Protein Topology Prediction through Parallel Constraint Logic Programming

Dominic A. Clark[1], Christopher J. Rawlings[1], Jack Shirazi[1], André Veron[2], and Mike Reeve.[2]

1. Biomedical Informatics Unit, Imperial Cancer Research Fund Laboratories, P.O. Box 123, Lincoln's Inn Fields, London, WC2A 3PX, UK.

2. European Computer-Industry Research Centre, Arabellastrasse, 17, D-8000, Munich, Germany.

dac@biu.icnet.uk, cjr@biu.icnet.uk, js@biu.icnet.uk, veron@ecrc.de, mjr@ecrc.de

Abstract

In this paper, two programs are described (CBS1e and CBS2e). These are implemented in the parallel constraint logic programming language ElipSys. These predict protein α/β-sheet and β-sheet topologies from secondary structure assignments and topological folding rules (constraints). These programs illustrate how recent developments in logic programming environments can be applied to solve large-scale combinatorial problems in molecular biology. We demonstrate that parallel constraint logic programming is able to overcome some of the important limitations of more established logic programming languages i.e. Prolog. This is particularly the case in providing features that enhance the declarative nature of the program and also in addressing directly the problems of scaling-up logic programs to solve scientifically realistic problems. Moreover, we show that for large topological problems CBS1e was approximately 60 times faster than an equivalent Prolog implementation (CBS1) on a sequential device with further performance enhancements possible on parallel computer architectures. CBS2e is an extension of CBS1e that addresses the important problem of integrating the use of uncertain (weighted) protein folding constraints with categorical ones, through the use of a cost function that is minimized. CBS2e achieves this with a relatively minor reduction of performance. These results significantly extend the range and complexity of protein structure prediction methods that can reasonably be addressed using AI languages.

Introduction

The most well developed methods for predicting the three dimensional structure of a protein from its amino acid sequence use model building by homology with a protein of known structure (e.g. Blundell *et al.* 1987). This method, however, requires that a protein exists that has an adequately resolved tertiary structure and an amino acid sequence that is sufficiently similar (typically > 30% identity) to the protein being studied, despite the fact that some classes of structurally similar proteins have very low sequence identity. In the absence of any strong sequence similarities, however, it is difficult to identify known structures suitable for model-building. As the number of protein sequences determined directly or by cDNA is increasing considerably more rapidly than the number of protein structures being solved by NMR or x-ray crystallography, the likelihood that the structure of any newly-determined sequence can be predicted by model-building is diminishing. Most software being developed for predicting protein structure does not directly address the problem of how to derive the most plausible and consistent predicted structure in the absence of a structurally-based sequence homology. It is not uncommon, however, for a laboratory scientist to collaborate with a protein structure specialist in order to combine their skills and, with the help of standard or specially constructed software, manually generate a plausible protein structure hypothesis.

From an analysis of a number of papers that were representative of this type of research we have argued (Clark *et al.*,1990b) that in the absence of an homologous protein of known structure, the predominant prediction strategy can be characterized as constraint generation and satisfaction. In these papers, a topological description of the protein structure was utilised both as a partial description of higher level structure, but also as a means to express commonly held beliefs about protein folding principles, both in general and in the specific case of the class of proteins being studied.

Protein topology denotes a level of protein structural organization intermediate to secondary and tertiary levels, based on sequential, adjacency and orientation relations between secondary structures (α-helices and β-strands in β-sheets). A topological prediction of a protein of unknown structure can also potentially be used for identifying proteins with similar topology which may provide a basis for model building.

Another important characteristic of the papers that we studied was that a great variety of knowledge about the protein was used in the considerations of which possible structures were plausible. This knowledge included evidence from biochemical experiments such as inhibition studies, site-directed mutagenesis and competitive antibody-binding as well as biophysical analysis of the protein by methods such as circular dichroism spectroscopy. This evidence was used to provide further constraints on the set of plausible structures.

We believe that the richness and variety of the knowledge used to predict protein structure in this way and the requirement to generate plausible structures and evaluate

This research was funded by the ICRF, ECRC and the CEC under ESPRIT III project 6708 "APPLAUSE"

them against all the available evidence clearly requires support from intelligent scientific decision support software. We also believe that a systematic investigation into the use of a wide variety of knowledge about protein structure will ultimately lead to more robust and hopefully also more accurate prediction methods.

In adopting a constraint-based approach to the protein structure prediction problem we both follow in the footsteps and extend the classic AI tradition for applications in the molecular sciences which goes back to DENDRAL (Buchanan and Feigenbaum, 1978) and extends through the MOLGEN project (Stefik, 1981a,b), the PROTEAN system (Hayes-Roth, 1987) to more recent programs for predicting RNA structure (Major *et al.* 1991).

In this paper we wish to illustrate the use of some recent developments in advanced logic programming (LP) methods that seem ideally suited to the development of intelligent systems in molecular biology.

