**Computer Science**
**Technical Reports**

CITY University

Technical Report No. 1997/04

# A Constraint Based Structure Description Language for Biosequences

Ingvar Eidhammer, David Gilbert, Inge Jonassen and
Madu Ratnayake

May 1997

City University
Dept. of Computer Science
Northampton Square
London EC1V 0HB
United Kingdom

ISSN 1364-4009

# A Constraint Based Structure Description Language for Biosequences

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

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

May 16, 1997

**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 the 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 ideas for an implementation of the CSP structure searching algorithm in C++.

Keywords: constraints, biostructures, description language, searching.

# Contents

# 1 Introduction

The aim of the work described in this paper is to investigate how constraint solving techniques can be used to search for structural patterns in sequences (or strings) of symbols over a finite alphabet $\Sigma$. 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 might be useful in other areas as well, e.g. signal processing or treating 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 for each component it contains a substring matching that component, such that all the constraints are satisfied.

A pattern can contain constraints ¿ of five types. There can be 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 three associated classes of patterns

- *Sequential*: patterns which do not include a correlation constraint. The patterns in the PROSITE data base [BBH95a] 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 of a D.
- *Pure structural*: patterns including at least one correlation constraint and no content constraints. One example is repetition, where the substrings matching two different components must be identical. Another example is a palindrome, two consecutive substrings of equal length must be the reverse of each other.
- *Structural*: patterns having at least one correlation constraint and one content constraint. One example is a palindrome, beginning with an a.

In terms of formal languages, the expressive power of sequential patterns is within that of the regular languages (not including Kleene closure), while structural patterns may describe context-free languages (e.g., stem-loops), or even languages beyond the expressive power of context-free grammars (e.g, repeats or pseudo-knots). As upper bounds on the length of biological sequences can be assumed, this is not strictly true (as all finite languages are regular), but we also require that the language description should be in some sense 'compact'.

We investigate structural patterns, but we have 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 $\Sigma$ 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.

We define a language to specify structural patterns. Although several such languages are already defined (see Section 3.1), in contrast to our language most of them do not permit the description of general structural patterns. However, the essential novelty in our work is the method used for searching for matching substrings in the input string. We define the patterns in a declarative way and show a naive method for solving it using Constraint Logic Programming. We then describe a possible implementation using techniques from solving Constraint Satisfaction Problems, which will be more efficient.

## 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. 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 strings for DNA/RNA molecules are called *nucleotide sequences*, and each element in such a sequence is called a *base*. 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 and g-c in a similar way. RNA and protein molecules fold into 3 dimensional structures enabling them to perform their structural/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 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 function of an RNA molecule or a protein from its sequence. Proteins may be grouped into families 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. Most languages used

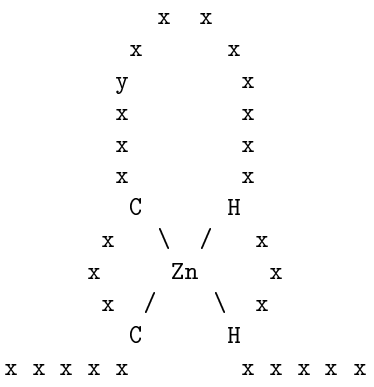


Figure 1: Schematic figure of the zinc finger c2h2 motif (accession number PS00028 in PROSITE). For this motif the sequential pattern C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H has been defined. (In the figure, x represents a position with any amino acid, and y a position with a more limited set of alternatives.

```
  c              g              x              x
 a-u            u-a            o-o            o-o
 g-c            a-u            o-o            o-o
 u-a            g-c            o-o            o-o
 c-g            c-g            o-o            c-o
augc   ggcau   aggc   ccgu    xx   xx        xx   xx

 (i)            (ii)           (iii)          (iv)
```

Figure 2: Illustration of structures and structural patterns: (i) and (ii) show two examples of structures (stem loops) that might be equivalent in RNA molecules. 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 pure 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 first nucleotide in the first part of the stem has to be a c – a restriction on the content of the substrings to match the pattern.

to define patterns for protein sequences permit only the definition of (what we have called) sequential patterns. Sequential patterns give sufficient expressive power to describe sequence features that are characteristic for many protein families. This is illustrated by the PROSITE protein family database which gives descriptive patterns for most of its families [BBH95a]. See Figure 1 for an example of a pattern from the PROSITE database.

For describing patterns in RNA sequences, one needs 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 [SBH+94]. Figures 2 (i) and (ii) shows two stem-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. 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. Finding a match to 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 than the one described by the pattern, might give a lower free energy.

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 dynamic programming algorithm which finds optimal as well as close to optimal secondary structures [Zuk89]. The algorithm has time complexity $O(l^3)$, where $l$ is the sequence length, and relies on some simplifications, for example, that no pseudo-knots are present.

Structural patterns should not be used alone to predict the secondary structure of RNA, but can be used in conjunction with structure prediction methods 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. We postulate that algorithms can be developed for finding conserved structural patterns in a set of RNA sequences analogous to algorithms for finding conserved sequential patterns in sets of protein sequences [BJEG95], and will investigate this in further work. In this way structural patterns allow for 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.

## 2.1 Example structures

In the structure description below $\alpha, \beta$ (with or without indices) are pattern components and $x$ is a wildcard (matching any one letter in an input string), $\alpha^r$ is the reverse of $\alpha$, and $\alpha^c$ is the complement of $\alpha$. $\alpha^{rc}$ is the reverse complement of $\alpha$. We have identified the following structures in the literature, see for example [Sea95, BCO$^+$95]. For each type we give one example. All examples are from DNA/RNA sequences, except for the last which is from a protein sequence.

- Tandem repeat $\alpha\alpha$ acgacg
- Simple repeat $\alpha\beta\alpha$ acgaaacg
- Multiple repeat $\alpha\beta\alpha\beta_1\alpha$ acgaaacguuacg
- Stem loop $\alpha\beta\alpha^{rc}$ acgaacgu
- Attenuator $\alpha\beta\alpha^{rc}\beta_1\alpha$ acgaacguauacg
- Palindrome, even $\alpha\alpha^r$ acggca
- Palindrome, odd $\alpha x\alpha^r$ acgagca
- Pseudoknot $\alpha_1\beta\alpha_2\beta_1\alpha_1^{rc}\beta_2\alpha_2^{rc}$ acgaaucugccguauaaga
- Sense - antisense $\alpha\beta\alpha^c$ IVLSPANHK

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

# 3 Previous approaches

Several programs have been developed for searching sequences for the presence of patterns. [DH95] gives an elaborate procedure for how to perform search 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 [DS90]. 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 she/he is going to search for. No explicit energy or stability aspects are taken into consideration, but some of them use structure predictions and biochemical properties. Some allow for mismatches and insertion of gaps, and have different ways for penalising mismatches and gaps.

Some (of the best known) programs or languages are:

**QUEST** [AEM+84] can only search for sequential patterns.

**Staden's program** [Sta90] is the first system that we are aware of in which one could search for structural patterns, though in a restricted way. He defines a pattern as comprising *motifs*. A pattern is built up and searched for by interactively specifying new motifs, by 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 a individual part of the structure, e.g. on the loop of a stem-loop. Percentage match and scoring matrices can be used in the 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** [SA90] 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 pattern, and where on a database sequence a specific part must match. A very limited form of dependencies constraints can be given, e.g. if position 4 is a small hydrophobic, then position 2 must be a `G`.

**OVERSEER** [SSA92] 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. 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.

**ANREP** [MM93] can only search for sequential patterns.

**PROSITE** [BBH95b] 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.

**PALM** [HS93] 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** [SD93] is the most general (implemented) system (to our knowledge) 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. GENLANG is implemented in Prolog, with hooks to C-code for the efficient caching of data that will be required during parsing. *String variables* are used to define structures. By letting $\sim$ be an operator denoting reverse complementarity, a pseudo-knot is for example specified by $X, \cdots, Y, \sim X, \cdots, \sim Y$. Constraints on the length and contents of the string variables can be specified.

**cBLISS** [Rat96] is an implementation in the constraint logic programming language Eclipse of the language of Brāzma and Gilbert [BG95] for describing constrained patterns in biosequences. This language is a formalisation and development of Staden's pattern language [Sta90]. 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 $\Sigma$ 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 not designed for the description of structures, the language can easily be extended for this purpose, since it is easy to add dependency constraints to the language.

# 4 Constraints

Constraint programming is a general term to describe problem solving techniques which computes solutions to problems by reducing the initial domains of the variables in the problem according to constraints expressed over those variables.

In general constraint solving techniques can be over infinite domains or over finite domains.

There are well-known techniques for the former, for example simplex solving over reals. Solving over finite domains is often achieved by techniques for solving Constraint Satisfaction Problems (CSP – see below). A general paradigm for describing and solving constraint problems is that of Constraint Logic Programming (CLP), a development of Logic Programming extended to domains other than just that of Herbrand terms. CLP systems can describe and solve problems both over infinite and finite domains and hence can utilise solvers based on techniques such as the simplex algorithm as well as those from the CSP world. Additionally some hybrid CSP-CLP systems exist such as CHIP which permit the user to explicitly use constraint satisfaction programming within a constraint logic programming environment. We will first describe CSP and then CLP.

