A Query Expansion Technique for the Functional Annotation in Biological Databases

Sangwon Yoo, Kangpyo Lee, Hyoung-Joo Kim
School of Computer Science & Engineering
Seoul National University
Seoul, Korea
swyoo@idb.snu.ac.kr, kplee@idb.snu.ac.kr, hjk@snu.ac.kr

Abstract

A functional annotation in biological databases describes the activities of the gene products. It plays a key role in the analysis of gene products. The problem is that different institutes use different annotation vocabularies. We introduce a simple SQL expansion to resolve this problem. Our approach enables a user to make a query against different hierarchical annotation vocabularies. Each annotation vocabulary has its own structure and mapping between them. Our query translation algorithm translates a user query into general SQL using this information. We implemented this mechanism and evaluated it on a real biological database.

Keywords*: DBMS, biological database, annotation, gene ontology, query expansion*

1. Introduction

1.1. Functional annotation

In the biology community, there is a huge amount of data which come from genomics and proteomics study. One of the major sequence databases, GenBank[4] announced that the public collections of DNA and RNA sequences reach 100Gigabases(letters) in 2005. There are more than 165,000 organisms which are completely or partially sequenced. There are 3 million sequence submissions in a month. We can expect that the increasing rate of biological data will be much higher in the future.

There is useful information besides the sequences themselves such as publication, lineage, function and so on. Knowledge acquisition from these sequences is more important than just accumulating them. Therefore, databases keep metadata as well as sequences. When a researcher submits a sequence or an experimental result to the public databases, related information is annotated to the gene products (gene, RNA, protein) such as organism, function and experimental condition.

There have been vast efforts to make a good quality of annotation in the biology community. For example, UniProt[19] is a protein resource combining three databases, Swiss-Prot, TrEMBL[6] and PIR[3]. They are trying to provide sequences and functional annotations which are manually inspected by the experts of the domain. They trace the biomedical publications manually to annotate the proper function on the protein. There are many other similar databases which focus on the certain organism or research field.

All this effort is intended to find the function of gene products. If a protein's function is found to be a "transcription regulator", this kind of database will record "transcription regulator" on the protein's functional annotation entry. This information can help a researcher study the protein of the same family. We can say that a functional annotation is a functional activity description of gene products.

1.2. The problem

Although many databases provide useful information in the form of a functional annotation, we need to consider several characteristics when making a query against the functional annotations.

The first problem is that there are numerous functional annotation schemes that describe a similar function. Each database usually has its own classification or controlled vocabulary [6, 13, 16]. The second problem is that the functional annotation scheme has hierarchy in itself[2, 3, 13].

The third problem is that they have mapping information between them. Each annotation scheme has

ID	Protein Name	Function	Function name
Q9NY61	AATF_HUMAN	GO:0003700	①Transcription factor activity
P05549	AP2A_HUMAN	GO:0003705	②RNA polymerase transcription factor activity
Q92876	HXB13_HUMAN	InterPro:IPR001356	③Homeobox
O08686	BARX2_MOUSE	InterPro:IPR000047	④Helix-turn-helix motif
P19622	HME2_HUMAN	InterPro:IPR000747	⑤'Homeobox' engrailed-type protein

Figure 1. A protein table with different functional annotation scheme

its own coverage and characteristic. Biological database curators annotated existing entries using mapping information in case of data integration[7, 8]. If we want to get a desired query result against functional annotations, we should consider the above problems.

Figure 1 shows an example of protein database which contains different annotation vocabularies. The first two rows are annotated with Gene Ontology[13] terms and the next three rows are annotated with InterPro[16] entry. Let us consider the question, "Find proteins which are related to transcription activity and its subfunction". She only knows that GO:0003700 means transcription factor activity for this question. There is actually hidden information. GO:0003705 is a child term of GO:0003700 and IPR001356 is equivalent to GO:0003700. If a user does not explicitly express this relation, the DBMS will not return the desired result. It will return the exact matching result, GO:0003700, instead.

1.3. Our Approach

In this paper, we propose and evaluate a practical approach to support different functional annotation schemes. We designed and implemented a SQL query expansion algorithm and annotation index. When a user makes a query, we provide 'EXPAND' clause to cover the different annotation schemes. We also use an annotation index for the fast search of related terms in query expansion.

