A Constraint Based Structure Description Language for Biosequences

Ingvar Eidhammer
Department of Informatics
University of Bergen
HIB
N-5020 Bergen Norway
email: ingvar@ii.uib.no

David Gilbert
Department of Computer Science
Northampton Square,
London EC1V 0HB,
United Kingdom,
email: drg@cs.city.ac.uk

Inge Jonassen
Department of Informatics
University of Bergen
HIB
N-5020 Bergen Norway
email: inge@ii.uib.no

Madu Ratnayake
Department of Computer Science
Northampton Square,
London EC1V 0HB,
United Kingdom,
email: drg@cs.city.ac.uk

Svenn Helge Grindhaug
Department of Informatics
University of Bergen
HIB
N-5020 Bergen Norway
email: svenn@ii.uib.no

Abstract

We report an investigation into how constraint solving techniques can be used to search for patterns in sequences (or strings) of symbols over a finite alphabet. We define a constraint-based structure description language for biosequences, and give the definition of an algorithm to solve the structure searching problem as a CSP. The methodology which we have developed is able to describe two-dimensional structure of biosequences, such as tandem repeats, stem loops, palindromes and pseudo-knots. We also report on an implementation of the language in the constraint logic programming language clp(FD), with test results of a simple searching algorithm, and results from a preliminary implementation in C++ using consistency checking techniques from solving CSP.

Keywords: constraints, biostructures, description language, searching.

1 Introduction

The aim of the work described in this paper is to use constraint solving techniques to search for structural patterns in sequences (or strings) of symbols over finite alphabets. The main

motivation is searching in biological sequences, and also in providing high-level descriptions of biosequence database contents, but we believe that programs for searching for such patterns also can be useful in other areas as well, e.g. signal processing or analysis of acoustics data.

We define a pattern as consisting of a logical expression on components and a set of unary and binary constraints on the components, where a component is a description of a string of symbols. An input string S matches a pattern if every component in the pattern is matched by some substring of S, such that all the constraints are satisfied.

A pattern can contain constraints on the

(1) *length* of a substring to match a specific component;

(2) *distance* (in the input string) between substrings to match the different components of a pattern;

(3) *contents* of a substring to match a component, e.g. the second symbol should be an a or a t;

(4) *positions* on the input string where a particular component can match;

(5) *correlation* between two substrings matching different components, e.g. the substrings should be identical, or the reverse of each other.

We also define two associated classes of patterns:

- *Sequential*: patterns which do not include a correlation constraint. The patterns in the PROSITE data base [4] are examples of this class; for example [AC]-x(2,3)-D describing a pattern comprising three components, the first being an A or a C, the second of length 2 or 3 and the last being a D.
- *Structural*: patterns having at least one correlation constraint, for example repetitions or palindromes.

In terms of formal languages, the expressive power of sequential patterns is within that of the regular languages (not including Kleene closure). Structural patterns may describe context-free languages or even languages beyond the expressive power of context-free grammars, although there may be some languages in these classes which they cannot describe. From the point of view of bioinformatics, sequential patterns can thus describe sequences, whereas structural patterns can describe "folded" strings, i.e. RNA structures such as stem-loops and pseudo-knots, and some topological descriptions of protein structures.

In the research reported in this paper we investigate structural patterns, but put restrictions on the allowed constraints:

- The length, position and distance constraints must be specified by intervals, and we represent these using finite domains over integers.
- The content constraints use sets, where a set specifies which symbols can be in a constrained position since the alphabet is finite but unordered.

- The correlations (relations) are binary, and are between components for which the matching substrings must be of equal length, since these relations are recursively applied to character pairs, one member of each pair from each substring.

Structure of the paper

We give some biological motivation in Section 2, review other pattern languages for biosequences and describe the use of constraint satisfaction in molecular biology in Section 3, and describe the relationship between CSP and CLP in Section 4. We then define a language to specify structural patterns in Section 5, and describe the general requirements for a system based on the language in Section 6. In Section 7 we describe an implementation in constraint logic and in Section 8 we describe a method for solving the structure searching problem using techniques from solving Constraint Satisfaction Problems. Testing of the CSP and CLP implementations is reported in Section 9, and Section 10 is a summary with pointers to further work.

2 Biological motivation

Biological macromolecules, DNA's, RNA's, and proteins, are chains of relatively small organic molecules. The different types of these organic molecules are few – there are 4 different bases for DNA's and RNA's and 20 different amino-acids for proteins. Thus a macromolecule can be coded as a string over an alphabet of size 4 (for DNA/RNA), or 20 (for proteins) starting from one end of the chain and moving towards the other –the direction is always important. The strings for DNA/RNA molecules are called *nucleotide sequences*, and each element in such a sequence is called a *base* (or *nucleotide*). The strings for protein molecules are called *protein sequences*, and each element in such a sequence is an *amino-acid* (*residue*). Collectively nucleotide and protein sequences are called *bio-sequences*, or just *sequences*. Sometimes we will also refer to them simply as strings.

Watson and Crick discovered in 1953 that DNA forms a double helix where a base in one strand is bonded to a *complementary* base in the other strand (chain), and the so-called Watson-Crick base pairs are a-t and g-c. The bases in RNA molecules can form bonds a-u, g-c in a similar way, and can in addition form weaker g-u base pairs. RNA and protein molecules fold into 3 dimensional structures enabling them to perform their functional role in the cell. The structures can be described at different levels. For RNA molecules, the secondary structure is the collection of base pairs which are formed in the folded molecule, and the tertiary structure is the complete 3 dimensional structure of the folded molecule. For proteins, the secondary structure is a description of which parts of the amino acid chain folds into alpha-helices and beta-sheets, which are the stable local conformations most often found in the core of the protein, and the tertiary structure is defined as for RNA molecules.

An important problem in molecular biology is the prediction of the biological properties of a macromolecule from its sequence, in particular the prediction of the structure and the function of an RNA molecule or a protein from its sequence. Proteins may be grouped into families

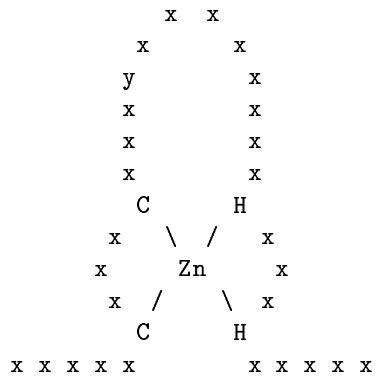

Figure 1: Schematic figure of the zinc finger c2h2 motif (accession number PS00028 in PROSITE). For this motif the sequential pattern C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H has been defined. (In the figure, x represents a position with any amino acid, and y a position with a more limited set of alternatives.)

where the members of a family have similar structures. If the structure of one family member is known, this helps in finding the structure of the other proteins in the family. Features that are common to the sequences of the proteins in a family can be expressed in a pattern, and a new sequence can be hypothesised to belong to the family if it fits the pattern. Sequential patterns give sufficient expressive power to describe sequence features that are characteristic for many protein families, and most languages used to define patterns for protein sequences permit only the definition of this type of pattern. This is illustrated by the PROSITE protein family database which gives descriptive patterns for most of its families [4]; Figure 1 is an example of a pattern from this database.

In order to describe patterns in RNA sequences, it is desirable to include dependencies between individual letters because the base-pairing interactions (most importantly a-u, g-c and g-u) play a dominant role in determining RNA structure and function [48]. Figures 2 (i) and (ii) shows two stem-loops, or hairpin loops, (defined later) that might be structurally and functionally equivalent in RNA molecules. It does not matter which bases (symbols) are present in the sequence in order for a stem-loop to be formed, as long as the sequence contains two substrings of some minimum and identical lengths and which are *reverse complement* of each other, i.e. one substring is equal to the result of reversing and complementing the other. We will call such patterns of dependencies *structures* in the sequences, and note that such patterns can be described using structural patterns as defined in the Introduction. We can also describe other structures found in RNA and DNA molecules such as clover-leafs and pseudo-knots (Figure 3).

