

A Study on Application of Deductive Database to Analysis of Genome Data

佐藤, 賢二

<https://doi.org/10.11501/3110980>

出版情報 : 九州大学, 1995, 博士 (工学), 論文博士
バージョン :
権利関係 :



Chapter 4

Similarity Search in a Deductive Database

In the previous chapter, it was shown that a deductive database system PACADE is fit to search for protein structures. Its users can describe flexibly their intended structure of proteins in the form of logical and declarative rules[16, 25]. The performance evaluation shows that the Magic Set method is effective in cutting down the cost of searching in the actual cases.

For protein structure analysis, the function that can only search for structures described by users is not enough and the function to search for similar structures is needed. However, the definition of similarity is ambiguous and some users define it differently. Moreover, the definition depends on what is to be searched.

Through practical searches, we found that in many cases, criteria of similarity are included in the binding values of variables occur in search rules. By memorizing the binding values of variables closely related to the structural features of the searched structure, one can combine them to the answers. Similar structures can be searched for by accompanying the binding values which represent the features of the structures with user-defined acceptable error range.

We describe herein the function added to PACADE to search for similar answers[26, 28].

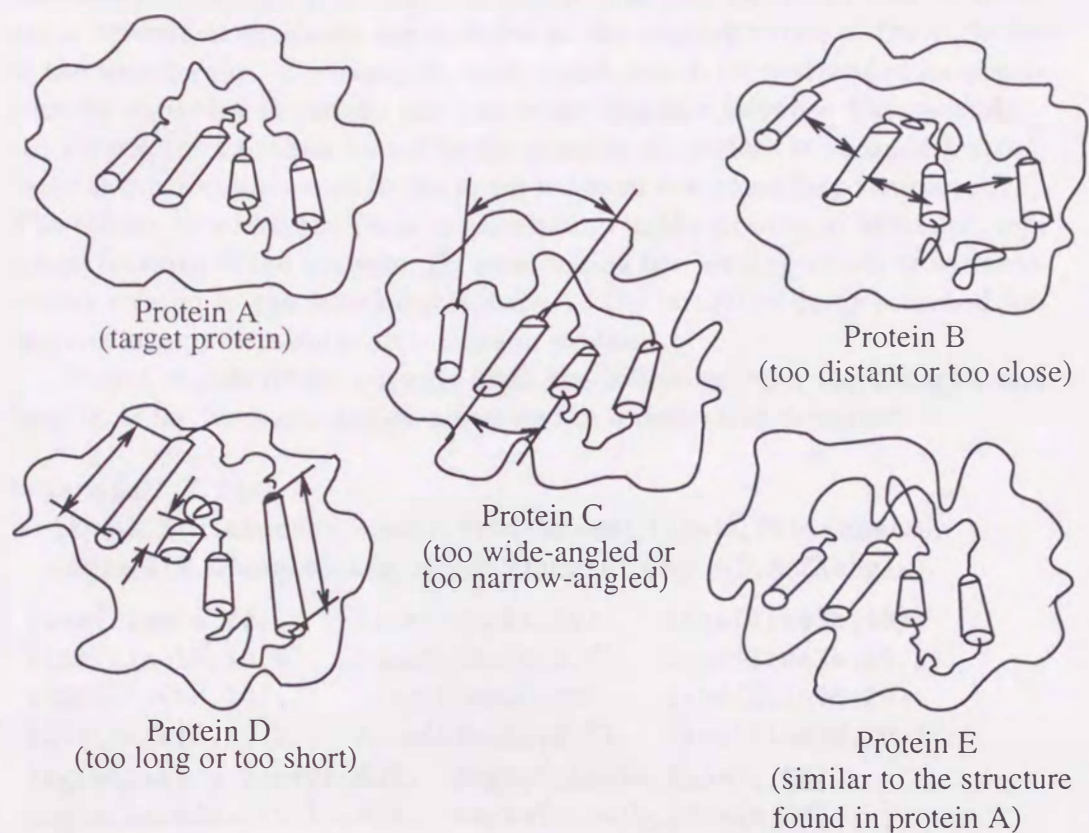


Figure 4.1: Conceptual figure of a search for a similar structure

4.1 Concept of Similar Structure Search

In general, when one searches for a structure in protein using PACADE, answers represent the instances of the structure. However, instances of structure are not always similar to each other. For close scrutinization, one needs a function which enables a search for only instances of the structure similar to the instances of the structure found in a protein which is aimed by a user (Figure 4.1).

However, the definition of similarity is ambiguous and may change, depending on what is being searched for. Moreover, some users may define it

differently. Through practical searches in [16, 25], we found that in many cases, criteria of similarity are included in the binding values of the variables in the search rules. For example, rules which search for n -stranded meanders contain variables bound to the minimum distance between two secondary structures, the variables bound to the number of residues in secondary structures and variables bound to the angle between two secondary structures[25]. The values, to which the variables are bound in the process of inference, represent features of the answers. By memorizing the binding values of variables closely related to the structural features of the structure being searched for, one can search for similar structures in protein.

To put it concretely, suppose that the following rules for triangles and base facts for lines and angles are stored in a deductive database:

```
triangle(X,Y,Z):-
  line(X,Xthickness),line(Y,Ythickness),line(Z,Zthickness),
  angle(X,Y,XYangle),angle(Y,Z,YZangle),angle(Z,X,ZXangle).

line(line1a,10).   line(line2a,10).   line(line3a,10).
line(line1b,10.5). line(line2b,9.7).   line(line3b,10.1).
line(line1c,10).   line(line2c,10).   line(line3c,20).
line(line1d,10.5). line(line2d,9.7).   line(line3d,10.1).
angle(line1a,line2a,60). angle(line2a,line3a,60).
angle(line3a,line1a,60). angle(line1b,line2b,60).
angle(line2b,line3b,30). angle(line3b,line1b,90).
angle(line1c,line2c,62). angle(line2c,line3c,55).
angle(line3c,line1c,63). angle(line1d,line2d,62).
angle(line2d,line3d,55). angle(line3d,line1d,63).
```

When a query :- triangle(X,Y,Z). is given to this database, deductive database system produces the following four answers:

[answer a] triangle(line1a,line2a,line3a).

[answer b] triangle(line1b,line2b,line3b).

[answer c] triangle(line1c,line2c,line3c).

[answer d] triangle(line1d,line2d,line3d).

These four answers are derived from the following four conjunctions of base facts, respectively:

```
[answer a'] line(line1a,10),line(line2a,10),line(line3a,10),
             angle(line1a,line2a,60),angle(line2a,line3a,60),
             angle(line3a,line1a,60)
```

```
[answer b'] line(line1b,10.5),line(line2b,9.7),line(line3b,10.1),
             angle(line1b,line2b,60),angle(line2b,line3b,30),
             angle(line3b,line1b,90)
```

```
[answer c'] line(line1c,10),line(line2c,10),line(line3c,20),
             angle(line1c,line2c,62),angle(line2c,line3c,55),
             angle(line3c,line1c,63)
```

```
[answer d'] line(line1d,10.5),line(line2d,9.7),line(line3d,10.1),
             angle(line1d,line2d,62),angle(line2d,line3d,55),
             angle(line3d,line1d,63)
```

Suppose a user wants to characterize these answers by the angles and thickness of lines in them. He can say that only answer c is similar to answer a if he can accept error ranges ± 6 for the angles and ± 1 for thickness of lines. This example illustrates that we can check similarity relations between answers through constants (binding values) in conjunctions of base facts which are equals to the answers.

4.2 Overview of a Search for Similar Structures

Figure 4.2 is a sketch of our approach to such a similarity search. Regarding rules as templates, and answers as instances which match template, the similarity search proceeds as follows:

1. Some instances (= answers of *the first search*) are found which match the same template (= rule set for the first search).
2. By analyzing characteristics of the instances, new templates (= rule set for *the second search*) are generated.

3. The instances (= answers of the second search) are found which match new templates. They are similar to the instances in the first search.

To realize such similarity search on a deductive database, we devised (1)an representation of similarity specification given by users, (2)an algorithm to search for similar structures based on the similarity specification, and (3)an implementation of the algorithm on PACADE.

4.2.1 Annotations in Rules and a Query

To search for the answers similar to an answer, a system has to allow a user to describe the following three information in addition to rules, query and base facts:

- values considered in the judgement of similarity (*similarity criterion*).
- the extent of acceptable error in the judgement of similarity (*error range*).
- the answers (objects) to which a user wants to search for the answers (objects) similar (*target domain (objects)*).

In the example in section 4.1, similarity criteria were angle and thickness of lines in a triangle. Error ranges were ± 6 for the angles and ± 1 for thickness of lines. Target domain defined only one answer:

```
triangle(line1a,line2a,line3a).
```

which represents a specific object.

PACADE allows a user to describe such information in rules and query as annotations called *prefixes*. An example of *prefixed query* is below:

```
:- triangle(@line1a,@line2a,@line3a).
```

where @s are annotations called *query prefixes* added to the constants line1a, line2a and line3a in the query. These prefixes are used to define target domain.

An example of *prefixed rule* is below:

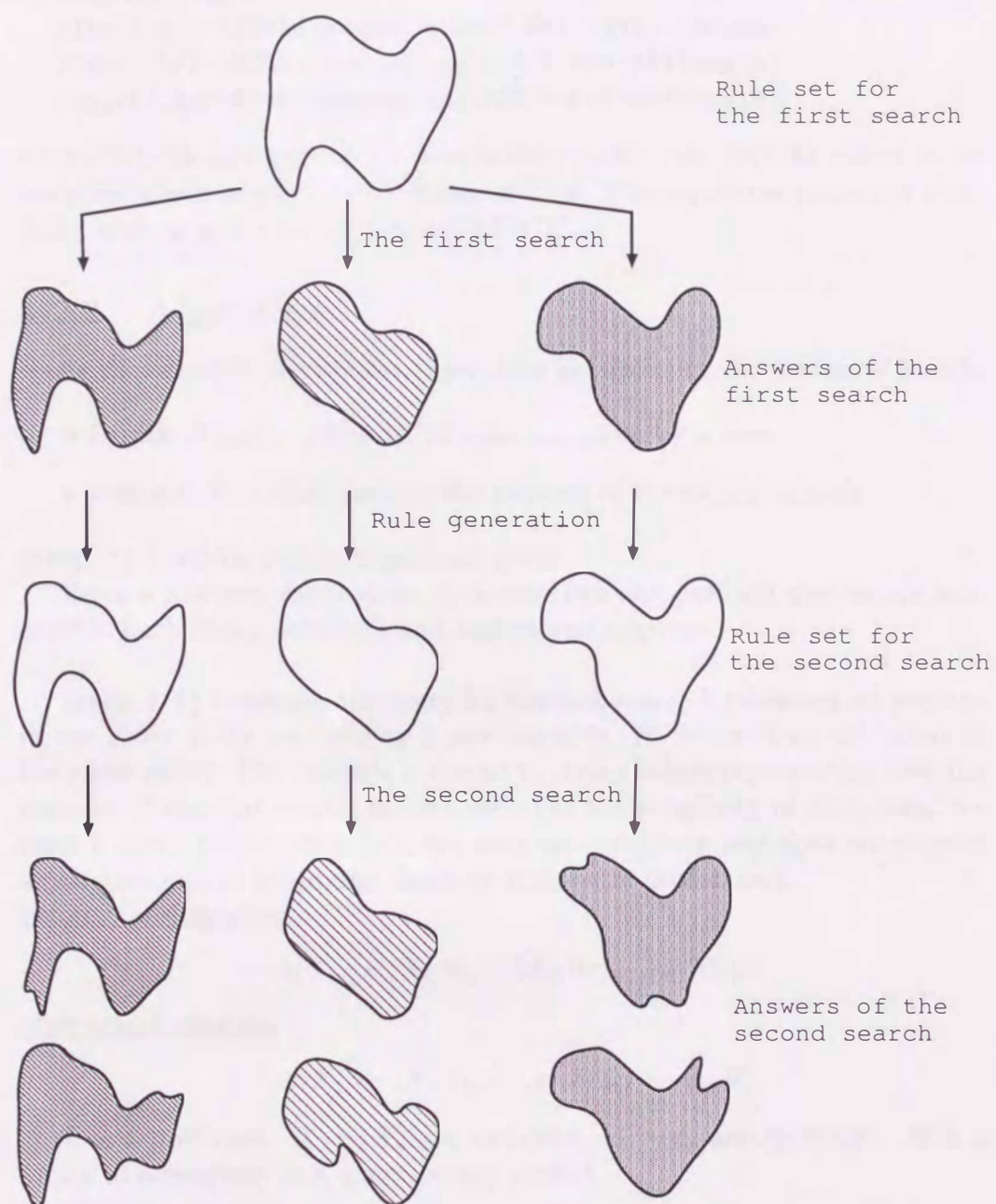


Figure 4.2: Sketch of our approach to search for similar structures.