### 4.1 Constraint satisfaction problems

A CSP (Constraint Satisfaction Problem) can be defined [Hen89] formally as: Let $X$ be a set of variables $x_1, x_2, ..., x_n$ which take their values from finite domains $D_1, D_2, ..., D_n$. Further let $C$ be a set of constraints, where a constraint $c_{i_1,...,i_h}(x_{i_1}, x_{i_2}, ..., x_{i_h})$ between $h$ variables from $X$ is a subset of the Cartesian product $D_{i_1} \times D_{i_2} ... \times D_{i_h}$, which specifies the values of the variables that are compatible with each other. A constraint among $h$ variables is called a $h$-constraint. The constraints are usually defined implicitly by equations, inequalities, programs etc. A solution to a CSP is an assignment of values to all variables, which satisfies all the constraints. Depending on the task, one or all solutions should be found. The general CSP is NP-hard [Nud83].

If the constraints are restricted to 1- and 2-constraints, the CSP is called *binary*, and binary CSP's are the most explored ones. A binary CSP is easily drawn as a graph, with the variables as the nodes, and edges drawn between variables which are mutually constrained. The edges represent the 2-constraints. A general CSP can be drawn as a hypergraph.

A CSP is usually solved by search with backtracking. Values are assigned to variables $x_{i_1}, x_{i_2}, ...$ as long as consistent values can be found. If the situation occurs that there exists a variable with no consistent value in its domain (consistent with the assignments done), backtracking has to be done. That means undoing some of the assignments, and trying alternative assignments. Conditions for a CSP to be solvable by a bactrack-free search have been developed [Fre82, DP88].

To reduce the backtracking, some consistency checking can be done before searching [Mes89]. The aim of this checking is to discover (and remove from being considered in the search) possible value assignments to one or several variables, which cannot be in any solution. This will reduce the search space, hence there is a trade off between consistency checking time, and search space reduction.

To formalise the consistency checking, *k-consistency* is introduced. A set of $n$ variables is k-consistent if each subset of $k-1$ variables with any values satisfying all the constraints among these $k-1$ variables, can be extended to include any of the other $n-(k-1)$ variables. The condition for inclusion is that the $k'th$ variable can be assigned a value such that all constraints among these $k$ variables are satisfied. The aim of the consistency checking is to get the set of variables k-consistent.

Many algorithms have been developed for achieving k-consistency. However, experiments have shown that achieving $k$-consistency for $k > 3$ is not cost effective in general [McG79, Nad88].

1-consistency is called *node-consistency*. Node-consistency is achieved by testing the values in the domains against the 1-constraints. 2-consistency is called *arc-consistency*, and is achieved by testing pair of values (from two different domains) against the corresponding 2-constraints. To achieve node- and arc-consistency values normally have to be removed from the domains. The best general algorithms for achieving arc-consistency have time-complexity $O(d^2e)$ [BC93], where $d$ is the size of each domain (all assumed equal), and $e$ is the number of pair of variables which are mutually constrained.

3-consistency is equivalent to *path-consistency* [Mon74], where path-consistency is defined as: for any variable pair $(x_i, x_j)$ each pair of values consistent with $c_{i,j}(x_i, x_j)$ must also be consistent with any other sequence of constraints $c_{i,i_1}(x_i, x_{i_1}), c_{i_1,i_2}(x_{i_1}, x_{i_2}) \ldots, c_{i_l,j}(x_{i_l}, x_j)$. When performing path-consistency checking (global) inconsistent pairs of assignments may be found, and can be added to the constraints. This might imply that the set of variables is no longer arc-consistent. For example, let $a$ be a member of $D_1$ and the only consistent value with that in $D_2$ be $b$. If, for achieving 2-consistency the possible assignments $(x_1 = a, x_2 = b)$ must be removed, then $a$ in $D_1$ is no longer consistent with any value in $D_2$, and must be removed to achieve 1-consistency. An algorithm for performing path-consistency checking is in [HL88]. In [Coo89] is an algorithm for achieving general $k$-consistency.

If the set of variables is $k'$-consistent for all $k' \leq k$, then it is *strong k-consistent*. For a CSP which is strong $n$-consistent all solutions are found without backtracking.

Performing consistency checking can also be done during the search, thus giving rise to different searching methods. In order to analyse that we consider a state in the search space where consistent values $v_{i_1}, v_{i_2}, ..., v_{i_r}$ are assigned to the variables $x_{i_1}, x_{i_2}, ..., x_{i_r}$. Let this set of variables be denoted by $U$, and let $W = X - U$ be the set of the $n - r$ other variables to which no values have yet been assigned. Different algorithms arise from:

- How much consistency checking is done for the variables after each assignment, e.g. arc-consistency between every pair of variables $(w, u), w \in W, u \in U$, (called Forward Checking) or in addition arc-consistency between each pair of variables in $W$ (called Looking Ahead). Some exploration are in [DM94, SF94].
- How much intelligence is used in the backtracking to decide where to backtrack. A number of different techniques have been proposed e.g. Backjumping [Gas77], Graph based backjumping [Dec90], Conflict-directed backjumping [Pro93], Backmarking [Gas77]. Evaluation of backtracking algorithms are in [Nad88, Gre94, Pro99, BR96].
- How the order in which values are assigned to the variables is decided, and how it is decided which of the possible values is to be assigned to each variable. This might have a great effect on the efficiency, and is explored in [DP88, DM89, FG89, BvR95, FD95, BR96].

A good book on CSP is [Tsa93].

## 4.2 Use of constraint satisfaction in molecular biology

Constraint based solving was used early in *map construction* [Ste78]. Several types of constraints were used to prune the search space during the search. In [LB92] a method for

genetic map construction is described. A number of constraints are defined, and constraint propagation are used to determine inconsistencies. Clark et. al. [CRD94] 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 program MC-SYM [MTG+91, FM95] 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 [GW94] have done the same for secondary structure prediction. To each base in the sequence there is associated a variable. The domain of a variable is 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 [CRS+93] is a program that *predicts protein $\alpha/\beta$-sheet and $\beta$-sheet topologies.* The variables represents different attributes associated with $\beta$-sheet, $\beta$-strands, $\alpha$-helices etc. The domains are values associated with those variables, and the constraints are known protein folding constraints. The program is written in ElipSys. An earlier version of this program [CSR92], is combined with explicitly representing the uncertainty of the rules in [Par95].

AUTOASSIGN [ZKM93] is a program which uses CSP to help in the *determination of protein structure* from NMR. The same is done by PROTEAN [AWN94] and TAM [LGF95]. The last is implemented in CHIP (Constraint Handling In Prolog), [Hen89].

We refer the interested reader to [CR94] which contains an introduction to constraint satisfaction in molecular biology and a more detailed explaination of some of the programs mentioned above.

## 4.3 Constraint Logic Programming

### 4.3.1 Introduction to Constraint logic programming

We base the following description on [JMSY92, Pou95]; a general text is [Hen89]. Constraint logic programming (CLP) is a based on constraint solving and the logic programming paradigm. In fact the CLP scheme describes a class of programming languages, of which Prolog is one member; in this sense, the CLP scheme is a generalisation of logic programming. In a CLP system the simple unification algorithm that lies at the heart of a logic programming system, for example Prolog, must be augmented by a dedicated solver for the particular domain of application, which can decide at any moment whether the remaining constraints are solvable. For efficiency's sake, solvers for CLP systems must be incremental, so that adding a new constraint to an already solved set does not force them all to be re-solved. Constraint solving algorithms are quite well understood from other branches of computing, for example Constraint Satisfaction Problems (CSP), described above in Section 4.1.

Some CLP languages which are widely available are:

- CLP($\mathcal{R}$), CLP over the reals, originated by J.Jaffar and J.L.Lassez, Monash University in Melbourne, Australia 1987. The present implementation is by IBM. [JMSY92]
- CHIP (Constraint Handling in Prolog), from ECRC Munich. Constraint solvers are over

finite arithmetic, linear rational and boolean domains [AB91].

- ECLiPSe, the ECRC Constraint Logic Parallel System, providing several libraries of constraint solvers: arithmetic constraints over finite atomic domains (CHIP compatible), finite set constraints, linear rational constraints, Propia (generalised propagation) and Constraint Handling Rules (CHR) [ECR95].
- clp(FD) from INRIA, France. This is a constraint logic programming language over finite domains, based on the wamcc Prolog compiler which translates Prolog to C via the WAM [CN95]
- SICStus Prolog, SICS, Sweden. Incorporates constraint solvers for reals, rationals, finite domains and booleans and a general constraint solver interface based on attributed variables [ea97].

### 4.3.2 Constraint logic programs

In constraint logic programs the basic components of the problem are constraints over an n-sorted algebra $\mathcal{A}$, which are composed together in order to describe the problem under consideration. An example of such an algebra is the two-sorted algebra which is the natural combination of real arithmetic terms and uninterpreted terms from the Herbrand universe.

We extend logic programs with dedicated predicate symbols, functors and constants over some specified domains. These domains may be finite or infinite, ordered or unordered, and possibly associated with a set of operations. The remaining functors of the program are interpreted as constructors of structures, possibly including elements of the domains, and the remaining predicate symbols of the program are interpreted as relations over the domains of such structures.

Thus in our example two-sorted algebra example the constraints may be either over the natural numbers or over Herbrand terms. It is common practice, but not necessary, to distinguish textually between constraint symbols over different domains; thus we may have the usual equality and inequality constraints over natural numbers ($=, <, >, \leq, \geq$) together with arithmetic operations on natural numbers (represented by the interpreted functors $+ - *$) and also unification over Herbrand terms ($=_{\mathcal{H}}$). (Pure) Prolog is thus a constraint logic programming language over Herbrand terms; the CLP($\mathcal{R}$) language has as its domain of discourse Herbrand terms and Real numbers. clp(FD) permits users to compute over integers and boolean domains and over Herbrand terms; Eclipse permits constraint computations over Herbrand terms, Real numbers and a variety of finite domains.