Combinatorial Analysis of β-sheet topology

A principal difficulty when predicting protein topology from secondary structure is that, in the absence of other constraining information, a vast number of topological conformations can potentially result from a single set of secondary structure assignments for a given topological folding class. Making the simplifying assumptions of considering: (a) only neighbourhood relations between strands (b) only left or right handed parallel (crossover) connections (c) only one type of antiparallel (hairpin) connection between consecutive strands, then a β-sheet or α/β-sheet of n strands has p possible strand topologies where p is given by $\mathbf{p= n!(3^{n-1})/2}$ (Table 1). Specifically, there are **n!/2** rotationally invariant strand orderings, and **n-1** connections each potentially of three types (hairpin, cross over left handed, cross over right handed).

Table 1: Number of topologies as a function of number of strands for single α/β-sheets and all β-sheets

strands (n)	2	3	4	5	6	...	10
topologies (p)	3	27	324	4,860	87,480	...	3.57×10^{10}

Much greater complexity arises where other aspects of structural organisation are considered, such as the layering of crossed connections, differing hairpin connections or extended class of topological folds.

Protein Topology Prediction as Constraint Satisfaction

Constraint satisfaction problems can be characterized as consisting of:

- objects (in protein topology, these are β-sheet, β-strands, α-helices etc).
- variables associated with those objects (i.e. the position of a strand in a sheet, it's orientation, the handedness of the connections to sequentially adjacent strands etc).
- values associated with those variables, and
- constraints on the values that variables of particular objects can take (i.e. protein folding rules expressed as ElipSys constraints).

The goal of constraint satisfaction algorithms is to find values for the variables for each object such that all constraints are simultaneously satisfied. A review of constraint satisfaction algorithms is provided in Nadel (1990).These can broadly be characterized by arc-consistency, search-based and hybrid methods.

We have previously shown that the LP language Prolog can be used to implement constraint-satisfaction algorithms (CBS1, Clark *et al.* 1991) for predicting protein topology from protein secondary structure. CBS1 was based on breadth-first tree search with pruning for rotational invariance and high level constraint evaluation. CBS1 also exploited the representation scheme developed for the TOPOL database of protein topology (Rawlings *et al*, 1985) with constraints represented as Prolog predicates exploiting the TOPOL relations. The advantage of this common representation scheme was that it enabled an evaluation of the validity and utility of the various topological folding rules used by Taylor and Green (1989) by direct comparison with known structures.

This paper illustrates the potential of parallel constraint logic programming (CLP) in ElipSys by describing developments of CBS1 that address some of its limitations; in particular the need to further improve the performance of the algorithm so that the constraint satisfaction methods can accommodate more complex protein combinatorics and a more sophisticated constraint satisfaction algorithm to deal with uncertain (partial) constraints.

Methods

Constraint Logic Programming

CLP can be understood as a generalization of LP where the basic operation of unification is replaced by constraint checking (Jaffar and Lassez, 1987; Van Hentenryck, 1991). CLP integrates efficient constraint-solving methods from algebra, artificial intelligence, operations research and logic to support, in a declarative way, computational paradigms such as partial enumeration, constraint satisfaction and branch and bound search. These employ both search and consistency methods. CLP languages therefore combine the advantages of LP (declarative semantics, relational form and non-determinism) with the efficiency of special purpose problem-solving (search) algorithms.

There are two main approaches for integrating constraints into logic programming. The first approach, formalised as CLP(X) (Jaffar and Lassez, 1987), is to replace the usual domain of computation with a new domain X.

From a user's perspective CLP(X) programs have great expressive power due to the constraints which they naturally manipulate. Program design is more intuitive because the programmer works directly in the domain of discourse. Moreover, the runtime efficiency of the resulting program is also enhanced because of the exploitation of computation techniques over specific domains.

```
constraint(c3,PositionList):-
    max_changes_in_winding(Atmost),
    changes_in_winding(PositionList,Current),
    Current #<= Atmost.

max_changes_in_winding(1).

false_constraint(c3,PositionList):-
    max_changes_in_winding(Atmost),
    changes_in_winding(PositionList,Current),
    Current #> Atmost.

?- parallel changes_in_winding/2.
changes_in_winding([P1,P2,P3|T], 0 + Current):-
    monotonic(P1,P2,P3),
    changes_in_winding([P2,P3|T],Current).
changes_in_winding([P1,P2,P3|T], 1 + Current):-
    nonmonotonic(P1,P2,P3),
    changes_in_winding([P2,P3|T],Current).

changes_in_winding([],0).
changes_in_winding([_],0).
changes_in_winding([_,_],0).

?- parallel monotonic/3.
monotonic(P1,P2,P3):-
    P1 #> P2, P2 #> P3.
monotonic(P1,P2,P3):-
    P1 #< P2, P2 #< P3.

?- parallel nonmonotonic/3.
nonmonotonic(P1,P2,P3):-
    P1 #> P2, P2 #< P3.
nonmonotonic(P1,P2,P3):-
    P1 #< P2, P2 #> P3.
```

Figure 1: ElipSys Representation of constraint c3. Note that (a) Negation conditions are explicitly defined (`false_constraint/2`, `nonmonotonic/3`), (b) non deterministic constraints can be annotated as parallel (`changes_in_winding/2`, `monotonic/3`, `nonmonotonic/3`), (c) The arithmetic constraints in `monotonic/3` and `nonmonotonic/3` implicitly delay.