A traditional approach to predicting the secondary structure of an RNA molecule is to find a set of base pairings that minimises the free energy. For example a popular program developed by Zuker uses a dynamic programming algorithm which finds optimal as well as close to optimal secondary structures [61], [60]. The algorithm has time complexity $O(l^3)$ and space complexity $O(l^2)$, where l is the sequence length, and relies on some simplifications, for example that no pseudo-knots are present. Pseudo-knots languages belong to the class of context-sensitive languages and can be parsed by a linear bounded automaton, and the

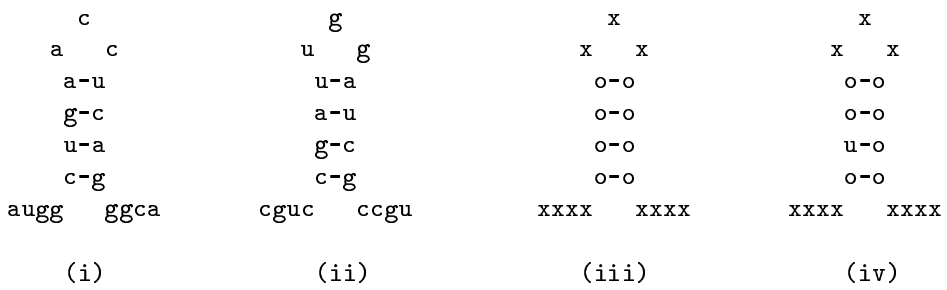

Figure 2: Illustration of structures and structural patterns: `(i)` and `(ii)` show two examples of structures (stem loops) in RNA molecules that have equivalent shapes. Watson-Crick base pairing is between `a` and `u`, and between `g` and `c`. Other base pairings are also possible. Figure `(iii)` shows a possible representation of a structural pattern matching the structures `(i)` and `(ii)`. The pattern can also be called a *consensus* for the structures in `(i)` and `(ii)`. The `o` and `x` symbols each match any one nucleotide symbol, and pairs of `o` symbols which are connected with a dash (-) should match pairs of symbols that can base pair. The `x` symbols are wildcards - one `x` matches any one symbol. Figure (iv) shows a structural pattern equivalent to the pattern shown in `(iii)` except that the second nucleotide in the first part of the stem has to be a `u` – a restriction on the content of the substrings to match the pattern; it matches (i) and not (ii).

number of possible derivations is exponentially large. Rivas and Eddy [47] have developed a dynamic programming algorithm for predicting optimal RNA secondary structure, including pseudoknots, with a worst case complexity of $O(l^6)$ in time and $O(l^4)$ in storage. They have demonstrated a one-to-one correspondence between the algorithm and a formal transformational grammar [46]. An implementation of the algorithm that generates the optimal minimum energy structure for a single RNA sequence uses standard RNA folding thermodynamic parameters augmented by a few parameters describing the thermodynamic stability of pseudoknots. Finding a match a structural pattern in an RNA sequence does not imply that the corresponding molecule in its native folded state will have the base pairing described by the pattern. It is believed that the native structure will be one with minimum free energy, and another set of base pairings other than the one described by the pattern, might give a lower free energy.

Structural patterns should not be used alone to predict the secondary structure of RNA, but are preferably employed in conjunction with structure prediction methods in order to provide hypothesises of possible folds. This can be done efficiently because matching a string against a structural pattern is computationally cheap compared to structure prediction. Another advantage of using structural patterns, is that they can be used to describe complex structures which are not allowed when using dynamic programming based structure prediction. Although many algorithms for finding conserved sequential patterns in sets of protein sequences exist [9], less work has been done to find conserved structural patterns in a set of RNA sequences [32], [31], and we will investigate this in further work. In this way structural patterns allow for the description of potentially interesting conserved structures in sets of related biosequences.

Structures are also found in DNA sequences that can be described using structural patterns but not using sequential patterns. This includes structures such as repeats and palindromes. Repeats are abundant in genomic DNA, both in coding and in non-coding areas, and for instance recognition sites for restriction enzymes are often palindromes.

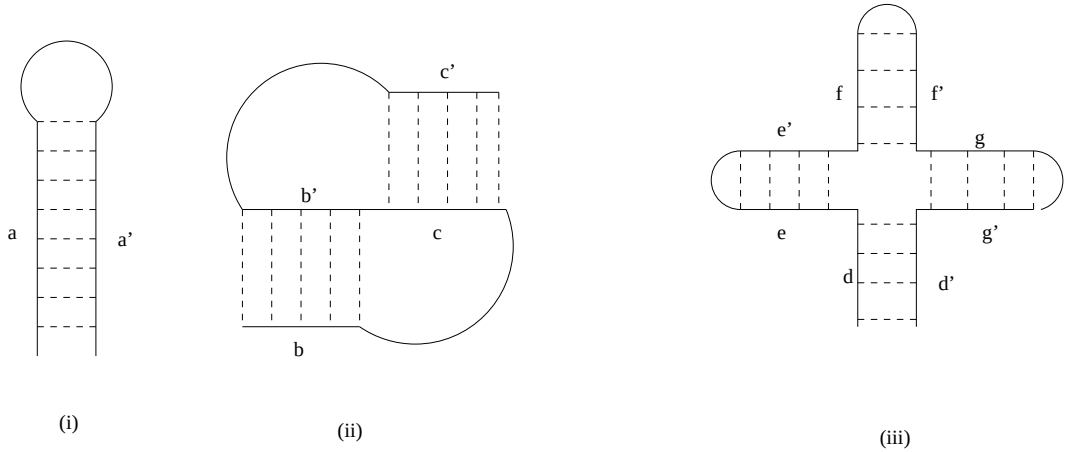

Figure 3: RNA structures: (i) Stem loop, (ii) Pseudo knot, (iii) Clover leaf. Base complement pairing indicated by dashed lines; reverse complement strings labelled by a, a' etc.

Stochastic context–free grammars (SCFG) [48] can describe the common primary and secondary structure of a set of sequences by enriching context-free grammars (CFG) with measures of probabilities applied to the production rules of the CFG. SCFG's can be used to implement Nussinov's dynamic programming algorithm for folding [41] as well as Zuker's energy-minimalisation approach [61], [59]. The probabilities in such grammars are generated by training on a set of sequences, and are then used to decide with what probability a given (new) sequence conforms to the grammar. Covariance models (CM's) are the SCFG analogue of profile Hidden Markov Models (HMMs); they specify a repetitive tree-like SCFG architecture suited for modelling consensus RNA structures whereas profile HMMs describe a repetitive linear HMM architecture well suited for modelling multiple sequence alignments [17], [5].

2.1 Example structures

We have identified the following structures in the literature, see for example [50, 6]. For each type we give one example. All examples are from DNA/RNA sequences, and strings corresponding to pattern components are underlined. In the structure descriptions α, β (with or without indices) are pattern components and x is a wildcard (matching any one letter in an input string), α^r is the reverse of α, and α^c is the complement of α. α^{rc} is the reverse complement of α.

- Tandem repeat $\alpha\alpha$ acg acg
- Simple repeat $\alpha\beta\alpha$ acg aaa acg
- Multiple repeat $\alpha\beta\alpha\beta_1\alpha$ acg aa acg uu acg
- Stem loop $\alpha\beta\alpha^{rc}$ acg aa cgu
- Attenuator $\alpha\beta\alpha^{rc}\beta_1\alpha$ acg aa cgu au acg
- Palindrome, even $\alpha\alpha^r$ acg gca
- Palindrome, odd $\alpha x\alpha^r$ acgagca
- Pseudoknot $\alpha_1\beta\alpha_2\beta_1\alpha_1^{rc}\beta_2\alpha_2^{rc}$ acg aa ucu gc cgu aua aga

More complicated structures can be obtained by combining the ones above, e.g. clover-leafs.

3 Other approaches

Several programs have been developed for searching sequences for the presence of patterns. We describe here their essential characteristics and refer to [14] for a description of a procedure to perform searches for patterns (or motifs) in RNA sequences using such programs.

The programs can be divided into two types: special and general purpose programs. The special programs are designed to search for specific patterns, e.g. candidates for trans-splicing sites [15]. Several of them use the minimal free energy principle, and stability measures. We concentrate here on general purpose programs.

3.1 General purpose search programs

The principle for most of the general purpose programs is that they include a language in which the user specifies the sequential and structural components of the pattern to be searched for. No explicit energy or stability aspects are taken into consideration, but some of the methods use structure predictions and biochemical properties. Several allow for mismatches and insertion of gaps, and have different ways for penalising these.

Some of the best known programs or languages are:

QUEST [1] and **ANREP** [39] can only search for sequential patterns.

PROSITE [3] accepts patterns described in a declarative notation, but only sequential patterns can be given. The expressive power of its specification language lies within the class of regular languages.

Staden's program [55] is the first system that we are aware of in which one could search for structural patterns, although in a restricted way. A pattern is defined to as comprise *motifs* and is built up and searched for by interactively specifying new motifs, giving the *class* to which a motif belongs. Nine classes are defined, of which two include structures: inverted repeat or stem-loop and (direct) repeat. Logical operators AND, OR and NOT can be used to specify whether each motif must be present, is an alternative to another, or must be absent.

Constraints can be specified on the length of a motif, the distance between two motifs and the contents of a motif; for the structure classes, constraints can be given on an individual part of the structure, e.g. on the loop of a stem-loop. Percentage match and scoring matrices can be used in searching. In Staden's system there is no possibility to define *general* correlations or relations between parts. The only relations are those which are included in the predefined classes.

SCRUTINEER [53] is an interactive program designed to search for patterns in protein sequence databases. It includes the use of structure prediction and biochemical properties. The user can give constraints on the length, contents and distances between parts of a pat-

tern, and where in a database sequence a specific part must match. A very limited form of dependency constraints can be given, e.g. if position 4 is a small hydrophobic, then position 2 must be a `G`.

