Published in the proc. of the SSDBM'99 international conference

A Query System in a Biological Database

Dunren Che, Yangjun Chen and Karl Aberer
GMD-IPSI, Dolivostr. 15, 64293 Darmstadt, Germany
{che, yangjun, aberer}@darmstadt.gmd.de

Abstract

In this paper, we present a query system that has been implemented in a practical biological database - GPCRDB. Distinguishing features of this system include: smart query relaxation and smooth integration of navigation with conventional language-based query functions. Query relaxation is required due to the fact that queries are not always effective (in other words, expected results are frequently not achieved), particularly in scientific databases like biological databases, in which complex domain knowledge is heavily used. On the other hand, navigation capability is desired as complex data sets are involved, especially in a WWW based environment where multiple hyperlinks are often employed. For efficient implementation, the "fuzzy equivalence class" concept has been applied that captures an important type of domain knowledge.

1. Introduction

Biological databases are among the most important classes of scientific databases. A variety of biological databases has been developed that provide database support for research activities conducted in different biological disciplines and practical applications in the pharmaceutical industry. Well-known examples include GDB [PMFR92], GenBank [NCBI92, WZ98]], GSDB (Genome Sequence Data Base), GCRDb [GC98], GPCR mutant database [GP98] and GPCRDB [GPCR98]. Protein or DNA sequence data are the primary data that reside in these databases while various related data such as annotations, mutant information and physico-chemical characteristics are often added as well. All these systems are equipped with information retrieval mechanisms to help biologists to solve their problems.

In this paper, we report a new query system that has been implemented in a practical biological database system - GPCRDB. In this system, query *relaxation* is important due to the fact that a query submitted to the system is usually domain knowledge related. As a consequence, "nothing found" may be frequently encountered to the user, whereas a different formulation of the query might have provided the data of interest. *Navigation* is another interesting aspect of our system which is implemented as an ideal supplement to the conventional query mechanism. It guides the user to the right subjects of his/her interest. Navigation helps the user to drill-down into a data sub-space for more details on a certain aspect. It is useful for GPCRDB users not only because the data sets to be handled are complex, but also because various hyperlinks that point to other GPCR-related resources available on the WWW are frequently involved.

Distinguishing features of the query system include:

(1) A *smart query engine* is developed that, when activated on user's request, can relax a user query intelligently (with the help of the domain knowledge). Different from the relaxation idea described in [YK98], we consider similarities among different concepts: similarity among keywords and among residue types. For the first, two hierarchies are constructed: *family tree* and *thesaurus hierarchy*. Then, the relaxation can be done bottom-up along the tree structures. For the second, a new concept, the so-called *fuzzy equivalence classes* is developed to capture the relationships among residue types.

(2) The *top-down* presentation style adopted in our query system allows a user to first get a general impression about the results of his/her query as quickly as possible; then use it as a starting point to obtain the data of his/her particular interest more efficiently.

(3) *Navigation* capability is smoothly integrated into our query system within the top-down presentation framework.

Both the top-down style and the navigation capability are strongly domain-oriented. Using them, a user can quickly get familiar with the complex domain knowledge and can easily access the relevant data sub-space.

(4) *Over-adaptation* is a common problem with regard to adaptive systems. In our case, this issue is concerned with *over-relaxation*, which can be avoided by conducting relaxation step by step. The first iteration explores the most credible alternative that relaxes the original query in a certain aspect; the next iteration favors another aspect, and so on. When all alternatives are tried, all possible relevant data regarding the original input query are retrieved. The advantage of this approach is that the user can interrupt the relaxation process as soon as the results of interest have been obtained.

(5) We have also developed suitable access structures for the main data - sequences and sub-sequences, by which redundant store of sequences is avoided and the sequence data (as a whole) can be quickly reconstructed from the sub-sequences whenever needed.

The remainder of the paper is organized as follows. First, in Section 2, a general description of the system is given. Then, in Section 3, we describe the main functions supported by the GPCRDB query system. Next, in Section 4, we show the advanced query interfaces that are representative and can reflect some of the design styles of our system. Section 5 reports some implementation issues. Finally, a short conclusion is set forth in Section 6.

2. GPCRDB system description

In this section, we describe the GPCRDB system in general. This includes the query system requirements, the data space of GPCRDB, and the query system architecture.

2.1. Query system requirements

GPCRDB has been developed as an advanced data management system for G-Protein Coupled Receptors (GPCRs) within the European Bioinformatics Program. The biological goal of this project is to rationalize the drug discovery process (which previously was mainly guided by "trial and error") as GPCRs are the main target in the human body for today's medicines. The system will be routinely queried by academic users (biologists) and industrial users (pharmacists) and is intended to be accessible on WWW. This cooperative project requires the gathering of data related to all aspects of GPCRs, the verification of data quality and consistency, the annotation of individual data items, and, last but not least, an effective query system. Within this large project we are mainly responsible for the query system.

This task needs to satisfy the following requirements:

(1) Appropriate handling of various, domain-specific, *non-standard* data types, e.g., G-protein sequences and hierarchical classification of the sequences.

(2) A simple, intuitive user-computer interface since the typical users of the GPCRDB are non-computer-experts and in particular not experienced in using either complex interfaces or query languages.

(3) Flexible query processing, in particular, smart *query relaxation* when the users demand more relevant matches. This function is highly desired since the users often have difficulty in systematically exploring all alternatives due to the domain complexity.

(4) Support of a suitable amount of *user interactions* so that the users can conduct the relaxation process toward the aspects of their particular interests.

(5) The capability of *navigation* among the obtained results as the users accessing the database often have difficulty in precisely specifying their information needs. They may get a large set of results but containing lots of unwanted or less interesting ones. This navigation capability, on the other hand, allows them to focus step by step on a specific point of interest. In other words, the system allows them to access the database in an *explorative* manner.