triangle(X,Y,Z):-

```

    line(X,&+1-1&Xthickness),line(Y,&+1-1&Ythickness),
    line(Z,&+1-1&Zthickness),angle(X,Y,&+6-6&XYangle),
    angle(Y,Z,&+6-6&YZangle),angle(Z,X,&+6-6&ZXangle).

```

where &+1-1& and &+6-6& are annotations called *rule prefixes* added to six variables about angles and thickness of lines. These prefixes point out similarity criteria and error ranges to PACADE.

4.2.2 Algorithm

Using the example in section 4.1, we show an algorithm for similarity search.

- **input:** A query and a set of rules are given by a user.
- **output:** PACADE returns the answers of the second search.

(step 1) Transformation of prefixed query

From a prefixed query given by a user, two non-prefixed queries are generated. Each query is for the first and second searches.

(step 1-1) Generate the query for the first search by deleting all prefixes in the given query and adding a new variable (R) which does not occur in the given query. The variable is bound to string values representing how the answers of the first search are derived. For the simplicity of discussion, we limit a query to the one which has only one predicate and does not require any arguments to be ground (such as arithmetic predicates).

before transformation

$$:-h(X_1, \dots, X_e, c_1, \dots, c_f, @c_{f+1}, \dots, @c_g).$$

after transformation

$$:-h(X_1, \dots, X_e, c_1, \dots, c_f, c_{f+1}, \dots, c_g, R).$$

h is a predicate. $X_1 \sim X_e$ are variables. $c_1 \sim c_g$ are constants. $@$ is a prefix to a constant in a query (query prefix).

example)

before transformation

`:-triangle(@line1a,@line2a,@line3a).`

after transformation

`:-triangle(line1a,line2a,line3a,R).`

Search space of the first search is limited to target domain by the constants line1a, line2a and line3a.

(step 1-2) Generate the query for the second search by replacing all prefixed constants in the given query by new variables which do not occur in the given query and differ from each other. Since h does not require any arguments to be ground, prefixed constants can always be replaced by variables.

before transformation

`:-h( $X_1, \dots, X_e, c_1, \dots, c_f, @c_{f+1}, \dots, @c_g$ ).`

after transformation

`:-h( $X_1, \dots, X_e, c_1, \dots, c_f, Y_{f+1}, \dots, Y_g$ ).`

example)

before transformation

`:- triangle(@line1a,@line2a,@line3a).`

after transformation

`:- triangle(X,Y,Z).`

By this replacement, search space of the second search become being not limited to target domain.

(step 2) Transformation of prefixed rule set

From the prefixed rule set given by a user, a non-prefixed rule set for the first search is generated.

(step 2-1) Analyze the prefixed rule set and check all predicates to determine whether extensional database (EDB) predicates or intensional database (IDB) predicates.

(step 2-2) Transform each rule in the prefixed rule set as follows:

before transformation

$$\begin{aligned}
h(X_1, \dots, X_k) :- \\
& b_{I_1}(p_1^{I_1} V_1^{I_1}, \dots, p_{m_{I_1}}^{I_1} V_{m_{I_1}}^{I_1}, V_{m_{I_1}+1}^{I_1}, \dots, V_{n_{I_1}}^{I_1}, c_1^{I_1}, \dots, c_{o_{I_1}}^{I_1}), \\
& \vdots \\
& b_{I_i}(p_1^{I_i} V_1^{I_i}, \dots, p_{m_{I_i}}^{I_i} V_{m_{I_i}}^{I_i}, V_{m_{I_i}+1}^{I_i}, \dots, V_{n_{I_i}}^{I_i}, c_1^{I_i}, \dots, c_{o_{I_i}}^{I_i}), \\
& b_{E_1}(p_1^{E_1} V_1^{E_1}, \dots, p_{m_{E_1}}^{E_1} V_{m_{E_1}}^{E_1}, V_{m_{E_1}+1}^{E_1}, \dots, V_{n_{E_1}}^{E_1}, c_1^{E_1}, \dots, c_{o_{E_1}}^{E_1}), \\
& \vdots \\
& b_{E_j}(p_1^{E_j} V_1^{E_j}, \dots, p_{m_{E_j}}^{E_j} V_{m_{E_j}}^{E_j}, V_{m_{E_j}+1}^{E_j}, \dots, V_{n_{E_j}}^{E_j}, c_1^{E_j}, \dots, c_{o_{E_j}}^{E_j}).
\end{aligned}$$

after transformation

$$\begin{aligned}
h(X_1, \dots, X_k, [R_0, R_{I_1}, \dots, R_{I_i}]) :- \\
& b_{I_1}(V_1^{I_1}, \dots, V_{m_{I_1}}^{I_1}, V_{m_{I_1}+1}^{I_1}, \dots, V_{n_{I_1}}^{I_1}, c_1^{I_1}, \dots, c_{o_{I_1}}^{I_1}, R_{I_1}), \\
& \vdots \\
& b_{I_i}(V_1^{I_i}, \dots, V_{m_{I_i}}^{I_i}, V_{m_{I_i}+1}^{I_i}, \dots, V_{n_{I_i}}^{I_i}, c_1^{I_i}, \dots, c_{o_{I_i}}^{I_i}, R_{I_i}), \\
& b_{E_1}(V_1^{E_1}, \dots, V_{m_{E_1}}^{E_1}, V_{m_{E_1}+1}^{E_1}, \dots, V_{n_{E_1}}^{E_1}, c_1^{E_1}, \dots, c_{o_{E_1}}^{E_1}), \\
& \vdots \\
& b_{E_j}(V_1^{E_j}, \dots, V_{m_{E_j}}^{E_j}, V_{m_{E_j}+1}^{E_j}, \dots, V_{n_{E_j}}^{E_j}, c_1^{E_j}, \dots, c_{o_{E_j}}^{E_j}), \\
& \text{make_string_rule} \\
& [^n h(X_1, \dots, X_k) : - \\
& b_{I_1}(V_1^{I_1}, \dots, V_{m_{I_1}}^{I_1}, V_{m_{I_1}+1}^{I_1}, \dots, V_{n_{I_1}}^{I_1}, c_1^{I_1}, \dots, c_{o_{I_1}}^{I_1}), \\
& \vdots \\
& b_{I_i}(V_1^{I_i}, \dots, V_{m_{I_i}}^{I_i}, V_{m_{I_i}+1}^{I_i}, \dots, V_{n_{I_i}}^{I_i}, c_1^{I_i}, \dots, c_{o_{I_i}}^{I_i}), \\
& b_{E_1}(V_1^{E_1}, \dots, V_{m_{E_1}}^{E_1}, V_{m_{E_1}+1}^{E_1}, \dots, V_{n_{E_1}}^{E_1}, c_1^{E_1}, \dots, c_{o_{E_1}}^{E_1}), \\
& \vdots \\
& b_{E_j}(V_1^{E_j}, \dots, V_{m_{E_j}}^{E_j}, V_{m_{E_j}+1}^{E_j}, \dots, V_{n_{E_j}}^{E_j}, c_1^{E_j}, \dots, c_{o_{E_j}}^{E_j})^n, \\
& d_1^{I_1}, \dots, d_{m_{I_1}}^{I_1}, d_1^{I_i}, \dots, d_{m_{I_i}}^{I_i}, \\
& d_1^{E_1}, \dots, d_{n_{E_1}}^{E_1}, d_1^{E_j}, \dots, d_{n_{E_j}}^{E_j}, " "], \\
& R_0).
\end{aligned}$$

b_I is a IDB predicate, and b_E is a EDB predicate. $V_1, \dots, V_m$ are prefixed variables bound to numeric values, and $p_1, \dots, p_m$ are prefixes to variables in rules (rule prefix). The general form of rule prefix p is as follows:

$$p = \&[+n_1][-n_2]\& \quad (0 \leq n_1, n_2)$$

where n_1 indicates upper limit of acceptable error range and n_2 indicates lower limit. If both are omitted, the prefix means an exact match.

d_α transformed from $p_\alpha V_\alpha$ is as follows.

if $p_\alpha V_\alpha$ is $\&\&$, d_α is $", V_\alpha = ", V_\alpha$.
 if $p_\alpha V_\alpha$ is $\& + n_1 \&$, d_α is $", V_\alpha > ", V_\alpha, ", V_\alpha < n_1 + ", V_\alpha$.
 if $p_\alpha V_\alpha$ is $\& - n_2 \&$, d_α is $", V_\alpha > -n_2 + ", V_\alpha, ", V_\alpha < ", V_\alpha$.
 if $p_\alpha V_\alpha$ is $\& + n_1 - n_2 \&$, d_α is $", V_\alpha > -n_2 + ", V_\alpha, ", V_\alpha < n_1 + ", V_\alpha$.

The built-in predicate *make_string_rule* requires that the first argument is bound and the second argument is free. The first argument must be a list which is not nested. The elements of the list must be strings or bound variables. When *make_string_rule* is evaluated, the elements of the list are examined. The values of variable elements are changed into strings. They are catenated with other string elements and the second argument is bound to the catenated string.

example)

before transformation

```
triangle(X,Y,Z):-
  line(X,&+1-1&Xthickness),line(Y,&+1-1&Ythickness),
  line(Z,&+1-1&Zthickness),angle(X,Y,&+6-6&XYangle),
  angle(Y,Z,&+6-6&YZangle),angle(Z,X,&+6-6&ZXangle).
```

after transformation

```
triangle(X,Y,Z,[R]):-
  line(X,Xthickness),line(Y,Ythickness),line(Z,Zthickness),
  angle(X,Y,XYangle),angle(Y,Z,YZangle),angle(Z,X,ZXangle),
  make_string_rule([
    "triangle(X,Y,Z):-
      line(X,Xthickness),line(Y,Ythickness),
      line(Z,Zthickness),angle(X,Y,XYangle),
      angle(Y,Z,YZangle),angle(Z,X,ZXangle)",
    ",Xthickness>-1+",Xthickness,
    ",Xthickness<1+",Xthickness,
    ",Ythickness>-1+",Ythickness,
```

```

", Ythickness<1+", Ythickness,
", Zthickness>-1+", Zthickness,
", Zthickness<1+", Zthickness,
", XYangle>-6+", XYangle, ", XYangle<6+", XYangle,
", YZangle>-6+", YZangle, ", YZangle<6+", YZangle,
", ZXangle>-6+", ZXangle, ", ZXangle<6+", ZXangle, "."] , R).