We assume that a user does not know the structure of the annotation scheme and their mapping relations A user makes a query which uses terms that are familiar to her. Our system uses these terms as a basis for query expansion. We provide the syntax that enables a user to specify the target annotation system and the range of search. When she issues a query to the DBMS using this syntax, our query translator translates this query into general SQL. The translated query includes the information about the search terms of the other annotation schemes.

In Figure 2, there is a simple architecture of our approach. In the bottom right, there are biological databases which use different annotation schemes. They are stored as relational tables.

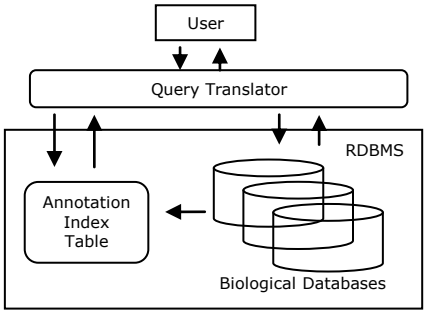


Figure 2. Architecture of query expansion

In the bottom left, there are annotation index tables. These tables have information about the structures of annotation schemes used in biological databases. There is also a mapping table which links annotation index entries.

The query translator accepts the query from the user. When it expands the user query, it utilizes the information in the annotation index table. It sends the translated SQL to the DBMS and delivers the result to the user.

The contribution of this research is that we provide a simple way to query the annotations in many databases. It is a common environment to import many public databases or integrate various resources. They have diverse vocabularies and structures among them. There are mapping relations among similar terms in different systems. This makes it too complex for a general user to query against these annotation systems.

Another contribution of this paper is that we utilize the relational DBMS and SQL features. Relational DBMS and SQL is a mature technology. They have been used for more than 30 years in academia and business. Many public biological databases are provided as a form of relational database. When a

researcher in this domain have to make her own local database, relational DBMS and SQL should be preferable.

The remainder of this paper is organized as follows. Section 2 introduces our annotation index and query translation algorithm. Section 3 shows the experiment data and its result. Section 4 describes the related work which addresses the issue of annotation in bioinformatics study and database research. Section 5 presents the conclusions and describes future work.

2. Query Expansion

2.1. Annotation Index

In Figure 1, there are relations between functional annotations as follows:

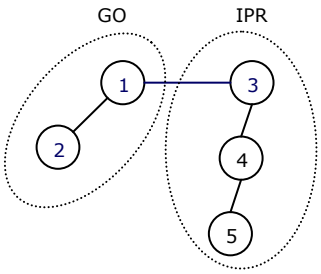


Figure 3. Hierarchical structure in annotations

A node represents an annotation term and the edge represents a relation between terms. Up-down edge shows parent, child relation and a horizontal edge shows the mapping relation between different annotation schemes. We model the annotation hierarchy as a DAG(Directed Acyclic Graph) structure. When a user query arrives, we reference this DAG structure to make an expanded search.

For example, a user submits a query as follows:

```
SELECT ID
FROM protein_table
WHERE Function = 'GO:0003700'
```

This query will return only 'Q9NY61'. If a user wants to know which proteins are related to transcription activity as well as wants to get information about the related protein family, the user should find a way to include this information in her query.

In this case, the hierarchical structures are obtained from each annotation scheme. If a query can scan the whole structure in Figure 3, the result will satisfy a user's intention.

Gene Ontology[13], InterPro[16], SwissProt[6] and EC numbers[2] are our target annotation systems. We choose these systems because they are widely used in the real databases. They have mapping information between Gene Ontology and the other systems[17]. They have complex hierarchies[13, 16].

They provide the relations shown in Figure 1 as follows:

```
<go:term rdf:about="GO:0003705">
 <go:is_a rdf:resource="GO:003700" />
 </go:term>

 <interpro id="IPR000047" type="Domain">
        <parent_list>
          <rel_ref ipr_ref="IPR001356" />
        </parent_list>
       <child_list>
          <rel_ref ipr_ref="IPR000747" />
       </child_list>
 </interpro>
```

Figure 4. Gene Ontology and InterPro relation

As you see in Figure 4, they have various types in its syntax such as XML, RDF and flat file. Their common information among them is the specification of relation types between terms used in its annotation system. We preprocessed the various formats of annotation systems and make it into a DAG structure.

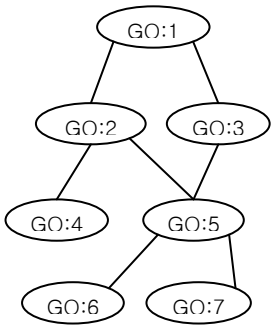
We build DAGs for GeneOntology and InterPro entries. EC number has a tree structure, while SwissProt does not have any explicit structure. Therefore, we do not need to make DAGs for these two systems.