To cover all these requirements, we have designed a simple, *form-based*, access to the database. Supported queries (forms) range from elementary keyword query, over a number of specialized queries focussing on a single or a combination of specific aspect(s) of GPCRs, to more advanced ones that can combine up to six different query criteria.

2.2. Data space of GPCRDB

Multidimensionality generally exists with real-world data. Data to be handled in GPCRDB involve various, domain-specific, and non-standard data types, including protein sequences, secondary structure motifs, and hierarchical classifications of the GPCR proteins (e.g., the so-called family tree).

The primary residents in the GPCR data space are the G-protein sequences, which are associated with various feature data (as mentioned above). These feature data or GPCR related data constitute a multidimensional data space. The various features are conceptually grouped into three clusters. *Textual feature* data such as sequence's identifiers, keywords, and a variety of annotation information that is made by the domain experts. *Hierarchical feature* data like family classification (the family tree) and the thesaurus hierarchy (according to which the sequence set is partitioned). *Discrete feature* data, e.g., residue mutants (relating to another database), ligand (binding information) and secondary structures (named sub-sequences) as well as other physicochemical characteristics of sequences.

Queries (for sequences or related data) can concern any of these features or sequences, e.g.,

- Retrieve sequences belonging to a specific (sub)family and matching a given keyword;
- Search for sequences that contain a specific residue pattern (given as a regular expression) and a specific secondary structure pattern;
- Find sequences that match a given residue pattern and contain mutations within the pattern;
- Display the ligand binding information of the sequences

that satisfy a particular condition (as above).

2.3. System architecture

As an implementation platform we chose the Informix Universal Server (IUS) database management system (see [IUS97]). IUS is an object-relational database management system that combines the efficiency, stability and scalability of relational database systems with the extensibility features (that allow to extend the built-in type system with user-defined data types) of OODB systems. We use the extensibility of IUS to support the domain-specific data types required in GPCRDB, such as the long sequence data type. The new data types are seamlessly integrated in the SQL query language, which we use to implement the database access functions. In addition, the IUS offers the WebDataBlade (a powerful framework for developing WWW-based database applications), which allows us to effectively develop WWW-based user interfaces to GPCRDB.

The architecture of the GPCRDB query system is depicted in Fig. 1. It is designed to run on WWW. On the client

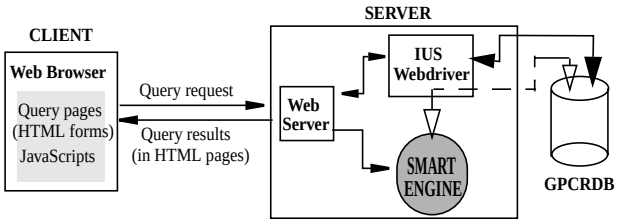

Fig. 1. GPCRDB query system architecture

side, users formulate their queries by simply filling out the query forms embedded in HTML pages. When submitted, the queries are first optimized according to some semantic knowledge on the spot by calling a corresponding *Java script*. This kind of optimization checks the validity of input parameters and reorders the condition terms involved in an input query in such a way that the cheapest terms will be evaluated first. This process uses the local computation resources and can quickly warn the user when any input is not appropriately offered. Valid queries are then transmitted to the Web server that further forwards the queries either to the IUS Webdriver (that calls for other WebDataBlade components, see [IUS97]) or to the **smart query engine**. The IUS Webdriver, as one of the WebDataBlade components, works as a gateway between Informix databases and the Web server while the smart query engine performs query relaxation and evaluation optimization if complex pattern matching (see below) is involved.

Queries submitted to the Web server are normally interpreted by the IUS Webdriver that accesses the GPCRDB database and returns query results as Web pages to the client's Web browser. However, if the user activates the smart query engine explicitly, the control will be taken over by the smart engine, and the following steps will be performed:

1. If a keyword is involved in a query, the thesaurus hierarchy (see 3.1) will be applied and the relaxation will be conducted as described in 3.2.1.
2. If a pattern, expressed as a regular expression, is involved, the sub-sequence (or sequence) and the corresponding position tree (see 5.2) will be retrieved and the relaxation will be conducted as described in 3.2.2.
3. If a secondary structure is involved, each relaxation iteration will lead to an expansion for both the two borders of the secondary structure by one residue position (see 3.2.3). That is, a larger secondary structure is generated and substitutes for the old one.

During the relaxation, users' interference is allowed so that they can conveniently choose among multiple alternatives and provide extra guidelines for subsequent relaxation of a query if the system's default method is not desired.

The smart query engine is implemented outside IUS; it uses IUS only for accessing data from GPCRDB database. When a user is not satisfied with the obtained results of his query, he can invoke the smart engine that will intelligently and iteratively relax the user's query using relevant domain knowledge. The smart engine will provide the user with an enlarged result set as long as the query is relaxable according to some domain knowledge. Users can also conduct the relaxation process toward a specific direction by making additional choices while activating the smart engine.

Navigation capability is also supported in our query system. According to the domain requirement, navigation mainly operates on the family tree. Climbing up the tree is to interactively relax the query results according to their families while climbing down the tree focuses on a specific subset of the query results. The latter is an example of the opposite of relaxation - *restriction*.

Retrieval of other related information is also possible. This includes the various feature data stored in GPCRDB and other related data (stored in other relevant Web databases). Access to related databases is accomplished as embedded hyperlinks, pointing to exact information items in related databases, which implies another type of navigation supported in our system.

3. Main functions

To systematically present the main functions of GPCRDB query system, we first characterize the main data modeled in GPCRDB in a formal way that will be used for later discussions, then describe the relaxation strategies, and finally the navigation support. The emphasis of this section is being placed on query relaxation.

3.1. Characterization of primary data in GPCRDB

for query relaxation

From an abstract point of view, a biological database is a set of protein sequences of characters from $\Re$ = {A, ..., Z}[1]. Each character stands for a "**residue type**". A biologist may issue a query against the database to get knowledge on sequences, sub-sequences, or any other relevance of a protein structure.

