BIOINFORMATICS

Vol. 21 Suppl. 2 2005, pages ii33–ii39
doi:10.1093/bioinformatics/bti1105

Databases

A query language for biological networks

Ulf Leser

Department for Computer Science, Humboldt-Universität zu Berlin, Berlin, Germany

ABSTRACT

Motivation: Many areas of modern biology are concerned with the management, storage, visualization, comparison and analysis of networks, but no appropriate query language for such complex data structures yet exists.

Results: We have designed and implemented the pathway query language (PQL) for querying large protein interaction or pathway databases. PQL is based on a simple graph data model with extensions reflecting properties of biological objects. Queries match subgraphs in the database based on node properties and paths between nodes. The syntax is easy to learn for anybody familiar with SQL. As an important feature, a query may require a certain structure in the database to exist but return a different subgraph. We have tested PQL queries on networks of up to 16 000 nodes and found it to scale very well.

Availability: The code is available on request from the author.

Contact: leser@informatik.hu-berlin.de

1 INTRODUCTION

Many areas of modern molecular biology deal with data that are structured in the form of networks (graphs). Metabolic pathways signal transduction pathways and networks of gene regulation are naturally modeled as graphs. In these graphs, nodes typically represent biological entities such as enzymes, genes or compounds, and edges represent some form of interaction or relationship. The availability of biological network data is currently increasing rapidly, both due to enhanced experimental techniques, such as two-hybrid systems (Legrain and Selig, 2000), and due to enhanced prediction methods (Mellor *et al*., 2002; von Mering *et al*., 2003). Furthermore, large protein interaction networks are extracted from publications based either on natural-language processing (Hakenberg *et al*., 2005; Marcotte *et al*., 2001) or on statistical properties (Jenssen *et al*., 2001).

In consequence, a large number of databases have emerged that collect and manage data on the interactions of biological entities (Bader *et al*., 2003; Hermjakob *et al*., 2004; Joshi-Tope *et al*., 2003; Kanehisa *et al*., 2004; Karp *et al*., 2002; Krishnamurthy *et al*., 2003; Salwinski *et al*., 2004). At the time of writing, an up-to-date list of pathway data resources lists no fewer than 171 entries (http://cbio.mskcc.org/prl). At the same time, the networks under analysis are becoming larger and larger. Although isolated pathways rarely have more than a hundred components and can thus be examined manually, many of the network resources described above are far bigger. At the time of writing, KEGG (Kanehisa *et al*., 2004) has $>20\,000$ pathway maps, and the BIND database (Bader *et al*., 2003) stores $>147\,000$ protein interactions. The largest current network is probably the PubGene database (Jenssen *et al*., 2001), containing >6 million associations extracted from the literature.

Clearly, users can review only small fractions of such networks at a time. However, users are highly specific about the information they are interested in. Typical queries against a pathway database include

- Find all reactions involving a certain substance.
- Find all paths, i.e. chains of reactions, connecting two given substances.
- Find the shortest path between two substances that includes a third substance.
- Given a set of molecules, extract the subgraph which contains all these elements and has the least number of nodes.

Despite the necessity for complex queries, current pathway databases support only very simple queries. Mostly, searching the database is restricted to full-text search of node names; sometimes it is also possible to search all paths between two given nodes. It is not possible to formulate conditions describing complex node and path patterns or to use conditions including functional annotation of biological entities. We believe that there is a strong need for a declarative language to specify clearly and succinctly queries on biological networks. In this paper, we propose such a language: PQL, the pathway query language.

PQL is a declarative language whose syntax is similar to SQL. PQL queries operate on a simple graph data model that is a generalization of many more specific data models; therefore, we believe that PQL can be very easily adopted for a broad range of existing systems. The result of a PQL query is itself a graph, which offers possibilities for nesting and composing PQL queries. Despite its syntactic simplicity, PQL is a powerful language capable of expressing graph isomorphism problems. Implementing PQL on top of a relational database is quite straightforward, which eases its porting to different systems and databases. We describe an implementation based on the commercial Oracle V9.2 database system. However, PQL is not capable of very complex graph operations such as the computation of spanning trees. In our understanding, such analysis should be confined to specialized applications using highly tuned data structures.

We think that this proposal can have many positive influences on the field:

- Talking about a language implicitly forces one to think about the requirements that exist for querying pathways. This discussion apparently has not yet started in the community, despite many papers mentioning various types of queries (see Section 4).
- A properly defined language can be used by many in pathway databases, reducing the amount of duplicate work. Users need to learn only one language and can use this language on their favorite database.

- A query language acts as an interface between applications and databases. PQL is thus a proposal for an interface between pathway applications and pathway databases. Having a clear interface fosters the development of database-independent methods for pathway analysis. Network algorithms, user interfaces and visualization tools may use this language and thus become more easily ported to other databases.
- Having a clear semantics of queries helps to integrate data from heterogeneous repositories since the same query can be shipped to different databases.