```

For the sake of readability, string constants are underlined in this rule.

(step 3) Evaluation of the query and the rule set for the first search

The query generated in step 1-1 and the rule set generated in step 2 are evaluated by the bottom-up evaluator. Each of the last arguments of the answers to the query keeps the derivation processes of the answers in the form of a nested list. The element of the nested list is the string which represents the step of bottom-up evaluation in the form of single rule. In addition to the step of bottom-up evaluation, it represents the binding values of the prefixed variables in the first search and the values are used to define the permitted errors in the second search.

example)

The answer to the example query for the first search is as follows.

```

triangle(line1a,line2a,line3a,
  ["triangle(X,Y,Z):-
    line(X,Xthickness),line(Y,Ythickness),
    line(Z,Zthickness),angle(X,Y,XYangle),
    angle(Y,Z,YZangle),angle(Z,X,ZXangle),
    Xthickness>-1+10,Xthickness<1+10,
    Ythickness>-1+10,Ythickness<1+10,
    Zthickness>-1+10,Zthickness<1+10,
    XYangle>-6+60,XYangle<6+60,
    YZangle>-6+60,YZangle<6+60,
    ZXangle>-6+60,ZXangle<6+60."]).

```

The process of derivation is memorized in the last argument. In other words, each of answers is characterized in terms of its process of derivation. Therefore, the first search can be referred to as "characterization search."

For the sake of readability, a string constant is underlined in this answer.

(step 4) Generation of the rule set for the second search

The derivation processes of answers are memorized as the string values of last arguments. The rule set for the second search is generated by matching the heads of the rules in the last arguments with the bodies of them. In the case of this example, it is straightforward because there is only one rule in the last argument.

example)

```
triangle(X,Y,Z):-  
  line(X,Xthickness),line(Y,Ythickness),  
  line(Z,Zthickness),angle(X,Y,XYangle),  
  angle(Y,Z,YZangle),angle(Z,X,ZXangle),  
  Xthickness > -1 + 10,Xthickness < 1 + 10,  
  Ythickness > -1 + 10,Ythickness < 1 + 10,  
  Zthickness > -1 + 10,Zthickness < 1 + 10,  
  XYangle > -6 + 60,XYangle < 6 + 60,  
  YZangle > -6 + 60,YZangle < 6 + 60,  
  ZXangle > -6 + 60,ZXangle < 6 + 60.
```

This rule is used so as to get answers similar to the answer of the first search.

(step 5) Evaluation of the query and the rule set for the second search

The query generated in step 1-2 and the rule set generated in step 4 are evaluated by the bottom-up evaluator. The answers of the second search are similar to those of the first search.

example)

The two lines below show answers derived from the rule above. Each of the answers is similar to the answer of the first search in the angles and the thickness of lines in it.

```
triangle(line1a,line2a,line3a).  
triangle(line1d,line2d,line3d).
```

4.3 Implementation

Figure 4.3 shows the system configuration of PACADE with CORAL extended by two new modules *preprocessor* and *rule generator* for similarity search.

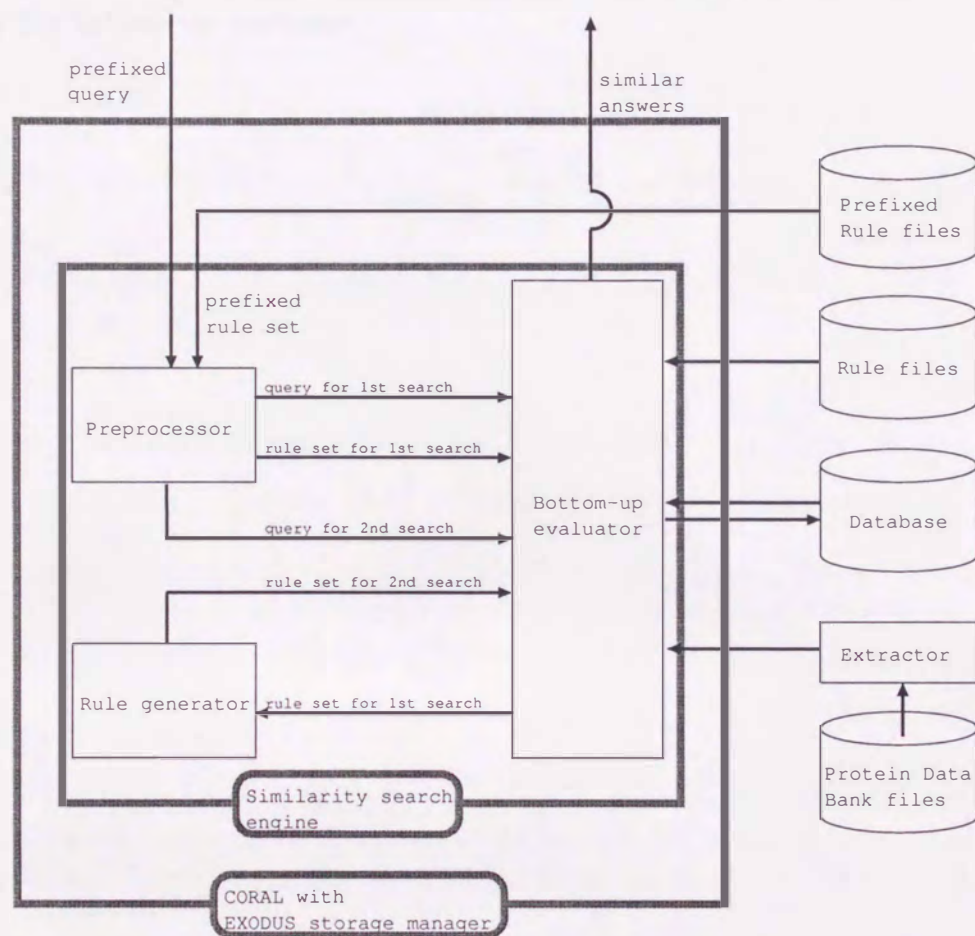


Figure 4.3: System configuration

[Preprocessor] From input (a prefixed query and prefixed rules), the preprocessor module generates non-prefixed rules for the first search and two non-prefixed queries for the first and the second search. They are sent to and used by the bottom-up evaluator.

[Rule generator] From input (answers of the first search), the rule generator generates non-prefixed rules for the second search. They are sent to and used by the bottom-up evaluator.

Applicability of Similarity Search Technique to Search for Similar Structures in Proteins

3.1 Examples of Searches for Similar Supersecondary Structures in Proteins

We applied the technique described in the previous section to search for similar supersecondary structures in proteins. In this section, we will describe the results of the searches for similar supersecondary structures in proteins.

3.1.1 Greek Key

The following search query and prefixed rules were used to search for similar supersecondary structures in proteins. The search results are shown in Table 3.1. The search results are shown in Table 3.1.

Query: $Q = \{ \text{GK} \}$
Rules: $R = \{ \text{GK} \}$
Results: $R = \{ \text{GK} \}$

Chapter 5

Applicability of Similarity Search Technique to Search for Similar Structures in Proteins

5.1 Examples of Searches for Similar Supersecondary Structures in Proteins

We applied the function to searches for similar supersecondary structures in proteins. In the examples below, we used the 134 proteins shown in subsection 3.3.2 to search for similar structures.

5.1.1 Greek Key

The following prefixed query and prefixed rule set are used to search for Greek keys similar to those found in protein 1sgt. The original (non-prefixed) version of the rule set is slightly modified from the rule set in subsection 3.2.5.

prefixed query

```
:- greek_key(A,B,C,Num,@"1sgt").
```

prefixed rule set

```
greek_key(A,B,C,Num,P):-greek_even_r(A,B,C,Num,P).
```

```

greek_key(A, B, C, Num, P):-greek_even_l(A, B, C, Num, P).

greek_key(A, B, C, Num, P):-greek_odd(A, B, C, Num, P).

greek_even_r(A, L, D, 4, P):-
    hairpin(B, C, P), hairpin(C, D, P), not_coils(A, B, P),
    neighbour(A, D, P), double_anti_parallel(A, D, P), L = [B, C].

greek_even_r(A, L, D, Num1, P):-
    greek_odd(B, L1, D, Num, P), not_coils(A, B, P), neighbour(A, D, P),
    double_anti_parallel(A, D, P), append([B], L1, L), Num1 = Num + 1.

greek_even_l(A, L, D, 4, P):-
    hairpin(A, B, P), hairpin(B, C, P), not_coils(C, D, P),
    neighbour(A, D, P), double_anti_parallel(A, D, P), L = [B, C].

greek_even_l(A, L, D, Num1, P):-
    greek_odd(A, L1, C, Num, P), not_coils(C, D, P), neighbour(A, D, P),
    double_anti_parallel(A, D, P), append(L1, [C], L), Num1 = Num + 1.

greek_odd(A, L, D, Num1, P):-
    greek_even_r(A, L1, B, Num, P), greek_even_l(C, L2, D, Num, P),
    append(L1, [B], L), append([C], L2, L3), L = L3, Num1 = Num + 1.

hairpin(A, B, P):-
    not_coils(A, B, P), neighbour(A, B, P), double_anti_parallel(A, B, P).

not_coils(A, B, P):-not_coil(A, P), B = A + 1, not_coil(B, P).

not_coils(A, B, P):-
    not_coil(A, P), A1 = A + 1, is_coil(A1, P), B = A1 + 1, not_coil(B, P).

is_coil(X, P):-secondary_structure(X, coil, &+3-3&Xnum, Cx, P).

not_coil(X, P):-secondary_structure(X, strand, &+10-10&Xnum, Cx, P).

not_coil(X, P):-secondary_structure(X, helix, &+10-10&Xnum, Cx, P).

```

```

neighbour(X, Y, P):-
    min_distance(X, Cx, Y, Cy, &+1-1&Dxy, P), Cx = Cy, Dxy < 20.0.

neighbour(X, Y, P):-
    min_distance(Y, Cy, X, Cx, &+1-1&Dxy, P), Cy = Cx, Dxy < 20.0.

double_anti_parallel(X, Y, P):-anti_parallel(X, Y, P).

double_anti_parallel(X, Y, P):-anti_parallel(Y, X, P).

anti_parallel(X, Y, P):-
    angle(X, Cx, Y, Cy, &+10-10&Ang, P), Cx = Cy, 120 < Ang, Ang < 180.

```

The protein 1sgt has four Greek keys We show the result of a similarity search of over 134 data on proteins listed in table 3.1 with the rule set above, in which the Greek keys in 1sgt are written as answer G_0^i ($1 \leq i \leq 4$).

```

[answer  $G_0^1$ ] greek_key(8, [10, 12], 14, 4, "1sgt").
[answer  $G_1^1$ ] greek_key(6, [8, 10], 12, 4, "1est").
[answer  $G_2^1$ ] greek_key(8, [10, 12], 14, 4, "1tgn").
[answer  $G_3^1$ ] greek_key(41, [43, 45], 47, 4, "2cga").

[answer  $G_0^2$ ] greek_key(10, [12, 14], 16, 4, "1sgt").

[answer  $G_0^3$ ] greek_key(4, [6, 8], 10, 4, "1sgt").
[answer  $G_1^3$ ] greek_key(4, [6, 8], 10, 4, "1ton").

[answer  $G_0^4$ ] greek_key(8, [10, 12, 14], 16, 5, "1sgt").