Formally, a biological database can be defined as a triple of the form: <*S*, H_F, H_T>, where *S* is a set of protein sequences belonging to $\Re^*$, H_F is a family tree imposed upon *S* and H_T is a thesaurus hierarchy.

For illustration, see a possible set of protein sequences as shown in Fig. 2.

s_1 = ADCFGKLPHGK
s_2 = DDCHFGKLNHGK
... ...
s_n = ...

Fig. 2. G-Protein sequences

In terms of the biological classification, *S* can be partitioned into several subsets which, in turn, can be organized into a hierarchy, called the *family tree* and denoted $H_F(S)$. Fig. 3 shows a fragment of the family tree stored in our GPCRDB database [GPCR98].

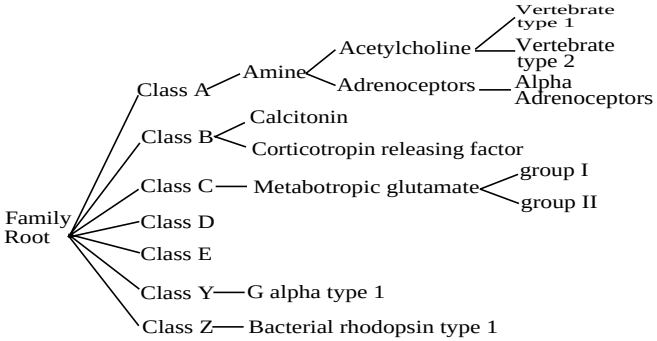

Fig. 3. A Fragment of the GPCR family tree

In this figure, each leaf node of a family tree is a type corresponding to a subset of *S*; the family root covers all the GPCR sequences, each of its child nodes corresponds to a biological category, a class of the GPCRs, which may further be partitioned into different families, subfamilies, and so on.

In addition, we can also partition *S* according to a hierarchically organized thesaurus, called the *thesaurus hierarchy* and denoted $H_T(S)$. See Fig. 4 for illustration.

1. Characters B, J, O, U, X and Z are excluded from this alphabet, $\Re$.

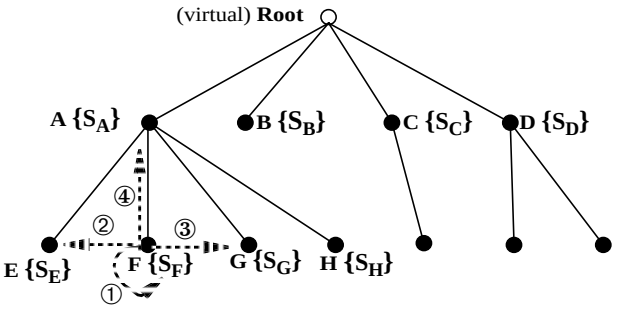

Fig. 4. Illustration of the GPCRDB thesaurus hierarchy

In Fig. 4, instead of giving a concrete example, we illustrate the generalization hierarchy first using an abstract representation. Let **A**, **B**, **C**, **D**, **E**, **F**, **G**, **H**, ... stand for the terms contained in this thesaurus, and $\mathbf{S_A}$, $\mathbf{S_B}$, $\mathbf{S_C}$, $\mathbf{S_D}$, $\mathbf{S_E}$, $\mathbf{S_F}$, $\mathbf{S_G}$, $\mathbf{S_H}$, ... for the synonym sets of term **A**, **B**, **C**, **D**, **E**, **F**, **G**, **H**, ..., respectively. For each main entry in the thesaurus, there is a corresponding node in the H_T hierarchy, and a set of synonyms of the term are associated. Numbers attached to dotted arrows indicate the relaxation steps to be tried during relaxation process (see 3.2.1). Following are several such term examples and their synonym sets:

A = *Melanocortin*

$\mathbf{S_A}$ = { };

E = *ARCH*

$\mathbf{S_E}$ = {*Adrenocorticotropic homone*, *Melanocortin-2*};

F = *Adrenocorticotropic homone*

$\mathbf{S_F}$ = {*ARCH*, *Adrenocorticotropin*};

G = *Melanocortin-2*

$\mathbf{S_G}$ = {*ARCH*};

H = *Adrenocorticotropin*

$\mathbf{S_H}$ = {*Adrenocorticotropic homone*};

A similarity relationship is defined between a pair of terms if one is a synonym of the other.

One important difference between H_F and H_T is that in the former a sequence is not allowed to belong to multiple (sub)families at the same level while in the latter, multiple memberships are allowed.

In fact, organizing data as above covers the main complex interrelationships of data: aggregation (similar to the family tree), generalization (the thesaurus hierarchy when seen from an epistemological point of view) and set-formation constructs.

In the biology domain, a sequence $s_i \in S$ is typically partitioned into several sub-sequences, the so-called *secondary structures*, reflecting particular biological properties of s_i; and very often a query is formulated using such sub-sequences as query conditions. (How to partition a sequence as

secondary structures is determined by biologists according to the 3D structure of the sequence and other biological properties. Normally, a secondary structure is named and the name indicates some biological meaning.)

Therefore, s_i is represented as

$$s_i = s_{i1}s_{i2} \ldots s_{in_i},$$

where each s_{ij} represents a sub-sequence.

At last, due to the similarity among the residue types, we partition alphabet $\Re$ into several *fuzzy equivalence classes* (*FEC*s) I_1, ..., I_m for some $m \geq 1$. The membership function of I_k ($k = 1, ..., m$) is defined as follows [Co93]:

$$f: 2^{\Re} \rightarrow [0, 1].$$

The following are several examples of *FEC*s.

I_1: {A}; $f(I_1) = f(\{A\}) = 1$.