A *goal* in a CLP language is defined as being

$\leftarrow C_1, \ldots, C_m, A_1, \ldots, A_n$

and a *program* clause to be

$B_0 \leftarrow C'_1, \ldots, C'_i, B_1, \ldots, B_j$

where

$C_1, \ldots, C_m$ and $C'_1, \ldots, C'_i$ are constraints and $A_1, \ldots, A_n$ and $B_i \ldots, B_j$ are atoms with ordinary predicate symbols, which may contain interpreted subterms.

A *derivation step* is defined as:

Pick ordinary subgoal, e.g. $A_1$ of the form $p(t_1, \ldots, t_k)$, and find a program clause

$p(s_1, \ldots, s_k) \leftarrow C'_1, \ldots, C'_i, B_1, \ldots, B_j$

where $C'_1, \ldots, C'_i$ are constraints; the derived goal is

$\leftarrow C_1, \ldots, C_m,\ t_1 = s_1, \ldots, t_k = s_k,\ B_1, \ldots, B_j,\ A_2, \ldots, A_n$

*if* the constraints and the equalities are solvable.

Thus for example given the program in a CLP language whose domain $\mathcal{A}$ is over the natural numbers and Herbrand terms

p(X,Y) :- X > Y, q(Y)
q(Y) :- Y = 3

and the goal

?- A < 20, p(A,B)

we can derive by one derivation step the goal

?- A<20, A=X, B=Y, X>Y, q(Y)

providing that the constraint A<20, A=X, B=Y, X>Y is solvable in $\mathcal{A}$. A derivation sequence comprises goals generated by one or more derivation steps; a derivation sequence is successful when the last goal comprises only solvable constraints which are the answer constraints constituting the output of the program. In our example above, the answer constraints are A<20, A>B, B=3.

Finitely failed sequences are those whose last goal cannot be expanded due to either an absence of a suitable rule defining one or more predicate symbols in the goal, or the fact that the constraints in the goal are not solvable.

### 4.3.3 Constraints, valuations and solutions

Constraints are interpreted with respect to some domain such as the real numbers, booleans, or strings, etc. An *atomic* constraint represents an element of the domain, whilst a (complex) constraint is a finite set of atomic constraints, intuitively considered as a conjunction. The subset of the domain it denotes may be

- *expressed in a shorthand* such as the finite representation X>5 of an infinite subset of the reals, or
- *explicitly enumerated*, as in Finite Domain problems and CSPs, for example
  - X::1..4 (where 1 and 4 are the minimum and maximum elements of the totally ordered set {1,2,3,4}) or
  - Y::[a,c,t,g] (where the set {a,c,t,g} is unordered).

It is required that the language of constraints includes equality, representing a singleton drawn from the domain. TRUE and FALSE are distinguished constraints, the former corresponding to the empty constraint.

A **valuation** is an assignment of one value from a domain to each variable in a constraint problem, and is said to satisfy a constraint if the constraint is true in that valuation. A **solution** to a problem is a valuation which satisfies all the constraints in the problem. We sometimes abuse this concept and consider a solution to be a set of complex constraints associated with all the variables in the problem such that all the complex constraints are true.

### 4.3.4 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 [Hen89, FHK+92]. 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.

K-consistency (and/or other CSP methods) involve 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.

Let $P$ be a program over 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 $i$ 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

## 5.1 Informal description

We base our structure language on a modified version of Brazma and Gilberts' pattern language [BG95] 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*, and denote them by the Greek letters $\alpha, \beta, \gamma, \ldots$ (possibly subscripted). A pattern is defined by a *structure specification*, which is a *string expression* followed by a set of constraints. The string expression specifies the string variables taking part in the pattern, and a logical expression on them using conjunction, disjunction and negation.

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

and inequality operations over the integer components of the constraints, with the arithmetic operations over integers. We further allow the user to describe complex structures by conjoining structure descriptions.

### 5.2 Length constraints

A length constraint restricts the length of a string variable to be within a particular range, and has the form length($\alpha$, L) where $\alpha$ is a string variable and L ranges over the positive integers such that the length of $\alpha$ 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.

Furthermore, we introduce two variants, maxlength($\alpha$, L) and minlength($\alpha$, L) such that the length of $\alpha$ is the maximum, respectively minimum value possible within the range denoted by L according to some mapping to a given input string. Redundant matches are avoided in the case of e.g. stemloops where substrings of the stem are not required.

### 5.3 Distance constraints

A distance constraint restricts the distance between two string variables, and are specified in a declarative and uniform way, e.g. start_start($\alpha, \beta, D$), end_start($\alpha, \beta, D$), start_end($\alpha, \beta, D$), end_end($\alpha, \beta, D$) where $\alpha$ and $\beta$ are string variables and D ranges over the integers. These relations constrain the distance between the start of $\alpha$ and start of $\beta$ (respectively end of $\alpha$ and start of $\beta$, start of $\alpha$ and end of $\beta$ end of $\alpha$ and end of $\beta$) to lie within the range denoted by D. A negative value for D indicates that the point of reference of $\alpha$ occurs after the corresponding point of reference of $\beta$ in the input string. We also permit the shorthand $\alpha.\beta$ to indicate that $\beta$ starts directly after $\alpha$. This shorthand is equivalent to $\alpha \wedge \beta$, end_start($\alpha$, $\beta$, 1) .

### 5.4 Content constraints

A content constraint restricts which symbols can be in a specific position on a string variable matching a component and is expressed thus: content($\alpha$,Pos,Set) where $\alpha$ is a string variable, Pos is a positive or negative (non-zero) integer representing $\alpha_{\mathsf{Pos}}$, the character from $\alpha$ at position Pos from the start (or end if Pos is negative) of $\alpha$, and Set is a (non-empty) set of characters to which $\alpha_{\mathsf{Pos}}$ may be bound, e.g. {a,t}.

### 5.5 Position constraints

A position constraint restricts the absolute positions of a string variable on the input string and is expressed as start($\alpha$,P) or end($\alpha$,P) where $\alpha$ is a string variable and P ranges over the positive integers such that the first (respectively last) character of $\alpha$ is located at position P on the input string.

### 5.6 Correlation constraints

A **correlation constraint** ("correlation" for short) defines the relation between the contents of two string variables. A correlation $C$ has the following properties:

- It relates two string variables $C(\alpha, \beta)$, the string variable $\alpha$ being called the *source*, and $\beta$ the *target*.
- The length of the two string variables must be equal (due to equal numbers of symbols in the matching substrings), implying that there is an implicit length constraint between the two strings.
- There is a *direction*-component $C_d$, written as the relation $C_d(\alpha, \beta)$. The two legal values for $C_d$ are 1 and -1. $1(\alpha, \beta)$ is satisfied iff $(\forall i : 1 \leq i \leq h : \beta_i$ is related to $\alpha_i)$. $-1(\alpha, \beta)$ is satisfied iff $(\forall i : 1 \leq i \leq h : \beta_i$ is related to $\alpha_{h-i+1})$, where $\alpha_i$ and $\beta_i$ are symbols from $\alpha$ and $\beta$, and $h$ is the length of the matching substrings. Note that this means that all positions of the string variables take part in the correlation.
- There is a *symbol*-component $C_s$. As part of this component a function $C_f$ is defined from $\Sigma$ to $2^{\Sigma}$. $C_s(\alpha, \beta)$ is satisfied iff $(\forall i : 1 \leq i \leq h : \beta_i \in C_f(\alpha_i))$
- Let $\mathcal{L}$ be the language of all strings with symbols from $\Sigma$. The correlation $C(\alpha, \beta)$ is satisfied iff $\exists x : x \in \mathcal{L} : C_d(\alpha, x) \wedge C_s(x, \beta)$.

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 between two string variables; when the mismatch is zero then we can omit this argument. We can use Hamming distance [Ham82], edit distance or more generally Levenshtein distance [Lev65] in order to implement approximate matching[1].

We define id($\alpha$,$\beta$) and reverse($\alpha$,$\beta$) as general correlation constraints over all alphabets, where $\beta$ is the identity (respectively, reverse) of $\alpha$, and assume that there is a library of correlations, and that a user may

- add a new correlation to the library by the command define_corr, for example define_corr('rev_compl_RNA',-1, {a→{u}, c→{g}, g→{c,u}, u→{a,g}})
- use a known correlation where a correlation between two string variables is simply specified by the name of the correlation, and the two variables as arguments, e.g. rev_compl_RNA($\alpha$, $\beta$, M) where M is the percentage approximate match.
- use (without storing in the library) an unnamed correlation in a specification correl($\alpha$, $\beta$,-1, {a→{t}, c→{g}, g→{c}, t→{a}}, M), where M is the percentage approximate match.