RNAmot [23] is a pattern-matching program by Gautheret et al. using primary and higher-order structural features based on a backtracking algorithm. The user can give constraints on the length of a part of a pattern, and the number of mismatches permitted on both parts and the whole of the pattern. The language can describe of helical regions using reverse-complement correlations which are restricted to pairs of pattern components. Constraints can be placed on the character contents of the pattern including exact instances of bases, class descriptions (purine or pyrimidine), wild cards and combinations of bases following the IUPAC-IUB notation ('not C', 'G' or 'U' etc). Character constraints can be neccessary or optional, the latter being used to establish a score for sequence matches. Matches are weighted using a combination of best sequence match, maximum helix length and minimum helix free energy. The weighted pattern-matching algorithm is used to compute the multiple alignment of a list of of homologous or highly similar sequences a user-provided pattern. The program was improved [33] by using a faster string matching algorithm and more efficient sequence scans. RNAbob [18] is a fast implementation of RNAmot, but with a different underlying algorithm using a nondeterministic finite state machine with node rewriting rules.

OVERSEER [54] is a program for searching in nucleic acids sequences. It is much like the system of Staden, in that the pattern (or *target*) is defined interactively using specific sub-targets (nine types, all sought by different algorithms). Only the logical operator AND can be used between sub-targets, and two structural sub-targets are defined, repeats and palindromes. Constraints can be given on the lengths, contents and distances between sub-targets, and where on the sequences the search should be done.

Correlation constraints can be given between two positions (by using boolean matrices), and between two substrings of equal length. The search can allow for mismatches.

PALM [27] is a powerful language in which general dependencies between parts (substrings) can be specified, and hence complicated structural patterns can be defined. Patterns are described in a declarative way, and PALM extends the notation used in the PROSITE motif database. PALM is capable of describing any context free language, and any language generated by a string variable grammar. Approximate matching can be specified. By allowing general procedures to be attached to and called from within a pattern, PALM can also recognise patterns describing any language in the Chomsky hierarchy. However, PALM has only been implemented as a prototype in Prolog.

GENLANG (Searls [52, 51]) is the most general system which has been implemented for searching for structural patterns in nucleotide sequences. It is based on formal language theory, and uses an indexed language which has an expressive power between context-free and context-sensitive languages. TAG (tree-adjoining grammar) languages are used to describe biologically relevant languages which are beyond context-free. Essentially, secondary structures are derived from the frontiers of the parse-trees that this approach generates. GENLANG is implemented in Prolog, with hooks to C-code for the efficient caching of data that is required during parsing. *String variables* are used to define structures; reverse complementarity is denoted by the operator $\sim$ and thus a pseudo-knot is specified by $X, \cdots, Y, \cdots, \sim X, \cdots, \sim Y$.

Constraints on the length and contents of the string variables can be specified.

PALINGOL [7] is a constraint programming language whose data types and search engines have been adapted for secondary structures in RNA, and is implemented directly in C; constraints are boolean expressions. The elementary object manipulated by Palingol is called a helix, comprising three physical elements – 'head', 'loop' and 'tail'; thus a helix is like a stem loop. Physical elements are like string variables in Searl's formalism. The Palingol system comprises two parts, a helix search program and a constraint checking program. Constraints can be local to a helix, for example on the length of the constituent elements, or their contents, and also correlations between them. Correlations can be weighted with a score, for example g-c=2, a-u=2, g-u=1. In addition global constraints can be expressed, for example on the distances between the start points of a pair of helices.

cBLISS [45] is an implementation in the constraint logic programming language Eclipse of the language of Brāzma and Gilbert [8] for describing constrained patterns in biosequences. This language is a formalisation and development of Staden's pattern language [55]. Brāzma and Gilbert follow the notations used by Staden, and consider a pattern to comprise *motifs* as the basic elements. A motif may be a simple string, $\alpha \in \Sigma^*$ for some alphabet Σ, or a more complex expression in some grammar. Motifs can be combined in a logical manner using AND, OR and NOT, and constraints can be given on the length, contents and distances between two motifs. As with Staden's language, there is no possibility to define dependencies between motifs, hence the structural possibilities lies within each motif. Although the language has not been designed for the description of structures, the language can easily be extended for this purpose by the addition of dependency constraints.

3.2 Use of constraint satisfaction in molecular biology

Constraint based solving was used early in *map construction* [56]; several types of constraints were used to prune the search space during the search. A method for genetic map construction is described in [35], where a number of constraints are defined, and constraint propagation are used to determine inconsistencies. Clark et. al. [11] have used ElipSys to develop a program for generating a physical genetic map from hybridisation fingerprinting data. ElipSys is a parallel CLP language which includes constraint handling on finite domains.

In the MC-SYM program [37, 20] the RNA (tertiary) structure prediction problem is formulated as a CSP. The set of variables is the set of nucleotides corresponding to an RNA sequence, and a domain is the set of Cartesian products of various permitted nucleotide conformations and 3-D transformational matrices. Gaspin and Westhof [22] have done the same for secondary structure prediction. Each base in the sequence is associated with a variable, the domain of a variable being the set of positions of other bases with which it can pair. The constraints comes from known restrictions on valid secondary structures, and are unary or binary.

CBS2E [12] is a program that predicts protein α/β-sheets and β-sheet topologies. Variables represent different attributes associated with β-sheet, β-strands, α-helices etc. Domains are values associated with those variables, and the constraints are known protein folding constraints. The program is written in ElipSys; in [42] an earlier version of this program [13] is

combined with the explicit representation of uncertainty rules.

AUTOASSIGN [58] is a program which uses CSP-solving techniques to help in the determination of protein structure from NMR. A similar task is performed by PROTEAN [2] and TAM [34], the latter being implemented in CHIP (Constraint Handling In Prolog), [28].

We refer the interested reader to [10] which contains an introduction to constraint satisfaction in molecular biology and a more detailed explanation of some of the programs mentioned above.

4 CLP and CSP

One way of constructing a constraint logic language is to extend an existing logic language with techniques from solving CSP (for finite domains). This is the case for CHIP and ECLiPSe, which extend Prolog [28, 21]. Solving constraint problems might be looked upon as searching for a valuation which is a solution. The searching is done by pruning the search space. As Prolog uses standard backtracking, only *a posterio* pruning is done (after the discovery of a failure). Extending the language with *k-consistency* checking implies *a priori* pruning of the search space, thus reducing the search space before failure.

The k-consistency checking method (and other CSP methods) involves more work at each node of the search tree than for Prolog or other logic programming languages which only compute over Herbrand terms. However the size of the tree to be searched is reduced, and hence there is a trade-off between amount of work at each node, and number of nodes visited. In practical terms, the introduction of constraint solving reduces backtracking – the possible improvement in the efficiency of the computation depends on the efficiency of the constraint solver.

The resolution step can be explained as follow: Let P be a program over the finite domains $d_1, \ldots, d_r$, and $\leftarrow A_1, ..., A_k, ..., A_m$ a goal. Further, let $x_1, ..., x_r$ be the arguments of A_k which are domain variables (i.e. taking values from one of the finite domains), the other arguments being ground. For each $i \in r$ define a set $e_i \subset d_i$ with $y \in e_i$ if it is found, by k-consistency checking, such that no solutions include the assignment $x_i = y$. Define $f_i = d_i - e_i$, and a new domain variable z_i with domain f_i. Then the new (derived) goal becomes $\leftarrow (A_1, ..., A_k, ..., A_m)\{x_1/z_1, ..., x_r/z_r\}$, where $\{x_1/z_1, ..., x_r/z_r\}$ is a substitution. If all z_i become ground, then the new goal is $\leftarrow (A_1, ..., A_{k-1}, A_{k+1}, ..., A_m)\{x_1/z_1, ..., x_r/z_r\}$.

5 The structure language

We base our structure language on a modified version of Brāzma and Gilberts' pattern language [8] but keep separate the information about the logical composition of the components from the set of constraints over these components. Since substrings of the input string have to match the components, we refer to the components as *string variables*, in a similar fashion to Searls [51], and denote them by the Greek letters $\alpha, \beta, \gamma, \ldots$ (possibly subscripted).

A pattern is defined by a *structure specification* of the form (S, C) where S is a *string expression* and C a set of constraints. The string expression S specifies the string variables taking part in the pattern, and a logical expression on them using conjunction and disjunction. We also define a *normal form* of structure specifications where S contains a disjunction is rewritten as a disjunction $(S_1, C_1) \vee \ldots \vee (S_n, C_n)$ where $S_1 \ldots S_n$ have been derived from S by the usual equivalences of first order logic and each contain only conjunctions. Each $C_i, i \in 1 \ldots n$ contains constraints solely over S_i.

5.1 Constraint types

A set of constraints can contain constraints over the five types: *length*, *distance*, *content*, *position* and *correlation* constraints, which are described below. In addition, we permit equality, inequality and arithmetic operations over the integer components of the constraints. We also permit the user to describe complex structures by conjoining structure descriptions.

Length constraints restrict the length of string variables to be within a particular range. They are of the form length(α, L) where α is a string variable and L ranges over the positive integers such that the length of α is constrained to be within the range of L. We permit the length of a string variable to be 0 in order to be able to describe null-strings.