The core of query expansion is traversing the relation between terms. It means we need an efficient way of traversing a DAG structure. We chose the simple encoding, Dewey Order[1]. Dewey Order is one of the simplest prefix based labeling schemes.

The multiple paths node has many labels in the DAG structure, whereas the node has only one label in the tree. This encoding scheme calculates every parent-child relation easily by simply comparing the prefix of each label. When we want to make a query expansion, we just reference the term label and traverse the ancestor descendant relations.

Our DAG structure is stored as a relational table and the parent-child relation is calculated using SQL operation. Figure 5 shows a DAG structure graph and its table representation. We made a table for an annotation system. The first column lists the term id or term name and the second column lists the label in the DAG structure. As we can see from this table, the parent-child relation is decided by the prefix comparing.

Node GO:4 is a child of node GO:2, because node



ID	Label
GO:1	03
GO:2	03.01
GO:3	03.02
GO:4	03.01.01
GO:5	03.01.02
GO:5	03.02.01
GO:6	03.01.02.01
GO:6	03.02.01.01
GO:7	03.01.02.02
GO:7	03.02.01.02

Figure 5. DAG structure and table representation

GO:4 and node GO:2 have common prefix '03.01'. We can also decide the ancestor-descendant relation through the same comparison. Each level encoding is separated by the comma and the length of prefix shows the distance between two nodes. GO:5, GO:6 and GO:7 have two labels while they are represented as a node in the graph. Dewey Order is initially made for the construction of a tree style classification. Therefore, there is no node which has multiple paths from the root to the terminal node. In the DAG, a node may have multiple paths. It is possible that an annotation node has two or more parents and occurs at many positions. To save the information about all these paths, we made labels for every path a node has. In Figure 5, GO:5, GO:6 and GO:7 are nodes for multiple labeling.

When calculating a parent id or ancestor ids in annotation index tables, we implemented a prefix function. We specified the desired level, and then it returned the prefix of a current label. For a child node and descendant nodes, we utilized LIKE function in SQL. In Figure 5, the fixed number of digits is assigned in each level. Thus SQL like

```
SELECT ID
FROM index
WHERE Label like '03.01.__'
```

will return the child of GO:2. If we specified the condition as '03.01.%', it would have retrieved all the descendant nodes.

Annotation index tables support the query expansion by efficient parent-child relation retrieval

2.2. SQL Expansion

When a user makes a query against functional annotations, the main concern is what kind of keywords she should include in the query. The proper term is finding the appropriate results. However, the annotation vocabularies are still large and complex to remember all the keywords.

In case of SQL, this term will be described in the WHERE clause. All the user can do is to add terms she knows in the where clause. There is no convenient way to support a parent-child relation or a mapping relation in functional annotations.

We provide the additional clause 'EXPAND' to SQL syntax, which enables a user to expand a query to the target annotation system. For example, there is a query to the protein table which uses Gene Ontology, InterPro and SwissProt annotation systems.

```
SELECT id
FROM protein_table
WHERE organism= 'Human' and
        function = 'GO:0003700'
EXPAND go>-3   ipr<+3   spkw=0
```

'go', 'ipr', 'spkw' mean target annotation systems and '+', '−' mean the direction toward descendants and ancestors in the relation respectively. '<' and '>' mean the range of search. '0' means a mapping to the equivalent target annotation term, no expansion to the parent-child relation.

In the above example, go>-3 will include three levels of query terms. They are original query term GO:0003700, its parent term and grand parent term. If GO:0003700 is on the multiple paths, it will have multiple parents and grand parents. ipr<+3 also include three levels of query terms. In the first place, it will search the mapping terms in InterPro entries which are equivalent to GO:0003700. If there exist mapping terms, it will search InterPro entries to add children and grand children terms in the query. spkw=0 will just find the mapping terms equivalent to the GO term in the SwissProt keywords. A user can specify the annotation system and the range of expansion through this EXPAND clause.

