, currently at Université Aix-Marseille II, and Robert Cedergren, at the Université de Montréal, are members of the MC-SYM development team. We thank Mark Cavanaugh and Tim Littlejohn for their careful reading of this manuscript and useful comments. Marcel Turcotte holds a studentship from the

Medical Research Council of Canada (MRC). François Major is a fellow of the Canadian Genome Analysis and Technology (CGAT) program and MRC. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, CGAT and MRC.

References

Berman, H. M., Olson, W. K., Beveridge, D. L., Westbrook, J., Gelbin, A., Demeny, T., Hsieh, S. H., Srinivasan, A. R. and Schneider, B. (1992) The nucleic acid database: A comprehensive relational database of three-dimensional structures of nucleic acids. *Biophys. J.* **63**: 751–759.

Bernstein, R., Koetzle, T. F., Williams, G. J., Meyer, E. F., Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T. and Tasumi, M. (1977) The Protein Data Bank: A computer-based archival file for macromolecular structures. *Eur. Biochem.*, **80**: 319–324.

Blumenthal, L. M. (1970) *Theory and Applications of Distance Geometry.* Chelsea.

Brown, R. S., Dewan, J. C. and Klug, A. (1985) Crystallographic and biochemical investigation of the lead(II)-catalyzed hydrolysis of yeast Phenylalanine t-RNA. *Biochemistry* **24**: 4785–4801.

Comarmond, M. B., Giege, R., Thierry, J. C., Moras, D. and Fischer, J. (1986) Three-dimensional structure of yeast t-RNA-Asp. I. Structure determination. *Acta Crystallogr., Sect.B.* **42**: 272–280.

Dincbas, M. (1986) Constraints, logic programming and deductive databases. In: *France-Japan Artificial Intelligence and Computer Science Symposium.*

Easterwood, T., Major, F., Malhotra, A. and Harvey, S. (1994) Orientation of transfer RNA in the ribosomal A and P sites. *Nucl. Acids Res.* **22**(18): 3779–3786.

Feeley, M., Lapalme, G. and Turcotte, M. (1994) Using Multilisp for solving constraint satisfaction problems: an application to nucleic acid 3D structure determination. *Lisp and Symbolic Computation*, **7**: 85–100.

Foucrault, M., Cedergren, R. and Major, F. (1995) Modeling the lead-activated ribozyme by intersection of conformational space. *In preparation.*

Frenkel, K. A. (1991) The human genome project and informatics. *Comm. ACM* **34**(11): 41–51.

Gautheret, D. and Cedergren, R. (1993) Modeling the three-dimensional structure of RNA. *FASEB J.* **7**(1): 97–105.

Gray, M. and Cedergren, R. (1993) The new age of RNA. *FASEB J.* **7**(1): 4–6.

Hartel, P., Feeley, M., Alt, M. *et al.* (1994) Pseudoknot: a float-intensive benchmark for functional compilers. *Submitted.*

Hentenryck, P. (1989) *Constraint Satisfaction in Logic Programming.* MIT Press.

Hingerty, B. E., Brown, R. S. and Jack, A. (1978) Further refinement of the structure of yeast t-RNA-Phe. *J. Mol. Biol.* **124**: 523.

Leclerc, F., Cedergren, R. and Ellington, A. D. (1994) A three-dimensional model of the Rev-binding element of HIV-1 derived from analyses aptamers. *Nature Structural Biology* **1**(5): 293–300.

Levitt, M. (1969) Detailed molecular model for transfer ribonucleic acid. *Nature* **224**: 759–763.

Major, F. (1995) Computer modeling of RNA 3-D structures. In: *Molecular Biology and Biotechnology: a comprehensive desk reference.* Robert A. Meyers Ed. VCH Publishers, New York, NY.

Major, F., Gautheret, D. and Cedergren, R. (1993) Reproducing the three-dimensional structure of a transfer RNA molecule from structural constraints. *Proc. Natl. Acad. Sci. (USA)* **90**: 9408–9412.

Major, F., Lapalme, G. and Cedergren, R. (1991a) Domain Generating Functions for Solving Constraint Satisfaction Problems. *J. Funct. Prog.* **1**(2): 213–227.

Major, F., Turcotte, M., Gautheret, D. *et al.* (1991b) The Combination of Symbolic and Numerical Computation for Three-Dimensional Modeling of RNA. *Science* **253**: 1255–1260.

Miranda System Manual (1989) Research Software Limited, Canterbury, UK.

Ousterhout, J. (1990) TCL: An embeddable command language. In: *1990 Winter USENIX Conference*, pp. 133–146.

Pereira, F. and Warren, D. (1980) Definite clause grammars for natural language analysis – a survey of the formalism and comparisons with augmented transition networks. *Artif. Intell.* **14**: 231–278.

Quintus Prolog 3.1: Reference Pages (1991) Quintus, Palo Alto.

Saenger, W. (1984) *Principles of Nucleic Acid Structure.* Springer-Verlag, New-York.

Steinberg, S., Misch, A. and Sprinzl, M. (1993) Compilation of tRNA sequences and sequences of tRNA genes. *Nucl. Acids Res.* **21**: 3011–3015.

Sussman, J. L., Holbrook, S. R., Warrant, R. W. *et al.* (1978) Crystal structure of yeast Phenylalanine t-RNA. I.crystallographic refinement. *J. Mol. Biol.* **123**: 607–630.

Van Hentenryck, P. and Dincbas, M. (1986) Domains in logic programming. In: *Proc. AAAI-86.*

Watson, J. D., Hopkins, N. H., Roberts, J. W. *et al.* (1987) *Molecular Biology of the Gene*, Vols. I & II. Benjamin Cummings.

Westhof, E., Dumas, P. and Moras, D. (1988) Restrained refinement of two crystalline forms of yeast Aspartic acid and Phenylalanine transfer RNA crystals. *Acta Crystallogr., Sect.A.* **44**: 112.

Westhof, E. and Sundaralingam, M. (1986) Restrained refinement of the monoclinic form of yeast Phenylalanine transfer RNA. temperature factors and dynamics, coordinated waters, and base-pair propeller twist angles. *Biochemistry* **25**: 4868.