The rest of this paper is organized as follows. Section 2 defines the data model that PQL is based upon. Section 3 defines the syntax and semantics of PQL and is the heart of this paper. Section 4 discusses a number of illustrative queries taken from publications on pathway databases and systems and describes how (if at all) they can be expressed in PQL. Section 5 sketches our prototypical implementation. Section 6 discusses related work. Section 7 concludes.

We omit formal definitions for brevity. These, together with more query examples and an extensive section on related work, can be found in Leser (2005).

2 PQL DATA MODEL

The basic PQL data model is a graph G with a set of nodes and directed edges. A node is either an interaction or a molecule. The graph need not be connected; i.e. it may fall into several unconnected subgraphs or even isolated nodes. The biological interpretation of such a graph is the following. G represents a network of molecules and their interactions. Molecules represent biochemical entities such as proteins, metabolites or genes. Interaction nodes may symbolize a chemical reaction with products and substrates, the formation of a compound from different proteins or the expression of a gene. Edges may connect (1) a molecule to an interaction, meaning that the molecule is necessary for the interaction to happen; (2) an interaction to a molecule, meaning that the molecule is a product of the interaction; (3) an interaction to an interaction. The latter situation arises when the first interaction influences the second, such as an enzymes inhibiting the catalytic effect of another enzyme. Interactions may involve any number of molecules either as input or as output, and molecules may be connected to any number of interactions.

The PQL data model is similar to the models used in pathway databases such as aMAZE (van Helden *et al*., 2000), KEGG (Kanehisa *et al*., 2004) and Reactome (Joshi-Tope *et al*., 2003). We give two examples of how biological data are represented in our model. First, consider the pyruvate metabolism pathway. Figure 1 is the original pathway from KEGG. Figure 2 shows a small fraction of the KEGG pathway in the PQL model. Enzymes and compounds are represented as molecules, and each arrow is transformed into an interaction with products and substrates.

Second, consider the leucine biosynthesis in yeast. Figure 3 is this pathway as represented in the aMAZE database. Here, names without surrounding rectangles represents metabolites, and rectangles represent reactions, with a reaction ID and the EC number of the enzyme catalyzing the reaction. In our model, enzymes and metabolites are molecules and reactions are interactions. Replacing each rectangle with a node for the reaction connected to one node for each enzyme translates the aMaze representation into the PQL data model (Fig. 4).

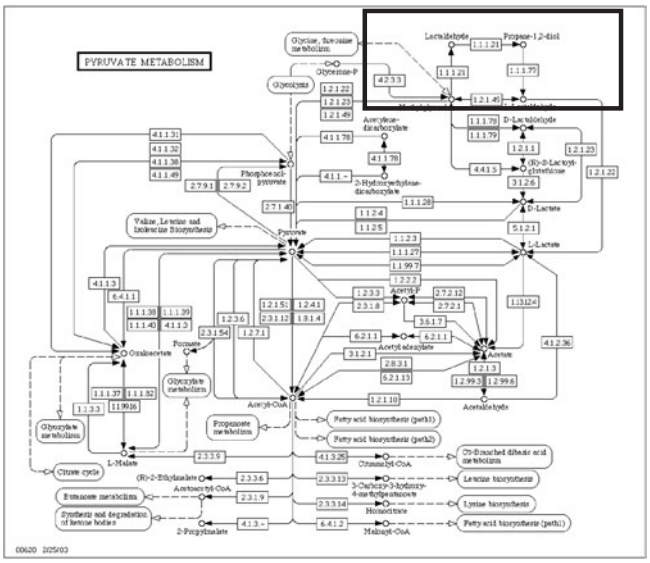

Fig. 1. Pyruvate metabolism from KEGG.

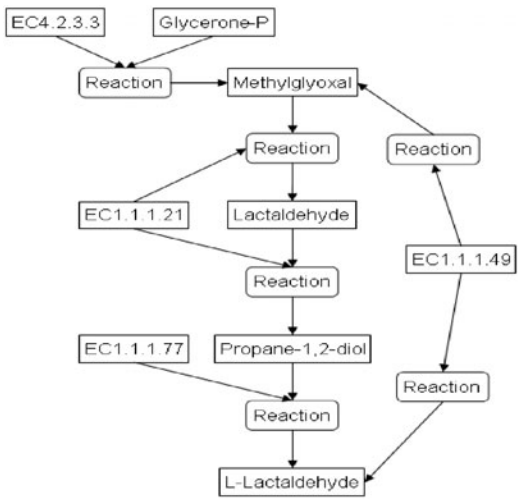

Fig. 2. Representation of the reactions in the upper right corner of the KEGG pathway using the PQL data model. Molecules are boxes, interactions are rounded boxes.