We can also constrain the length of a string variable α to be the maximum value within some range L, using the constraint maxlength(α, L). This constraint is required when searching for structures containing some component of a maximal length - an example is a stem-loop, where we may wish to ensure that the stem region identified is the "largest" possible. Thus in Figure 2(i) the stem region `a-u, g-c, u-a, c-g` cannot be extended further since final character `g` in the prefix `augg` does not match the initial character `g` in the suffix `ggcau`. Likewise we can constrain the length of a string variable to the minimum of a range using minlength(α, L).

Distance constraints restrict the distance between two string variables. They are of the form dist(α, β, D) where D is the distance between the start positions of each of the two string variables α and β. D ranges over the positive and negative integers; a negative value indicates that the start of α occurs after that of β in the input string. We also permit the shorthand $\alpha\beta$ to indicate that β starts directly after α. This shorthand is equivalent to $\alpha \wedge \beta$, length(α,L), distance(α, β, L+1) .

Content constraints restrict which symbols can be at specific positions in a string variable matching a component. They are expressed as: content(α,Pos,Set) where α is a string variable. Pos is a positive or negative (non-zero) integer representing α_{Pos}, the character from α at position Pos from the start (or end if Pos is negative) of α. Set is a (non-empty) set of characters from Σ to which α_{Pos} may be bound, e.g. {a,t}.

Position constraints restrict the absolute positions of a string variable on the input string and are expressed as start(α,P) or end(α,P). P ranges over the positive integers such that the first (respectively last) character of the string variable α is located at position P on the input string.

Character correlation constraints describe constraints that can be placed on characters and are symmetric relations over some alphabet Σ. They are normally given in set notation c= $\{\phi(\sigma,\tau) \mid \sigma,\tau \in \Sigma\}$ where ϕ is the symbol for the constraint. For example, we define the reverse character complement for RNA, where Σ = {a,c,g,u} by {a-u, c-g, g-u} where - is the infix symbol for the reverse complement. Character equality for RNA is defined by {a=a, c=c, g=g, u=u}.

String correlation constraints define the relation between the contents of two string variables. They take the form C(Dir, α, β, c) where C is the name of the string correlation, Dir is a direction whose value can be 1 or -1, α and β are string variables, and c is a character constraint. A string correlation C over two string variables α and β each with the same length h is satisfied iff Dir=1$\Rightarrow$ $(\forall i : 1 \leq i \leq h : \mathsf{c}(\alpha_i, \beta_i)$ or D=-1$\Rightarrow$ $(\forall i : 1 \leq i \leq h : \mathsf{c}(\alpha_i, \beta_{h-i+1})$. An example string correlation constraint is `rev_comp_RNA(-1,`α`,`β`,{a-u, c-g, g-u})`.

We define id(α,β), and reverse(α,β) as general string correlation constraints over all alphabets, where β is the identity (respectively, reverse) of α, and also not_id(α,β) where α and β differ in at least one character position.

Furthermore, we define a notion of approximate matching, given as an argument to the appropriate correlation constraints. This argument ranges over the interval 0..100 and represents the percentage mismatch allowed between two string variables; when the mismatch is zero then this argument can be omitted. We can use Hamming distance [26], edit distance or more generally Levenshtein distance [36] in order to implement approximate matching[1].

The definitions of reverse with approximate matching is
$\forall\alpha\forall\beta\forall M$ (reverse(α, β, M) $\leftrightarrow$ $\exists\gamma$ (reverse(α, γ) $\wedge$approximate_match(γ, β, M)))
complement and reverse_complement are defined in a similar manner.

Weighted matches in order to compare correlations, and thus to be able to rank structures found, we define scoring over string correlations. The form is score(Corr,α,β,Score) where the value of the score is the sum of all the weights of correlated character pairs respecting the correlation Corr. For example, as with Palingol [7], we use a table of character complements and associated weights for RNA as (g,c,3), (a,u,2) and (g,u,1). A more sophisticated form is to weight by positions, following Zuker [60] and McCaskill [38].

Strings We write quoted strings in place of string variables where appropriate in constraints, thus id(α, "ctg") stands for length(α,3), content(α,1,{c}), content(α,2,{t}), content(α,2,{g}).

[1] Minimum transformation costs calculated for: Hamming distance: substitution only, edit distance: insertion and deletion only, Levenshtein distance: substitution, deletion and insertion.

In a similar way we can express the constraint that a string variable is not equal to a particular string: not_id(β,"tata").

5.2 User-defined grammars and constraint libraries

We further define a macro language permitting the user to store and re-use definitions of, i.e. grammars for, specific structures, as well as correlations. A general stem loop can be defined by
stem_loop α,β :: $\alpha\gamma\beta$, rev_compl_RNA(α, β).

The specific stem loop in Figure 2(i) is described by
stem_loop α,β :: $\alpha\gamma\beta$, length(α, 4), length(γ, 3), content(γ, 2, {c}), rev_compl_RNA(α, β)
assuming a library definition of rev_compl_RNA, and where α and β form the stem, with γ the loop.

The definition of a general pseudoknot is
pseudo_knot α,β,γ,δ :: $\alpha\tau_1\beta\tau_2\gamma\tau_3\delta$, rev_compl_RNA($\alpha$, γ), rev_compl_RNA(β,δ)
where τ_1, τ_2 and τ_3 are "spacer" string variables of any length.

5.3 Semantics

As noted earlier, the grammars permitted by our language are up to context sensitive, since we can describe copy languages (e.g. for pseudo-knots) as well as palindromes (the basis of stemloop languages). Parsers for context-free languages are $O(n^3)$ or better, and can be linear-time for many applications. However, parsers for context-sensitive languages are very complicated, and very inefficient. Searls [49] has proposed that nucleic acids can be described by languages in between context-free and context-sensitive, since structures like pseudo-knots have finite bounds on their length, and has developed his GENLANG system in order to operate up to this zone of complexity. We also presume the size limits on such structures in our implementation.

In general, we characterise the structures created during our matching as "string-graphs", that is to say strings with relations (constraints) over some components. In general processing of these structures can be quite efficient if these constraints are determined in advance of the parse, as is the case for example with some representations of protein structures where we have obtained good efficiencies by pruning the match on these precompiled constraints [25]. In the case of our present language we do not do this precompilation, and thus have to compute the constraints "on the fly" making for more inefficiency.

6 Searching for structures

In order to search in sequences for structures which have been described in the language, a pattern matching system is required. Thus, given the following sequence of bases from an RNA sequence database:

auggcugaaccucagggca a search for a stemloop of the form in Figure 2(i):
stem_loop α,β :: $\alpha\gamma\beta$, length(α, 4), length(γ, 3), content(γ, 2, {c}), rev_compl_RNA(α, β)
should result in α being mapped to the substring cuga starting at position 5 and ending at 8, β to ucag starting at 12 and ending at 15, and γ to acc starting at position 10.

In general, there may be more than one way in which a given pattern and string match, and it may be desirable to rank them according to some measure, for example the scoring based on correlation weights described in Section 5.1.

In the following sections we describe two approaches to the design and construction of pattern matching algorithms for the language. The first is a simple inefficient algorithm using integer constraints, implemented in a constraint logic programming language over finite (integer) domains using the built-in integer constraint solver. The second is a more sophisticated CSP algorithm implemented in C++.

In general any usable system should provide an environment with a user interface which permits the user to enter descriptions of the structures on which to search, or to use definitions from libraries, and to give the string in which to search. The matching program then maps the description to a given input string and then returns the results of the mapping. Queries are handled by a query evaluator, which checks the syntax of the queries, expands macros, and stores any macro definitions made by the user, and translates the queries into an internal form. This form is passed down to a constraint engine which sets up the data structures, imposes the constraints on them and uses a matching algorithm to solve the constraints. Results of matching are displayed as the locations of strings and the strings themselves.

7 Implementation in CLP

7.1 String variables and string expressions

We represent the string variables by a data structure which is a sequence of *string-characters* with maximum length m. These string-characters comprise pairs whose first element *Chars* is a set of characters drawn from some alphabet Σ (of bases or nucleotides) and whose second element *Pos* is a set of integers in $1 \ldots m$. We omit brackets for singleton sets where no confusion arises. The first element of each pair represents the possible values of the characters to be found on the input string at the locations indicated by the second element. Moreover, we assume that the successor relation holds between the second elements of neighbouring members of the sequence, in the normally accepted direction of ordering.

Thus a string-variable $S = c_1 \ldots c_n$ where $c_i = (s_i, p_i)$ $(i \in 1 \ldots n)$, and initially $s_i = \Sigma$ and $p_i > 0, p_{i+1} = p_i + 1$. Our matching algorithm first of all sets up a correspondence between the string characters by imposing constraints over the character variables, and then generates values for the character and position variables.