The definitions of reverse, complement and reverse_complement with approximate matching are
$\forall\alpha\forall\beta\forall M$ (reverse($\alpha$, $\beta$, M) $\leftrightarrow \exists\gamma$ (reverse($\alpha$, $\gamma$) $\wedge$approximate_match($\gamma$, $\beta$, M)))
$\forall\alpha\forall\beta\forall M$ (complement($\alpha$, $\beta$, M) $\leftrightarrow \exists\gamma$ (complement($\alpha$, $\gamma$) $\wedge$approximate_match($\gamma$, $\beta$, M)))
$\forall\alpha\forall\beta\forall M$ (rev_compl($\alpha$, $\beta$, M) $\leftrightarrow \exists\gamma\exists\delta$ (reverse($\alpha$, $\gamma$) $\wedge$complement($\gamma$, $\delta$) $\wedge$approximate_match($\delta$, $\beta$, M)))

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

### 5.7 Examples

A description of the stem loop with exact matching in Figure 2(iv) is
$\alpha.\gamma.\beta$, maxlength($\alpha$, 4), length($\gamma$, 1), content($\alpha$, 1, {c}), rev_compl_RNA($\alpha$, $\beta$)
assuming a library definition of rev_compl_RNA as above, and where $\alpha$ and $\beta$ form the stem, with $\gamma$ the loop. A longer version without using the shorthand $\alpha.\gamma.\beta$ would be
$\alpha \wedge\beta \wedge\gamma$, maxlength($\alpha$, 4), length($\gamma$, 1), end_start($\alpha$, $\gamma$, 1), end_start($\gamma$, $\beta$, 1), content($\alpha$, 1, {c}), rev_compl_RNA($\alpha$, $\beta$, 0)

### 5.8 User queries

Finally, queries can be formulated where an input string is appended to a structure description and some *mapping algorithm* used to map the description to the string. Thus the user may enter the following query:
$\alpha.\gamma.\beta$, maxlength($\alpha$, 4), length($\gamma$, 1), content($\alpha$, 1, {c}), rev_compl_DNA($\alpha$, $\beta$), tatacctgtcaggtata
which will result in $\alpha$ being mapped to the substring cctg starting at position 5 and ending at 8, $\beta$ to cagg starting at 10 and ending at 13, and $\gamma$ to t at position 9. Queries may be optionally prefaced by a description of the alphabet of characters which are permitted in the input string.

### 5.9 Macro language

We further define a macro language permitting the user to store and re-use definitions of, i.e. grammars for, specific structures. The syntax of this language is similar to that of logic programs; for example the following grammars define languages for stem loops and pseudoknots:
stemloop($\alpha$,$\gamma$,$\beta$):- $\alpha.\gamma.\beta$, rev_compl_RNA($\alpha$,$\beta$)
pseudoknot($\alpha$,$\beta$,$\gamma$,$\delta$):- $\alpha.\omega_1.\beta.\omega_2.\gamma.\omega_3.\delta$ , rev_compl_RNA($\alpha$,$\gamma$), rev_compl_RNA($\beta$,$\delta$)

### 5.10 Syntax — BNF

Note: we indicate terminals by enclosing them in single quotes. Optional items are indicated in square brackets.

```
query          ::=  [alphabet] specs ‘,’ inputstring
specs          ::=  spec | spec ‘;’ specs
alphabet       ::=  ‘sigma(’CharSet‘)’
spec           ::=  strexp ‘,’ constraints
constraints    ::=  constraint | constraint ‘,’ constraints
constraint     ::=  length_c | dist_c | content_c | pos_c | corr_c |
                    arith_c
length_c       ::=  ‘length(’stringvar ‘,’ IntVar ’)’ |
                    ‘maxlength(’stringvar ‘,’ IntVar ’)’ |
                    ‘minlength(’stringvar ‘,’ IntVar ’)’
dist_c         ::=  ‘start_start(’stringvar ‘,’ stringvar ‘,’ IntVar±
                    ’)’ |
                    ‘end_start(’stringvar ‘,’ stringvar ‘,’ IntVar±
                    ’)’ |
                    ‘start_end(’stringvar ‘,’ stringvar ‘,’ IntVar±
                    ’)’ |
                    ‘end_end(’stringvar ‘,’ stringvar ‘,’ IntVar± ’)’
content_c      ::=  ‘content(’ stringvar ‘,’ Int± ‘,’ CharSet ‘)’
pos_c          ::=  ‘start(’ stringvar ‘,’ IntVar ‘)’ | ‘end(’
                    stringvar ‘,’ IntVar ‘)’
corr_c         ::=  ‘define_corr(’CorrName ‘,’ Direction ‘,’
                    CharMapping ‘)’ |
                    CorrName‘(’ stringvar ‘,’ stringvar [‘,’ IntVar]
                    ‘)’ |
                    ‘correl(’ stringvar ‘,’ stringvar ‘,’ CharMapping
                    [‘,’ IntVar] ‘)’ |
                    gen_corr
gen_corr       ::=  ‘id(’stringvar ‘,’ stringvar [‘,’ IntVar] ‘)’ |
                    ‘reverse(’stringvar ‘,’ stringvar [‘,’ IntVar] ‘)’
arith_c        ::=  IntExp IntComp IntExp
IntComp        ::=  ‘=’ | ‘<’ | ‘≤’ | ‘>’ | ‘≥’
IntExp         ::=  IntVar IntOp IntExp | ‘(’ IntExp ‘)’ | IntVar
IntOp          ::=  ‘+’ | ‘−’ | ‘*’ | ‘/’
Direction      ::=  ‘1’ | ‘-1’
CharMapping    ::=  ‘{’ CharMappings ‘}’
CharMappings   ::=  CharToSet | CharToSet ‘,’ CharMappings
CharToSet      ::=  Char ‘->’ CharSet
CharSet        ::=  ‘{’ Chars ‘}’
Chars          ::=  Char | Char ‘,’ Chars
Char           ::=  character
IntVar         ::=  Var | Int
IntVar±        ::=  Var | Int±
Int            ::=  positive_integer
Int±           ::=  positive_integer | negative_integer
Var            ::=  variable_over_integers
strexp         ::=  stringvar | stringvar OP strexp | ‘not’ ‘(’ strexp
                    ‘)’
OP             ::=  ‘and’ | ‘or’ | ‘.’
stringvar      ::=  ‘A’ ...‘Z’
```

### 5.11 Towards an implementation

We plan that the language will be used in an environment where there is a user interface which permits the user to enter descriptions of the structures that he is interested in, or to use definitions from libraries, to map the description to a given input string and then will return the results of the mapping to the user. The queries will be handled by a query evaluator, which will check the syntax of the queries, expand macros, and store any macro definitions made by the user, and translate 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 could be output in various ways, ranging from the locations of strings, and optionally the strings themselves, to some graphical representation of the structures found.

Such a processor for the language may be implemented in any programming system; in the next two sections we describe an implementation of the processor in constraint logic programming and also the design for a solver using CSP. Our program in clp(FD) implements all these stages but employs a naive and inefficient algorithm to map a specification onto an input string. We plan to improve the existing CLP implementation by integrating the CSP solver within it.

## 6 Implementation in CLP

### 6.1 String variables and string expressions

We have chosen to represent string variables ($SV$), i.e. components, by sequences of maximum length $m$ of *string-characters* ($SC$). These comprise pairs whose first element *Chars* is a set of characters drawn from some alphabet $\mathcal{A}$ (of bases or nucleotides) and whose second element *Pos* is a set of integers in $1 \ldots m$, i.e. $SV = \text{seq}(\mathcal{A} \times 1 \ldots m)$. Each pair represents the possible values of the characters to be found on the input string at the locations indicated by the second element of the pair. 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; given a projection function proj2: $x \times y \rightarrow y$ then following the set-oriented specification method of [SP87] we define a successor constraint on string character positions by
$\forall a : SV \;.\; \forall i : [1 \ldots \text{len}(a-1)] \;.\; \forall x : 1 \ldots m | (x \in \text{proj2}(a_i) \wedge x < m) \;.\; \text{succ}(x) \in \text{proj2}(a_{\text{succ}(i)})$

We have chosen constraint logic programming over finite domains [HD91b] 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) [DC93] as the basis for our implementation because it has a specialised operation for complementation over genomic alphabets (see below). Moreover, the clp(FD) system 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 [Wal89], [Ger94] or [Raj94].

## 6.2 Constraints

*Length* constraints are defined in the usual backtracking manner for lists although ideally we would like to use a list solver (for example [Raj94]). *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 a specialised solving routine compl/4 in clp(FD). For example RNA, whose alphabet a, c, g and u we represent by 1, 2, 3 and 4 respectively, has complements {a→{u}, c→{g}, g→{c,u}, u→{a,g}}. We represent this by

```
complement_char(Char1,Char2):-
    compl(Char1,1,Char2,[4]), compl(Char1,2,Char2,[3]),
    compl(Char1,3,Char2,[2,4]), compl(Char1,4,Char2,[1,3]).
```

where the definition of compl/4 is

```
compl(A, Char, B, Chars):-
     A=Char <=> Val1, B in Chars <=> Val2,
     Val1 in 0 .. max(Val2), Val2 in min(Val1) .. 1
```

## 6.3 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 $\mathcal{A}$ (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). For example the RNA sequence of act of bases which is the sequence {(1,a),(2,g),(3,t)} is mapped to the sequence {(1,({a},{1})), (2,({g},{2})), (3,({t},{3}))} and then to the list [({1},{1}),({3},{2}),({4},{3})] using our numeric representation of the base alphabet.

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. We assume two types of correlation: $c$ (normal correlation) and $r$ (reverse correlation), and a function p1: $x \times y \rightarrow x$. variables efficiently.