The nodes in a PQL database are, thus, biological entities or biochemical interactions. These need to be described further to allow biological rather than purely abstract queries. The PQL data model therefore defines a small set of properties of nodes and edges. Nodes, i.e. molecules and interactions, are associated with one or more types (e.g. 'gene', 'enzyme', 'inhibition', 'catalysis') and functions. The vocabulary of all types and functions is modeled as a directed acyclic graph of concepts, i.e. in the same way as the Gene Ontology. Nodes further have a unique ID and a name. Nodes can be queried on their name or their annotation using the operators `ISA` and `HASFUNC`.

Many extensions to this model are possible. For instance, interactions could be annotated with kinetic parameters, molecules could

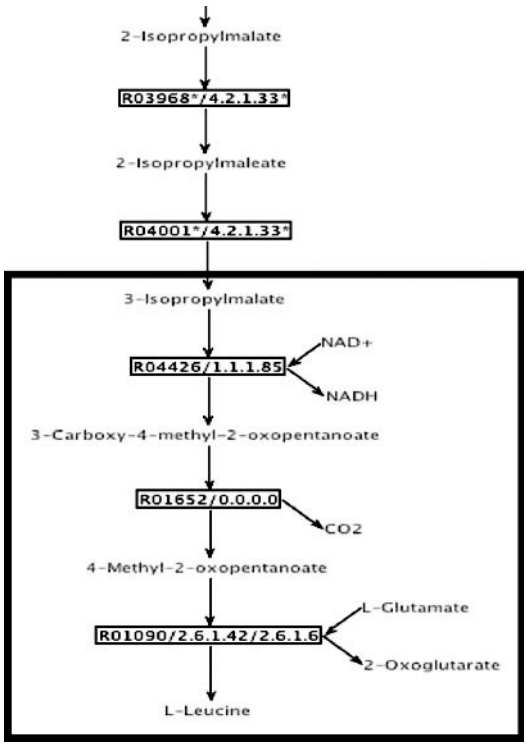

Fig. 3. Yeast leucine biosynthesis (only partly shown) taken from aMaze.

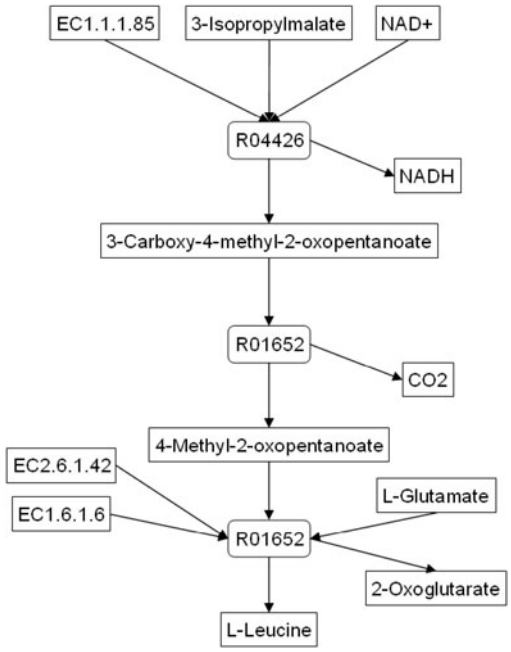

Fig. 4. Lower part of the leucine biosynthesis represented in the PQL model. Molecules are boxes, interactions are rounded boxes.

be annotated with links to external databases and interactions could be annotated with the cellular compartment they occur in. However, the current focus of PQL is on matching structures in networks. We leave those extensions to future work.

3 PATHWAY QUERY LANGUAGE

The purpose of PQL is to extract subgraphs with certain properties from a graph. The power of the language is determined by the types of subgraph properties it can express. Imagine a graph with n nodes and m edges. For instance, a simple query could extract all nodes with a certain name, which can be implemented in O(log(n)) as a binary search over a sorted list of all node labels. A more complex query could extract the shortest path from a fixed node n_1 to a fixed node n_2, which for dense graphs is possible in O(n^2) using Dijkstra's algorithm. Other complex queries could ask for the minimal spanning tree of the graph, which requires O($m * \log(n)$).

PQL allows for medium complexity queries. More precisely, it can extract subgraphs that are characterized by node and edge properties and by the existence and properties of the paths they contain. Thus, PQL goes beyond the search options available in most interaction databases but is not capable of computing the properties of the entire graph. This restriction has the advantage that PQL queries can be evaluated efficiently.

We introduce PQL in four steps. The first three assume cycle-free graphs. Section 3.1 discusses the basic syntax and semantics of PQL. We introduce expressions describing conditions on paths in Section 3.2. Section 3.3 describes how the subgraph returned by a PQL query is determined. Finally, in Section 3.4 we explain the semantics of PQL queries in graphs containing cycles.