Thus for example the stemloop $\alpha\gamma\beta$, `length(`α`,4), length(`γ`,3), content(`γ`,2,{c}), rev_compl_RNA(`α`,`β`)`
would be represented by

$(c_1, p_1), (c_2, p_2), (c_3, p_3), (c_4, p_4), (c_5, p_5), (c_6, p_6), (c_7, p_7), (c_8, p_8)(c_9, p_9), (c_{11}, p_{11})$
where $c_1 = \{\texttt{a,c,g,u}\}, \ldots, c_{11} = \{\texttt{a,c,g,u}\}$, $p_1 > 0$, $p_2 = p_1 + 1, \ldots, p_{11} = p_{10} + 1$, char_comp($c_1$,$c_{11}$), ..., char_comp($c_4$,$c_8$), $c_6 = \texttt{c}$, where char_comp defines character complement over the RNA alphabet.

We have chosen constraint logic programming over finite domains [30] as a paradigm for implementation because of the declarative nature of our structure language and the use which it makes of finite domain constraints. In our implementation sequences are represented as lists, and thus string variables comprise lists whose elements are pairs of (Chars,Pos). We choose also to map alphabets onto (dense subsets of) natural numbers, so that for example for DNA we represent a, c, g, t by 1, 2, 3 and 4 respectively. In this way we can use any finite constraint logic programming language which does not permit operations over arbitrary finite domains. We have used clp(FD) [16] as the basis for our implementation since it is is freely available, small in size and can compile to executable code. Ideally we would also like to be able to use a string solver, along the lines of [57], [24] or [44].

7.2 Constraints

Length constraints are defined in the usual backtracking manner for lists although ideally we would like to use a list solver (for example [44]). *Distance* constraints are defined simply by referring to the position elements of character pairs. *Content* constraints are implemented by imposing constraints on the integer sets representing the characters using the sparse representation of finite domain variables in clp(FD) to describe non-continuous domains. *Position* constraints are straightforwardly implemented by constraining the position element of a string-character pair.

General correlation constraints (those independent of any alphabet) are coded in clp(FD) as follows.

- The id constraint constrains the corresponding *characters* in the string characters pairs to be equal. Note that the position elements in each corresponding pair are not constrained by this relation, since the string variables may be mapped to different places on the input string.
- The reverse constraint first of all reverses one of the string variables and then constrains it to be identical to the other string variable.

Approximate matching between string variables is implemented using Hamming distance and relating this to the length of the list representing the string variable.

Complementation constraints are implemented using integer constraint operations. Thus, for example RNA, whose alphabet a, c, g and u we represent by 1, 2, 3 and 4 respectively, has complements a-u, g-c and g-u which we can map as
char_comp_RNA(c_1,c_2) $\leftrightarrow (c_1 \in \{1,2,3,4\} \wedge c_2 \in \{1,2,3,4\} \wedge (c_1 + c_2) \in \{5,7\})$.
For example if $c_1 = 3$ then this relation will constrain $c_2 \in \{2,4\}$. This constraint is implemented in clp(FD) by the definition

```
char_comp_RNA(C1,C2) :- C1 in 1..4, C2 in 1..4, C1+C2 #= C, C in 5..5:7..7.
```

7.3 Mapping a specification to an input string

We have implemented a processor for our language using clp(FD), and have also produced a front-end which permits users to specify constraints on stemloops in an interactive fashion. The system then sets up the data structures for the components, and imposes the constraints given by the user.

The aim of a processor for our language is to match a structure description on to an input string, in order to determine the contents and locations of those substrings of the input string which match the components of the description. Thus a solution to a mapping of a string expression onto an input string is a valuation (an assignment to each constraint variable in the string expression of one value from the domain of the variable) such that all the constraints are satisfied. Each element of all string-character pairs must be a singleton set satisfying the constraints on that element; an empty set indicates a failure to produce a solution. In our problem domain we are interested in producing *all* the solutions (mappings) possible of a given string expression onto an input string.

An input string I comprises a sequence of characters drawn from some alphabet Σ (of bases or nucleotides); we limit the maximum length of any string to be less or equal to some maximum integer m. In order to perform mapping we first convert the input string into a string-variable, i.e. a list whose elements are pairs of (Chars,Pos).

We have defined a naive procedure to map a specification *Spec* (i.e. a *constrained* string expression SE) onto an input string I using backtracking, and give this for conjunctive string expressions. We assume two types of correlation: c (normal correlation) and r (reverse correlation), and a function $p_1 : x \times y \rightarrow x$.

Normal correlation c

for each pair of string variables (α, β) in SE correlated by correlation c **do**
    find members of I s.t. $\alpha_1 = I_j$ and $\beta_1 = I_k$
    $i := 1$
    while $c(p_1(\alpha_i), p_1(\beta_i))$ and $i \leq \text{length}(\alpha)$ **do**
        $i := i + 1$ and $j := j + 1$ and $k := k + 1$
        $\alpha_i = I_j$ and $\beta_i = I_k$
    end
end

Reverse correlation r

for each pair of string variables (α, β) in SE correlated by correlation r **do**
    find members of I s.t. $\alpha_1 = I_j$ and $\beta_l = I_k$ and set $i_1 := 1$, $i_2 := \text{length}(\beta)$

while $c(p_1(\alpha_{i_1}), p_1(\beta_{i_2}))$ and $i_1 \leq \text{length}(\alpha)$ **do**
$i_1 := i_1 + 1$ and $i_2 := i_2 - 1$ and $j := j + 1$ and $k := k - 1$
$\alpha_{i_1} = I_j$ and $\beta_{i_2} = I_k$
end
end

8 Solving the structure searching problem using consistency checking

We now describe a method for solving the structure searching problem using techniques from solving Constraint Satisfaction Problems. Only conjunction are permitted in the string expression, and only exact matching in the correlations. An initial implementation in C++ has been constructed, where the constraints are represented as objects. We first describe the method, then give a short example as help for understanding the method, and then present some results from the preliminary implementation.

The method, described in detail below, comprises the following four steps:

1. Represent the structure searching problem as a constraint problem,
2. Perform consistency checking to remove search alternatives,
3. Reformulate the problem,
4. Search for solutions.

8.1 Representation of the constraints

To make the constraint propagation clear we introduce *constraint* variables. Each constraint variable has a corresponding finite domain.

Two types of constraint variables are introduced:

- For each string variable α two *distance* constraint variables L_α, U_α, with domains subsets of $[1, n]$, where n is the length of the input string S. In a solution the start position and the end position of α in S are assigned to these variables, respectively.

- For each correlation C between α, β a set $\mathcal{X}^{(\alpha,\beta)}$ of *correlation* constraint variables $\{X_{L_\alpha}^{(\alpha,\beta)}, ..., X_{U_\alpha}^{(\alpha,\beta)}\}$, with domains subsets of $[1, n]$. At most one correlation between one pair of string variables is allowed for. Let $D_i^{(\alpha,\beta)}$ be the domain of $X_i^{(\alpha,\beta)}$, where i corresponds to position i on S. i ranges over the interval $[min(D_{L_\alpha}), max(D_{U_\alpha})]$, where D_{L_α} is the (current) domain of L_α. The correlation constraint variables are used to constrain the position of β in S when restrictions on the position of α are known, and vice versa. In a solution $X_{L_\alpha+j}^{(\alpha,\beta)} = L_\beta + j$

if $C_d = 1$, and $X^{(\alpha,\beta)}_{L_\alpha+j} = U_\beta - j$ if $C_d = -1$. Note that L_α and U_α are themselves constraint variables. Such use of constraint variables corresponds to the use in [22].

A string variable α in the constraint system thus has the following associated constraint variables: L_α, U_α and a set $\mathcal{X}^{(\alpha,\beta)}$ for each correlation where α is a source.

The *correlation* constraints are represented using a new constraint construction s_c (for Sequence Constraint). Let $\delta = \{\delta_i\}$ be an ordered set of variables, and l, u, h integers or integer variables. Then

$$s\_c(\delta, l, u, h) \equiv (\forall i : l < i \leq u : \delta_i = \delta_{i-1} + h)$$

A correlation C between α, β, is then represented as $s\_c(\mathcal{X}^{(\alpha,\beta)}, L_\alpha, U_\alpha, C_d)$. This constraint has some similarities with the cardinality operator of van Hentenryck [29], and the value constraint in [19].

8.2 Consistency checking and constraint propagation

The effect of *position* constraints on a string variable α is performed by reducing the domains of L_α and/or U_α. This means performing node-consistency checking, and the constraints need not be saved any longer. The same is the case for the *content* constraints. Let a content constraint be such that the i'th position of a string variable α must be one of the symbols in a set E. Then the domain of L_α is restricted to be a subset of the set $\{j | S_{j+i-1} \in E\}$. If α occurs as source in a correlation (α, β), then a content constraint on α implies constraints on either L_β or U_β, which of them depends on the specification of the constraint and the direction component. If α occurs as target, a content constraint implies in the same way a content constraint on the source.

The *distance* constraints are binary, and easily used in arc consistency checking, the same for the *length* constraints on single string variables. A constraint on the distance between two string variables includes four constraint variables. It can however, be looked upon as a binary constraint between two string variables, which might be cost effective.

The restriction of the domains are propagated, through performing arc-consistency checking, to other distance constraint variables.