I_2: {S, T}; $f(I_2) = f(\{S, T\}) = 0.9$.

... ...

I_4: {E, D}; $f(I_4) = f(\{E, D\}) = 0.9$.

I_5: {G, P}; $f(I_5) = f(\{G, P\}) = 0.9$.

I_6: {E, I, W}; $f(I_6) = f(\{E, I, W\}) = 0.8$.

... ...

With regard to relaxation, we consider queries that contain:

- keywords and/or
- secondary structures and/or
- residue patterns expressed as regular expressions or as a fuzzy regular expression (see below).

Residue patterns may apply either to a particular secondary structure or a sequence as a whole.

For evaluating a query containing keywords, H_T will be searched and H_F may be navigated by the user according to the query results obtained from the query system.

To find the sequences matching a pattern, we have built a position tree over each protein sub-sequence to speed up the evaluation (see 5.2).

As has been mentioned before, the users sometimes need some help or a clue how to obtain the results of interest. This lead to the development of an additional component of the GPCRDB: smart query engine. It iteratively and intelligently relaxes (reformulates) the user's query using pertinent domain knowledge (e.g., the fuzzy equivalence classes of residue types) on the above three aspects of a query.

In the following subsection, we first concentrate on the relaxation of keywords (along H_T), then on residue types, and finally on secondary structures.

3.2. Query relaxation

For query relaxation, we concentrate on the condition part of queries. So we simply represent a query as a condition in the form: $(c_{11} \wedge ... \wedge c_{1i_1}) \vee ... \vee (c_{j1} \wedge ... \wedge c_{ji_j})$, where each c_{kl} is a basic query form (discussed below).

Generally speaking, relaxation can be done either syntactically or semantically. Typical syntactic relaxation is made by replacing a '$\wedge$' with '$\vee$', or simply dismissing a less important $\wedge$*–term* (if the terms are weighted), say, some c_{kl}. By the semantic relaxation, pertinent domain knowledge is used.

One typical form of c_{kl} is $p(a,v)$, where p stands for a concrete predicate such as *"equal to"* or *"similar to"*, a indicates an attribute of an interesting type of data items, and v is typically a value. Generally, either p, a, or v can be relaxed (replaced by a more general counterpart). In our case, both keyword and residue pattern relaxation will relax v, while secondary structure relaxation relaxes a (in addition to v).

Consequently, three kinds of basic query forms are distinguished in our system: those containing a keyword, those containing a residue pattern (expressed as a regular expression), and those containing a secondary structure. In the following, we discuss the relaxation technique for each of them. First, in 3.2.1, the keyword query is addressed. In 3.2.2, we give a formal description of the pattern relaxation. Finally, in 3.2.3, we briefly outline the relaxation of secondary structures.

3.2.1. Keyword relaxation

The keyword query form is expressed just as a keyword *kw*.

For such a query form, the relaxation is very straightforward. We calculate the level numbers for all nodes in the hierarchy H_T (the root is level 0.) When a keyword query *kw* is issued, all the sequences below it in H_T will be returned. If the user expects more matches, the query is relaxed in the following way. Assume that *kw* is at level *k*. First, the sequences below all the synonyms of *kw* are returned. During the next iteration, one of the closest nodes to *kw*, say the left brother node *v* (if existent, otherwise the right brother node is used) of *kw* is called upon and all the sequences below *v* will provide the additional output. During the third iteration, another node, the right brother node of *kw* if existent, which is regarded as the second closest to *kw*, will be considered. During the fourth iteration, we jump to the parent node of *v* no matter whether an additional brother exists, and all sequences covered by the parent node are returned. This process is repeated until the expected results are obtained or the involved keywords are not further generalizable due to the absence of more brothers or parent nodes. But an exception is made for the sub-root nodes at level 1, because generalization of these nodes to the unique root node will make a

keyword query return all the sequences stored in the GPCRDB, which trivializes the keyword query and leads to a tremendous burden for the query engine. Note also that the third brother (if existent) in a subtree is always bypassed during this generalization process. The order of trials made during this process is depicted in Fig. 4. That is, first ① is tried, then ②, next ③, and last ④; the same process can be repeated next time with the newly interpolated parent node.

3.2.2. Residue pattern relaxation

A pattern query form is represented as a pair of the form: <*ss*, *E*> or <_, *E*>, where *ss* is a sub-sequence name, *E* is a regular expression representing a pattern and "_" means "any sub-sequences". If the query form is <_, *E*>, the database will be searched for those sequences containing the pattern *E*. If <*ss*, *E*> is submitted, *ss* will be checked to see whether it contains *E*. If so, all the sequences containing *ss* as a sub-sequence will be returned. The following are two examples of pattern queries:

<Transmem-1, A*D*C>,

<N-terminus, [AD]*TT*>,

where "Transmem-1" and "N-terminus" are two sub-sequence names and A*D*C and [AD]*TT* are two regular expressions (see [AHU74]).

For the purpose of relaxation, we define the following concepts.

Definition 3.1 (*fuzzy regular expression*) Let *E* be a regular expression. Let l_i (i = 1, 2, ..., m) be the different characters appearing in *E*. By replacing each l_i with its fuzzy equivalence class I_{k_i}, we get another regular expression *E*', called the fuzzy regular expression (*FRE*).

Definition 3.2 (*mixed regular expression*) Let *E* be a regular expression. Let l_i (i = 1, 2, ..., m) be the different characters appearing in *E*. By replacing some l_{i_j}'s (j = 1, ..., h; $h < m$) with the corresponding fuzzy equivalence classes we get another regular expression *E*', called the mixed regular expression (*MRE*).

Note that each fuzzy equivalence class *I* is associated with its membership function value *f*(*I*), based on which a value for a mixed (fuzzy) regular expression is calculated.