```

The answers G_j^i ($j \geq 1$) are similar to the answer G_0^i . The answer G_0^2 has no similar answer. Through a graphical viewer of proteins called BIORESEARCH/3D, we confirmed that three dimensional shapes of these structures are actually similar. For example, figure 5.1 shows 4-stranded Greek keys represented by G_0^1 , G_1^1 and G_2^1 , using the external graphical viewer.



Figure 5.1: The 4-stranded Greek keys in three proteins, 1sgt, 1est and 1tgn.

5.1.2 *n*-stranded Meander

The following prefixed query and prefixed rule set are used to search for *n*-stranded meanders similar to those found in protein 2sbt. The original (non-prefixed) version of the rule set is slightly modified from the rule set in subsection 3.2.5..

prefixed query

```
:- meander_n(A,B,Num,@"2sbt").
```

prefixed rule set

```
meander_n(A,L,3,P):-hairpin(A,B,P),hairpin(B,C,P),L=[B,C].
```

```
meander_n(A,L1,Num1,P):-
```

```
    hairpin(A,B,P),meander_n(B,L,Num,P),Num1=Num+1,L1=[B|L].
```

```
hairpin(A,B,P):-
```

```
    not_coils(A,B,P),neighbour(A,B,P),double_anti_parallel(A,B,P).
```

```
not_coils(A,B,P):-not_coil(A,P),B=A+1,not_coil(B,P).
```

```
not_coils(A,B,P):-
```

```
    not_coil(A,P),A1=A+1,is_coil(A1,P),B=A1+1,not_coil(B,P).
```

```
is_coil(X,P):-secondary_structure(X,coil,&+3-3&Xnum,Cx,P).
```

```
not_coil(X,P):-secondary_structure(X,strand,&+10-10&Xnum,Cx,P).
```

```
not_coil(X,P):-secondary_structure(X,helix,&+10-10&Xnum,Cx,P).
```

```
neighbour(X,Y,P):-
```

```
    min_distance(X,Cx,Y,Cy,&+1-1&Dxy,P),Cx=Cy,Dxy<20.0.
```

```
neighbour(X,Y,P):-
```

```
    min_distance(Y,Cy,X,Cx,&+1-1&Dxy,P),Cy=Cx,Dxy<20.0.
```

```
double_anti_parallel(X,Y,P):-anti_parallel(X,Y,P).
```

double_anti_parallel(X, Y, P):-anti_parallel(Y, X, P).

anti_parallel(X, Y, P):-

angle(X, Cx, Y, Cy, &+15-15&Ang, P), Cx = Cy, 120 < Ang, Ang < 180.

The protein 2sbt has 15 meanders. We show the result of a similarity search of over 134 data on proteins listed in table 3.1 with the rule set above, in which the meanders in 2sbt are written as answer M_0^i ($1 \leq i \leq 15$).

we obtained the following answers:

[answer M_0^1] meander_n(8, [10, 12], 3, "2sbt").

[answer M_0^2] meander_n(10, [12, 14], 3, "2sbt").

[answer M_1^2] meander_n(10, [12, 14], 3, "1sbc").

[answer M_2^2] meander_n(10, [12, 14], 3, "1st2").

[answer M_3^2] meander_n(10, [12, 14], 3, "1tec").

[answer M_0^3] meander_n(12, [14, 16], 3, "2sbt").

[answer M_1^3] meander_n(14, [16, 18], 3, "2prk").

[answer M_0^4] meander_n(14, [16, 18], 3, "2sbt").

[answer M_0^5] meander_n(16, [18, 20], 3, "2sbt").

[answer M_1^5] meander_n(16, [18, 20], 3, "1sbc").

[answer M_2^5] meander_n(16, [18, 20], 3, "1st2").

[answer M_0^6] meander_n(8, [10, 12, 14], 4, "2sbt").

[answer M_0^7] meander_n(10, [12, 14, 16], 4, "2sbt").

[answer M_0^8] meander_n(12, [14, 16, 18], 4, "2sbt").

[answer M_0^9] meander_n(14, [16, 18, 20], 4, "2sbt").

[answer M_0^{10}] meander_n(8, [10, 12, 14, 16], 5, "2sbt").

[answer M_0^{11}] meander_n(10, [12, 14, 16, 18], 5, "2sbt").

[answer M_0^{12}]meander.n(12, [14, 16, 18, 20], 5, "2sbt").

[answer M_0^{13}]meander.n(8, [10, 12, 14, 16, 18], 6, "2sbt").

[answer M_0^{14}]meander.n(10, [12, 14, 16, 18, 20], 6, "2sbt").

[answer M_0^{15}]meander.n(8, [10, 12, 14, 16, 18, 20], 7, "2sbt").

Answers M_j^i ($j \geq 1$) are similar to answer M_0^i and answers M_0^1 , M_0^4 and $M_0^6 \sim M_0^{15}$ have no similar answers.

Likewise the result on 4-stranded Greek keys, three dimensional shapes of these structures are actually similar. Figure 5.1 shows 4-stranded Greek keys represented by M_0^2 , M_1^2 and M_2^2 , using the external graphical viewer.

5.2 Similarity Search for Arbitrary Structure

In the previous study, we used rule sets which represent the topologies of supersecondary structures (i.e. Greek key and n -stranded meander). These are useful to show that some iterative structures can be represented and searched in PACADE. However, they are of no benefit to a user who cannot represent the topology of a specific part of a protein as rules.

The rule set for n -stranded meander includes the following conditions to be tested, concerning two adjacent secondary structures, say, X and Y:

1. The angle between X and Y must range from 120 to 180 degrees.
2. The distance between X and Y must be smaller than 20.0 angstroms.
3. Regarding a protein as sequences of secondary structures, X and Y are also adjacent in the sequence. One random coil may be put between them.

Since almost every protein consists of secondary structures held together with random coils, the third condition is weak. Therefore, if we remove the

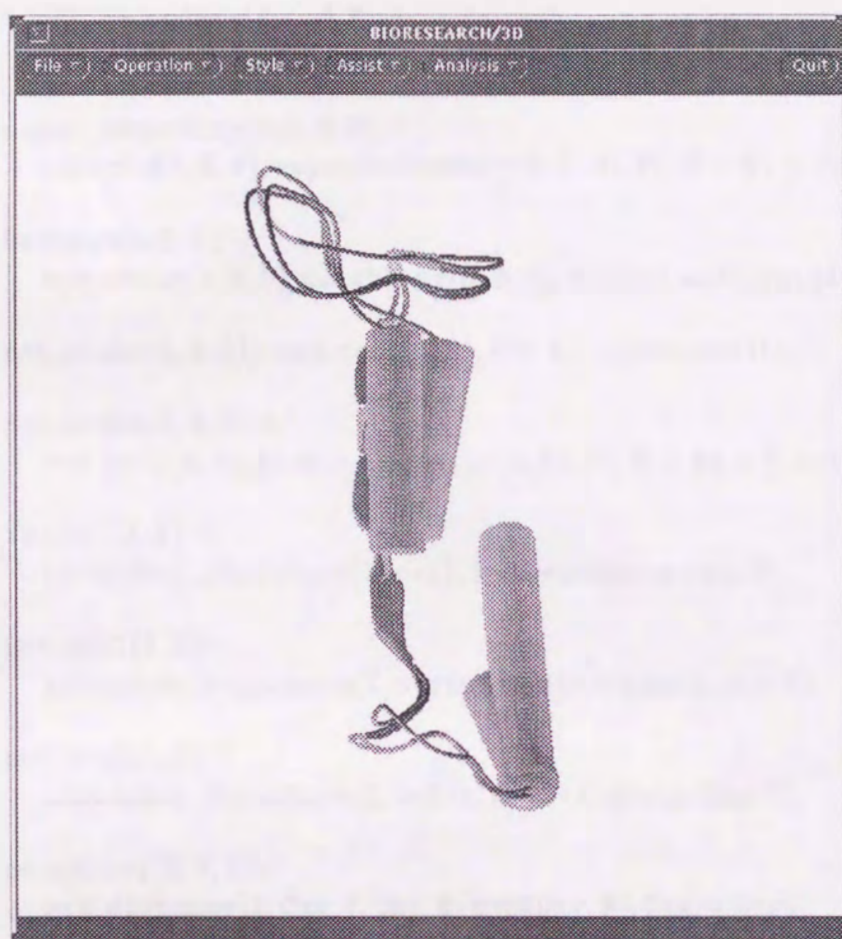


Figure 5.2: The 3-stranded meanders in three proteins, 2sbt, 1sbc and 1st2.

first and the second conditions, the rule set accepts almost every structure continuous in the sense of sequence.

In this way, we relaxed restrictions of the rule set used in our previous study to search for n -stranded meanders[27].

```

super_secondary(A, L, N, P):-
    hairpin(A, B, P), hairpin(B, C, P), L = [B, C], N = 3.

super_secondary(A, L, N, P):-
    hairpin(A, B, P), super_secondary(B, L, N1, P), N = N1 + 1, L = [B|L1].

hairpin(A, B, P):-
    not_coils(A, B, P), neighbour(A, B, P), double_anti_parallel(A, B, P).

not_coils(A, B, P):-not_coil(A, P), B = A + 1, not_coil(B, P).

not_coils(A, B, P):-
    not_coil(A, P), A1 = A + 1, is_coil(A1, P), B = A1 + 1, not_coil(B, P).

is_coil(X, P):-
    secondary_structure(X, coil, &-15+15&Xnum, Cnx, P).

not_coil(X, P):-
    secondary_structure(X, strand, &-15+15&Xnum, Cnx, P).

not_coil(X, P):-
    secondary_structure(X, helix, &-15+15&Xnum, Cnx, P).

neighbour(X, Y, P):-
    min_distance(X, Cnx, Y, Cny, &-6+6&Dxy, P), Cnx = Cny.

neighbour(X, Y, P):-
    min_distance(Y, Cny, X, Cnx, &-6+6&Dxy, P), Cny = Cnx.

double_anti_parallel(X, Y, P):-anti_parallel(X, Y, P).

double_anti_parallel(X, Y, P):-anti_parallel(Y, X, P).

```

```

anti_parallel(X,Y,P):-
    angle(X,Cnx,Y,Cny,&-6+6&V3x,&-6+6&V3y,
          &-45+45&Ang1,&-45+45&Ang2,P),
    Cnx = Cny.

```

Using the above rules, a user can perform a similarity search for any supersecondary structure. In this case, error ranges on the length of helices, strands and random coils are set to ± 15 residues, and on the distance of two adjacent strands or helices to ± 6 angstrom, and so on. A user can readily change these values of error ranges by modifying the rule set.

The query for the above rules must be given with care. If a user gives a query with no constant bindings, the processing of the query will result in many answers because almost any structure of a protein would be accepted. A user of the rules is recommended to write a query with few variables or no variables like the following example of prefixed query:

```
:- super_secondary(@16,@[18,20,22],4,@"1ifb").
```

This query shows that the user intends to search for structures similar to a specific structure, that is, a structure which consists of four helices or strands with sequential numbers 16, 18, 20 and 22, respectively, and they are in a specific protein whose mnemonic code is "1ifb." By changing such prefixed constants in the query and the rule set, a user can search for structures similar to a specific structure to be examined, with intended error ranges.