The second main approach to integrating constraints in logic programming uses the standard, syntactic, domain of computation, except that the variables may be restricted to explicitly range over finite subsets of the universe of values (finite domain variables, Van Hentenryck and Dincbas 1986). In this approach, inaugurated by CHIP (Dincbas *et al* 1988), it is the proof system that is extended, using techniques from Artificial Intelligence. The more recent type of controlled inference is termed generalized propagation. Informally, generalized propagation aims at exploiting program-defined predicates as constraints. It operates by looking ahead at yet unsolved goals to see what locally consistent valuations there remain for individual problem variables. Such constraint techniques can have a dramatic effect in cutting down the size of the search space (Le Provost and Wallace, 1992).

ElipSys

ElipSys (ECRC Logical Inferencing Parallel System) is the most recent of a series of CLP languages developed at ECRC that builds upon the experience of CHIP and which employs generalized propagation on terms and parallelism. ElipSys parallelism provides the opportunity for parallel evaluation during search through process parallelism. The parallel execution model employed by ElipSys allows programs to be run concurrently on shared memory and distributed memory computer architectures. Moreover, the declarative nature of the language hides most of the details of the parallel programming task so that load-balancing and synchronization are invisible to the application designer. The exploitation of parallelism is enabled by user annotations of non-deterministic predicates (Figure 1) or through the use of built-in parallel predicates such as `par_member/2`. A useful way of viewing the interplay between the constraint propagation and parallel execution strategies is that the constraints provide the mechanism for *a priori* pruning of the hypothesis space through propagation, while parallelism allows the search tree it to be traversed more efficiently.

Data Structure & Representation of Constraints.

The protein folding constraints in CBS1e (Table 2) are taken from Taylor and Green (1989).

Table 2: Protein Folding Constraints used by CBS1e

Name	Description and Reference
c1	For parallel pairs of β-strands, β-α-β and β-coil-β connections are right handed (Richardson, 1977; Sternberg and Thornton, 1977)
c2	The initial β-strand in the amino acid sequence is not an edge strand in the sheet (Branden, 1980)
c3	Only one change in winding direction occurs (Richardson, 1981)
c4	The β-strands associated with the conserved patterns lie adjacent in the sheet (Walker *et al.* 1982).
c5	All strands lie parallel in the β-sheet.
c6	Unconserved strands are at the edge of the sheet.
f2	Parallel β–coil–β connections should contain at least 10 residues in the coil.

The representation of β–sheets in CBS1e is similar to that of CBS1. Specifically, a sheet of N strands is represented as a list of N terms each describing the way in which one strand is positioned in the sheet using the notation: `position(P,S,O,C)`, where `P` is the position of that strand in the sheet template in the range *[1..N]*, `S` refers to the sequential position of the strand in the sequence in the range *[1..N]*, `O` is the orientation of the strand (up or down) and `C` is the nature of the connection to the preceding strand (left cross over, right cross over, hairpin or undefined).

```
generate_topologies(N,Constraint_list,Topologies_List):-
    create_strands_template_with_N_slots(N,Topologies_List),
    apply_unary_logical_constraints(Topologies_List),
    apply_set_theoretic_logical_constraints(N,Topologies_List)),
    order_empirical_constraints(Constraint_List,OC_List),
    apply_empirical_constraints(N,OC_list,Topologies_List),
    instantiate(Topologies_List).

apply_unary_logical_constraints([]).
apply_unary_logical_constraints([position(F,Y,Z,C)|T]):-
    first_strand_is_orientation_up(Y,Z),
    first_strand_is_of_undefined_chirality(Y,C),
    apply_unary_logical_constraints(T).

apply_set_theoretic_logical_constraints(N,Topologies_List):-
    all_different_template_positions(Topologies_List),
    strictly_orderedYs(Topologies_List),
    first_strand_is_on_the_left(N,Topologies_List).
```

Figure 2: Top Level Schema of CBS1e: The strategy of "constraint and generate" involves (a) defining hypothesis space by producing a solution template, (b) applying logical constraints to this template which are part of the problem structure, then ordering and applying empirical constraints and finally instantiating any solution templates with domain variables.

Figure 1. shows the ElipSys representation of constraint c3. There are two essential differences between an ElipSys and Prolog specifications reflecting the difference between parallel CLP and LP.

- In ElipSys, equalities and disequalities may be specified as arithmetic constraints which are not simply instantiated and then checked, but which are implemented using a look ahead inference rule. This means, for example, that it is not necessary to compute the total number of changes in winding direction before comparing against the maximum. Further, such arithmetic constraints are implicitly delayed (see goal suspension below)
- In ElipSys, the predicates with choice points (`changes_in_winding/2`, `monotonic/3` and `nonmontonic/3`) may be annotated as suitable for execution in parallel.

Goal Suspension

The standard execution strategy of Prolog and similar LP systems is the so called "fixed left to right & depth-first" strategy in which subgoals are executed in the order specified. This procedural model of Prolog execution is important for some aspects of programming, particularly those to do with input/output, but they can obscure the declarative semantics when a more abstract representation of a problem is required. This is often the case when developing a knowledge-based program.