Definition 3.3 (*value of mixed regular expression*) Let *E* be an *MRE*(*FRE*). Let I_i (i = 1, ..., h) be the fuzzy equivalences appearing in *MRE*(*FRE*). Then the value of the *MRE*(*FRE*) is defined to be $\min\{f(I_1), ..., f(I_h)\}$.

Based on this definition, the relaxation of a pattern query can be specified as follows.

Let *M* be the set of all *MRE*s generated from the regular expression to be evaluated. We sort *M* such that the *f*(*I*) values of all elements in *M* are in a non-increasing order. Assume that M_1, ..., M_h are ordered *MRE*s in *M*, which can be dynamically produced and evaluated during a relaxation process. Below is a simplified description of the algorithm.

Algorithm *pattern-query-relaxation*

{
$i \leftarrow 1$; *success* $\leftarrow 0$;
repeat
  evaluate M_i;
  if result is O.K. **then** *success* $\leftarrow 1$
  else $i \leftarrow i + 1$;
until *success* = 1 **or** M_i does not exist
}

In the next section, we will discuss the evaluation optimization of the pattern queries expressed as *MREs* (*FREs*).

3.2.3. Secondary structure relaxation

For queries involving secondary structures, not only the residue patterns can be relaxed, but also the borders of secondary structures can be expanded.

Assume that *s* is a sequence, s_i, s_j, s_k are three adjacent secondary structures of *s*, and s_j is one of the interesting secondary structures in a query. That is,

$s = ...s_i s_j s_k$ ...,

$s_i = s_{i1}s_{i2} ... s_{im}$, with s_{i1} and s_{im} as borders,

$s_j = s_{j1}s_{j2} ... s_{jn}$, with s_{j1} and s_{jn} as borders,

$s_k = s_{k1}s_{ik} ... s_{kp}$, with s_{k1} and s_{kp} as borders.

The border expansion for s_j will generate a new substitute s_j' for s_j,

$s_j' = \boldsymbol{s_{im}} s_{j1}s_{j2} ... s_{jn}\boldsymbol{s_{k1}}$.

The length (i.e., the number of contained residues) of the secondary structure is increased by 2.

Border expansion of secondary structures is relatively simple. When this function is switched on by the user, the borders of each involved secondary structure are expanded as above. This process can be repeated up to five times as further expansion is considered uninteresting.

3.3. Navigation

Navigation among the data space of a database system has been widely accepted as a very useful supplement to language-based query mechanisms. The two strategies represent completely different styles of accessing information repositories. Navigation is more explorative and user-friendly as compared with language based mechanisms, but is criticized for its low efficiency. The two strategies can benefit from each other if a smooth integration of them in a single system can be obtained, which whereas appears still to be a knotty problem, and quite often to be more craft than sci-

ence.

To achieve this goal in our system, random navigation within the data space is not allowed. In other words, navigation in the GPCRDB query system is restricted in the following way:

(1) Navigation within the sequence space is only allowed along the family tree, i.e., climbing up (to visit a super family) or drilling down (to focus on a specific subfamily). For both, a starting point, i.e., an entrance node into the family tree has to be located first (see below).

(2) The family tree entrance node is determined by a user-issued query after evaluation by the system. That is, the user query needs first be evaluated and the obtained results are grouped according to their family information that constitute an interesting data sub-space (called result-space) and serve as an entrance into the entire family tree, and thus into the whole database for navigation.

In other words, there exist two navigation spaces, one is the entire database (organized into the family tree), the other is the result space which is also organized into a family subtree. Navigation conducted in this way turns out to be very reasonable and efficient (as compared with random navigation).

In fact, a third type of navigation, i.e., hyperlink based navigation on the Web is conveniently supported. The result presentation is deployed as follows. At the top level, the top summary page involves hyperlinks pointing to sub-summary pages (at a lower level) if the result space is sufficiently large. Each sub-summary page in turn contains all the matches of the query in a specific subfamily (corresponding to a concrete entrance node into the family tree); or if the result sub-space is still large, the sub-summary page may again contain hyperlinks to a set of further detailed sub-sub-summary pages, and so on. Sub-summary pages at each level may contain various hyperlinks pointing to relevant entrances in a related (remote) GPCR database such as tinyGRAP [GP98], and links to complex data fields of the sequences such as the 3D structures and the sequences themselves which are normally large and do not easily fit into the same page.

In short, the design of our system (and interfaces) follows the "*top-down*" style. Let's use the metaphor of "finding interesting trees in a forest" - first, try to get some general knowledge about the forest as a whole; then use this knowledge to help localize to one part of the forest; next to a more restricted part, and so on, until the real interesting individual trees are reached.

4. System interface

In addition to the domain requirements mentioned in 2.1, attention to some epistemological and psychological principles should be paid to construct a user-friendly query system. Following are the additional elements of our consideration:

1. A frequently encountered troublesome situation during an information search is the response of "*nothing found*" to a user's query. In practice, "nothing found" may be the result of an improperly formulated query or of really non-existent data. To reduce the false negatives (no data found due to ineffective query), intelligent query relaxation has been developed.

2. An important issue with regard to query relaxation is that careful design should be made to avoid *over-relaxation*. In our system, *prompted iterations* have been recognized as a practical approach to this issue. (An action is *prompted* if it follows an explicit user request [BMR96]).

3. In a certain sense, a user's query represents his self-esteem. The behavior of the query system should first strictly conform to the original form of the user's query; the decision whether or not to apply the relaxation function to a query is at best left to the user. Here, a useful metaphor for the role of our smart query engine is to only ask "May I help?" and be prepared to offer the help to a user.

4. Navigation has to take place in the *right situation* and at the *right time*, i.e., when it is necessary. Smooth integration of this function into a language-based query system is still a knotty problem for designing a concrete system.