5.3 Discussion

5.3.1 Comparisons with Other Approaches

Our approach to searches for similar structures is novel, in the following points:

- it is based on techniques of deductive databases
- target (original) structure is specified indirectly using a rule set for the first search as template
- similarity criterion, error range and target protein are specified subjectively, by the user

The first point implies the flexibility of our approach. Rules are written in denotational style, and a rule prefix does not destroy the style because it can be added and transformed, independently of other rules.

The second point may differ the most from other techniques. For example, in approaches based on root mean square deviation (e.g. [2]), target structure is specified directly by pointing start and end point of an amino acid sequence which forms the target structure. Our approach has the following merits and demerits:

[merits]

- low calculation cost because is needed no combinatorial calculation
- can search target structure which consists of discrete amino acids like a dipole moment
- a user can flexibly specify similarity criteria and error ranges

[demerits]

- a user has to be able to represent searched structure in the form of rule
- it is hard to evaluate the degree of similarity on the basis of a unified standard
- if a user wants to search for more real (unabstracted) structures, he has to write more complex rules and specify more similarity criteria

Since the two approaches have merits and demerits, they can coexist. Our indirect specification method is suitable for cases when a user has logical structures on which he is focusing. On the other hand, the approach based on root mean square deviation is suitable for cases when a user is focusing on structures which cannot be written in logical form and he needs a precise estimation of similarity.

5.3.2 Extensions of Descriptive Ability of Prefixes

The following extensions will be accomplished by the improvement of pre-processor.

1. Currently, the function does not allow to prefix variables bound to non-numerical values. Obviously, this limitation may be lessened for other types which can be totally ordered.
2. To limit the search space of the first search, a query must have at least one constant prefixed by "@". Extending descriptive ability of query prefix to allow && type prefixes, like rule prefixes, is useful.

5.3.3 Utilization of Waste (Discarded) Bindings

In the function described in the previous and this chapter, any variables in body atoms which have numerical binding values can be prefixed. A variable which occurs only once in a body atom of a rule and does not occur in the head of the rule can also be prefixed. If such variables are not prefixed, their binding values are not used to produce new intermediate facts and are discarded when rule set and query are evaluated in the bottom-up evaluator. Through the function, such waste bindings can be utilized to characterize answers of the first search.

5.3.4 Reduction of Prefixes

A user may write redundant or meaningless rule prefixes in one rule such as

`anc(X, Y, N):-anc(X, &+5-2&Z, N), par(&+2-5&Z, Y, Age, N).`

To eliminate such prefixes, one way is to analyze and rewrite the prefixed rule set, and another is to analyze and rewrite the rule set for the second search. When the first search returns many answers, the latter approach takes more time than the former. On the other hand, it is easy to analyze and eliminate redundant prefixes in the latter approach as the bodies of rules for the second search are independent because of the lack of an IDB predicate.

Chapter 6

Computation of Closures of Similarity Relationships

To make PACADE an even more useful system for molecular biologists, we introduced a **novel** function which enables automatic searches for similar structures in proteins. In examinations on biological functions of proteins, it is important to investigate structural similarity between structures of proteins. Through this function, a user can search for structures of proteins similar to the one being examined. From experimental results on the application of this function to clustering problems, it was shown that a user can discover clusters of structures which are similar. Biologists want to compare these clusters based on structural similarity with clusters mainly based on biological functions of proteins[28].

However, there are shortcomings in the application of this function to clustering problems, one of them is that to obtain a cluster (i.e. a closure of indirect similarity relationship), all structures in search space have to be examined concerning similarity. Since combinatorial computation is involved, the task would be too time consuming.

To compute a closure of structures which includes a specific structure of a protein, it is insufficient and inefficient to use the similarity search function of PACADE, combinatorially. In this chapter, we describe a function newly introduced into PACADE to solve the problems. This function enables a non-combinatorial computation of a cluster of indirectly similar structures, which includes a specific structure given by a user in the form of a query.

6.1 Closure of Similarity Relationships

6.1.1 Regarding Answers as Closed Queries

As shown in section 5.2, if all the arguments in a query are bound, the similarity search starts from only one specific structure. In this case, the query and the answers to it have ground atoms of the same predicate name if prefixes in the query are ignored.

[prefixed query]

```
:- super_secondary(@16,@[18,20,22],4,@"1ifb").
```

[answers to the query]

```
super_secondary(16,[18,20,22],4,"1ifb").
```

```
super_secondary(6,[8,10,12],4,"1stp").
```

```
super_secondary(19,[21,23,25],4,"2fbp").
```

Figure 6.1 shows the similarity search starting from the query. In this figure, "o" means a structure and an arrow between two "o"s, say, A and B, represents that there is a similarity relationship between A and B (B is similar to A). A grey region is an abstract representation of an error range. Regarding the above answers as closed queries, one can search for structures similar to corresponding ones (see Figure 6.2). The structures newly found are indirectly similar to the original structure. They are represented as "●"s in this figure.

By repeating such operation, one can successively search for indirect similarity relationships. More formally, starting from a set of answers A_0 which consists of only one element, that is, the original structure, sets of answers $A_1, A_2, \dots$ can be obtained one after another where each element of A_i is similar to an element of A_{i-1} ($i \geq 1$). Thus, A_∞ can be obtained by repeating such operation until new answers are no longer forthcoming (see Figure 6.3). A_∞ represents the set of structures which are indirectly similar to the original structure. In other words, one can compute a closure of indirect similarity relationships as an ordinary bottom-up evaluator computes a fixed point of ancestor relations.

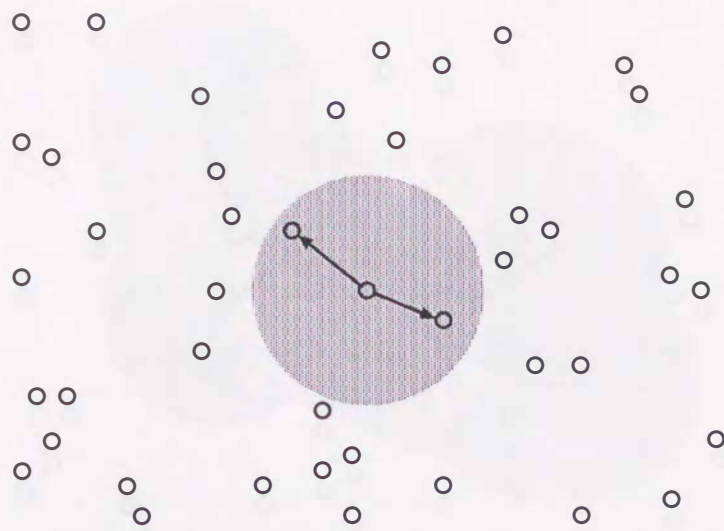


Figure 6.1: Search space (after round 1)

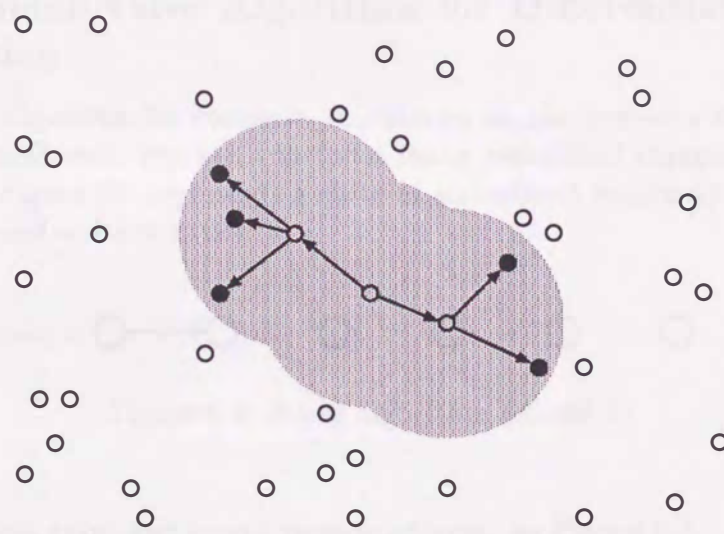


Figure 6.2: Search space (after round 2)

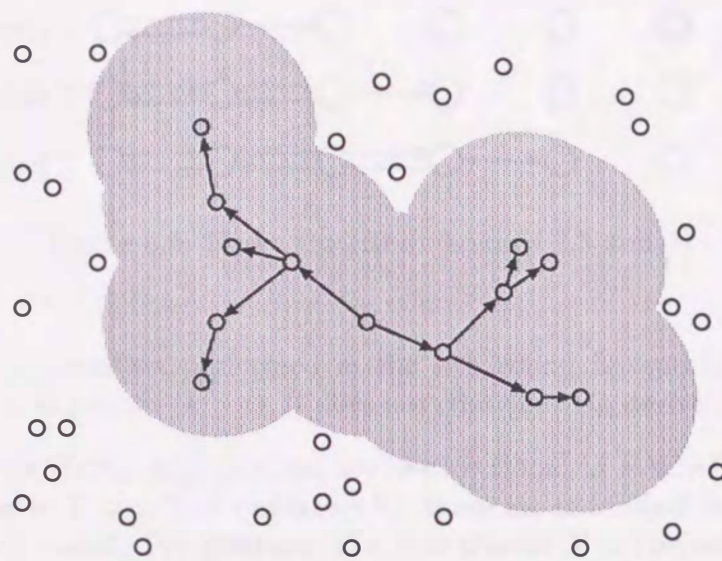


Figure 6.3: Search space (after many rounds)

6.1.2 Semi-Naive Algorithm for Differential Evaluation

The naive algorithm to compute A_∞ shown in the previous subsection is obviously inefficient because it includes many redundant computations. Assume that Figure 6.4 represents a state of an indirect similarity search after the first round of its iteration.

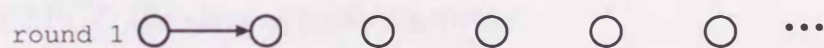


Figure 6.4: Naive algorithm (round 1)

Then, second, third and fourth rounds proceed, as Figure 6.5.

This shows that there are at least two kinds of redundant computations in the naive algorithm.

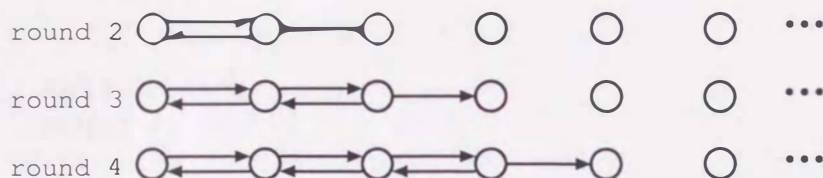


Figure 6.5: Naive algorithm (rounds 2,3 and 4)

1. The computation performed in the i -th round is repeatedly done in every k -th rounds ($k > i$). It does not contribute to derive new answers.
2. Since similarity relationships are bi-directional in PACADE (i.e. X is similar to $Y \iff Y$ is similar to X), there are redundant computations in each round. For instance, if a new answer Y is derived in the i -th round from answer X in the previous round, X is derived as a candidate of new answers in the $(i+1)$ th round, although it must not be new.

In this subsection, an algorithm optimized against the first kind of redundancy is described. It incrementally searches for indirect similarity relationships using only queries corresponding to newly derived answers of the previous round. It is based on the same idea of a the semi-naive evaluation algorithm of the ordinary bottom-up evaluator[31].