This problem is most easy to understand with the use of arithmetic operators. For example, when using the disequality operator as in *"A > B"* where *A* and *B* are two logical variables it is required that both variables are instantiated before the statement can be evaluated. This means that arithmetic statements in general (in Prolog) have to be confined to the last subgoals in the body of a predicate. In ElipSys it is possible to preserve more of the declarative semantics of a program because the runtime system uses a goal suspension mechanism in which the order of subgoals execution depends on the state of instantiation of the arguments rather than their textual position in the source program. In Figure 1, for example, the constraint P1 #> P2 will be "woken up" and used in propagation only when either P1 or P2 becomes ground.

CBS1e

A natural way to express a CLP program is to employ the constrain and generate paradigm. This involves (a) defining the hypothesis space, (b) specifying the constraints on that hypothesis space and (c) initiating a search (or labelling) procedure. In CBS1e, we distinguish between logical constraints which are part of the problem specification and empirical constraints which are independent of the problem specification *per se*, but which are relevant (selected by the user or program to constrain the hypothesis space, Figure 2).

The CBS1e algorithm operates as follows

1. Creation of a sheet template with the appropriate number of strand slots.
   The call `create_strands_template_with_N_slots(4,Solution)` causes ElipSys to unify the logical variable `Solution` with a list of finite domain terms which represents the possible assignment of (in this case) a β–sheet with 4 strands. This is thus the situation before the finite domains (variable lists of possible values of a variable as a list in curly braces "{}")

```
generate_topologies(N,Constraint_list,Topologies_List):-              % (++,++,-)
    create_strands_template_with_N_slots(N,Topologies_List),
    apply_unary_logical_constraints(Topologies_List),
    apply_set_theoretic_logical_constraints(N,Topologies_List),
    order_empirical_constraints(Constraint_List,OC_List),
    copy_term(Topologies_List,Min_Cost_Topology),
    determine_min_cost(N,OC_list,Min_Cost_Topology,Min_Cost,_TCS),
    generate_according_to_cost(N,OC_list,Topologies_List,Min_Cost,TCS).

determine_min_cost(N, OC_list, Min_Cost_Topology, Min_Cost, _TCS):-   % (++,++,+,-,-)
    build_cost_function(OC_list,ListCostVars,Costfunction),
    minimize(
        (apply_empirical_constraints(N,OC_list,Min_Cost_Topology,ListCostVars, _TCS),
        instantiate(Min_Cost_Topology)),
        Costfunction),
    Min_Cost is Costfunction.

generate_according_to_cost(N,OC_list,Topologies_List,Min_Cost,TCS):-  % (++,++,+,-,-)
    build_cost_function(OC_list,ListCostVars,Costfunction),
    Costfunction #= Min_Cost,
    apply_empirical_constraints(N,OC_list,Topologies_List,ListCostVars, TCS),
    instantiate(Topologies_List).
```

```
?- parallel eval_constraint / 4.
eval_constraint(Constraint,Topologies_List,TrueScore,true(Constraint)):-
    cost_val(Constraint,TrueScore,_),
    eval_constraint_true(Constraint,Topologies_List).
eval_constraint(Constraint,Topologies_List,FalseScore,false(Constraint)):-
    cost_val(Constraint,_,FalseScore),
    eval_constraint_false(Constraint,Topologies_List).
```

Figure 3: Top Level Schema of CBS2e: The "constraint and generate" strategy of CBS1e is extended to (a) firstly determine the cost of a minimum cost topology defined as a weighted sum across all empirical constraints and (b) determine all solutions with this cost. TCS is the particular Truth Conditional State of the constraints. The symbols '++', '+' and '-' are comments with '++' indicating variable is ground, '+' partially instantiated (in this case as finite domain variables) and '-' uninstantiated. Parallelism is exploited in the evaluation of constraints through a non-deterministic predicate eval_constraint/4 which may evaluate constraints as true or false returning he appropriate cost and truth conditional state.

have been reduced by the application of any constraints:

```
Solution=[
  position(_{1,2,3,4},_{1,2,3,4},_{u,d}, _{l,r,a,u}),
  position(_{1,2,3,4},_{1,2,3,4},_{u,d}, _{l,r,a,u}),
  position(_{1,2,3,4},_{1,2,3,4},_{u,d}, _{l,r,a,u}),
  position(_{1,2,3,4},_{1,2,3,4},_{u,d}, _{l,r,a,u})
].
```

2. Application of logical (structural) constraints.
   In this formulation of β–sheet topology the first strand is arbitrarily assigned to the left hand side the sheet and oriented up (with undefined chirality). This constrains two independent degrees of rotational invariance. The other strands have either left or right handed connections. A further logical constraint ensures that all strands are assigned different template positions and that the solution gives the strand numbers in sequence order. The hypothesis space is therefore reduced through propagation to:

```
Solution=[
  position(_{1,2},     1,   u,      u),
  position(_{1,2,3,4}, 2, _{u,d}, _{l,r,a}),
  position(_{1,2,3,4}, 3, _{u,d}, _{l,r,a}),
  position(_{1,2,3,4}, 4, _{u,d}, _{l,r,a})]
```

Note that this provides the most general representation of the hypothesis/solution space.