The GPCRDB query system allows users to query the GPCR database through a variety of ways: "family", "keyword", "secondary structure" (sub-sequence), "ligand" (binding information) and "mutants". In addition, an advanced (compound) query page is provided for formulating query expressions consisting of up to six different conditions that can be logically interconnected with "$\wedge$" or "$\vee$", constituting a classic "and-or" normal form. For example, if $c_1 \vee c_2 \wedge c_3$ is input, it will be treated as $c_1 \vee (c_2 \wedge c_3)$. That is, "$\wedge$" takes precedence over "$\vee$".

To show our design style and the features of our query system, the advanced query page is taken as a representative example.

The interface shown in Fig. 5 contains a query that can be restated as follows:

> *Find GPCR sequences that have the keyword 'ARCH' (contained in any one of the annotation fields; to each sequence a number of annotation fields are attached) and contain the pattern 'R*SS'; furthermore, the secondary structure (sub-sequence) "N-terminus" of the sequences contains the pattern 'PP'.*

The first response of the query system to an input query is a top-level summary page that presents to the user links to var-

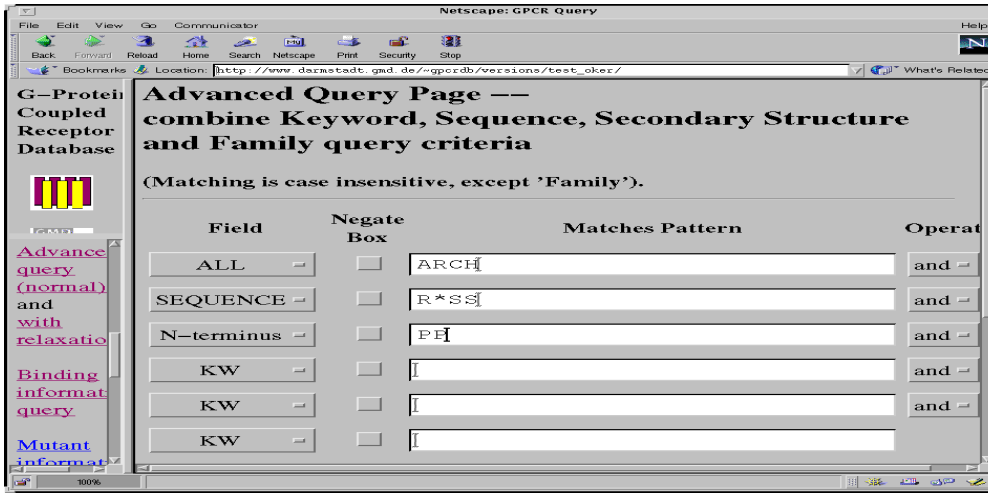

Fig. 5. The Advanced query page (forms) of GPCRDB

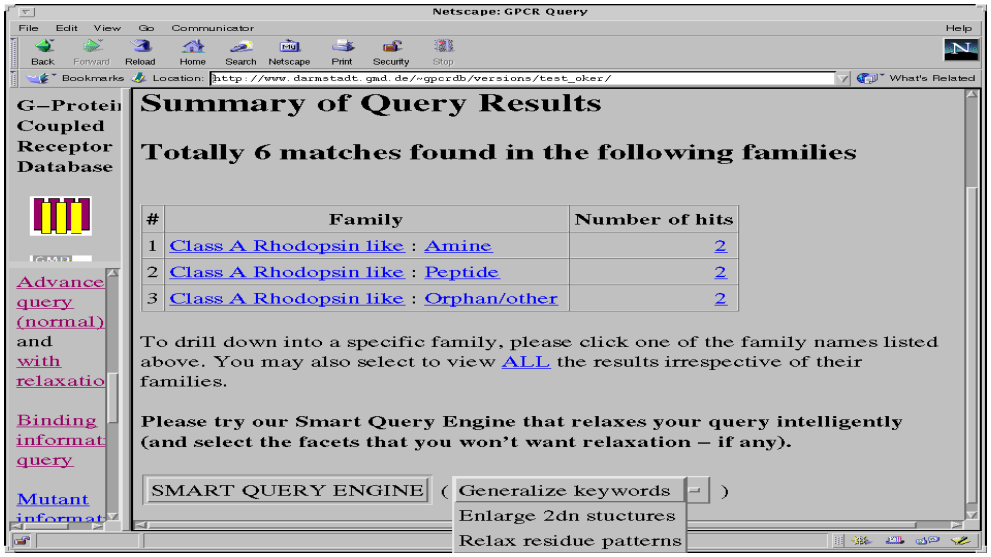

Fig. 6. Result page of ordinary query (without relaxation)

ious pages containing more details about all hits of the query (the number of the sequences satisfying the query). Fig. 6 is an example of such summary pages. In the displayed *family* column, two links are provided pointing to the top level family and a subfamily, respectively, and serve as entrances to the family tree. In the *number of hits* column, the numbers (also links) are clickable and navigate to a lower level summary page of the obtained matches in a specific subfamily. This summary page also contains a button at the bottom that the user can use to invoke the smart query engine to relax a query if he/she wants more relevant results.

When activating the smart engine, users may make extra selections from an additional option button just beside the *Smart Query Engin*e button (see Fig. 6). This additional button allows a user to select the aspects that should not be relaxed. This means that the three kinds of relaxation (operating on a different aspect of a query) discussed in the last section can be freely switched on/off. By default, all the three kinds of relaxation are to be performed if the smart engine is invoked.