Here is a formal description of a semi-naive algorithm to compute a closure of indirect similarity relationships:

INPUT: the same as naive algorithm

OUTPUT: the same as naive algorithm

ALGORITHM:

```

 $i := 0; Q_0^p := Q_{orig}^p; A := \emptyset;$ 
repeat
   $A' := sim\_search(R^p, Q_i^p, B_1, \dots, B_k);$ 
   $\Delta A' := A' - A$ 
   $A := A \cup \Delta A'$ 

```

```

     $i := i + 1;$ 
     $Q_i^p := a\_to\_q(\Delta A');$ 
until  $\Delta A' = \emptyset;$ 
output  $A$ 

```

where Q_i^p is a prefixed query for a similarity search. A , A' and $\Delta A'$ are sets of answers. The function *sim_search* performs similarity searches for each element in Q_i^p , using a prefixed rule set R^p and base relations $B_1, \dots, B_k$ stored in PACADE. It returns the union of sets of the answers to queries in Q_i^p . On the other hand, the function *a_to_q* converts a set of answers into a set of corresponding queries. Generally, the conversion is as follows:

[before conversion]

$$p(c_1, \dots, c_m, c'_1, \dots, c'_n).$$

[after conversion]

$$: -p(@c_1, \dots, @c_m, c'_1, \dots, c'_n).$$

where $c_1, \dots, c_m$ are the constants in arguments filled with prefixed constants in Q_{orig}^p , while the arguments for $c'_1, \dots, c'_n$ are filled with normal (non-prefixed) constants.

If the current set of answers does not contribute to derive new answers, this algorithm terminates. The condition on termination of algorithm is the same one that the naive evaluation algorithm of the bottom-up evaluator [31] have.

Since the function *sim_search* performs direct similarity search through bottom-up evaluation, this algorithm can be regarded as “meta-level bottom-up evaluation.”

The differential computation through this semi-naive algorithm is illustrated in Figure 6.6.

6.1.3 Optimizing Characterization Search Using Magic Set Method

In addition to optimization through differential evaluations at the meta-level, optimization of a direct similarity search inside the *sim_search* should be examined. As shown in chapter 4, the similarity search mechanism of PACADE

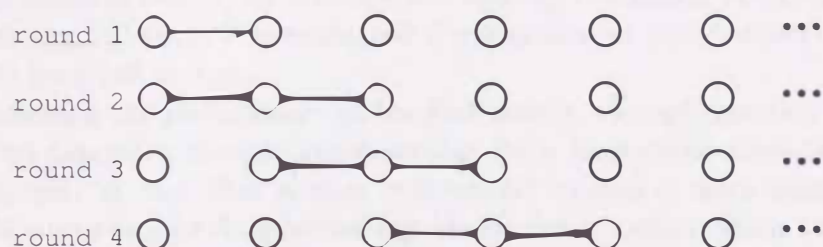


Figure 6.6: Semi-naive algorithm (rounds 1,2,3 and 4)

is based on the ordinary bottom-up evaluation. For this reason, existing optimization techniques based on rule rewriting should be examined whether they can be used together with a similarity search mechanism of PACADE.

Suppose that the computation of a closure of similarity relationships starts from one structure, that is, a prefixed query which does not include variables like the example query : $-p(@c_1, \dots, @c_m, c'_1, \dots, c'_n)$. in subsection 6.1.2. From the prefixed query, the preprocessor generates the following non-prefixed query to perform the first search:
[a query for the first search]

$$: -p(c_1, \dots, c_m, c'_1, \dots, c'_n, R).$$

where R is a new variable which is bound to the process of derivation through bottom-up evaluation. When *sim_search* is evaluated again at the next round of iterative computation of indirect similarity relationships, prefixed queries in Q_1^p are transformed into non-prefixed ones in the same manner as the original prefixed query. Thus, in each round of iteration, all arguments of queries for the first search are filled with constants, except the last one. These constant bindings are available to optimize the first search in each round, using the rule rewriting technique called Magic Set method[5].

In [12], we have already proved that Magic Set transformation is query equivalent, even when the rule set to be transformed is the one for the first search and it includes the special built-in predicate `make_string_rule` which is sensitive to process of derivation. Details of the proof cannot be discussed here for lack of space. The key of the proof is that, since the body atom `make_string_rule(...)` is the rightmost body atom of the

rules for the first search, all bindings are already computed before the atom `make_string_rule(...)` is evaluated if evaluation of body atoms of a rule proceeds from left to right.

Concerning the performance of the first search, the optimization through Magic Set transformation is indispensable for a reasonable execution time. For example, in case that similar n -stranded meanders were searched for over 134 proteins, the Supplementary Magic Set transformation cuts down the execution time to approximately 1/400 [12].

On the other hand, there is only a little possibility to optimize the second search through Magic Set transformation. When prefixed queries are transformed into queries for the second search, all prefixed constants are replaced by variables. Thus, the number of constant bindings available for optimization decreases. Moreover, rules for the second search are not suitable for Magic Set transformation. They are in a special class of Horn clauses, that is, they all have the same head predicate as the one in queries and their body atoms are all base predicates or evaluable predicates. Therefore, there are no recursive rules and Magic Set transformation does not contribute to optimization.

6.2 System Configuration

To compute the closure of indirectly similar structures, we added the iterator module to PACADE (Figure 6.7). The similarity search engine is the same module described in section 4.3

The iterator module receives a prefixed query Q_{orig}^p and returns a closure of structures which are indirectly similar to the original structure specified in Q_{orig}^p . During the computation of the closure, it repeatedly performs the following tasks to execute the semi-naive algorithm described in the previous section:

1. It calls the similarity search engine with a prefixed query Q_i^p .
2. It receives answers representing similar structures.
3. It accumulates answers and converts only new answers into prefixed queries Q_{i+1}^p .

This module is implemented as a C++ module outside CORAL.

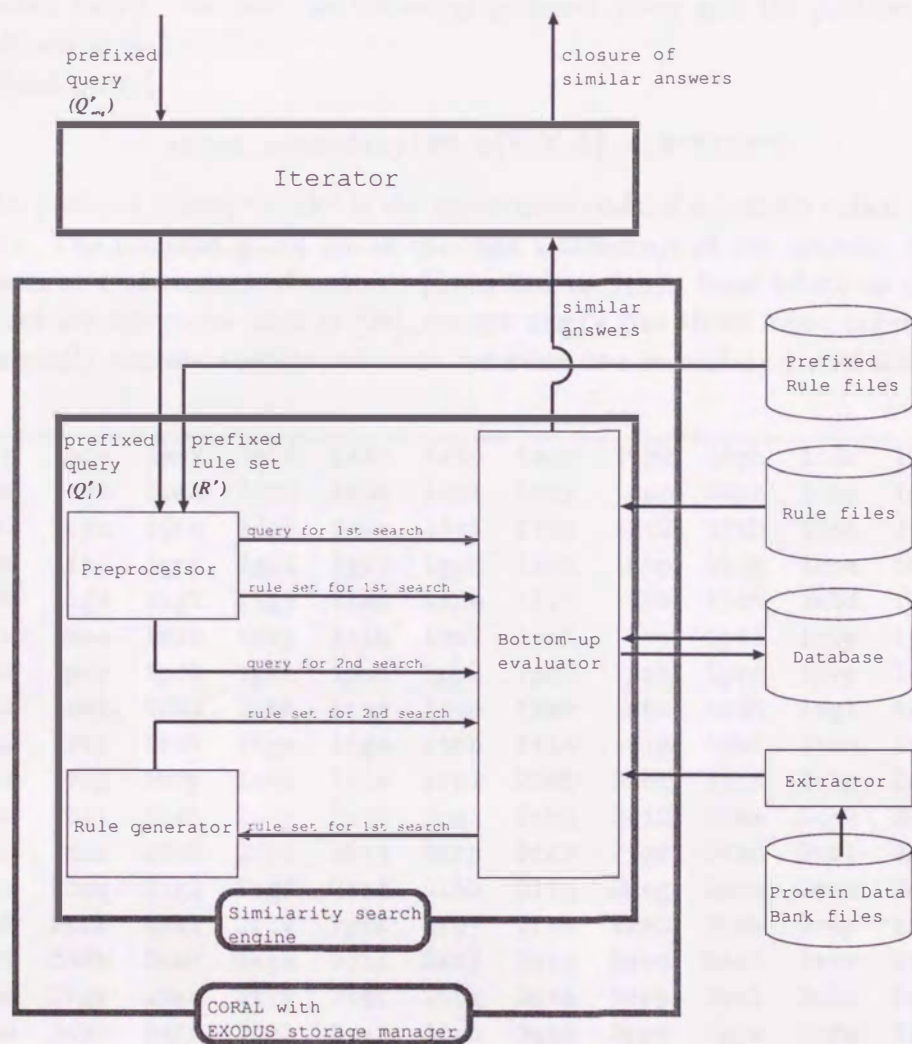


Figure 6.7: System configuration of PACADE

6.3 An Example of Indirect Similarity Search

An experimental result of computing a closure of indirectly similar structures is shown below. We used the following prefixed query and the prefixed rule set shown in section 5.2:

[prefixed query]

```
:- super_secondary(@3,@[5,7,9],4,@"3il8").
```

In the prefixed query, "3il8" is the mnemonic code of a protein called interleukin. The prefixed query above specifies a structure of the protein, which consists of 4 secondary structures (from 3rd to 9th). Base relations in the rule set are the same ones in [28], except angle has three more arguments to precisely express angular relations between two secondary structures.

1abp	1ace	1acx	1ald	1atn	1atp	1azu	1bp2	1bph	1c2r	1cc5	1ccr
1cd4	1cdt	1cms	1cn1	1coh	1col	1cox	1cpc	1cph	1crn	1cse	1ctx
1cy3	1cyc	1dfn	1dph	1est	1fai	1fbp	1fc2	1fcb	1fdh	1fdl	1fnr
1fxa	1fxi	1gcr	1gd1	1gp1	1gpd	1hho	1hhp	1hip	1hoe	1hrh	1hsa
1ifb	1ige	1igi	1igj	1lap	1lpe	1lyd	1lym	1lzt	1mbd	1mbs	1mcp
1mcw	1mee	1msb	1mvp	1nih	1nn2	1nxb	1ovo	1p01	1p2p	1pal	1paz
1pbx	1pcy	1pfk	1phh	1pmb	1pp2	1ppd	1ppt	1prc	1pyp	1q21	1r1a
1rbb	1rei	1rhd	1rnh	1rns	1rop	1sar	1sbc	1sdh	1sgt	1sn3	1srn
1st2	1stp	1tab	1tgn	1tgs	1thb	1tie	1tlp	1tnf	1ton	1tpk	1trc
1ula	1vsg	1wrp	1wsy	1xis	1ypi	256b	2abx	2act	2alp	2apr	2atc
2aza	2bjl	2cab	2ccy	2cdv	2cga	2cha	2ci2	2cna	2cpk	2cpp	2cyp
2dhb	2dhf	2f19	2fb4	2fbj	2fbp	2fcr	2fgf	2had	2hfl	2hbb	2hip
2hla	2hmq	2ig2	2igf	2kai	2lhb	2ltm	2mcg	2mcm	2mev	2mhb	2mlt
2pab	2pfk	2pka	2plv	2prk	2r07	2rhe	2rm2	2rs5	2rsp	2rus	2sar
2sbt	2sdh	2sec	2sga	2sic	2sni	2sns	2sod	2ssi	2stv	2taa	2tbv
2tec	2tgp	2tmn	2trx	2tsc	2utg	2zta	3app	3bcl	3c2c	3chy	3cla
3cna	3cyt	3dfr	3enl	3est	3fab	3gap	3gpd	3grs	3hfm	3il8	3ins
3pcy	3pgk	3pgm	3psg	3rlx	3rn3	3rp2	3sdp	3sgb	3tln	4ape	4blm
4ca2	4cna	4cts	4er4	4gpd	4hvp	4mdh	4pep	4ptp	4rhv	4rlx	4sbv
4sgb	4tms	4tpi	4xia	5acn	5cpa	5csc	5cyt	5fbp	5ldh	5pti	5rub
5rxn	5tim	6adh	6at1	6ebx	6fab	6q21	7at1	7rsa	7wga	7xia	8cat
9api	9lpr	9wga									

Table 6.1: List of proteins

In a search of data on 255 proteins listed in Table 6.1, a closure of the following indirectly similar structures was obtained:

```

----- round 0 -----
(a0) super_secondary(3,[5,7,9],4,"3il8").

----- round 1 -----
(a1) super_secondary(21,[23,25,27],4,"1ppd").

----- round 2 -----
(a2) super_secondary(21,[23,25,27],4,"1t1p").
(a3) super_secondary(25,[27,29,31],4,"2act").

----- round 3 -----
(a4) super_secondary(20,[22,24,26],4,"2tmn").
(a5) super_secondary(21,[23,25,27],4,"3tln").