In general, an expanded SQL query has the form as follows.

```
SELECT select_list
FROM from_list
WHERE where_list
EXPAND expand_list
```

When translating an expanded SQL query into a general SQL, we consider the case which has various

kinds of annotations in a table. So the core of query translation algorithm is how to add the annotation values in the where_list. In the above example, there is no need to change select_list or from_list because every annotation is stored in a table. The output SQL query has the following form:

SELECT id
FROM protein table
WHERE organism='Human' and
(function='GO:0003700' or function='GO:0003677' or function='GO:0003676' or function= 'GO:0030528' or function='IPR001356' or function='IPR000047' or function='IPR001827' or function='IPR000747' or function = 'KW-0803')

In the WHERE clause, there are 3 more terms in Gene Ontology, 4 more terms in InterPro and 1 more term in SwissProt keyword than the original query. In this way, the user's query is expanded to include the related terms in other annotation systems.

If we have many tables that have different annotation systems, it is also simple to translate the original query. All we need to do is simply add the extra terms with OR conditions in the WHERE clause. Added terms will search each table according to which table the original term belongs to.

Query translation algorithm works as follows.

Input: expanded SQL
Output: SQL

1. Scan expand_list
2. Retrieve and save the names and ranges of annotation systems
3. Scan where_list
4. Retrieve and save the original query term with its annotation system name
5. Search the value in 4 from the mapping table
6. A mapping table returns the value which belongs to the annotation system in 2
7. Search the annotation index within the range of 2 from the values returned in 6
8. Add retrieved results in 7 to the where_list

The EXPAND clause in steps 1~2 is described in the previous example in detail. Step 3 through step 8 shows how to use mapping table and annotation index during the query expansion. In steps 3~4, we extract the information a user specified as original annotation values. These values are the starting points of query expansion. To find the mapping terms to this value we search the mapping table in step 5. Mapping table has information between terms of different annotation systems. In our case, this table has information about the mapping relation between Gene Ontology and other annotation systems. We could infer some indirect relations such as SwissProt keyword has a mapping to EC number or InterPro entry. However, only the binary relation between Gene Ontology and other systems are considered in this paper. It is not a matter of technical problem but a semantic problem because such kinds of inferred relations are not verified by the domain experts.

In step 6, a mapping table search returns the values which are counterparts of the term the user gave. The terms in different systems have m:n mappings each other. There are terms which have no mapping relation to other systems. Therefore, the number of mapping terms varies from 0 to dozens.

In step 7, we already have many expanded terms for each system from step 6. Using the information from step 2, we can choose which annotation index table to look up. We can get ancestor descendant terms in this index using the range the user specified. If a user just wants the mapping terms, we do not need to search the annotation index.

In the final step, the expanded term list is added in the WHERE clause of SQL. Although the user simply puts down the name that she knows in the expanded SQL, the result SQL includes many related terms. The query processing and answering is the portion of the DBMS.

3. Experiments

3.1. Data and Environment

In this study, we chose Gene Ontology[13], InterPro[16], Swiss-Prot keyword[6] and EC number[2] as the functional annotation schemes. These schemes has been used in GOA project[7, 8] for large scale assignment of Gene Ontology terms to the existing database. UniProt[19] database initially had no annotations described in Gene Ontology. Through the GOA project, they have annotated their data with Gene Ontology. They have made mapping relations and updated them. Their latest statistics reports(2006/12) that they have more than 11 million associations comprised of Gene Ontology, InterPro, Swiss-Prot keyword and EC number.

Gene Ontology has more than 17,000 terms and InterPro has more than 13,000 entries. Gene Ontology has a 'is-a' relation between terms. InterPro has a 'parent-child' relation of protein families. EC number has 3600 numbers and 4 levels of classification for enz

ymes. SwissProt keywords have more than 800 keywords. Each annotation scheme is getting larger and