3.1 PQL basic syntax and semantics

PQL queries resemble the syntax of SQL, though the semantics of queries is quite different. Like an SQL query, a PQL query has three parts—a SELECT clause, a FROM clause and a WHERE clause. Like SQL, where queries have relations as input and generate a relation as output, PQL queries are evaluated on a graph and result in a graph. The general syntax of PQL is as follows:

```
SELECT subgraph-specification
FROM node-variables
WHERE node-condition-set
```

Query evaluation binds node variables to nodes of the graph such that all node-conditions in the WHERE clause evaluate to TRUE. The query result is constructed from these variable bindings according to the subgraph-specification of the SELECT clause. Note that binding of node variables, which essentially means matching parts of the graph, does not directly determine the subgraph returned. For now, we assume that the SELECT clause is a '*' and returns all nodes from the subgraph determined by the FROM and WHERE clause. Consider the pathway shown in Figure 4. The following query returns a graph consisting of the two nodes '`3-Isopropylmalate`' and '`EC1.1.1.85`':

```
SELECT *
FROM A, B
WHERE A.name='3-Isopropylmalate' AND
      B.name='EC1.1.1.85'
```

The semantics of a PQL query is intuitively defined as follows (for details see Leser (2005)). Query evaluation considers each node-variable in the FROM clause. For each of these variables, all possible assignments of a variable to nodes of the graph are determined for which the conditions of the WHERE clause evaluate to TRUE. Node variables are equally assigned to molecules and interactions. Once all

bindings for each node variable have been computed, the Cartesian product of these sets is built. All instances are removed for which the entire WHERE clause evaluates to FALSE, thus enforcing conditions that include more than one node variable (such as a condition `A.name=B.name`). Next, all distinct assignments (node variables to database nodes) from the remaining elements of the Cartesian product are combined to form the so-called match graph. Thus, the set of database nodes in the match graph is always a subset of the nodes of the underlying database. The match graph does not contain any edges (see Section 3.3). Consider the following query and Figure 4:

```
SELECT *
FROM A, B
WHERE A.name='R04426' AND
      (B.name='NAD+' OR B.name='CO2')
```

It returns the nodes '`R04426`', '`NAD+`', and '`CO2`', since only the bindings (A→`'R04426'`, B→`'NAD+'`) and (A→`'R04426'`, B→`'CO2'`) fulfill the conditions in the WHERE clause.

3.2 Path expressions

We enrich the language with possibilities to match subgraphs in PQL by using path expressions. A path is a set of nodes such that between each pair of consecutive nodes an edge exists. Paths must be acyclic for now. The length of a path is the number of edges it contains. We introduce path expression by an example using the network of Figure 2:

```
SELECT  *
FROM  B, C, D
WHERE D.name='L-Lactaldehyde' AND
      B ISA 'Enzyme' AND B[−2]D AND
      B[−*]C[−*]D AND
      C.name='Lactaldehyde'
```

A condition of the form '$X[-n]Y$', with X and Y being node variables, means that there must exist a path between the bindings of X and Y in the underlying database. This path can have arbitrary length if n is '*' and must be of length n if n is a number. Hence, the above query matches a subgraph of three nodes B, C and D such that D has name `L-Lactaldehyde`, B is an enzyme, there exists a path of length 2 between node B and D and there exists a path of arbitrary length from B through C to D, where C must be `Lactaldehyde`. Thus, the query finds all instances of the graph sketched in Figure 5.

Path expressions are existential conditions. They require that certain paths exist between the nodes assigned to node variables. They do not require that all such paths adhere to the stated conditions. Each path expression considers a path between only two nodes, but since a PQL query may contain arbitrary many path expressions that may share node variables, many types of subgraphs can be expressed. However, path expressions are not capable of describing all possible subgraphs. For instance, we cannot state a condition on the shortest path between two nodes, and we cannot state that different paths have the same length.

3.3 The SELECT clause—specifying the output

So far we have assumed that the SELECT clause is simply a '*', thus returning the set of all bindings of node variables. This is not sufficient, since we are interested in finding subgraphs and not just

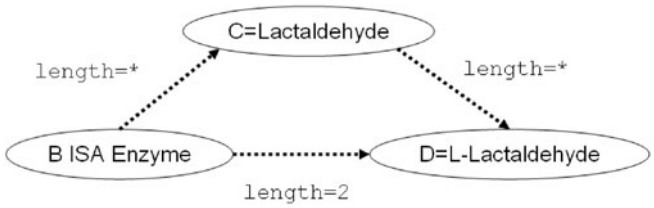

Fig. 5. Graphical representation of the query given in the text. Dashed lines represent path expressions.