```

where answer a1 is newly obtained in the first round of iteration, a2-a3 in the second round and a4-a5 in the third round. In each round, similarity relationships between these structures are derived as follows:

```

----- round      1 -----
a0 is similar to a0.
a1 is similar to a0.

----- round      2 -----
a0 is similar to a1.
a1 is similar to a1.
a2 is similar to a1.
a3 is similar to a1.

----- round      3 -----
a1 is similar to a2.
a2 is similar to a2.
a4 is similar to a2.
a5 is similar to a2.
a1 is similar to a3.
a3 is similar to a3.

```

```

----- round      4 -----
a2 is similar to a4.
a4 is similar to a4.
a5 is similar to a4.
a2 is similar to a5.
a5 is similar to a5.
a4 is similar to a5.

```

Figure 6.8 illustrates the similarity relationships. Trivial similarity relationships, that is, " a_i is similar to a_i ," are omitted in this figure. Note that the relationships between a_4 and a_5 are obtained in round 4, however, they do not contribute to obtain new structures. Since round 4 derives new similarity relationships but no new structures, the system stops iterative computation after the round. The graphical representations of a_0 - a_5 reflect the fact that a_2 , a_4 and a_5 are similar to each other (Figure 6.9).

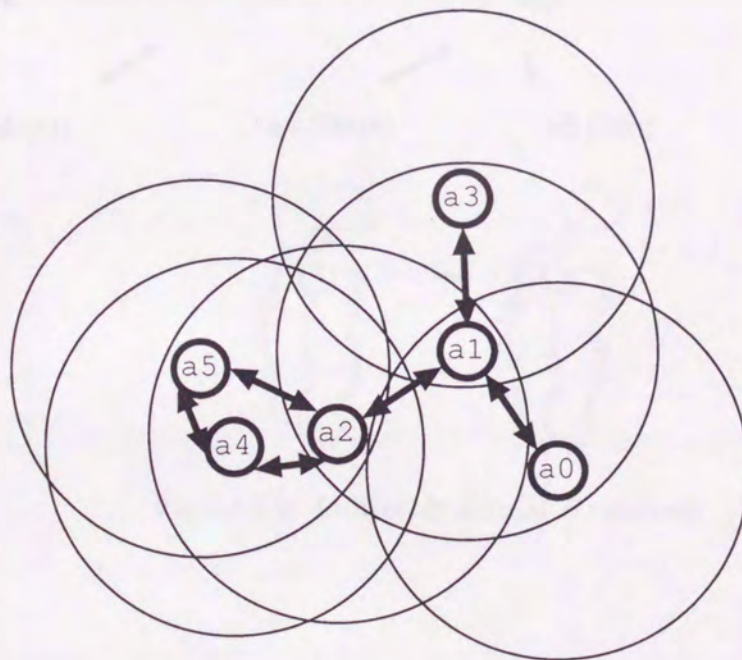


Figure 6.8: Similarity relationships among a_0 - a_5

6.1 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm for protein structure comparison. We compare the performance of the proposed algorithm with the existing algorithms, i.e., DALI, CE, and TM-align. The performance is evaluated in terms of the number of correctly identified similar structures and the execution time.

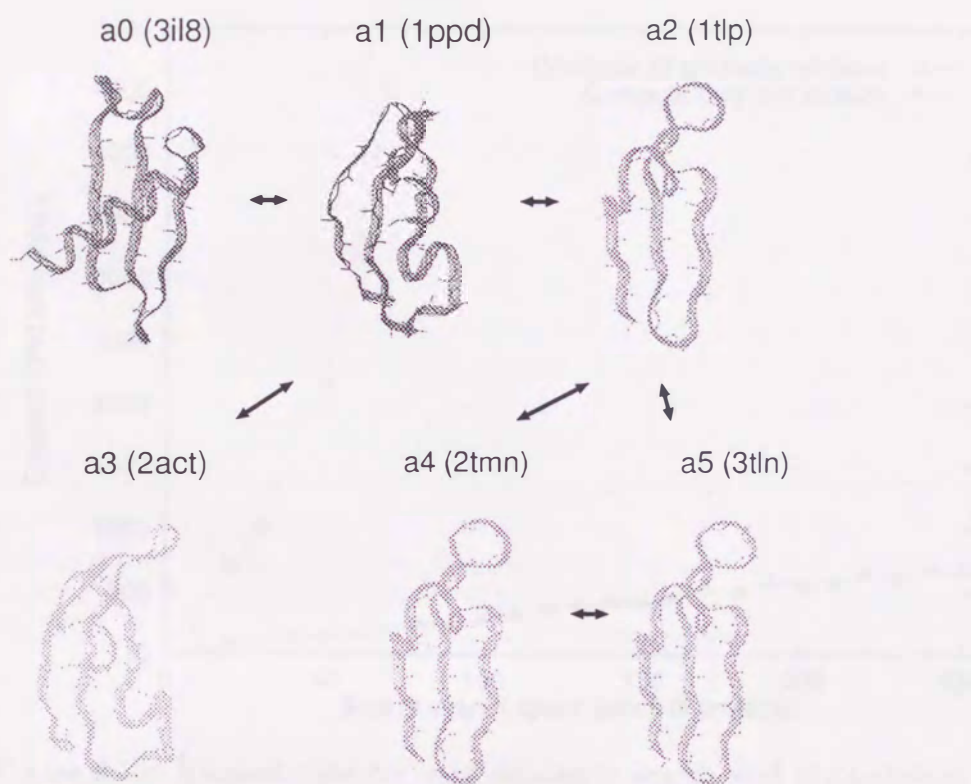


Figure 6.9: Indirectly similar structures

6.4 Performance Evaluation

In this section, we show that through two optimization technique above, i.e. Magic Set transformation and differential evaluation, a closure of structures which includes a specific structure of a protein can be computed in short time even if the search space is large.

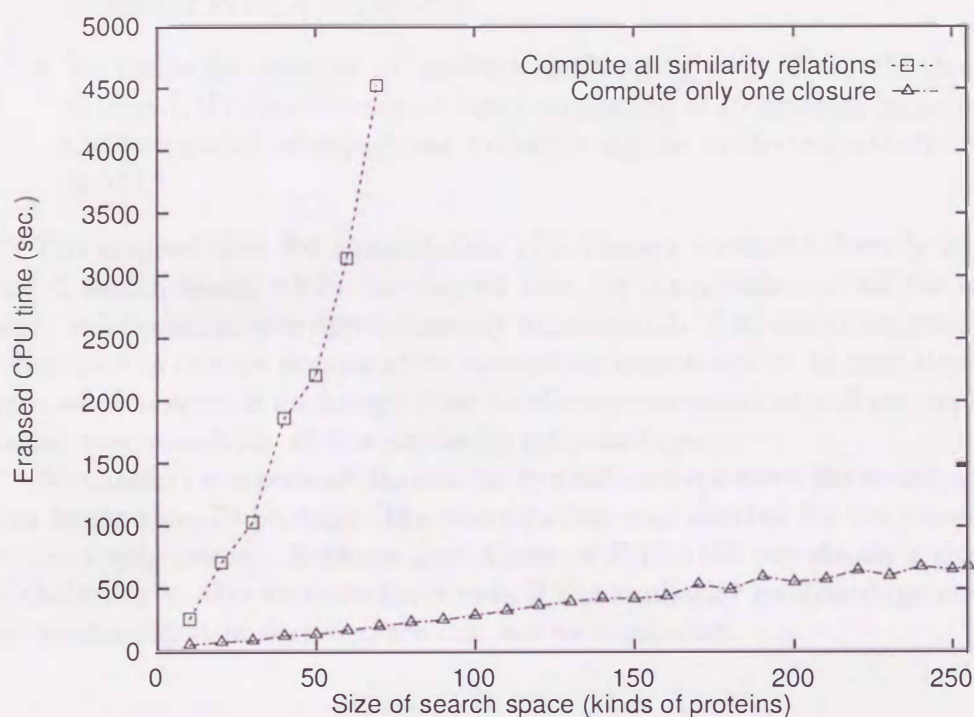


Figure 6.10: Elapsed time for vast similarity search and computation of a closure

Figure 6.10 shows two elapsed time for:

- computation of one closure of 4-stranded substructures of proteins which have indirect relationships to the specific substructure, i.e. from 3rd to 9th secondary structure in the protein 3i18.
- computation of all the similarity relationships among all 4-stranded substructures in search space.

by enlarging search space from 10 proteins to 255 proteins. This experiment was carried out on SPARCstation 20 with 512MB main memory.

In this experiment, we used the prefixed rule set in section 5.2. In each search space, the proteins in it were randomly selected from Table 6.1, but satisfies the following two conditions.

- The search space consisting of $10N$ proteins includes the search space consisting of $10(N-1)$ proteins.
- To make the number of answers unchanging even if search space is enlarged, the smallest search space consisting of 10 proteins includes all the 4-stranded substructures indirectly similar to the original structure in 3i18.

The elapsed time for computation of a closure increased linearly to the size of search space, while the elapsed time for computation of all the similarity relationships was approximately exponential. This result suggests the robustness of closure computation against its search space. In case that the number of answers is unchange, time for closure computation will not explode unlike vast search for all the similarity relationships.

We couldn't compute all the similarity relationships when the search space was larger than 70 proteins (the computation was aborted for the reason of memory exhaustion). It shows that a user of PACADE can obtain a closure of indirectly similar substructures even if the similarity relationships among all substructures in search space can not be computed.

Chapter 7

Conclusion

In this thesis, we discussed two major topics on the application of deductive database to the analysis of three-dimensional structure of protein. The results obtained through this study are as follows.

- *Development of PACADE*: A prototype deductive database system, which stores data on three-dimensional structure of protein, was developed. Some real-world examples, including hydrophobic clusters, dipole moments and supersecondary structures, could be described on the system. The results of performance evaluations show that even if a database consists of large amount of data and a rule set including recursive rules, the examples could be processed in short time. In the cases of supersecondary structures, the Magic Set method dramatically improved the efficiency of query processing (in average case, approximately $1/72$ for n -stranded meander rules and approximately $1/36$ for Greek key rules). This result indicates that deductive database is, by its nature, highly applicable to flexible analysis of large genome data.
- *Similarity Search on Deductive Database*: We developed a novel framework for a kind of approximate search, i.e. similarity search, on deductive database. It is an example of ambiguous search request for highly complex entities expressed by heterogeneous data. By extending the prototype system, it was enabled to search for similar structures in proteins with simple query description.