3. Constraint Ordering.
   In general the order in which constraints are applied in a search algorithm can affect the performance of that algorithm (e.g. see Clark *et al*, 1991). In CBS1e, the empirical constraints are ordered such that those which are non-deterministic (contain choice points) are evaluated last, in this case c3. The rationale for this is to maximize the amount of resolution (propagation) that the system performs to minimize the search space. This is a principal difference between CLP and LP approaches. To take a trivial example, if we have A and B are finite domain variables {1,2,3} and 2*A #= B, then

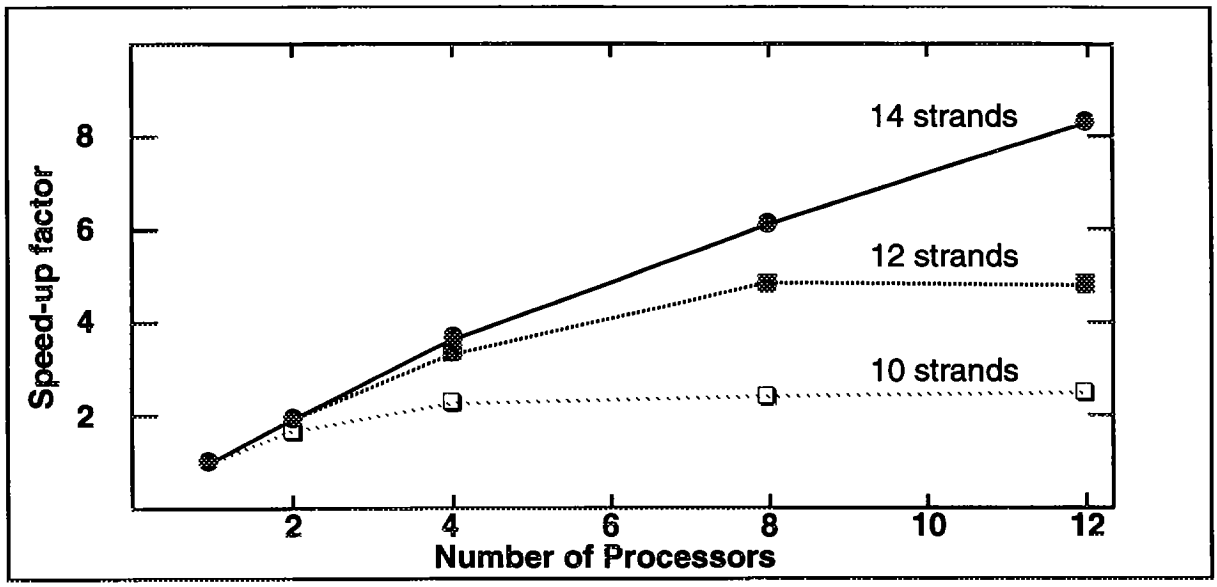

Figure 4: Benefits of Parallelism as a function of search Space. The graph shows the speed-up due to parallelism corresponding to three cases of increasing protein size (search space). Results show that the advantages of parallelism increase with search space.

ElipSys immediately infers A=1, B=2. There is no search.

4. Application of Empirical Constraints
   The empirical constraints are then applied and their effects progagated through the search tree, with the following partial solution generated as constraints are added:

With constraints c2 only (the first strand is non edge) we have:

```
Solution=[
  position( 2,      1,   u,      u),
  position(_{1,3,4}, 2, _{u,d}, _{l,r,a}),
  position(_{1,3,4}, 3, _{u,d}, _{l,r,a}),
  position(_{1,3,4}, 4, _{u,d}, _{l,r,a})]
```

With constraints c5 (all strands are parallel), c1 (all parallel connections are right handed), and c2 we have:

```
Solution=[
  position(2,       1, u, u),
  position(_{1,3,4}, 2, u, r),
  position(_{1,3,4}, 3, u, r),
  position(_{1,3,4}, 4, u, r)]
```

With constraints c1,c2, c5 and c3 (at most 1 change in winding direction) and only now introducing labelling/ search, we have the following set of solutions:

```
Solution1=[
  position(2, 1, u, u),
  position(1, 2, u, r),
  position(3, 3, u, r),
  position(4, 4, u, r)];

Solution2=[
  position(2, 1, u, u),
  position(3, 2, u, r),
  position(4, 3, u, r),
  position(1, 4, u, r)];

Solution3=[
  position(2, 1, u, u),
  position(4, 2, u, r),
  position(3, 3, u, r),
  position(1, 4, u, r)]
```

CBS2e: Protein Topology Prediction with Weighted Constraints.

Many of the empirical constraints employed by molecular biologists in published predictions of protein topology are non categorical, being violated by known structures (Clark *et al* 1991). The ability to represent and reason with this uncertainty in PTP is important in order to (a) produce the least uncertain prediction or set of predictions and (b) to resolve potential inconsistencies between secondary structural data and constraints. CBS2e is an extension of CBS1e that deals with weighted constraints (Figure 3). Its operation is an extension to CBS1e in which:

(a) The failure conditions of each constraint must be each specified in addition to the success conditions (i.e. `false_constraint/2` in Figure 1).