**for** each pair of string variables $(\alpha, \beta)$ in $SE$ correlated by correlation $c$ **do**
    find members of $I$ s.t. $\alpha_1 = I_j$ and $\beta_1 = I_k$ and set $i = 1$
    **while** $c(\mathrm{p1}(\alpha_i), \mathrm{p1}(\beta_i))$ and $i \leq \mathrm{length}(\alpha)$ **do**
        $i := i + 1$ and $j := j + 1$ and $k := k + 1$
        $\alpha_i = I_j$ and $\beta_i = I_k$
    **end**
**end**
**for** each pair of string variables $(\alpha, \beta)$ in $SE$ correlated by correlation $r$ **do**
    set $l = \mathrm{length}(\beta)$
    find members of $I$ s.t. $\alpha_1 = I_j$ and $\beta_l = I_k$ and set $i1 = 1$, $i2 = l$
    **while** $c(\mathrm{p1}(\alpha_{i1}), \mathrm{p1}(\beta_{i2}))$ and $i1 \leq \mathrm{length}(\alpha)$ **do**
        $i1 := i1 + 1$ and $i2 := i2 - 1$ and $j := j + 1$ and $k := k - 1$
        $\alpha_{i1} = I_j$ and $\beta_{i2} = I_k$
    **end**
**end**

However, in the algorithm for the general case (including disjunction and negation) we do not do this pairwise mapping:

**proc** map($SE$)
    **if** $SE = A \wedge B$ **then do** proc($A$) and proc($B$) **end**
    **if** $SE = A \vee B$ **then do** proc($A$) or proc($B$) **end**
    **if** $SE = \neg A$ **then do** not proc($A$) **end**
    **if** $SE$ is a string variable $\gamma$ **then do**
        find a member of $I$ s.t. $\gamma_1 = I_j$
        **while** $i \leq \mathrm{length}(\gamma)$ **do**
            $i := i + 1$ and $j := j + 1$
            **if** $\gamma_i = I_j$ **then** true **else** fail **end**
        **end**

```
    end
end
```

### 6.4 Testing

Our source program is 388 lines (10K) of clp(FD) code; we have compiled our program to 370K of stand-alone sun-sparc code using the clp(FD) system [DC93], and have used this to test the detection of stem-loops from a variety of databases, including entry with ID CXSTPLUC2 (accession number X87994) from the EMBL nucleotide sequence database release 49 (Nov 1996),
URL: http://www2.no.embnet.org/srs/srsc?[EMBL-id:CXSTPLUC2]+-sf+GCG. For example our program took 40 ms on a Sun IPX to find the stem-loop cccgtcca, gctcggct, tggacggg at position 20–43 (perfect matching), and 90 ms to find the stem-loop cagctcg, gcttgga, cgggctg at position 26–46 (mismatch of 14%) in a string of nucleotides from positions 1–60.

### 6.5 Obtaining the program

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

## 7 Representing and solving the structure searching problem as a CSP

A method for solving the structure searching problem using techniques from solving Constraint Satisfaction Problems is described. In this first version only conjunction are allowed for in the string expression, and only exact matching in the correlations. In addition only intervalls are allowed for in the length constraints. Inputs are the input string $S(1:n)$, and a structural pattern.

The method consists of four steps:

1. Represent the problem as a constraint problem.
2. Perform consistency checking to remove search alternatives.
3. Reformulate the problem to a new CSP.
4. Search for solutions (a solution is occurrences of the structure in the string).

### 7.1 Representation of the constraints

In the first step the constraint variables will be defined, and the constraints represented. Five types of constraints are defined in 5. However, in this section the length constraint is included in the distance constraint. This is because they are represented in a similar way.

### 7.1.1 The constraint variables

The following constraint variables are introduced:

- For each string variable $\alpha$ two *distance* constraint variables $L_\alpha, U_\alpha$, with domains subsets of $[1, n]$. In a solution the start position and the end position of $\alpha$ in $S$ are assigned to these variables, respectivily.
- For each correlation $C$ between $\alpha, \beta$ a set $\mathcal{X}^{(\alpha,\beta)}$ of *correlation* constraint variables $\{X^{(\alpha,\beta)}_{L_\alpha}, ..., X^{(\alpha,\beta)}_{U_\alpha}\}$, with domains subsets of $[1, n]$. (We allow at most one correlation between any pair of string variables.) Let $D^{(\alpha,\beta)}_i$ be the domain of $X^{(\alpha,\beta)}_i$, where $i$ corresponds to position $i$ in the input string $S$. The correlation constraint variables are used to constrain the position of $\beta$ in $S$ when restrictions on the position of $\alpha$ are known, and vice versa. For example, let $D^{(\alpha,\beta)}_7 = \{5, 9, 13\}$. If $\alpha$ includes $S_7$, and $C_d = 1$, then the position of the corresponding symbol in $\beta$ must be in $\{5, 9, 13\}$. In a solution $X^{(\alpha,\beta)}_{L_\alpha+j} = L_\beta + j$ if $C_d = 1$, and $X^{(\alpha,\beta)}_{L_\alpha+j} = U_\beta - j$ if $C_d = -1$.

  Note that $L_\alpha$ and $U_\alpha$ are themselves constraint variables. Such use of constraint variables corresponds to the use in [GW94].

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

### 7.1.2 The constraints

Most of the *distance* constraints are represented as (binary) relational expressions between distance constraint variables. The only exception is the implicit length equality between the variables in a correlation.

Bound on the length of a string variable $\alpha$ is represented as a binary constraint between $L_\alpha, U_\alpha$. Bound on the distance between two string variables $\alpha, \beta$ is represented as a binary constraint between one of $L_\alpha, U_\alpha$ and one of $L_\beta, U_\beta$, depending on how the bound is specified. The implicit distance constraint in each correlation (length equality between $\alpha$ and $\beta$), is also represented as constraints between four distance constraint variables $L_\alpha, U_\alpha, L_\beta, U_\beta$.

The *position* constraints are used to restrict the domains of the corresponding distance constraint variables. This is wholly done in the consistency checking step, before searching. The same is the case for the *content* constraints.

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 $\alpha, \beta$, is then represented as $s\_c(\mathcal{X}^{(\alpha,\beta)}, L_\alpha, U_\alpha, C_d)$.

This constraint has some similarities with van Hentenrycks cardinality operator [HD91a], and the value constraint in [Eid93].

## 7.2 Consistency checking and constraint propagation

In this step of the algorithm the domains of the constraint variables will be found, some redundant constraints can be introduced, and consistency checking will be done.

### 7.2.1 Position constraints

The effect of position constraints on a string variable $\alpha$ is done by reducing the domains of $L_\alpha$ and/or $U_\alpha$. This means performing node-consistency checking, and the constraints need not be saved any longer.

### 7.2.2 Distance constraints

Consistency checking over the distance constraint variables can be done. In doing so the distance constraint variables might be constrained against the end points of the input string (1 and $n$). For example, if $|\alpha| \geq r$ for a string variable $\alpha$, then $L_\alpha \leq n - r + 1$.

It might be useful, for reasons of efficiency, to have redundant constraints, such that there are explicit constraints between distance constraint variables corresponding to each pair of string variables appearing in a correlation. This can be found by propagating other distance constraints. For example if $U_\alpha < L_\gamma \wedge U_\gamma - L_\gamma \geq 4 \wedge U_\gamma < L_\beta$, we can deduce $L_\beta \geq U_\alpha + 6$.

Most of the distance constraints are binary, and easily used in arc consistency checking. However, the implicit distance constraint from a correlation between $\alpha$ and $\beta$ are between four distance constraint variables, but can be seen as a binary constraint between $|\alpha|$ and $|\beta|$. An example might clarify that: Suppose there is a correlation between $\alpha$ and $\beta$, $2 \leq |\alpha| \leq 4$, and $D_{L_\alpha} = \{2, 3, 5\}, D_{U_\alpha} = \{6, 8\}, D_{L_\beta} = \{12, 16\}, D_{U_\beta} = \{14, 16, 19\}$. We see that $|\alpha|$ can never be 3, and $U_\beta = 14$ only if $|\beta| = 3$, hence 14 cannot be in the domain of $U_\beta$. Treating such an implicit distance constraint as a binary constraint between $\alpha$ and $\beta$ might be cost effective.

### 7.2.3 Content constraints

The content constraints are used to restrict the domains of the distance constraint variables.

Let a content constraint be such that the $i$'th position of a string variable $\alpha$ must be one of the symbols in a set $E$. Then the domain of $L_\alpha$ is restricted to be a subset of the set $\{j | S_{j+i-1} \in E\}$. If the constraint is on the i'th last symbol, the domain of $U_\alpha$ is restricted in a similar way: $\{j | S_{j-i+1} \in E\}$.

If $\alpha$ occurs as source in a correlation $(\alpha, \beta)$, then a content constraint on $\alpha$ implies constraints on either $L_\beta$ or $U_\beta$, which of them depends on the specification of the constraint and the direction component. For example if the correlation is reverse complement, then $D_{U_\beta}^{(\alpha,\beta)}$ is a subset of $\{j | S_{j-i+1} \in \mathit{compl}(E)\}$.

If $\alpha$ occurs as target in a correlation $(\gamma, \alpha)$, then a content constraint on $\alpha$ implies content constraints on $\gamma$. Let the symbol function of the constraint be $C_f$ (see Section 5.6). If $C_d = 1$, then the domain of $L_\gamma$ is restricted to be a subset of $\{j | (C_f(S_{j+i-1}) \cap E) \neq \emptyset\}$. If $C_d = -1$, then the domain of $U_\gamma$ is restricted to be a subset of $\{j | (C_f(S_{j-i+1}) \cap E) \neq \emptyset\}$.