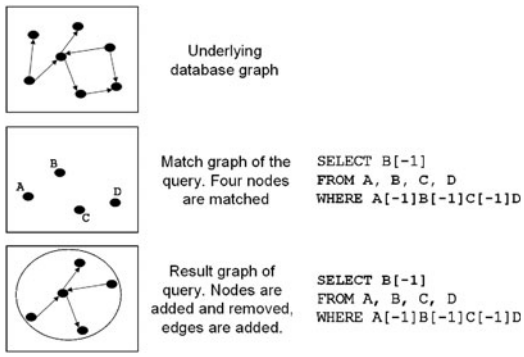

Fig. 6. Steps during the evaluation of a PQL query.

isolated nodes. These subgraphs may stretch further than the nodes in the match graph, for instance including the vicinity of a matched node up to a certain range. Furthermore, a query often only requires some nodes to exist but does require the concrete bindings to be retrieved.

To fulfill these requirements, PQL defines a variety of different expressions in the SELECT clause, called select functions. Select functions determine how the result graph, i.e. the query result, is derived from the match graph. The match graph depends on the underlying database, the node variables and the conditions in the WHERE clause; select functions determine the result graph by adding nodes and edges to the match graph or by removing uninteresting bindings (Fig. 6).

PQL defines the following select functions:

- Addition of all ('*') variable bindings or only bindings of specific variables.
- Addition of paths between bindings of any two node variables, including fixed-length paths, shortest path and longest path.
- Addition of the vicinity around the bindings of a variable with given radius.

For instance, the following query returns the vicinity of radius 2 (all nodes that can be reached from a node or that reach that node by a path of length at most 2) around all bindings of A and all paths between bindings of A and bindings of B:

```
SELECT A[−2], A[−*]B
FROM A, B
WHERE A[−<5]B AND A ISA 'enzyme' AND
      B.name='Propane-1,2-diol
```

Note that path expressions in the WHERE clause are existentially quantified, whereas path expressions in the SELECT clause require that all existing paths are included in the result graph.

3.4 Networks with cycles

So far we have assumed cycle-free graphs and hence cycle-free paths. However, biological networks often contain cycles, such as feedback and feedforward loops (Yeger-Lotem *et al*., 2004).[1] In fact, any reversible reaction introduces a cycle.

PQL allows cycles in graphs, but queries are evaluated only on cycle-free paths. There are three reasons for this. First, if paths may contain cycles, the notion of 'all paths' between two nodes becomes undefined, as there may be infinitely many. Second, in any network with too large a number of cycles, many PQL queries containing select functions with paths would always return an unreasonably large fraction of the network. With an increasing number of nodes involved in at least one cycle, chances rise that a path between any pair of nodes touches a node that is part of a cycle. In this case, the cycle and all its constituents immediately become part of the result, although it is probably irrelevant for the query at hand. Third, cycle-free paths are more efficient to deal with during the computation of a query result.

However, demanding paths to be cycle-free does not constrain the power of PQL (Leser, 2005). The only case in which this restriction affects PQL queries is queries for loops, i.e. expressions of the form 'A[−*]A' in the SELECT or WHERE clause. Such expressions always return an empty answer. However, it is easy to reformulate them into the semantically equivalent form 'A[−*]B[−*]A', which is perfectly computable in PQL.

4 EXEMPLARY BIOLOGICAL QUERIES

To evaluate whether or not PQL is sufficiently expressive for biological questions, we have analyzed a number of publications on metabolic databases with respect to the types of queries they list as being crucial. In this section, we describe queries taken from those publications and give their equivalent in PQL, if it exists. This should help the reader to better understand the strengths and limitations of PQL and serves as a guideline for future extensions. Queries are cited from van Helden *et al*. (2000), Karp *et al*. (2002) and Schaefer (2004).

'Find all genes whose expression is directly or indirectly affected by a given compound.'

Ignoring the difficulty in defined the intented meaning of the word 'affected', the question can be answered as follows (assume the given compound is `L-Glutamate`):

```
SELECT B
FROM   A, B
WHERE A.name='L-Glutamate' AND
      A[−*]B and B ISA 'gene'
```

'In the complete set of metabolic reactions, find all feedback loops including a given compound.'

This is answered by the following PQL query (assume the given compound is `Methionine`):

[1]Note that all motifs described in this reference can be found in a pathway database using a single PQL query.

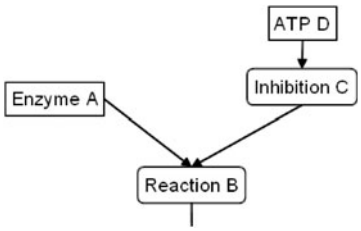

Fig. 7. Enzymes whose catalyzed reaction is inhibited by ATP.

```
SELECT A[−*]B[−*]A
FROM   A, B
WHERE  B.name='Methionine' AND
       A[−*]B[−*]A
```

'The user specifies a set of nodes . . . and prompts the system to extract the . . . sub-graphs that interconnect each pair of seed nodes via the smallest number of . . . links.'