(b) Weight assignment: Each constraint is assigned a pair of values (weights or penalties) corresponding to the truth conditional states of the constraint (i.e. a value for the constraint being true and a value for the constraint being false) which are each positive integer.

As an illustration, Table 3 gives a set of weight assignments based on the empirical analysis of 8 nucleotide binding proteins by Clark *et al.* (1991)

Table 3: Weights for Topological Folding Constraints

Constraint	Penalty if false [0,8]	Penalty if true [0,8]
c1	8	0
c2	7	1
c3	6	2
c5	5	3
f2	6	1

(c) Cost function definition. The cost assigned to any set of topological hypotheses is defined as the sum of the weights (penalties) corresponding to the truth conditional states of the constraints to which it corresponds. In Table 3, for example, the set of topologies for which all constraints are true would have a total cost of 9 assuming a simple cost function.It should be emphasized that although adoption of a linear cost function makes the assumptions that the constraint weights can be combined independently and additively, many more complex weighting procedures can be reduced, via the appropriate transformation, to a linear cost function (e.g. probabilities though the use of a function such as mod(-K*log*(odds))).

(d) Cost minimization: A minimum value of this cost function is then determined using an ElipSys built-in branch and bound procedure (`minimize/2` in Figure 3).

(e) Finally all topologies with this minimum value are generated using the same propagation/search procedure called without the minimization function but with the additional explicit constraint that the precise value of the cost function be equal to the minimum already determined.

Results

Comparison of CBS1 and CBS1e.

CBS1 and CBS1e were evaluated on the set of benchmarks reported in Clark *et al.* (1991) using a Sun SparcStation1 (Table 4). Results showed that for small problems CBS1e yielded a modest improvement over the CBS1 algorithm. However, for a larger Benchmark III, where the required solution involves identifying 28 valid topologies from a potential $3.57.10^{10}$, CBS1e was approximately 60 times quicker:.

Table 4: Benchmark Comparison between CBS1 and CBS1e

Benchmark III[a]	**CBS1**: 136s	**CBS1e**: 2.26s

a. Find all solutions for a ten stranded sheet with constraints [c1,c2,c3,c5,c6], Clark *et al* (1991).

Parallelism in CBS1e

The potential benefits of parallelism were investigated on a Sequent Symmetry with 12 Intel 86386 processors using benchmark III, or variants thereof. This was done using the predicates marked as parallel in Figure 1 and by replacing the definition of `instantiate/2` with a call to `par_member/2` in the code listed in Figure 2. Extensions to the search space were simulated by running the same query with 12 and 14 strands giving 45 and 66 solutions respectively. Median speed-up figures are shown in Figure 4.

Using speed-up as the dependent variable and size and number of processors as the independent variables, Analysis of Variance for this data showed:

- Larger queries took longer, $F(2,30) = 50.72$, $p < 0.001$.
- Speed-up increased with the number of processors $F(4,30) = 78.38$, $p < 0.001$.
- There was a significant interaction between the size of the problem and the number of processors in determining speed-up ($F(8,30) = 13.13$, $p < 0.001$), such that the benefits of parallelism increased with search space. Thus parallelism offers greater advantages for larger search spaces.

More recently a similar pattern of results has been found for much larger search spaces using an ICL DRS6000, with 4 SPARC2 processors, extending the analysis up to 24 simulated processors.

Performance of CBS2e

To determine the overhead associated with the use of weighted constraints relative to categorical constraints, benchmark III was evaluated by CBS2e using the weights in Table 2, for c1, c2, c3, c5 and f2, arbitrarily assigning weights of {5,3} to c4 and c6 (where detailed conservation data is not available). For this query, the minimum cost is 9 corresponding to the truth functional state in which all constraints are true. CBS2e found this cost and the 28 solutions and corresponding truth functional states in a median time of 4.42s, representing a substantial increase in functionality for a small performance overhead (factor of 2 relative to the time for CBS1e). For different cost matrices in which the minimum total cost may occur when some constraints are false the overhead can vary.

Because this approach uses a branch and bound algorithm, it has a worse case complexity of O^n, where n is the number of constraints. However, since the number of constraints is relatively small, this is not currently viewed as a limitation.

Predicting the Topology of a Nucleotide Binding Domain with CBS2e

The prediction of the topology of the nucleotide binding domain of cation transporting ATPases by Taylor and Green (1989) was reconsidered using the weights in Table 2 for {c1,c2,c3,c5,f2} with weights for c4 and c6 were arbitrarily given the values {5,3}. As stated above the minimum value for the cost matrix occurs when all constraints are true. Thus, running the query for one of their sets of secondary

structure assignments produced the result in Figure 5, endorsing their original prediction:

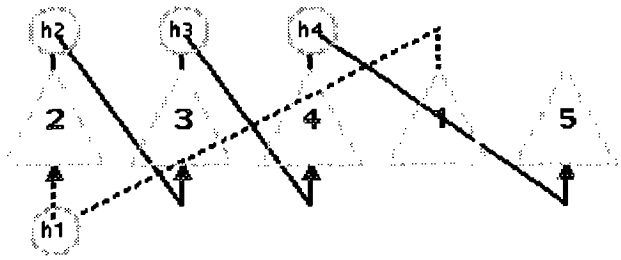

Figure 5: Topological prediction for the nucleotide binding domain of cation transporting ATPases using CBS2e and weighs from Table 2. Note that in this diagram strands are represented as triangles, helices as circles and connections as lines either filled (above the plane of the sheet) or dotted (below the plane of the sheet). The relative orientations of strands are indicated by the position of the vertex.