When activated, the smart query engine will return to the user the first alternative (query results) of relaxing his/her query, as is illustrated in Fig. 7 (the *keyword generalization* is switched off.) With the same input query as in Fig. 6, the result page in Fig. 7 contains a much larger set of matches (22 hits *vs.* the original 6 hits in Fig. 6). This page also presents the original query predicate and the relaxed one (just executed) as an explanation to the achieved relaxation. This helps the user not only to understand the performed relaxation but also to formulate more pertinent queries in the future. In the presented relaxed query predicate (see Fig. 7), expression *rp('*R*SS*', '*R*[ST][ST]*',2)* shows that the input residue pattern '**R*SS**' is relaxed as '**R*[ST][ST]**' using the fuzzy equivalence class numbered 2 (i.e., residue type '*S*' is replaced in this pattern by its fuzzy equivalence class '*[ST]*'), and *ss(DOMAIN1)* indicates that the border expansion for secondary structure *DOMAIN1* [2] in this query has been performed once (if the border expansion has been performed two times, we will have *ss(ss(DOMAIN1))* instead of *ss(DOMAIN1)*.)

5. Implementation

In this section, we address two aspects concerning the system implementation: sequence storage structure and pattern matching optimization.

5.1 Sequence storage structure

2. *DOMAIN1* is the old name of *N-terminus* that is still used internally in our system, but update will be made soon.

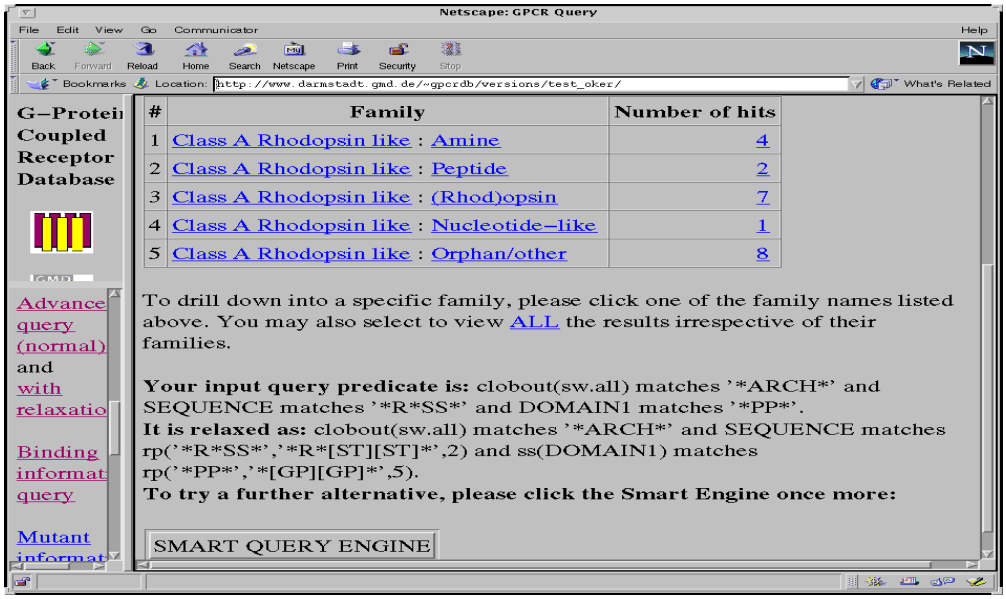

Fig. 7. Result page of a relaxed query returned by GPCRDB Smart Query Engine

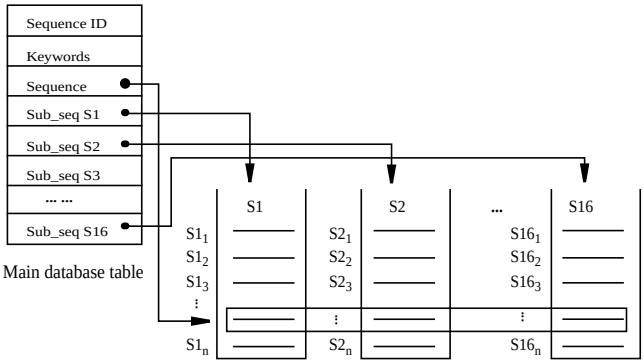

Fig. 8. Storage structure of GPCR sequence database

Data to be handled in the GPCRDB involve various, domain-specific and non-standard data types. These include protein sequences, secondary structure motifs (sub-sequence variants), and hierarchical classifications of the GPCR proteins (i.e., the so-called family tree), mutant and ligand (binding information), and other chemical/physical characteristics. These data are classified as two types: i.e., the GPCR sequence data and GPCR-related data. Most of the GPCR related data are annotations made by the domain experts, such as keywords, descriptors, ligand (binding information), and mutant data. Typical queries issued by the intended users indicate a search in the database for particular sequences, e.g., those containing a given residue pattern in the sequences or in a specific range (secondary structure) of the sequences. GPCR related data may be queried in a similar way but have to be combined with sequence information. As long as the sequences have been identified, other related data of the sequences can be easily obtained through the IDs (identifiers) of the identified sequences.

Therefore, in the following, we only show the main storage structure of sequences (depicted in Fig. 8).

For each sequence, a unique ID, a number of keywords, the primary sequence data, and 16 substructures (sub-sequences) are logically defined. A sequence is physically decomposed as 16 sub-sequences (i.e., secondary structures). They are separately stored in 16 buckets, S1, S2, ..., S16. This matrix storage structure can efficiently support the two primary types of sequence matching that are of interest to the users of GPCRDB:

1. Search for sequences that contain a given residue pattern (a regular expression).
2. Search for sequences that contain a given residue pattern in a specific sub-sequence (a secondary structure) of the sequences.

In the main database table (see Fig. 8), the "sequence" field simply holds the internal sequential number assigned by the system for each sequence. Using this number, the system can easily locate the 16 sub-sequences and quickly construct the sequence data (as a whole) using the 16 sub-sequences when they are needed to perform an entire sequence matching. The matrix storage structure greatly facilitates quick pattern-matching of sequences as a whole and of any of the 16 sub-sequences. In our system, pattern search in sub-sequences is even more frequently required for the evaluation of user queries.

If a query initiates a pattern matching with entire sequences, the 16 sub-sequence buckets are read out in parallel; whole sequences are constructed on the spot, and pattern matching is then carried out sequentially against each of the sequences. Once a match is found, the relative position of each involved sub-sequence in its sub-sequence stream is mapped to the ID of the corresponding sequence that is unique in our database.