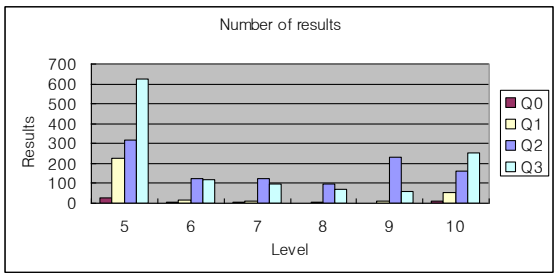


Figure 6. Number of results in query expansion

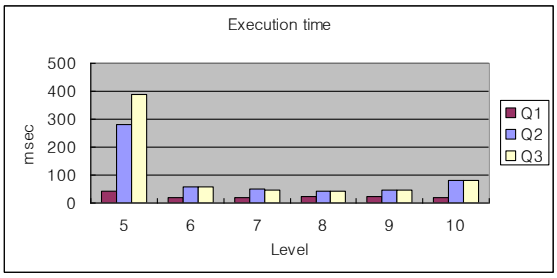


Figure 7. Execution time in query expansion

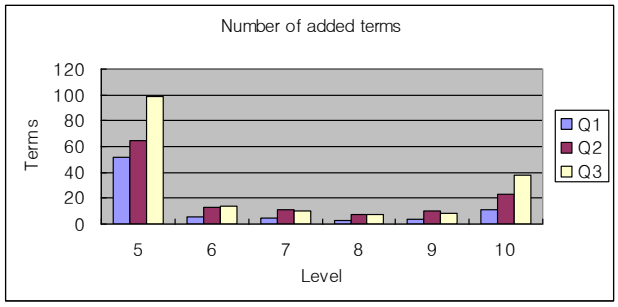


Figure 8. Number of added terms in query expansion

published periodically.

We used SwissProt Database[6] in UniProt[10] as a query expansion test database. SwissProt is one of the highest quality protein database that has been manually annotated by the domain experts. It has many kinds of annotations. We made a table that has 4 kinds of functional annotations among them and it had 2.2 million annotatons.

We used MySQL5.0 DBMS and Java language for the query translation. The experiment sever has 2.8GHz dual CPU, 4G RAM and Linux OS.

3.2. Experimental Result and Discussion

We set up several queries against human-related proteins. "Find human proteins which are doing molecular functions described in a Gene Ontology term." The query can be expanded into 3 categories.

Q0: Original query
Q1: Given a term, expand the query using just mapping
(EXPAND ipr=0 ec=0 spkw=0)
Q2: Given a term, expand the query using parent and ancestor relationship.
(EXPAND go>-3 ipr>-3 ec>-3 spkw=0)
Q3: Given a term, expand the query using child and descendant relations
(EXPAND go<+3 ipr<+3 ec<+3 spkw=0)

In Q1~Q3, a Gene Ontology term is given as an original term. SwissProt keyword has a flat structure. Therefore, we do not apply parent-child relations to SwissProt keywords in Q2 and Q3. In Q1, we add InterPro, EC and SwissProt terms which have a mapping relation to the original term. In Q2, we add GO, InterPro and EC terms which have a parent or a grand parent relation to the original term. In Q3, we add GO, InterPro and EC terms which have a child or a descendant relation to the original term.

In Figure 6~Figure8, the level shows the level of the original term in the Gene Ontology DAG structure. We randomly select a term from the Gene Ontology in the level 5~10 and tested query expansion 1000 times to obtain an average.

In Figure 6, the graph shows the number of results in the query expansion. This number means the count of proteins as a query result. When we make a query with the original term(Q0), the maximum number of result is only 27 in level 5. When we apply query expansions(Q1, Q2, Q3), the number of query result is greatly increased especially in Q2 and Q3. If we do not apply the query expansion, many useful results will be missing.

Figure 7 shows the graph of the overhead for the query translation and the query processing.

As a result, it shows that there is almost no cost for query expansion. The scale of time is msec and the result shows that the worst case(level 5) does not take a second. We can conclude that the power of DBMS treats query expansion very efficiently.

Figure 8 shows the number of added terms during the query expansion. We can see that it affects the size of result and the query execution time. If the annotation graph has a tree-like structure the added terms will be increasing mostly in Q3. Our test result shows that there is not much difference between Q2 and Q3. We found out that InterPro entries had many multiple paths and it is not a tree-like structure. It explains why there is little difference between Q2 and Q3.

In this experiment, our conclusion is that expanded query using SQL shows little overhead but gives much more answers. RDBMS has its own index on the annotation column. It ensures that the query execution time does not take long.

Each annotation system which is used in this experiment has its own purpose. Gene Ontology is made for the description of gene product's function, process, and location. InterPro entry classification is made for the protein family and domain information. Enzyme number is used for the enzyme classification. SwissProt keyword has its coverage in multiple areas. It is hard to combine queries against these complex systems without proper support. Our approach shows that the simple extension of SQL can solve this kind of problems efficiently.

4. Related Work

Database research community shows much interest in the annotation management in DBMS[5, 12, 18]. They are trying to manage annotations inside the DBMS. They assume scientific domains, especially biology data for annotation management because annotations play a key role in knowledge sharing.

In their researches, [5] addressed the problem of annotation propagation. There might be multiple annotations for the same entity. Query result might miss the needed annotations. They proposed a SQL extension to handle this situation. In [12], they introduced the annotation mechanism which could annotate the value association in the record. They tried to capture value association with annotation algebra. [18] investigated the problem of annotation insertion and deletion history. They provided the provenance tracking method for the annotation.