A *correlation* is a constraint between two string variables, α and β. We define a **local solution** for such a correlation to be two substrings (of S) which satisfy all the constraints between and on α and β. The correlation (together with other constraints) constrain the solutions. If there exist bounds on the distance between α and β, these bounds can be used to find constraints on $X^{(\alpha,\beta)}_i$ (bounds on the difference $X^{(\alpha,\beta)}_i - i$). If for example $l \leq L_\beta - U_\alpha \leq L$ and $|\alpha| \leq R$ and $C_d = -1$ then $l \leq X_i - i \leq 2R - 2 + L$. Several such rules are developed. We can also use the minimum and maximum value of the distance constraint variables to find rules restricting the domains of some $X^{(\alpha,\beta)}_i$.

If we represent $D^{(\alpha,\beta)}_i$ as bit vectors, finding local solutions can be done by finding diagonals with succeeding ones in a matrix with $D^{(\alpha,\beta)}_i$ as the rows.

8.3 Reformulating the problem

The local solutions define new constraints between pairs of *string* variables, or (expressing it another way) between four distance constraint variables. Before searching for a global solution, we can perform consistency checking between individual local solutions. To formalise this we found it convenient to regard this operation as solving a binary CSP with constraints between the string variables:

- The variables are the string variables $(\alpha, \beta, ...)$, with the components L_α, U_α, L_β, $U_\beta, \ldots$
- The constraint between two variables is the distance constraint, and if there is a correlation between them, the local solutions found by the procedure above.
- The domains are the sets of consistent values for L_i and $U_i, i = \alpha, \beta, ...$

Arc-consistency checking can be performed, but note that arc-consistency is assured between string variables for which there is a correlation. As an example: let a structure specification contain the string variables $\alpha, \beta, \gamma, \delta, \eta, \phi$, and the correlations be $C_1(\alpha, \gamma), C_2(\gamma, \eta), C_3(\delta, \phi)$. At the beginning of this step arc-consistency is assured for those pairs. Arc-consistency to β is also assured from all the other string variables, but not in the other direction (from β). The reason for this is that the domains of L_β, U_β are not reduced by the procedure performing the correlation constraints.

When full arc-consistency is achieved, searching can start. This might be done using forward checking or looking ahead, or other types of searching with consistency checking.

8.4 The Procedure

The procedure is here formulated as an algorithm:

1. Define the distance constraint variables for all string variables (initialise the domains to $[1, n]$), and represent the constraints.

2. Use the position constraints and the end positions (1 and n) to reduce the domains of the distance constraint variables. Propagate to other distance constraint variables, using the represented constraints.

3. Use the content constraints:

for each content constraint on a string variable α **do**
    reduce the domain of L_α or U_α
    for each correlation to another string variable β **do**
        reduce the domain of L_β or U_β
    propagate to other distance constraint variables

4. Treat the correlation constraints:

for each correlation (α, β) **do**
define the variables $\{X_i^{(\alpha,\beta)}\}; i = min(D_{L_\alpha}), \ldots, max(D_{U_\alpha})$ with domains
$D_i^{(\alpha,\beta)} = [min(D_{L_\beta}), max(D_{U_\beta})]$
reduce the domains according to the rules found in Section 8.2
use the s_c-constraint and the distance constraint to find the local solutions.

5. Reformulate the problem, and perform consistency checking on the reformulated problem.

6. Perform searching for global solutions.

The local solutions are found individually in the procedure for each correlation. The efficiency may increase if local solutions which have already been found are used to guide the search for local solutions to other correlations, which are consistent with earlier found solutions. For example, if there are two correlations $C1(\alpha, \beta), C2(\gamma, \alpha)$, the local solutions found for α in the first correlation should be used to guide the search for consistent solutions in the second correlation.

8.5 Example

We give the following example as an illustration of the method. A structure is defined by four string variables $\{\alpha, \beta, \gamma, \delta\}$, where $3 \leq max\_length(\alpha) \leq 5$, $2 \leq end\_start(\alpha, \beta) \leq 4$, $1 \leq end\_start(\beta, \gamma) \leq 6$, $1 < end\_start(\gamma, \delta)$. For convenience we have defined $end\_start(\alpha, \beta) = 1 + distance(\alpha, \beta) - length(\alpha)$. The correlations are $id(\gamma, \alpha)$, $R\_C(\beta, \alpha)$, $id(\delta, \beta)$, where I is identity, and R_C is reverse complement. There is one content constraint – the next last symbol in β is t.

The input is the DNA string attagtacta tagctagcta actagcgcgc tata (n=34, spaces in the presentation after each 10). The following local solutions are found using obtained local solutions to restrict other local solutions, the solutions are specified by the start and end positions:

1. Correlation $(\alpha, \gamma), C_d = 1$

   (a) $\alpha = [3, 5]$ $\gamma = [11, 13]$
   (b) $\alpha = [3, 5]$ $\gamma = [15, 17]$
   (c) $\alpha = [11, 14]$ $\gamma = [23, 26]$ (subsolutions not given, e.g. $\alpha = [11, 13]; \gamma = [23, 25]$)

2. Correlation $(\alpha, \beta), C_d = -1$

   (a) $\alpha = [3, 5]$ $\beta = [8, 10]$
   (b) $\alpha = [11, 14]$ $\beta = [17, 20]$

3. Correlation $(\beta, \delta), C_d = 1$

(a) $\beta = [8, 10]$ $\delta = [14, 16]$
(b) $\beta = [8, 10]$ $\delta = [18, 20]$
(c) $\beta = [8, 10]$ $\delta = [22, 24]$
(d) $\beta = [8, 10]$ $\delta = [30, 32]$
(e) $\beta = [17, 20]$ $\delta = [29, 32]$

From this we see that the domains are:

$D_\alpha = \{[3, 5], [11, 14]\}$

$D_\beta = \{[8, 10], [17, 20]\}$

$D_\gamma = \{[11, 13], [15, 17], [23, 26]\}$

$D_\delta = \{[14, 16], [18, 20], [22, 24], [29, 32], [30, 32]\}$

The reformulated problem is now node-consistent, we then check for arc-consistency. We know that arc-consistency is satisfied for $(\alpha, \gamma), (\alpha, \beta), (\beta, \delta)$. The other pairs with explicit distance constraints are (β, γ) and (γ, δ). From this we find that $\delta = [14, 16]$ cannot be in any solution, on removing this the whole system becomes arc-consistent.

Searching is then performed, and if the variables are assigned values in the given order, all solutions are found without backtracking:

$\alpha = [3, 5]$ $\beta = [8, 10]$ $\gamma = [11, 13]$ $\delta = [18, 20], [22, 24], [30, 32]$
$\alpha = [3, 5]$ $\beta = [8, 10]$ $\gamma = [15, 17]$ $\delta = [22, 24], [30, 32]$
$\alpha = [11, 14]$ $\beta = [17, 20]$ $\gamma = [23, 26]$ $\delta = [29, 32]$

9 Testing the CSP and CLP implementations

A preliminary implementation has been made, without arc consistency checking in the reformulation step, and the final searching is pure backtracking. Testing has been done on a Sun Ultra 1.

9.1 The example in Section 8.5

The 6 solutions for the example in Section 8.5 is found in 0.02 seconds. To test the program on larger sequences, we enlarged the string with random nucleotides, achieving the results as shown in Table 1.

9.2 Sequence aaorf92a in EMBL

We ran the program searching for stem-loops in the DNA sequence AAORF92A, which is document to have a stem-loop, in the EMBL database (http://www.ebi.ac.uk/htbin/emblfetch?aaorf92a),

Nr	Sequence length	Nr. of solutions	CLP (sec) Total	CSP (sec) Consistency checking	CSP (sec) Search	CSP (sec) Total
1	34	6	0.02	0.01	0.01	0.02
2	68	23	0.05	0.02	0.04	0.06
3	136	27	0.1	0.08	0.12	0.20
4	272	63	0.22	0.39	0.53	0.92
5	544	160	0.67	2.44	3.48	5.92

Table 1: Number of solutions and running time for searching for the structure defined in Section 8.5

Nr	Content restrictions on pos. in α	Nr. of solutions	CLP (sec) Total	CSP (sec) Consistency checking	CSP (sec) Search	CSP (sec) Total
0	0	2	0.26			
1	1	2	0.16	0.26	0.26	0.52
2	1 and 2	2	0.12	0.27	0.28	0.55
3	1, 2 and 3	1	0.12	0.27	0.28	0.55
4	1, 2, 3 and 4	1	0.08	0.25	0.25	0.50
5	1, 2, 3, 4 and 5	1	0.1	0.18	0.18	0.36
6	1, 2, 3, 4, 5 and 6	1	0.1	0.14	0.15	0.29

Table 2: Number of solutions and running time for searching for the stem-loop in EMBL sequence AAORF92A

total length 582 characters.

The stem-loop was defined by two string variables, α, β, with length(α)=6, end_start(α, β)=25, R_C(α, β), and different number of content constraints, as shown in Table 2.