This is the same solution as obtained with CBS1e in which constraints are treated as categorical. However, now for illustrative purposes only, we are able to vary the weights such that if the weights for c2 were changed from {1,7} to {7,1}, the following solution (Figure 6) would be generated, demonstrating the sensitivity of the solution set to the precise weights applied.

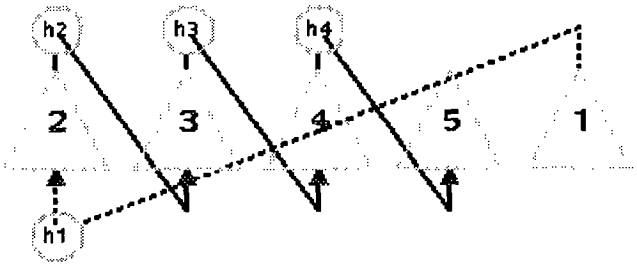

Figure 6: Topological prediction for the nucleotide binding domain of cation transporting ATPases using CBS2e and an alternative weigh set.

Discussion and Conclusions

In this paper we have used protein topology prediction as an application having many of the characteristics that make building intelligent knowledge-based systems in molecular biology a challenge. We have focused on three aspects of the use of the ElipSys parallel CLP language: the representation of the protein folding rules and other features using finite domain constraints, the potential for building large scale, scientifically realistic applications and an approach to uncertainty management using one of the special purpose optimization algorithms (branch and bound).

The use of finite domain constraints are central to the implementation of CBS1e and CBS2e. They seem relatively straightforward to use and complement the deductive features of ElipSys as a logic programming language, particularly when used in the representation of constraints most naturally expressed in terms of discrete values (e.g. the positions of objects). The goal suspension mechanisms enable arithmetic constraints to be expressed in a more declarative style than would be the case in traditional LP languages.

By comparing the performance of the ElipSys and Prolog implementations we have showed that the constraint handling enables significant increases in performance and that further increases in speed can be achieved, particularly in the case of large problem spaces, by exploiting parallelism on multi-processor computer systems. We are very encouraged by these results and believe that the paradigm of "constrain-and-generate" (in contrast to the more common "generate and test") is much more suited to intelligent systems design for molecular biology.

By using ElipSys and with these significant improvements in the efficiency of existing methods of PTP we are in a position to significantly extend the set of problems that may be addressed - particularly those generating higher order combinatorics. More specifically we can consider addressing the following problems which will be potentially important developments in protein structure prediction:

- allow for multiple secondary structural assignments to be considered simultaneously.
- explore α/β-sheets at a greater level of detail, and
- simultaneously consider multiple possible topological motifs using the appropriate parameterization (e.g. Murzin and Finkelstein, 1988, for globular α-helical proteins).

The extensions to CBS1e in the program CBS2e showed that without significant loss of performance, we could enhance the functionality to provide a methodology for handling uncertainty in the protein folding rules. This is an important result from the point of view of developing scientifically realistic software.

The technique used for handling uncertainty in CBS2e also provides a mechanism for resolving inconsistencies between sets of constraints and the input data (secondary structure assignments) when these can be assigned confidence factors. It has not escaped our attention that the method therefore potentially provides a means of revising (selecting among) secondary structure predictions if these can be assigned costs commensurate with those attached to the constraints. A specific example being to distinguish between alternative consensus secondary structure predictions for sets of aligned sequences.

For non-weighted potentially inconsistent constraints, CBS2e can be used if uniform weights are assumed. The result is then the set of topological structures which contradict the least number of constraints and might be therefore considered the most plausible structures.

In the longer term it is anticipated that research on PTP can also be employed to enhance protein secondary structure prediction by identifying parts of the predicted secondary

structure that lead to inconsistencies when attempting to apply sets of reliable folding rules. For example, if there are grounds for believing that a pair of predicted strands are parallel and it is known that all loop regions between such strands contain at least ten residues, and the observed number of residues between the predicted strands is less than 10, then one way to resolve the inconsistency is to adjust the termini of the predicted strands to make the loop region ten residues or more. An alternative strategy might be to reassign one of the predicted strands as helix or loop.

A further feature of ElipSys that makes it attractive as an environment for building practical intelligent systems in molecular biology is that it is implemented on the top of the Megalog deductive database which provides efficient storage and retrieval of very large logic databases. In order to extend and evaluate the protein folding rules that we can represent as constraints and incorporate in CBS2e we have developed a large deductive database of protein topology. This database presently consists of ~500,000 clauses derived from the Brookhaven database (Bernstein *et al.*, 1977) release 60 in ElipSys/MegaLog. This database contains data similar to that used by the **TOPOL** system (Rawlings *et al.* 1985), but is specialised for the purposes of assessing the validity of folding rules against representative sets of structures and identifying known structures that are similar to predicted protein topologies. Because of the fast access times, sets of constraints can be evaluated against the entire database in a matter of seconds.