Assuming a direct graph as the underlying data structure, PQL is capable of such queries if the number of seed nodes is fixed at query time. Note that the query does not ask for the minimal spanning tree of the set of nodes, which cannot be computed by a PQL query (though the use of PQL can considerably easy the computation). Assume we have four seed nodes A, B, C and D. Then, the question would be answered by the following query ('A[−s]B' computes the shortest path between nodes A and B):

```
SELECT  A[−s]B, A[−s]C, A[−s]D, B[−s]C,
       B[−s]D, C[−s]D
FROM   A, B, C, D
WHERE  A[−*]B[−*]C[−*]D
```

'Find all processes that lead from node A to node B in less than Max steps, and more than Min steps.'

This query cannot readily be expressed in PQL because it is not possible to specify in one query both maximum (N) and minimum (M) lengths of paths between two nodes. The following query does not compute the desired result:

```
SELECT A[−*]B
FROM A, B
WHERE A[−>M]B AND A[−<N]B
```

Instead, the query computes all nodes A and B for which there exists at least one path between them longer than M and at least one path shorter than N. Having multiple conditions on paths is part of future work on PQL.

'Find all enzymes for which ATP is an inhibitor.'

The following PQL query finds the desired enzymes[2] (see Fig. 7 for illustration):

```
SELECT   A
FROM   A, B, C, D
WHERE  A ISA 'enzyme' AND
       D.name='ATP' AND
       A[−1]B AND D[−1]C[−1]B AND
```

[2]Since enzymes are usually used synonymously for the reactions they catalyze, the original formulation is slightly ambiguous. We assume that we are looking for reactions catalyzed by an enzyme and inhibited by ATP.

```
B ISA 'reaction' AND
C ISA 'inhibition'
```

'Retrieval of all interactions that involve any of a set of molecular species as immediate participant.'

Such a query can easily be expressed in PQL using the vicinity operation with a radius of 2 (radius 1 retrieves only interactions not the interacting molecules).

'Retrieval of a connected graph that includes a set of specified interactions.'

Such queries cannot be computed using PQL, because the order of elements in a path expression is fixed with the query. Again, having more elaborate path conditions is part of future work on PQL.

5 IMPLEMENTATION OF PQL

We have implemented PQL on top of Oracle Server V9.2.[3] The implementation consists of three parts: (1) a model for storing the PQL data model, (2) procedures performing certain precomputations on the data to improve the performance of PQL queries and (3) a compiler for PQL queries.

A PQL query is compiled into a PL/SQL stored procedure. This procedure finds all possible bindings given the conditions of the query and computes the result graph using the select functions in the SELECT clause. Since the result of a PQL query is a graph, it cannot be simply returned as the result of the procedure. We could encode the result into a single table and use a table function, but table functions are currently supported only by commercial database systems, thus impeding the portability of PQL. Therefore, the result graph is stored in two tables. If PQL queries were used in client programs, we envisage that these tables would be read and turned into a graph representation by the middleware.

The relational data model used in our PQL implementation is very simple. Nodes are stored in the `Node` table, edges in the `Edge` table. An edge is represented by the foreign keys of the two nodes it connects. Annotations are stored in the `Function` and `Type` tables. Since both the type and the function hierarchies are DAGs, the structure of the concepts is encoded in two separate tables.

Evaluating a PQL query has two phases: computing the match graph and computing the result graph. Computing the match graph conceptually requires that assignments of node variables to nodes are enumerated and tested. We push as much of this work as possible into the database, i.e. express it using SQL operations. Conditions on node names and node IDs are efficiently supported by any relational database, e.g. by using indices. Therefore, the efficiency of computing a match graph is dominated by two factors: (1) the cost of evaluating path expressions and (2) the cost of computing the ISA and HASFUNC operations, i.e. traversing a DAG. To speed up these operations, the PQL compiler assumes some helper tables to be filled before executing the query. With their help, path expressions and DAG operations can be answered in approximately logarithmic time.

The helper tables store all cycle-free paths that exist in the graphs. Not only the existence of a path is stored—this would speed up only path expressions of the form '`A[-*]B`'—but also its length and all nodes it contains. Using this data, path expressions of the form '`A[->3]B`' and '`A[-4]B`' can be evaluated using a single lookup. Further, this information helps to compute the select functions for shortest path and vicinity. Computing all cycle-free paths is achieved by iteratively computing paths of increasing length. This is a costly operation, but computing all paths is necessary only once, and current database technology makes it feasible even for large graphs. For instance, our current implementation computes all ~208 000 paths between the ~16 000 nodes and ~23 000 edges of the Gene Ontology (GeneOntology Consortium, 2001) in ~5 min on a laptop. However, we cannot possibly compute all paths in a network of all interactions between genes and proteins extracted from Medline, which is one of our envisaged applications. For such graphs, future versions of PQL must use recursive or hybrid methods for path enumerations.