9.3 Pseudoknots in tobacco mosaic virus RNA

We also tested the program on tobacco mosaic virus RNA, the structure is described in [43]. In the positions 106-179 it contains three pseudoknots, as shown in Figure 4.

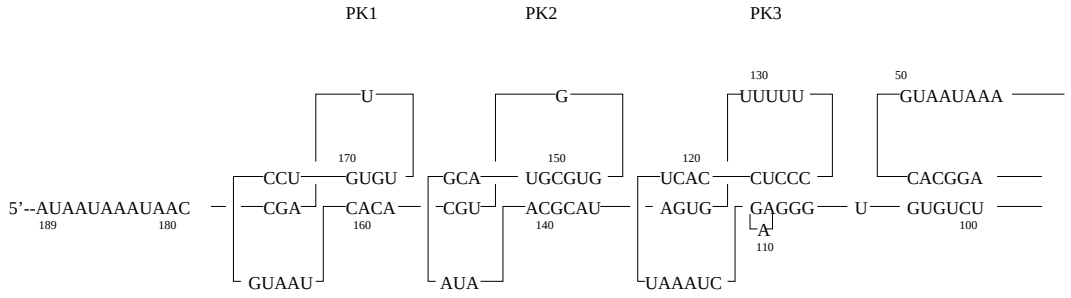

Figure 4: The folding of a part of tobacco mosaic virus (TMV) RNA. The upstream pseudoknot domain consists of three consecutive pseudoknots (PK1, PK2, PK3) at nucleotides 106-179.

The pseudoknot PK3 contains a bulge (A at position 110). Since our program does not yet deal with bulges, we removed nucleotide 110 before testing.

To search for the pseudoknots we defined a structure consisting of four string variables, $\{\alpha, \beta, \gamma, \delta\}$, where $4 \leq max\_length(\alpha) \leq 6$, $3 \leq max\_length(\beta) \leq 4$, $4 \leq end\_start(\alpha, \beta) \leq 7$, $end\_start(\beta, \gamma) = 1$, $2 \leq end\_start(\gamma, \delta) \leq 6$. The correlations are $R\_C(\alpha, \gamma)$, $R\_C(\beta, \delta)$.

Using the CSP implementation, with the content constraint $content(\beta, 2, \{c\})$ we found 5 solutions (including the correct ones) in 0.33 seconds. With content constraint $content(\beta, -2, \{ac\})$ we found 13 solutions in 0.61 seconds. Using both of the constraints resulted in 5 solutions found in 0.32 seconds. Approximately 3/4 of the time was used in the search, hence more efficient methods for searching (forward checking, looking ahead) [40], will probably in considerably degree reduce the time further.

Restricting the search to positions 100-188 in the sequence reduced the time to approximately one fifth of the time when searching in the whole string.

The CLP implementation produced similar results: 0.73 sec with no content constraints, 0.58 sec with the first content constraint, 0.6 sec with the second content constraint and 0.55 sec with both content constraints, on the entire string. Restricting the search to positions 100-188 reduced the search time to around one third.

9.4 Discussion

For the two real sequences, the CSP implementation was superior to the CLP implementation for the tobacco mosaic virus. In this sequence we searched for the most complex structure, had the highest number of solutions, and it was the shortest one. It might indicate that CSP is best for complexed structures and several solutions, which is not surprising. The search in the artificial sequence, however, does not follow this trend, as the CSP-implementation is relatively slower when the length to solution is decreasing. A more comprehensive investigation is neccessary for finding the relations of the two implementations.

9.5 Obtaining the program

The executable form of the CLP program can be used interactively and also obtained from http://www.soi.city.ac.uk/~drg/systems/structures/structures.html.

10 Summary and conclusions

In this research we have taken a grammar-based approach to describe biosequence patterns and structural features because we feel that the representational richness of grammars permits a more natural representation compared with Hidden Markov Models and profiles. We have investigated how constraint based techniques can be used to describe and search for patterns in sequences of symbols over finite alphabets, and we have defined a declarative constraint-

based language in which a user specifies the search pattern. These patterns can range from strings and regular expressions to more complex structures such as palindromes, repeats, stem loops and pseudo-knots. The expressive power of the language is beyond that of the regular languages, and it is deterministic in the sense that a pattern either does or does not match a given sequence. In the language the user can specify what we call a *structural* pattern, which means it can include correlations between different components of the pattern.

A pattern consists of a logical expression over *components* and a set of *constraints* on the components, where a component is a description of a sequence of symbols. An input string matches a pattern if for each component, it contains a matching substring such that all the constraints are satisfied with respect to the logical expression over the components. It is possible to constrain the length of a component, the distance between two components (relative to a matching input string), the symbols of a substring matching a component, the position on the input string matching a component, and the relation over the contents of two components.

We have defined an interpreter for this language as a constraint logic program over finite domains and implemented the interpreter in several constraint logic programming systems. A naive backtracking matching algorithm is used in this implementation which results in inefficient behaviour; however we have tested our implementation on some real biological sequences with encouraging results.

We have also designed a matching algorithm based on constraint satisfaction solving techniques, and made a preliminary implementation of its effectiveness.

The advantage of pattern-based searching for motifs are twofold: users can construct patterns are interesting based on their expertese, and secondly RNA motifs are difficult to derive from computation. Although several languages are already defined for biosequences (see Section 3.1), our language permits the description of more general structural patterns than most of the others. However, the essential novelty in our work is the method used for searching for matching substrings in the input string. We are able to define the patterns in a declarative way and have described a simple method for matching a pattern to a string using Constraint Logic Programming. We have also given an algorithm for matching using techniques from solving Constraint Satisfaction Problems, which will be more efficient.

Further work

We will improve the preliminary CSP-implementation, and will use a more efficient search method (forward checking, looking ahead) [40]. We further intend to investigate the possibility of interfacing this solver to the high-level implementation which has been made using constraint logic programming. This will involve defining a common interface and using the foreign-language facilities of the CLP language chosen for this.

We also intend to carry out more pactical testing of the language regading recognition of structures, comparing our results with those of existing systems, e.g. [52], [23] and validating them over large data sets. We are in the process of improving the efficiency of our system by improving our matching algorithms in order to be able to carry this out.

A more challenging task for the future will be to develop a structure discovery algorithm, and we will need to decide whether we will wish to find *conserved* structures. We intend to base our approach on the framework that we have developed in [9].

We also plan to develop a language for the schematic description of the spatial structure of proteins, broadly based on the approach which we have developed in this research. A first step in this direction could be the definition of a 'regular-expression' language over string variables, and also the definition of string constraints, for example the substring relation. The language would be used for describing the spatial structure of proteins at different levels of structural granularity (atoms, amino-acids, secondary and tertiary structures, etc.). A first step has been made regarding a language and associated pattern matching algorithm [25] for descriptions of protein structures at the level of topology.

Acknowledgements

We wish to thank Bernie Cohen for his help with the set-theoretic description of our language, and Daniel Diaz, author of the clp(FD) package, for his help with designing some of the routines needed by our solver. This work has been carried out as part of a project financed by the British Council and the Norwegian Research Council, which provided funding for the research visits. In addition, Inge Jonassen's research post is financed by the Norwegian Research Council.

References

[1] R. M. Abarbanel, P. R. Eiencke, E. Mansfield, D. A. Jaffe, and D. L. Brutlag. Rapid searches for complex patterns in biological molecules. *Nucleic Acids Research*, 12(1):263–280, 1984.

[2] R. B. Altman, B. Weiser, and H. F. Noller. Constraint Satisfaction Techniques for Modeling Large Complexes: Application to the Central Domain of 16S Ribisomal RNA. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings Second International Conference on Intelligent Systems for Molecular Biology*, pages 10–18. AAAI Press, 1994.

[3] A. Bairoch, P. Bucher, and K. Hofman. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24:189–196, 1995.

[4] A. Bairoch, P. Bucher, and K. Hofman. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24(1):189–196, 1996.

[5] P. Baldi and S. Brunak. *Bioinformatics: the machine learning approach.* MIT Press, 1998.

[6] L. Baranyi, W. Campell, K. Ohshima, S. Fujimoto, M. Boros, and H. Okada. The antisense homology box: A new motif within proteins that encodes biologically active peptides. *Nature Medicine*, 1(9):894–901, 1995.

[7] B. Billoud, M. Kontic, and A. Viari. Palingol: a declarative language to describe nucleic acids' secondarys tructures and to scan sequence databases. *Nucleic Acids Research*, 24(8):1395–1403, 1996.

[8] A. Brāzma and D. Gilbert. A Pattern Language for Molecular Biology. Technical Report 11, Department of Computer Science, City University, London, 1995.

[9] A. Brāzma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304, 1998.

[10] D. A. Clark and C. J. Rawlings. Constraint Satisfaction in Molecular Biology. Tutorial at Second International Conference on Intelligent Systems for Molecular Biology, 1994.

[11] D. A. Clark, C. J. Rawlings, and S. Doursenot. Genetic Map Construction with Constraints. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings Second International Conference on Intelligent Systems for Molecular Biology*, pages 78–86. AAAI Press, 1994.