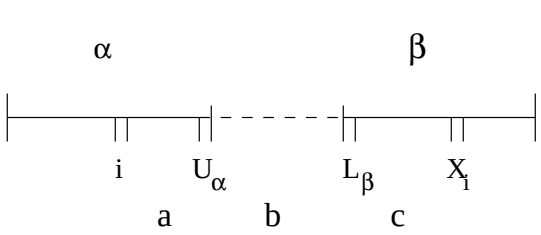


Figure 3: Figure showing the limits of $\delta_i = X_i - i$. $a = U_\alpha - i, b = L_\beta - U_\alpha, c = X_i - L_\beta$

The effect of the content constraints is to individually reduce the domains of distance constraint variables, hence performing node consistency. This means that the whole effect is taken care of in this step, and the content constraints can thus be ignored in the further processing.

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

### 7.2.4 Correlation constraints

A correlation is a constraint between two string variables, $\alpha$ and $\beta$. In this step the **local solutions** for each correlation are found. A local solution is two substrings which satisfy all the constraints between and on $\alpha$ and $\beta$. For simplicity in the rest of this section we omit the superscript $(\alpha, \beta)$ on the correlation constraint variables.

For a correlation between $\alpha$ and $\beta$ the following can be done. Let

- $m_{I_J} = min(D_{I_J})$ and $M_{I_J} = max(D_{I_J}); I \in \{L, U\}, J \in \{\alpha, \beta\}$.
- $r \leq |\alpha| \leq R$, i.e $r$ and $R$ are the lower and upper bound on the length of $\alpha$ (and $\beta$).

The (actual) constraint correlation variables are $X_i, i \in [m_{L_\alpha}, M_{U_\alpha}]$, with $D_i = [m_{L_\beta}, M_{U_\beta}]$. However, some of the variables might be constrained more:

1. If there exists bounds on the distance between $\alpha$ and $\beta$, these bounds can be used to find constraints on $X_i$ (bounds on the differance $X_i - i = \delta_i$). There are four ways to specify the distance between $\alpha$ and $\beta$. However, two of them have the same effect.
   - Let $l \leq L_\beta - U_\alpha \leq L$
     - Let $C_d = 1$. Figure 3 shows how the bounds on $\delta_i$ can be calculated when $C_d = 1$. $\delta_i = a + b + c$, from $r - 1 \leq a + c \leq R - 1$ and $l \leq b \leq L$ follows $r - 1 + l \leq \delta_i \leq 2R - 2 + L$
     - Let $C_d = -1$. Now $\delta_i$ is not equal for each $i$. The minimum value for $\delta_i$ is for $i = U_\alpha$, and the maximum for $i = L_\alpha$. Hence $l \leq \delta_i \leq 2R - 2 + L$
   - Let $l \leq U_\beta - L_\alpha \leq L$. In a similar way as above we find
     - For $C_d = 1$: $l - R + 1 \leq \delta_i \leq L - r + 1$
     - For $C_d = -1$: $l - 2R + 2 \leq \delta_i \leq L$
   - Let $l \leq L_\beta - L_\alpha \leq L$ or $l \leq U_\beta - U_\alpha \leq L$ In a similar way we find
     - For $C_d = 1$: $l \leq \delta_i \leq L$

- For $C_d = -1$: $l - (R-1) \leq \delta_i \leq L + R - 1$

2. We can also use minimum and maximum values of the distance constraint variables.

   - For $C_d = 1$, from $X_{m_{U_\alpha}} \leq M_{U_\beta}$ follows $X_{m_{U_\alpha}-j} \leq M_{U_\beta} - j; 1 \leq j \leq r$
   - For $C_d = 1$, from $X_{M_{L_\alpha}} \geq m_{L_\beta}$ follows $X_{M_{L_\alpha}+j} \geq m_{L_\beta} + j; 1 \leq j \leq R$
   - For $C_d = -1$, from $X_{m_{U_\alpha}} \geq m_{L_\beta}$ follows $X_{m_{U_\alpha}-j} \geq m_{L_\beta} + j; 1 \leq j \leq r$
   - For $C_d = -1$, from $X_{M_{L_\alpha}} \leq M_{U_\beta}$ follows $X_{M_{L_\alpha}+j} \leq M_{U_\beta} - j; 1 \leq j \leq R$

By use of these constraints we restrict the domains $D_i, i \in [m_{L_\alpha}, M_{U_\alpha}]$.

We then find the values for $L_\alpha, U_\alpha$ where $s\_c(\mathcal{X}, L_\alpha, U_\alpha, C_d) \wedge r - 1 \leq U_\alpha - L_\alpha \leq R - 1$ is satisfied.

The procedure above guarantees to find solutions satisfying the constraints on the distance between $\alpha$ and $\beta$ if and only if the constraint are in the form $l \leq L_\beta - L_\alpha \leq L$ or $l \leq U_\beta - U_\alpha \leq L$, and $C_d = 1$. For all other cases the result must be checked against the distance constraints to be sure that the found values are local solutions. An example will clarify. Let
$l \leq L_\beta - U_\alpha \leq L$ and
$r = 3, R = 5, l = 1, L = 2, D_1 = \{6, ...\}, D_2 = \{7, ...\}, D_3 = \{8, ...\}, D_4 = \{10, ...\}$. We find $3 \leq X_i - i \leq 6$, and a solution to the correlation constraint is $\alpha = S(1:3), \beta = S(6:8)$, but this does not satisfy the distance constraint.

Developing efficient algorthm for finding the local solutions are the clearly most difficult and challenging task.

The local solutions define new constraints between pairs of string variables, or between 4 distance constraint variables. Before searching for a global solution, we can perform consistency checking between individual local solutions. To formalise this we reformulate the problem.

### 7.3 Reformulating the problem

We reformulate the problem as a new binary CSP with constraints between the string variables:

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

We can now perform arc-consistency checking.

Note that arc-consistency are assured between string variables for which there is a correlation.

### 7.3.1 Example

Let a structure specification contain the string variables $\alpha, \beta, \gamma, \delta, \eta, \phi$, and the correlations are $C_1(\alpha, \gamma), C_2(\gamma, \eta), C_3(\delta, \phi)$. At the beginning arc-consistency is assured for those pairs. Arc-consistency to $\beta$ are also assured from all of the other string variables, but not the other direction (from $\beta$). The reason for this is that the domains of $L_\beta, U_\beta$ are not reduced by the procedure performing the correlation constraints.

## 7.4 Searching for solutions

When all the consistency checking is done, we can start searching. This might be done using forward checking.

## 7.5 The Procedure

Now we give the whole procedure as an algorithm.

1. Define the distance constraint variables for all string variables, and initialise the domains to $[1, n]$.
2. Represent the constraints (including the implicit distance constraints from the correlation constraints).
3. 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.
4. Use the content constraints:

   **for** each content constraint on a string variable $\alpha$ **do**
   reduce the domain of $L_\alpha$ or $U_\alpha$
   **for** each correlation to another string variable $\beta$ **do**
   reduce the domain of $L_\beta$ or $U_\beta$
   **end**
   propagate to other distance constraint variables
   **end**

5. Treat the correlation constraints:

   **for** each correlation $(\alpha, \beta)$ **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 7.2.4
   use the $s\_c$-constraint and the distance constraint to find the local solutions.
   **end**

6. Reformulate the problem, and perform consistency checking on the reformulated problem.
7. Perform searching for global solutions.

### 7.5.1 Increase the efficiency

The local solutions are in the procedure found individually for each correlation. The efficiency may increase if already found local solutions are used to guide the further search for consistent local solutions to other correlations. For example, if there are two correlations $C1(\alpha, \beta), C2(\gamma, \alpha)$, the local solutions found for $\alpha$ in the first correlation should be used as guide for finding consistent solutions in the second correlation or vice versa.

## 7.6 Example

*In this example the domains are written as sets of integers. When $x \ldots y$ appears inside a set, it means all integers between and included $x$ and $y$.*

A structure is defined by four string variables $\{\alpha, \beta, \gamma, \delta\}$, where
$3 \leq length(\alpha) \leq 5, \quad 2 \leq end\_start(\alpha, \beta) \leq 4, \quad 1 \leq end\_start(\beta, \gamma) \leq 6, \quad 1 \leq end\_start(\gamma, \delta)$.

The correlations are $\gamma = I(\alpha), \quad \beta = R\_C(\alpha), \quad \delta = I(\beta)$,
where $I$ is identity, and $R\_C$ is reverse complement. There is one content constraint, the next last symbol in $\beta$ is t.

The input string is
attagtacta tagctagcta actagcgcgc tata (n=34, spaces in the presentation after each 10).