[3] At the time of writing, the implementation was missing the following features. (1) Only AND may be used in the WHERE clause. (2) ISA and HASFUNC are not implemented. (3) Complex path expressions are not recognized by the parser.

6 RELATED WORK

To the best of our knowledge, this is the first suggestion for a query language directly targeted at biological networks. None of the pathway and interaction databases we have examined so far has a query language. BIND (Bader *et al*., 2003), DIP (Salwinski *et al*., 2004) and IntAct (Hermjakob *et al*., 2004) have only keyword search and no notion of paths or subgraphs. Reactome (Joshi-Tope *et al*., 2003) and Kinase Pathway Database (Koike *et al*., 2003) have an additional module for finding paths between two given nodes. In both cases it is not clear whether or not paths must lie within one pathway or may cross pathways. Sohler *et al*. (2004) describe the TopNet system for pathway visualization and analysis. The authors mention an XML-based pathway query language but give too few details for a comparison.

The pathway database system (Krishnamurthy *et al*., 2003) is a complete system for modeling, visualizing and editing pathways. It is based on hyper-graphs, i.e. edges are reactions connecting sets of nodes (molecules). The data model is very rich, including pathways, species, reaction properties etc. The system supports canned queries for simple selection based on node or edge properties, simple paths (only two ends) and neighborhood queries. Queries are always evaluated in memory; i.e. the entire database is loaded into main memory at startup. A similar system is PathDB, built around the ISYS system for application integration in the life sciences (Siepel *et al*., 2001). Queries are parameterized functions that do not go beyond the capabilities of the previously described system. EcoCyc (Karp *et al*., 2002) is a pathway databases based on a frame-based knowledge representation and reasoning systems. Arbitrary queries may be programmed in LISP, PERL or Java, but no query language is provided.

Graph databases have been an area of research in the database community for some years. Güting (1994) proposed a data model and query language for graph data in spatial applications. The data model is an extension of an object-oriented model with special class types for links and paths. However, the query language was defined only in fragments and no implementation is available. GraphLog (Consens and Mendelzon, 1990) is a visual graph query language whose expressions are translated into rules of a DATALOG program. GraphLog is considerably more expressive than PQL. However, GraphLog returns only tuples of matched nodes, not graphs, and cannot be implemented easily on top of relational databases. Also, some commercial databases support network-type data. For instance,

Oracle has recently released an extender for Network Data Management as part of the spatial extender in version 10g (Oracle Corp., 2003). With this extender, networks of various types can be generated and managed. Access is either through a PL/SQL or a Java API. However, there is no notion of network queries. In Das *et al*. (2004), Oracle reports on an experimental system for querying ontologies (read: DAGs). Similar to PQL, the implementation is based on the pre-computed transitive closure of the graph, but it is unclear whether Oracle's approach can be extended to full graphs. Finally, there has been extensive work in the area of query languages for semi-structured data and XML [see e.g. Buneman (1997); the most prominent language today is XQuery], but these languages operate only on trees.

Our current focus for PQL is on the definition of an appropriate, expressive and simple query language for querying biological networks. Not much effort has yet been invested into an efficient implementation. For instance, faster algorithms are known for computing the transitive closure of a graph (Agrawal and Jagadish, 1987). However, one must be careful whether or not these algorithms can treat cyclic paths and really compute all paths, not only all connections.

7 CONCLUSIONS

We have described PQL, a language for querying biological networks. The power of PQL comes from mainly three sources: the adoption of a biological data model, the ability to define conditions on paths in a query and a variety of functions to give a detailed specification of the desired result constructed from the matching subgraphs.

PQL's syntax is derived from SQL because we believe that many programmers are familiar with this language. However, PQL has a different semantics and allows queries to be formulated in a concise way that would require extensive programming using only SQL. A drawback of defining a new language is that mixing of SQL and PQL in a single query is not possible.

Our analysis of requirements for such a language, based on a literature review and presented in Section 4, has shown that there is a strong need to define more clearly the exact nature of a biological question against a biological network than is possible with current languages and that PQL needs further extensions to fulfill all such needs. These extensions include named paths, nesting of PQL queries and the explicit modeling of pathways, organisms and tissues. Also, many interesting queries require that certain conditions hold on all elements of a path, such as a query asking for all paths between two enzymes that do not contain a certain third enzyme. To enable such conditions, PQL must be extended to include path bindings in addition to the currently available node bindings. Another line of necessary improvement is an efficient and complete implementation of PQL. The current implementation should be considered a prototype to support language definition. Specifically, it must be equipped with a graphical interface for query specification and result visualization.