[12] D. A. Clark, C. J. Rawlings, J. Shirazi, A. Veron, and M. Reeve. Protein Topology Prediction through Parallel Constraint Logic Programming. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings First International Conference on Intelligent Systems for Molecular Biology*, pages 83–91. AAAI Press, 1993.

[13] D. A. Clark, J. Shirazi, and C. J. Rawlings. Protein topologi prediction through constraint-based search and the evaluation of topological folding rules. *Protein Engineering*, 4:751–760, 1992.

[14] T. Dandekar and M. W. Hebtze. Finding the hairpin in the haystack: searching for RNA motifs. *Trends in Genetics*, 11(2):45–50, 1995.

[15] T. Dandekar and P. R. Sibbald. Trans-splicing of pre-mRNA is predicted to occur in a wide range of organisms including vertebrates. *Nucleic Acids Research*, 18(16):4719–4725, 1990.

[16] D. Diaz and P. Codognet. A Minimal Extension of the WAM for clp(FD). In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 774–790, Budapest, Hungary, 1993. The MIT Press.

[17] R. Durbin, S. Eddy, A. Krough, and G. Mitchison. *Biological sequence analysis.* CUP, 1998.

[18] S. Eddy. RNAbob User Guide. unpublished.

[19] I. Eidhammer. Extending Constraint Satisfaction Problems with Value Constraints. Technical Report 90, Department of informatics, University of Bergen, 1993.

[20] M. Foucrault and F. Major. Symbolic Generation and Clustering of RNA 3-D Motifs. In C. J. Rawlings, D. A. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors, *Proceedings Third International Conference on Intelligent Systems for Molecular Biology*, pages 121–126. AAAI Press, 1995.

[21] T. Frühwirth, A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy, and M. Wallace. Constraint Logic Programming: An informal introduction. In G. Comyn, N. E. Fuchs, and M. J. Ratcliffe, editors, *Logic Programming in Action*, LNCS 636, pages 3–35. Springer-Verlag, 1992. (Also available as Technical Report ECRC-93-5).

[22] C. Gaspin and E. Westhof. The Determination of the Secondary Structures of RNA as a Constraint Satisfaction Problem. In S.Schultze-Kremer, editor, *Advances in Molecular Bioinformatics*, pages 103–122. IOS Press, 1994.

[23] D. Gautheret, F. Major, and R. Cedergren. Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Computer Applications in the Biosciences*, 6:325–331, 1990.

[24] C. Gervet. Conjunto: constraint logic programming with finite set domains. In Maurice Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, pages 339–358, Massachusetts Institute of Technology, 1994. The MIT Press.

[25] D. R. Gilbert, D. R. Westhead, N. Nagano, and J. M. Thornton. Motif–based searching in tops protein topology databases. *Bioinformatics*, 15(4):317–326, 1999.

[26] R. Hamming. *Coding and Information Theory.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[27] C. Helgesen and P. Sibbald. PALM - a pattern language for molecular biology. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings First International Conference on Intelligent Systems for Molecular Biology*, pages 172–180. AAAI Press, 1993.

[28] P. V. Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[29] P. V. Hentenryck and Y. Deville. The cardinality operator: a new logical connective for constraint logic programming. In *Proceedings Eight International Conference on Logic Programming*, 1991.

[30] P. V. Hentenryck and Y. Deville. Operational semantics of constraint logic programming over finite domains. In J. Małuszyński and M. Wirsing, editors, *PLILP91*, number 528 in LNCS, pages 395–406. Springer-Verlag, aug 1991.

[31] I. L. Hofacker and P. F. Stadler. Automatic detection of of conserved base pairing patterns in RNA virus genomes. *Computation and Chemistry*, 23(3–4):401–414, 1999.

[32] I.L. Hofacker, M. Fekete, C. Flamm, M. A. Huynen, S. Rauscher, P. E. Stolorz, and P. F. Stadler. Automatic detection of conserved RNA structure elements in complete RNA virus genomes. *Nucleic Acids Research*, 26(16):3825–3836, 1998.

[33] A. Laferrière, D. Gautheret, and R. Cedergren. An RNA pattern matching program with enhanced performance and portability. *Computer Applications in the Biosciences*, 10(2):211–212, 1994.

[34] S. Leishman, P. M. D. Gray, and J. E. Fothergill. A Constraint-based Assignment System for Automatic Long Side Chain Assignments in Protein 2D NMR Spectra. In C. J. Rawlings, D. A. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors,

Proceedings Third International Conference on Intelligent Systems for Molecular Biology, pages 231–239. AAAI Press, 1995.

[35] S. Letovsky and M. B. Berlyn. CPRPO: A rule-based program for constructing genetic maps. *Genomics*, 12:435–446, 1992.

[36] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii nauk SSSR (in Russian)*, 163(4):845–848, 1965. Also in Cybernetics and Control Theory, vol 10, no. 8, pp 707–710, 1996.

[37] F. Major, M. Turcotte, D. Gautheret, G. Lapalme, E. Fillion, and R. Cedergren. The combination of symbolic and numerical computation for 3D modelling of RNA. *Science*, 253:1255–1260, 1991.

[38] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6–7):1105–1119, 1990.

[39] G. Mehldau and G. Myers. A system for pattern matching applications on biosequences. *Computer Applications in the Biosciences*, 9(3):299–314, 1993.

[40] B. A. Nadel. Constraint Satisfaction Algorithms. Technical report, Wayne State University, 1988. CSC-88-005.

[41] R. Nussinov, G Piecznik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal of Applied Mathematics*, 35:68–82, 1978.

[42] S. Parsons. Softening constraints in constraint-based protein topology prediction. In C. J. Rawlings, D. A. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors, *Proceedings Third International Conference on Intelligent Systems for Molecular Biology*, pages 268–276. AAAI Press, 1995.

[43] C. W. A. Pleij. RNA pseudoknots. *Current Opinion in Structural Biology*, 4:337–344, 1994.

[44] A. Rajasekar. Applications in constraint logic programming with strings. In Alan Borning, editor, *PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, Seattle WA, May 1994.

[45] M. Ratnayake. Constrained Pattern Recognition in Biosequences. Department of Computer Science, City University, London, 06 1996. B.Eng. (Honours) Degree in Software Engineering.

[46] E. Rivas and S. R. Eddy. The language of RNA: A formal grammar that includes pseudoknots. *Bioinformatics*. in press.

[47] E. Rivas and S. R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285:2053–2068, 1999.

[48] Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjoelander, R. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modelling. *Nucleic Acids Research*, 22:5112–5120, 1994.

[49] D. Searls. The computational linguistics of biological sequences. In Larry Hunter, editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press, 1993.

[50] D. Searls. The Computational Linguistics of Biological Sequences. Tutorial at Third International Conference on Intelligent Systems for Molecular Biology, 1995.

[51] D. Searls. String variable grammar: A logic grammar formalism for the bioligical language of DNA. *Journal of Logic Programming*, 24(1–2):73–102, July/August 1995.

[52] D. Searls and S. Dong. A syntactic pattern recognition system for DNA sequences. In C. R. Cantor H. A. Lim, J. Fickett and R. J. Robbins, editors, *Proceedings Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 89–101. World Scientific, 1993.

[53] P. R. Sibbald and P. Argos. Scrutineer: a computer program that flexibly seeks and describes motifs and profiles in protein sequences databases. *Computer Applications in the Biosciences*, 6(3):279–288, 1990.

[54] P. R. Sibbald, H. Sommerfeldt, and P. Argos. Overseer: a nucleotide sequence searching tool. *Computer Applications in the Biosciences*, 8(1):45–48, 1992.

[55] R. Staden. Searching for Patterns in Protein and Nucleic Acid Sequencies. In R. F. Doolittle, editor, *Methods in Enzymology, Vol. 183*, pages 193–211. Academic Press, 1990.

[56] M. Stefik. Inferring DNA Structures from Segmentation Data. *Artificial Intelligence*, 11:85–114, 1978.

[57] C. Walinsky. CLP(Σ^*): Constraint logic programming with regular sets. In Giorgio Levi and Maurizio Martelli, editors, *ICLP'89: Proceedings 6th International Conference on Logic Programming*, pages 181–196, Lisbon, Portugal, June 1989. MIT Press.

[58] D. E. Zimmerman, C. A. Kulikowski, and G. T. Montelione. A Constraint Reasoning System for Automating Sequence-Specific Resonance Assignments from Multidimensional Protein NMR Spectra. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings First International Conference on Intelligent Systems for Molecular Biology*, pages 447–455. AAAI Press, 1993.

[59] M. Zuker. Computer prediction of RNA structure. *Mehtods in Enzymology*, 180:189–225, 1989.

[60] M. Zuker. On Finding All Foldings of an RNA Molecule. *Science*, 244:48–52, 1989.

[61] M. Zuker and P. Stiegler. Optimal folding of large RNA sequences using thermodynamics and auxilliary information. *Nucleic Acids Research*, 9:133–148, 1981.