### 7.6.1 Distance constraints

From the bounds in the structure specification we get:
$2 \leq U_\alpha - L_\alpha \leq 4, \quad 2 \leq U_\beta - L_\beta \leq 4, \quad 2 \leq U_\gamma - L_\gamma \leq 4, \quad 2 \leq U_\delta - L_\delta \leq 4$, and
$2 \leq L_\beta - U_\alpha \leq 4, \quad 1 \leq L_\gamma - U_\beta \leq 6, \quad 1 \leq L_\delta - U_\gamma$

By constraint propagation and using the end positions (1 and 34), we get the following constraints:

$1 \leq L_\alpha \leq 22 \qquad 3 \leq U_\alpha \leq 24$
$5 \leq L_\beta \leq 26 \qquad 7 \leq U_\beta \leq 28$
$8 \leq L_\gamma \leq 29 \qquad 10 \leq U_\gamma \leq 31$
$11 \leq L_\delta \leq 32 \qquad 13 \leq U_\delta \leq 34$

In addition we have $U_\alpha - L_\alpha = U_\beta - L_\beta = U_\gamma - L_\gamma = U_\delta - L_\delta$

### 7.6.2 Content constraint

The next last in $\beta$ has to be a t, hence $S_{U_\beta - 1}$ =t, which means $S_{U_\delta - 1}$ =t, and $S_{L_\alpha + 1} = compl(\mathsf{t})$ =a. We then decrease the domains for these constraint variables: $D_{U_\beta} = \{10, 12, 16, 20, 24\}$, $D_{U_\delta} = \{16, 20, 24, 32, 34\}$, $D_{L_\alpha} = \{3, 6, 9, 11, 15, 19, 20\}$.

From $\alpha = \gamma$ we get $D_{L_\gamma} = \{9, 11, 15, 19, 20, 23\}$.

The effect of the decreased domains can be propagated using the distance constraints, and the result is (the notation $\{i...j\}$ is used for [i,j]):

$D_{L_\alpha} = \{3, 6, 9, 11\} \qquad D_{U_\alpha} = \{5...15\}$
$D_{L_\beta} = \{7...18\} \qquad D_{U_\beta} = \{10, 12, 16, 20\}$
$D_{L_\gamma} = \{11, 15, 19, 20, 23\} \qquad D_{U_\gamma} = \{13...27\}$
$D_{L_\delta} = \{14...32\} \qquad D_{U_\delta} = \{16, 20, 24, 32, 34\}$

### 7.6.3 Correlation constraints

To restrict the correlation constraint variables, we find (see the definition of $m, M$ in 7.2.4):

$m_{L_\alpha} = 3, M_{L_\alpha} = 11$ $\quad$ $m_{U_\alpha} = 5, M_{U_\alpha} = 15$
$m_{L_\beta} = 7, M_{L_\beta} = 18$ $\quad$ $m_{U_\beta} = 10, M_{U_\beta} = 20$
$m_{L_\gamma} = 11, M_{L_\gamma} = 23$ $\quad$ $m_{U_\gamma} = 13, M_{U_\gamma} = 27$
$m_{L_\delta} = 14, M_{L_\delta} = 32$ $\quad$ $m_{U_\delta} = 16, M_{U_\delta} = 34$

We now use the developed formulas (Point 1 and 2 in 7.2.4) to decrease some of the domains:

- For correlation $\gamma = I(\alpha)$ we find $l = 5, L = 14$, and get the following constraints ($C_d = 1$):
  - $7 \leq X_i - i \leq 18$
  - $X_{5-j} \leq 27 - j; \quad 1 \leq j \leq 3$
  - $X_{11+j} \geq 11 + j; \quad 1 \leq j \leq 5$
- For correlation $\delta = I(\beta)$ we find $l = 4$, and get the following constraints ($C_d = 1$):
  - $6 \leq X_i - i$
  - $X_{10-j} \leq 34 - j; \quad 1 \leq j \leq 3$
  - $X_{18+j} \geq 14 + j; \quad 1 \leq j \leq 5$
- For correlation $\beta = R\_C(\alpha)$ we find $l = 2, L = 4$, and get the following constraints ($C_d = -1$):
  - $2 \leq X_i - i \leq 12$
  - $X_{5-j} \geq 7 + j; \quad 1 \leq j \leq 3$
  - $X_{18+j} \leq 20 - j; \quad 1 \leq j \leq 5$

We can now find the domains for $X_i^{(I,J)}, I, J \in \{\alpha, \beta, \gamma, \delta\}$, using the developed constraints, as shown in the following table.

Note that for $X_i^{(\alpha,J)}, J \in \{\beta, \gamma\}$, $i$ is in [3,15], and for $X_i^{(\beta,\delta)}$, $i$ is in [7,20]. The third row shows the domains before the constraints are used. The forth row shows the constraints from point 1 in 7.2.4, and columns 3,5 and 7 shows the results from using point 2.

| $i$ | | | $D_i^{(\alpha,\gamma)}$ | | $D_i^{(\alpha,\beta)}$ | | $D_i^{(\beta,\delta)}$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | | | $\{11..27\}$ | | $\{7..20\}$ | | $\{14..34\}$ |
| | | | $7 \leq X_i - i \leq 18$ | | $2 \leq X_i - i \leq 12$ | | $6 \leq X_i - i$ |
| 1 | a | | | | | | |
| 2 | t | | | | | | |
| 3 | t | $\leq 25$ | $\{11, 15, 19\}$ | $\geq 9$ | $\{10, 12\}$ | | |
| 4 | a | $\leq 26$ | $\{12, 16, 20, 21\}$ | $\geq 8$ | $\{9, 11, 15\}$ | | |
| 5 | g | $\leq 27$ | $\{13, 17\}$ | | $\{8, 14\}$ | | |
| 6 | t | | $\{15, 19, 23\}$ | | $\{10, 12, 16\}$ | | |
| 7 | a | | $\{16, 20, 21, 24\}$ | | $\{9, 11, 15, 19\}$ | $\leq 31$ | $\{16, 20, 21, 24\}$ |
| 8 | c | | $\{18, 22, 26\}$ | | $\{13, 17\}$ | $\leq 32$ | $\{14, 18, 22, 26, 28, 30\}$ |
| 9 | t | | $\{19, 23\}$ | | $\{12, 16, 20\}$ | $\leq 33$ | $\{15, 19, 23, 31, 33\}$ |
| 10 | a | | $\{20, 21, 24\}$ | | $\{15, 19\}$ | | $\{16, 20, 21, 24, 32, 34\}$ |
| 11 | t | | $\{19, 23\}$ | | $\{16, 20\}$ | | $\{19, 23, 31, 33\}$ |
| 12 | a | $\geq 12$ | $\{20, 21, 24\}$ | $\leq 19$ | $\{15, 19\}$ | | $\{20, 21, 24, 32, 34\}$ |
| 13 | g | $\geq 13$ | $\{25\}$ | $\leq 18$ | $\{18\}$ | | $\{25, 27, 29\}$ |
| 14 | c | $\geq 14$ | $\{22, 26\}$ | $\leq 17$ | $\{17\}$ | | $\{22, 26, 28, 30\}$ |
| 15 | t | $\geq 15$ | $\{23\}$ | $\leq 16$ | $\{\}$ | | $\{23, 31, 33\}$ |
| 16 | a | | | | | | $\{21, 24, 32, 34\}$ |
| 17 | g | | | | | | $\{25, 27, 29\}$ |
| 18 | c | | | | | | $\{26, 28, 30\}$ |
| 19 | t | | | | | $\geq 15$ | $\{31, 33\}$ |
| 20 | a | | | | | $\geq 16$ | $\{32, 34\}$ |
| 21 | a | | | | | | |

### 7.6.4 Using the $s\_c$-constraints

Now we can, for each correlation, find for which $i$ the $s\_c$-constraint is satisfied. Note that the length of each string variable is in [3,5]. We don't show subsolutions, e.g. $\alpha = [11, 13]; \gamma = [23, 25]$ is a subsolution of $\alpha = [11, 14]; \gamma = [23, 26]$. The notation used is that $\alpha = [i, j]$ means that $\alpha$ is the substring starting at position $i$ and ending at $j$. By applying the $s\_c$-constraint to each correlation we find (Remembering the earlier reduced domains for the distance constraint variables):

1. Correlation $(\alpha, \gamma), C_d = 1$
   - $\alpha = [3, 5]; \gamma = [11, 13], [15, 17]$
   - $\alpha = [11, 14]; \gamma = [23, 26]$ Note: subsolutions not given
2. Correlation $(\alpha, \beta), C_d = -1$

- $\alpha = [3, 5]; \beta = [8, 10]$
- $\alpha = [11, 14]; \beta = [17, 20]$

3. Correlation $(\beta, \delta), C_d = 1$
   - $\beta = [7, 10]; \delta = [21, 24]$
   - $\beta = [8, 10]; \delta = [18, 20]$
   - $\beta = [8, 12]; \delta = [30, 34]$
   - $\beta = [13, 16]; \delta = [29, 32]$
   - $\beta = [17, 20]; \delta = [29, 32]$

Note that the solutions for each correlation were found independently, hence we can perform consistency-checking. However, if we use the solutions found for $\beta$ in the second correlation $(\alpha, \beta)$ as guide for the third $(\beta, \delta)$, we only find

- $\beta = [8, 10]; \delta = [22, 24]$
- $\beta = [8, 10]; \delta = [18, 20]$
- $\beta = [8, 10]; \delta = [30, 32]$
- $\beta = [17, 20]; \delta = [29, 32]$

We then have:

1. Correlation $(\alpha, \gamma), C_d = 1$
   - $\alpha = [3, 5]; \gamma = [11, 13], [15, 17]$
   - $\alpha = [11, 14]; \gamma = [23, 26]$
2. Correlation $(\alpha, \beta), C_d = -1$
   - $\alpha = [3, 5]; \beta = [8, 10]$
   - $\alpha = [11, 14]; \beta = [17, 20]$
3. Correlation $(\beta, \delta), C_d = 1$
   - $\beta = [8, 10]; \delta = [22, 24]$
   - $\beta = [8, 10]; \delta = [18, 20]$
   - $\beta = [8, 10]; \delta = [30, 32]$
   - $\beta = [17, 20]; \delta = [29, 32]$

### 7.6.5 Reformulating the problem

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 $(\beta, \gamma)$ and $(\gamma, \delta)$. We find that arc-consistency is satisfied. Hence, the whole system is arc-consistent.