We hope that PQL is the start of a common effort to develop query languages for biological network data. Repeating our plea from the introduction, we think that such a discussion is badly needed to support the exchange of network data, to allow for a concise specification of data to be extracted from a pathway database and to reduce duplication of work in different systems. We believe that a declarative query language should be a natural component of any pathway database.

ACKNOWLEDGEMENTS

This work is supported by BMBF grant no. 0312705B (Berlin Center for Genome-Based Bioinformatics). We thank Silke Trissl and Lukas Faulstich for many fruitful discussions.

Conflict of Interest: none declared.

REFERENCES

Agrawal,R. and Jagadish,H.V. (1987) Direct algorithms for computing the transitive closure of database relations. In *Proceedings of the 13th Conference on Very Large Databases*, Brighton, UK, pp. 255–266.

Bader,G.D. *et al*. (2003) BIND: the biomolecular interaction network database. *Nucleic Acids Res.*, **31**, 248–250.

Buneman,P. (1997) Semistructured Data. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, Tuscon, AZ, pp. 117–121.

Consens,M.P. and Mendelzon,A.O. (1990) GraphLog: a visual formalism for real life recursion. In *Proceedings of the ACM Symposium on Principles of Database Systems*, Nashville, TN, pp. 404–416.

Das,S., Chong,E., Eadon,G. and Srinivasan,J. (2004) Supporting ontology-based semantic matching in RDBMS. In *Proceedings of the 30th Conference on Very Large Databases (VLDB04)*, Toronto, Canada, pp. 1054–1065.

GeneOntology Consortium. (2001). Creating the gene ontology resource: design and implementation. *Genome Res.*, **11**, 1425–1433.

Güting,R.H. (1994) GraphDB: modeling and querying graphs in databases. In *Proceedings of the 20th Conference on Very Large Databases*, Santiago de Chile, pp. 297–308.

Hakenberg,J., Plake,C. and Leser,U. (2005) Optimizing syntax patterns for discovering protein–protein interactions. In *Proceedings of the ACM Symposium on Applied Computing*, Santa Fe, NM, Bioinformatics Track 1, pp. 195–201.

Hermjakob,H. *et al*. (2004) IntAct: an open source molecular interaction database. *Nucleic Acids Res.*, **32** (Database issue), D452–D455.

Jenssen,T.K. *et al*. (2001) A literature network of human genes for high-throughput analysis of gene expression. *Nat. Genet.* **28**, 21–28.

Joshi-Tope,G. *et al*. (2003) The Genome Knowledgebase: a resource for biologists and bioinformaticists. *Cold Spring Harb. Symp. Quant. Biol.*, **68**, 237–243.

Kanehisa,M. *et al*. (2004) The KEGG resource for deciphering the genome. *Nucleic Acids Res.*, **32** (Database issue), D277–D280.

Karp,P.D. *et al*. (2002) The EcoCyc database. *Nucleic Acids Res.*, **30**, 56–58.

Koike,A. *et al*. (2003) Kinase pathway database: an integrated protein-kinase and NLP-based protein-interaction resource. *Genome Res.*, **13**, 1231–1243.

Krishnamurthy,L. *et al*. (2003) Pathways database system: an integrated system for biological pathways. *Bioinformatics*, **19**, 930–937.

Legrain,P. and Selig,L. (2000) Genome-wide protein interaction maps using two-hybrid systems. *FEBS Lett.*, **480**, 32–36.

Leser,U. (2005). A query language for biological networks. Technischer Report 187, Department for Computer Science, Humboldt-Universität Berlin.

Marcotte,E.M. *et al*. (2001) Mining literature for protein–protein interactions. *Bioinformatics*, **17**, 359–363.

Mellor,J.C. *et al*. (2002) Predictome: a database of putative functional links between proteins. *Nucleic Acids Res.*, **30**, 306–309.

Oracle Corp. (2003) Oracle Database 10*g* Network Data Model.

Salwinski,L. *et al*. (2004) The database of interacting proteins: 2004 update. *Nucleic Acids Res.*, **32** (Database issue) D449–D451.

Schaefer,C.F. (2004) Pathway databases. *Ann. N. Y. Acad. Sci.*, **1020**, 77–91.

Siepel,A. *et al*. (2001) ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources. *Bioinformatics*, **17**, 83–94.

Sohler,F. *et al*. (2004) New methods for joint analysis of biological networks and expression data. *Bioinformatics*, **20**, 1517–1521.

van Helden,J. *et al*. (2000) Representing and analysing molecular and cellular function using the computer. *Biol. Chem.*, **381**, 921–935.

von Mering,C. *et al*. (2003) STRING: a database of predicted functional associations between proteins. *Nucleic Acids Res.*, **31**, 258–261.

Yeger-Lotem,E. *et al*. (2004) Network motifs in integrated cellular networks of transcription-regulation and protein–protein interaction. *Proc. Natl Acad. Sci. USA*, **101**, 5934–5939.