Our research overlaps with [5, 12] in terms of the use of RDBMS environment. However, the problem of processing relations between annotations has not been addressed. In this study, our theme focuses on the management of structural annotation relations.

Tree node labeling and DAG node labeling is extensively studied in the context of XML query processing[9, 15]. They focused on how to decide parent-child relation efficiently using the label of graph nodes. In our annotation index, we are also interested in deciding the parent-child or ancestor-descendant relation in a short time. The different situation is that we are not sensitive to the order between siblings or global order in the document. Relabeling cost is not our concern either because annotation vocabularies are not updated as often as database itself. Therefore, we judged that the dewey encoding method is suitable for our application.

In the biology community, there is a great effort in making annotation schemes and annotating data using these schemes. Gene Ontology[13] consortium consists of 14 public DB groups and provide controlled vocabularies for functional annotations. The GOA project[7, 8] is a large scale data annotation in UniProt[19] with Gene Ontology, which includes manual and electronic associations. They also provide web based application program such as AmiGO[14] and QuickGO[11]. They have a web interface for ontology navigation and keyword searching. They do not support any facilities for querying multiple hierarchical structures at a time.

Their main contribution is making the data rather than searching them. In that respect, our proposed method is a good fit for efficient search of their data.

5. Conclusion and Future Work

We have described a query expansion technique for the functional annotations in biological databases. We showed that this technique helps users get more results without much overhead of time. We investigated the relation between different annotation schemes. From this observation, we found the need to cross reference differently annotated data. Our proposed method supports this need by the query expansion against the hierarchical structure of different annotation schemes.

We proposed a simple extension of SQL. SQL and RDBMS have been friendly to the users. They are also generally used for the biological data integration environment. We can use the existing mature technologies to its full extent.

There are relation types other than the parent-child relation. This is a good point of extension to our

approach. Ranking of the results could be a good question in the query expansion. It will help users refine their query result.

6. Acknowledgements

This research was supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031

7. References

[1] Dewey Decimal Classification "http: //www. oclc. org/dewey/"

[2] A. Bairoch, "The ENZYME database in 2000", *Nucleic Acids Research*, 2000, pp. 304-305.

[3] W. C. Barker et al., "The Protein Information Resource (PIR)", *Nucleic Acids Research*, 2000, pp. 41-44.

[4] D. A. Benson et al., "GenBank", *Nucleic Acids Research*, 2006, pp. D16-20.

[5] D. Bhagwat et al., "An annotation management system for relational databases", *VLDB*, 2005, pp. 373-396.

[6] B. Boeckmann et al., "The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003", *Nucleic Acids Research*, 2003, pp. 365-370.

[7] E. Camon et al., "The Gene Ontology Annotation (GOA) Project: Implementation of GO in SWISS-PROT, TrEMBL, and InterPro", *Genome Research*, 2003, pp. 662-672.

[8] E. Camon et al., "The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology", *Nucleic Acids Research*, 2004, pp. D262-266.

[9] C. Edith et al., "Labeling dynamic XML trees", *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002, pp. 271-281

[10] Gene Ontology Annotation(GOA) Database, "http: //www.ebi.ac.uk/GOA/"

[11] QuickGO, "http://www.ebi.ac.uk/ego/"

[12] F. Geerts et al., "MONDRIAN: Annotating and querying databases through colors and blocks", *ICDE*, 2006, pp. 82

[13] C. Gene Ontology, "The Gene Ontology (GO) project in 2006", *Nucleic Acids Research*, 2006, pp. D322-326.

[14] AmiGO, "http://www.godatabase.org/cgi-bin/amigo/go.cgi"

[15] T. Igor et al., "Storing and querying ordered XML using a relational database system", *SIGMOD*, 2002, pp. 204-215

[16] N. J. Mulder et al., "InterPro, progress and status in 2005", *Nucleic Acids Research*, 2005, pp. D201-205.

[17] Mappings of External Classification Systems to GO, "http://www.geneontology.org/GO.indices.shtml"

[18] B. Peter et al., "Provenance management in curated databases", *SIGMOD*, 2006, pp. 539-550

[19] C. H. Wu et al., "The Universal Protein Resource (UniProt): an expanding universe of protein information", *Nucleic Acids Research*, 2006, pp. D187-191.