Basic idea of this framework is “recycle of derivation processes”. This extension did not destroy denotationality of query description which

is one of the advantages of deductive database. Queries in this extended system could also be optimized through the Magic Set method (in case of n -stranded meander, it cuts down the execution time to approximately $1/400$). As to the more extended version of similarity search function which automatically compute a closure of indirect similarity relationships among protein structures, cost of computation tended to be linear to the size of search space.

This result indicates that deductive database is suitable for managing ambiguous search requests over genome data.

Future works of this thesis are as follows.

- *Development of a Graphical User Interface (GUI) for PACADE*: For the comfortable analysis of genome data, PACADE must be furnished with a high-level GUI. The present state of PACADE is still "flexible search engine prototype", and GUI is essential in order to make PACADE a more practical research tool for biologists.
- *Knowledge Discovery in Deductive Database*: Besides flexible and high-speeded search of large amount of genome data, methods for automated knowledge discovery from them are desired under the situation that they have been increasing exponentially. In the research area of database, studies called *knowledge discovery* or *data mining* in database have been active in these years. In a sense, searches of genome data might be done in order to find out biological knowledges and the rules behind the large amount of experimental results.

In recent years, some studies in this research area become to have great importance, which are called *inductive logic programming (ILP)*. Typically, ILP systems find out logical rules from a set of base facts, a set of facts as positive or negative examples and a set of background knowledges described in the form of logical rules. Though ILP systems perform completely opposite computation in comparison with deductive database systems, we believe that PACADE have to evolve into the system in which discovery of biological knowledges can be performed as well as deductive searches of data.

However, there are difficulties in applying knowledge discovery methods to large amount of genome data:

- In some methods, computation time tends to be exponential to the sizes of domains of values. This feature may be fatal because many genome data include numerical values (integer, real, etc.) determined through experiments and they often have large domains.
- In the framework of ILP, a user has to give positive or negative examples, in the form of facts, to an ILP system. It means that, before using the system, the user must already know "what" he wants to discover. This means that almost every existing ILP systems could not be applicable to the areas like discovery of biological knowledges concerning about relationships between structure and function of protein because in such an area, the knowledges which should be discovered are not or could not be specified clearly.

In our opinion, a knowledge discovery framework proposed by Agrawal et al.[1] will be effective to the first difficulty. On the other hand, as to the second difficulty, an elegant extensions of ILP framework made by De Raedt et al.[20] will be effective as well as Agrawal's.

Bibliography

- [1] Agrawal, R., Imielinski, T. and Swami, A.N.: Mining Association Rules between Sets of Items in Large Databases, *ACM SIGMOD*, pp.207-216 (1993).
- [2] Akutsu, T.: Efficient and Robust Three-Dimensional Pattern Matching Algorithms Using Hashing and Dynamic Programming Techniques, in *Proc. of the 27th Hawaii International Conference on System Sciences*, pp.225-234 (1994).
- [3] Bancilhon, F. and Ramakrishnan, R.: An Amateur's Introduction to Recursive Query Processing Strategies, in *Readings in Artificial Intelligence and Databases*, Eds. Mylopoulos and Brodie, Morgan Kaufmann, 376-430 (1989).
- [4] Barker, W.C., George, D.G., Hunt, L.T. and Garavelli, J.S.: The PIR protein sequence database, *Nucleic Acids Research*, 19, pp.2231-2236 (1991).
- [5] Beer, C., and Ramakrishnan, R.: On The Power of Magic, *Proc. 6th ACM PODS*, pp.269-283, San Diego, California (1987).
- [6] Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer, E.F., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T. and Tasumi, M.: The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures, *Journal of Molecular Biology*, 112, pp.535-542 (1977).
- [7] French, J.C., Jones, A.K. and Pfaltz, J.L.: NSF Workshop on scientific database management, *SIGMOD Rec.*, Vol.19, No.4, 32-40(1990).

- [8] Burks,C., Cassidy,M., Cinkosky,M.J., Cumella,K.E., Gilna,P., Hayden,J.E.-D., Keen,G.M., Kelley,T.A., Kelly,M., Kristofferson,D. and Ryals,J.: Genbank, *Nucleic Acids Research*, 19, pp.2221-2225(1991).
- [9] Codd,E.F.: A relational model for large shared data banks, *Comm. ACM*, 13:6, pp.377-387(1970).
- [10] Codd,E.F.: Relational completeness of data base sublanguages, *ibid*, pp.65-98(1972).
- [11] Gray,P.M.D., Patton,N.W., Kemp,G.J.L. and Fothergrill,J.E.: An object-oriented database for protein structure analysis, *Protein Engineering*, 3, pp.235-243 (1990).
- [12] Hashimoto,S.: Studies on Optimization of Similarity Search in a Deductive Database (in Japanese), Bachelor thesis of Kyushu University, (1994).
- [13] Islam,S.A. and Sternberg,M.J.E.: A Relational Database of Protein Structures Designed for Flexible Enquiries about Conformation, *Protein Engineering*, 2, pp.431-442 (1989).
- [14] Kabsh, W. and Sander, C.: Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features, in *Biopolymers*, 22, pp.2577-2637 (1983).
- [15] Kuhara, S., Matsuo, F., Futamura, S., Fujita, A., Shinohara, T., Takagi, T., and Sakaki, Y., A.: GENAS: a database system for nucleic acid sequence analysis, in *Nucl. Acids Res.*, Vol. 12, pp.89-99 (1984).
- [16] Kuhara,S., Satou,K., Furuichi,E., Takagi,T., Takehara,H. and Sakaki,Y.: A deductive database system PACADE for the three dimensional structure of protein, in *Proceedings of the 24th Hawaii International Conference on System Sciences*, pp.653-659 (1991).
- [17] Lloyd,J.W.: Foundations of Logic Programming 2nd edn., Springer-Verlag (1987).
- [18] Morffew,A. and Todd,S.: The Use of Prolog as a Protein Querying Language, *Computers and Chemistry*, 10, pp.9-14 (1986).

- [19] Nicholson, H., Becktel, W. J., and Matthews, B. W., A.: Enhanced protein thermostability from designed mutations that interact with α -helix dipole, in *Nature*, Vol. 336, pp.651-656 (1988).
- [20] De Raedt, L. and van Laer, W.: Inductive Constraint Logic, in Jantke et al. (Eds.) *Lecture Notes in Artificial Intelligence 997 (ALT'95)*, pp.80-94 (1995).
- [21] Ramakrishnan, R., Srivastava, D. and Sudarshan, S.: CORAL-Control, Relations And Logic, *Proc. 18th VLDB*, pp.238-250 (1992).
- [22] Rawlings, C.J., Taylor, W.R., Nyakairu, J., Fox, J. and Sternberg, M.J.E.: Using Prolog to represent and reason about protein structure, in Shapiro, E. (ed.) *Third International Conference on Logic Programming*, Springer-Verlag, Berlin, pp.536-543 (1986).
- [23] Read, R. J. and James, N. G., A.: Refined crystal structure of *Streptomyces griseus* Trypsin at 1.7 Å resolution, in *J. Mol. Biol.*, Vol. 200, pp.523-551 (1988).
- [24] Richardson, J.S.: The anatomy and taxonomy of protein structure, in *Advances in Protein Chemistry*, Academic Press, pp.283-306 (1981).
- [25] Satou, K., Furuichi, E., Takiguchi, K., Takagi, T. and Kuhara, S.: A deductive database system PACADE for analyzing three dimensional and secondary structures of protein, *Computer Applications in Bioscience*, 9, pp.259-265 (1993).
- [26] Satou, K., Furuichi, E., Takagi, T., Kuhara, S. and Ushijima, K.: Similar Structure Search in a Deductive Database, In *Proceedings of International Symposium on Next Generation Database Systems and Their Applications*, pp.130-137 (1993).
- [27] Satou, K., Furuichi, E., Kuhara, S., and Takagi, T.: Searches for Topologically and Three Dimensionally Similar Structures in Proteins, *Proceedings of Genome Informatics Workshop IV*, pp.25-35 (1993).
- [28] Satou, K., Furuichi, E., Takiguchi, K., Kuhara, S. and Takagi, T.: Application of a Deductive Database System PACADE toward Discovery of

- Clusters of Similar Structures in Proteins, in *Proc. of the 27th Hawaii International Conference on System Sciences*, pp.160-169 (1994).
- [29] Schwartz, R.M. and Dayhoff, M.O., A.: in *Atlas of Protein Sequence and Structure*, Vol. 5 (ed. Dayhoff, M.O.) pp.353-358 (1972).
- [30] Takagi,T., Goto,S., Suzuki,T. and Ushijima,K.: Applicability of a Deductive Database to CAD Systems, in *Proceedings of Supplement 7th IEEE International Conference on Data Engineering*, pp.51-58(1991).
- [31] Ullman,J.: Principles of Database and Knowledge-Base Systems volume I and II, COMPUTER SCIENCE PRESS (1989).

List of Publications

1. Kuhara,S., Satou,K., Furuichi,E., Takagi,T., Takehara,H. and Sakaki,Y.: A deductive database system PACADE for the three dimensional structure of protein, in *Proceedings of the 24th Hawaii International Conference on System Sciences*, pp.653-659 (1991).
2. Satou,K., Furuichi,E., Takiguchi,K., Takagi,T. and Kuhara,S.: A deductive database system PACADE for analyzing three dimensional and secondary structures of protein, *Computer Applications in Bioscience*, 9, pp.259-265 (1993).
3. Satou,K., Furuichi,E., Takagi,T., Kuhara,S. and Ushijima,K.: Similar Structure Search in a Deductive Database, In *Proceedings of International Symposium on Next Generation Database Systems and Their Applications*, pp.130-137 (1993).
4. Satou,K., Furuichi,E., Kuhara,S. and Takagi,T.: Searches for Topologically and Three Dimensionally Similar Structures in Proteins, *Proceedings of Genome Informatics Workshop IV*, pp.25-35 (1993).
5. Satou,K., Furuichi,E., Takiguchi,K., Kuhara,S. and Takagi,T.: Application of a Deductive Database System PACADE toward Discovery of Clusters of Similar Structures in Proteins, in *Proc. of the 27th Hawaii International Conference on System Sciences*, pp.160-169 (1994).
6. Nishikawa,A., Satou,K., Furuichi,E., Kuhara,S. and Ushijima,K.: Equipments for Statistical Analysis in a Deductive Database System -their Implementation and their Application to Protein Structural Analysis-, in *Proc. of International Symposium on Advanced Database Technologies and Their Integration*, pp.237-244 (1994).

7. Nishikawa,A., Satou,K., Furuichi,E., Kuhara,S. and Ushijima,K.: Data Classification Component in a Deductive Database System and its Application to Protein Structural Analysis, *Special Issue on Advanced Database Technologies of IEICE Transactions on Information and Systems*, Vol.E78-D, No.11, pp.1377-1387 (1995).
8. Satou,K., Furuichi,E., Hashimoto,S., Tsukamoto,Y., Kuhara,S., Takagi,T. and Ushijima,K.: Development of a Deductive Database System for Computing Closures of Similarity Relationships among Protein Structures, *Journal of Japanese Association for Artificial Intelligence*, Vol.11, No.3 (to appear).