### 7.6.6 Searching

Searching is now performed, and for this example all solutions are found without backtracking.

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

# 8 Further work

We are in the process of making an object-oriented design for the CSP solver based on the specification given in Section 7 and implementing it in C++. The design process includes the development of data structures and algorithms for representing and propagating constraints and for the search process.

We further intend to investigate the possibility of interfacing this solver to the high-level impementation 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 are working on a formal semantics for our language, including the semantics of negation, as well as of disjunction over both the set expressions as well as over the constraints.

More pactical testing of the language is in progress regading recognition of structures, comparing our results with those of existing systems, e.g. [SD93]. We are in the process of improving the efficiency of our system by improving our matching algorithms.

A more challenging task for the future will be to develop 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 [BJEG95].

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.).

# 9 Summary and conclusions

During this research we have investigated how constraint based techniques can be used to describe and search for patterns in sequences of symbols over finite alphabets. We have

defined a declarative constraint-based language in which a user specifies the pattern he wishes to search for. 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. We use a naive backtracking matching algorithm in this implementation which results in inefficient behaviour. However we have tested our implementation on some real biological sequences with encouraging results.

We have designed a matching algorithm based on constraint satisfaction solving techniques which will enable the user to efficiently search for structures in biological sequences.

**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

[AB91] Abderrahmane Aggoun and Nicolas Beldiceanu. Overview of the CHIP compiler system. In Koichi Furukawa, editor, *ICLP'91: Proceedings 8th International Conference on Logic Programming*, pages 775–789. MIT Press, 1991.

[AEM+84] 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.

[AWN94] 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.

[BBH95a] A. Bairoch, P. Bucher, and K. Hofman. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24(1):189–196, 1995.

[BBH95b] A. Bairoch, P. Bucher, and K. Hofman. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24:189–196, 1995.

[BC93] C. Bessiere and M.O. Cordier. Arc-Consistency and Arc-Consistency again. In *Proceedings of the AAAI*, 1993.

[BCO+95] 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.

[BG95] A. Brazma and D. Gilbert. A Pattern Language for Molecular Biology. Technical Report 11, Department of Computer Science, City University, London, 1995.

[BJEG95] A. Brāzma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. Technical Report TCU/CS/1995/18, Department of Computer Science, City University, 1995. Also Technical Report 113, Department of Informatics, University of Bergen, Bergen, Norway.

[BR96] C. Bessière and J-C. Régin. MAC and Combined Heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In E. C. Freuder, editor, *Second International Conference on Principles and Practice of Constraint Programming (CP96)*. Springer-Verlag, 1996.

[BvR95] F. Bacchus and P. van Run. Dynamic variable ordering in CSPs. In *First International Conference on Principles and Practice of Constraint Programming (CP95)*, pages 258–275, Cassis, France, 1995.

[CN95] Philippe Codognet and Guiseppe Nardiello. Enhancing the Constraint-Solving Power of clp(FD) by means of Path-Consistency Methods. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer, 1995. (Châtillon-sur-Seine Spring School, France, May 1994).

[Coo89] M.C. Cooper. An Optimal $k$-Consistency Algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[CR94] D. Clark and C. Rawlings. Constraint Satisfaction in Molecular Biology. Tutorial at ISMB-94, 1994.

[CRD94] 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.

[CRS+93] D. A. Clark, J. R. 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.

[CSR92] 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.

[DC93] D. Diaz and P. Codognet. A Minimal Extension of the WAM for clp(FD). In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 774–790, Budapest, Hungary, 1993. The MIT Press.

[Dec90] R. Dechter. Enhancedment Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[DH95] T. Dandekar and M. W. Hebtze. Finding the hairpin in the haystack: searching for RNA motifs. *TIG*, 11(2):45–50, 1995.

[DM89] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings of IJCAI*, pages 271–277, 1989.

[DM94] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[DP88] R. Dechter and J. Pearl. Network-Based heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34(34):1–38, 1988.

[DS90] 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.

[ea97] Mats Carlsson et al˙ *SICStus Prolog User's Manual Version 3.5*. Swedish Institute of Computer Science, Kista, Sweden, 1997.

[ECR95] Munich ECRC. Eclipse 3.5 User Manual, 1995.

[Eid93] I. Eidhammer. Extending Constraint Satisfaction Problems with Value Constraints. Technical Report 90, Department of informatics, University of Bergen, 1993.

[FD95] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *IJCAI'95*, pages 572–578, Montréal, Canada, 1995.

[FG89] R. Feldman and M. C. Golumbic. Constraint Satisfiability Algorithms for Interactive Student Scheduling. In *Proceedings of IJCAI*, pages 1010–1016, 1989.

[FHK+92] Thom Frühwirth, Alexander Herold, Volker Küchenhoff, Thierry Le Provost, Pierre Lim, Eric Monfroy, and Mark 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).

[FM95] M. Foucrault and F. Major. Symbolic Generation and Clustering of RNA 3-D Motifs. In C. Rawlings, D. 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.

[Fre82] E. C. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29(1):24–32, 1982.

[Gas77] J. A. Gaschnig. A general Backtracking Algorithm that Eliminates Most Redundant Tests. In *Proceedings of the Fifth IJCAI*, 1977.

[Ger94] 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.

[Gre94] K. A. Gregorz. A Theoretical Evolution of Selected Backtracking Algorithms. Technical report, Department of Computer Science, University of Alberta, Canada, 1994. 52 pages.

[GW94] 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.

[Ham82] R. Hamming. *Coding and Information Theory.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[HD91a] 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.

[HD91b] 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.

[Hen89] P. V. Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[HL88] C-C. Han and C-H. Lee. Comments on Mohr and Hendersons's Path Consistency Algorithm. *Artificial Intelligence*, 36:125–130, 1988.

[HS93] 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.

[JMSY92] Joxan Jaffar, Spiro Michayov, Peter Stuckey, and Roland Yap. The CLP($\mathcal{R}$) Language and System. *TOPLAS: ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[LB92] S. Letovsky and M. B. Berlyn. CPRPO: A rule-based program for constructing genetic maps. *Genomics*, 12:435–446, 1992.

[Lev65] 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.

[LGF95] 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. Rawlings, D. 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.

[McG79] J. McGregor. Relational consistency algorithms and their applications in finding subgraphs and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.

[Mes89] Pedro Meseguer. Constraint Satisfaction Problems: An Overview. *AICOM*, 2(1):3–16, 1989.

[MM93] G. Mehldau and G. Myers. A system for pattern matching applications on biosequences. *CABIOS*, 9(3):299–314, 1993.

[Mon74] U. Montanari. Network of constraints: fundamental properties and applications to picture processing. *Information Sciences*, (7):95–132, 1974.

[MTG+91] 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.

[Nad88] B. A. Nadel. Constraint Satisfaction Algorithms. Technical report, Wayne State University, 1988. CSC-88-005.

[Nud83] B. Nudel. Consistent-Labeling Problems and their Algorithms: Expected- Complexities and Theory-Based Heuristics. *Artificial Intelligence*, 21:135–178, 1983.

[Par95] S. Parsons. Softening constraints in constraint-based protein topology prediction. In C. Rawlings, D. 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.

[Pou95] Dick Pountain. Constraint Logic Programming. *BYTE*, Feb 1995.

[Pro93] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, (9):268–299, 1993.

[Pro99] P. Prosser. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping. Technical Report 95-177, Department of Computer Science, University of Startclyde, 1999.

[Raj94] 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.

[Rat96] M. Ratnayake. Constrained Pattern Recognition in Biosequences. Department of Computer Science, City University, London, 06 1996. B.Eng. (Honours) Degree in Software Engineering.

[SA90] P. R. Sibbald and P. Argos. Scrutineer: a computer program that flexibly seeks and describes motifs and profiles in protein sequences databases. *CABIOS*, 6(3):279–288, 1990.

[SBH+94] 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 Res*, 22:5112–5120, 1994.

[SD93] D. B. 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.

[Sea95] D. Searls. The Computational Linguistics of Biological Sequences. Tutorial at Third International Conference on Intelligent Systems for Molecular Biology, 1995.

[SF94] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *PPCP'94: Second Workshop on Principles and Practice of Constraint programming*, Seattle WA, May 1994.

[SP87] S. A. Schuman and D. H. Pitt. Object oriented subsystem specification. In Meertens, editor, *Program Transformation: Proc. IFIP Working Conf.* North Holland, 1987.

[SSA92] P. R. Sibbald, H. Sommerfeldt, and P. Argos. Overseer: a nucleotide sequence searching tool. *CABIOS*, 8(1):45–48, 1992.

[Sta90] 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.

[Ste78] M. Stefik. Inferring DNA Structures from Segmentation Data. *Artificial Intelligence*, 11:85–114, 1978.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[Wal89] C. Walinsky. CLP($\Sigma^*$): 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.

[ZKM93] 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.

[Zuk89] Michael Zuker. On Finding All Foldings of an RNA Molecule. *Science*, 244:48–52, 1989.