Finally, it is anticipated that the control structures outlined in this paper can be generalised to cover a number of other constraint satisfaction problems in molecular biology and genetics for example genetic map construction and restriction mapping.

Acknowledgements: We would like to thank Dr. Mark Wallace (ECRC) and two anonymous referees for comments on an earlier draft of this paper. We would also like to thanks all members of the ElipSys team at ECRC.

References

Bernstein, F. C. Koetzle, T. F. Williams, G. J. B. and Meyer, Jr, E. F. Brice, M. D. Rodgers, J. R. Kennard, O. Shimanouchi, T. and Tasumi, M. (1977) **The Protein Databank: A Computer-Based Archival File for Macromolecular Structures,** J. Mol. Biol., 112, 535-542.

Blundell T.L, Sibanda, B. L, Sternberg, M. J. E. and Thornton, J. M. (1987) **Knowledge-based prediction of Protein structures and the design of novel Molecules,** Nature, 326, 347-352.

Branden, C-I, (1980) **Relationship between structure and function of α/β-proteins** *Quart. Rev. Biophys*, 13, 3, 317-338.

Buchanan, B. G. and Feigenbaum, E. A. (1978) **"DENDRAL and Meta-DENDRAL: Their Applications Dimension"** *Art. Int.* 11, 5-24.

Clark, D. A, Rawlings, C. J, Barton G. J. and Archer, I. (1990a) **Knowledge-Based Orchestration of Protein Sequence Analysis and Knowledge Acquisition for Protein Structure Prediction,** *Proceedings AAAI Symposium on AI and Molecular Biology*, 28-32, Stanford, March 1990.

Clark, D. A, Barton G. J. and Rawlings, C. R. (1990b) **A Knowledge-Based Architecture for Protein Sequence Analysis and Structure Prediction,** *Journal of Molecular Graphics*, 8(3), 94-107.

Clark, D. A, Shirazi, J., and Rawlings, C. (1991) **The Prediction of Protein Topology using Constraint Based Search and the Evaluation of Topological Folding Rules,** Protein Engineering, 4(7), 751-760.

Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Berthier, F. (1988) **The Constraint Logic Programming Language CHIP,** Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), 693-702, Tokyo, Japan.

Hayes-Roth, B., Buchanan, B.G., Lichtarge, O. et al. (1986) **PROTEAN: Deriving Protein Structure from Constraints** *Proceedings of American Association of Artificial Intelligence*, 5 904-909

Jaffar, J. and Lassez, J-L. (1987) **Constraint Logic Programming** In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, Germany.

Le Provost, T. and Wallace, M.(1992) **Domain Independent Propagation,** In Proceedings International Conference on Fifth Generation Computer Systems, Tokyo, Japan.

Major, F., Turcotte, M., Gautheret, D., Lapalme et al. (1991) **The combination of Symbolic and Numerical Computation for three-dimensional Modeling of RNA** *Science, 253, 1255-1260*

Murzin, A. G. and Finkelstein, A. V. (1988) **General Architecture of the α-Helical Globule,** J. Mol. Biol. 204, 749-769.

Nadel, B. A. (1990) **Constraint Satisfaction Algorithms** *Computational Intelligence*, 5(4), 188-224.

Rawlings, C. J, Taylor, W. R, Nyakairu, J, Fox, J. and Sternberg, M. J. E. (1985) **TOPOL** *J. Mol. Graphics* 3(4), 151-157.

Richardson, J. S. (1977) **β-sheet relatedness and the relatedness of proteins.** Nature, 268, 495-500.

Richardson, J. S. (1981) **The Anatomy and Taxonomy of Protein Structure.**, Advances in Protein Chemistry, 34, 167-339.

Stefik, M. (1981a) **Planning with Constraints [MOLGEN: Pt 1]** *Artificial Intelligence*, 16, 111-140

Stefik, M. (1981b) **Planning and meta-planning [MOLGEN: Pt 2]** *Artificial Intelligence*, 16, 141-169

Sternberg, M. J. E. and Thornton, J. M. (1977) **On the Conformation of Proteins: The Handedness of the Connection between Parallel β-Strands,** J. Mol. Biol, 110, 269-283.

Taylor, W. R. and Green, N. M. (1989) **The predicted secondary structure of the nucleotide-binding sites of six cation-transporting ATPases leads to a probable tertiary fold.** European Journal of Biochemistry, 179, 241-248.

Van Hentenryck, P. and Dincbas, M. (1986) **Domains in Logic Programming** In Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI'86), Philadelphia, PA, AAAI Press.

Van Hentenryck, P. (1991) **Constraint Logic Programming,** Knowledge Engineer Review, 6(3), 151-194.

Walker, J. E, Saraste, M, Runswick, M. J. and Gay, N. J. (1982) **Distantly related sequences in the α- and β- subunits of ATP synthase, myosin, kinases and other ATP-requiring enzymes and a common nucleotide binding fold.**, EMBO Journal, 1(8) 945-95.