If a pattern matching is against a sub-sequence, only the corresponding bucket of the sub-sequence needs to be read out, and pattern matching is then carried out in a similar way.

For border expansion of a sub-sequences, the predecessor and successor (if existent) of the sub-sequences are additionally read out for the construction of an expanded sub-sequences' stream.

5.2. Query evaluation optimization

We distinguish between two kinds of query optimization in our system. They are

1. the semantic query optimization and
2. efficient (fuzzy) pattern matching.

In general, an issued query takes the form: $(c_{11} \wedge ... \wedge c_{1i_1}) \vee ... \vee (c_{j1} \wedge ... \wedge c_{ji_j})$, where each c_{kl} is a basic query form. It is either a keyword or a pair of the form: <*ss*, *E*>, where *ss* is a sub-sequence name or a symbol "_" meaning "any" and *E* is a (fuzzy) regular expression representing a pattern.

By the semantic optimization, the order of the conjunctive and disjunctive basic query forms in the original query will be changed so that the keyword queries are always evaluated first. Then, the pattern queries with sub-sequence names will be executed. Next, the pattern queries with no sub-sequence names will be performed on the results returned from the previous execution. This kind of optimization is performed on the client site so that the network transmissions are minimized and the server resources remain available.

To make the (fuzzy) pattern matching efficient, we have built a *position tree* [We73, AHU74, CR94] for each sequence (due to space limitation, we omit the details of constructing a position tree). Since the regular expressions submitted to the system are normally short, we first transform it into a deterministic finite automaton (*DFA*) which takes $O(2^{|P|})$ time, where *P* is the regular expression, for which the *DFA* is constructed. Then, the position tree will be searched against the *DFA* to check whether the regular expression is satisfied.

6. Conclusion

In this paper, we have described the GPCRDB query system that is designed according to the top-down epistemological style for result presentation and has an embedded smart query engine. The smart engine takes care of the query processing from IUS query engine upon user request - it iteratively relaxes and optimizes a query, then evaluates it and presents the results as a (top) summary page to the user. Three kinds of relaxations are handled: keyword relaxation (concept generalization), residue pattern relaxation, and secondary structure expansion. The implementation of residue pattern relaxation is based on a new concept - fuzzy equivalence classes which capture the similarities among residue types. In addition, a navigation mechanism has been smoothly integrated into our system. Navigation is conducted mainly by climbing-up or climbing-down the family tree hierarchy, by which any detail on any aspect of a hit (a relevant sequence) can be obtained.

The GPCRDB [GPCR98] and the query system are now operable and available on WWW. The last published working version of the GPCRDB query system can be invoked at **URL**: http://www.darmstadt.gmd.de/~gpcrdb/.

Acknowledgments: The authors are very glad to acknowledge that Hannelore Eisner has did a lot of implementation work in the GPCRDB query system. We also would like to show our gratitude to other partners of this project.

References

[AHU74]Aho, A.V., Hopcroft, J.E. and Ullman, J.D. *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Com., London, 1969.

[BMR96]Brajnik, G., Mizzaro, S., Rasso, C. Evaluating User Interfaces to Information Retrieval Systems: A Case Study on User Support, in *Proc. of the 19th Anual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1996), pp 128-136.

[Co93]Cottawald, S. *Fuzzy Sets and Fuzzy Logic: the foundations of application - from a mathematical point of view.* Braunschweig, Wiesbaden: Vieweg, 1993.

[GC98]GCRDb, URL: http://www.gcrdb.uthscsa.edu/.

[GP98]GPCR mutant database, URL: http://www-grap.fagmed.uit.no/GRAP/homepage.html.

[GPCR98]GPCRDB, URL: http://www.sander.embl-heidelberg. de/7tm/.

[GBS92]Gonnet, G.H., Baeza-Yates, R.A. and Snider, T. New Indices for Text: PAT trees and PAT arrays, *Information Retrieval*, ed.: Frakes, W.B., Baeza-Yates, R.A., Prentice Hall, New Jersey, 1992, pp. 66-83.

[HU69]Hopcroft, J.E. and Ullman, J.D. *Formal Language and Their Relation to Automata*, Addison-Wesley Publishing Com., London, 1969.

[Kn73]Knuth, D.E. *The Art of Computer Programming: Sorting and Searching*, Addison-Wesley Publishing Com., London, 1973.

[IUS97]*Informix-Universal Server - Informix Guide to SQL*, Informix Press, Menlo Park, CA, USA, 1997.

[Mo68]Morrison, D.R. PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *Journal of Association for Computing Machinary*, Vol. 15, No. 4, Oct. 1968, pp. 514-534.

[NCBI92]National Center for Biotechnology Infromation. ENTREZ: Sequences User's Guide, *National Library of Medicine*, Bethesda, MD, 1992, Release 1.0.

[PMFR92]Pearson, P., Matheson, N., Flescher, N., and Robbins, R.J. The GDB huan genome data base anno 1992, *Nucleic Acids Research* 20 (1992), 2201-2206.

[We73]Weiner P. Linear Pattern Matching Algorithms. *Conf. Recorder, IEEE 14th Annual Symposium on Switching and Automata Theory*, 1973, pp. 1-11.

[WZ96]Williams, H. and Zobel J., Indexing Nucleotide Dtabases for Fast Query Evaluation, in *Proc. of 5th Int. Conf. on Extending Database Technology*, Avignon, France, March 1998, pp. 275-288.

[YK98]Yoon, J. and Kim S.-H. A. Three-Level User Interface of Multimedia Digital Libraries with Relaxation and Restriction, in *Proc. of IEEE International Forum on Research and Technology Advances in Digital Libraries* (*ADL'98*), April 22-24, 1998, Santa Barbara, California, pp206-215.