bcnQL: A Query Language for Biochemical Network

Hong Yang, Rajshekhar Sunderraman, Hao Tian

Computer Science Department
Georgia State University
Outline

• Introduction
  • Graph Data Model
  • Query Language for Biochemical Network (bcnQL)
• Examples
• Implementation
• Conclusion
Introduction

- Biochemical network is considered as an alternative for investigating genomic information.
- Biochemical network has been used to predict enzyme functions for newly sequenced genes.
- Biochemical Network which integrates metabolic, regulatory and signal transduction information.
  - Nodes typically represent biochemical entities such as enzymes, genes, or compounds, and edges represent some form of chemical interaction or relationship.
Introduction

Pyruvate metabolism from KEGG
Introduction

Variety of databases store the biochemical network data:
- **KEGG**: a genome database which contains information on proteins and enzymes whose gene sequence has been determined.
- **CSNdb**: focuses on signaling pathways of human cells.
- **EcoCyc**: an organism-specific pathway/genome database that describes the metabolic and signal-transduction pathways of Escherichia coli, its enzymes, its transport proteins and its mechanisms of transcriptional control of gene expression.
- **aMAZE**: is a database for the representation of information on networks of cellular processes: genetic regulation, biochemical pathways, signal transductions.
- **Swiss-Prot**: a manually curated biological database of protein sequences.
Introduction

• Typical queries about biochemical networks:
  – Find all reactions involving a certain substance.
  – Find all processes that lead from node A to node B in less than X steps and more than Y steps.
  – Find the shortest path between two substances that includes a third substance.
  – Given a set of molecules, extract the sub-graph which contains all these molecules and has the least number of nodes.
Introduction

- The tools available for querying pathway databases are restricted to the full-text search for node names and paths search for given two given nodes:
  - PathCase provides pre-defined queries executed through dialog box.
  - aMAZE possesses a metabolic pathfinding tool that is able to find the shortest path between two nodes.
  - BioCyc encompasses a query page that allows users to query pathway database by specifying conditions on the entity’s name.
Introduction

• The available query tools cannot have conditions specified on paths or graphs.
• The programming effort to process queries for one pathway database system cannot be shared for another.
Introduction

• Graph data model
  – Be able to reflect the features of biochemical pathways.
  – Possess the general features of variously specific data models.

• A declarative language which requires following features is strongly needed on biochemical networks:
  – Able to construct queries existing in the current biochemical network database system.
  – Be able to query biological entities, interactions, processes and pathways with conditions on them.
  – Could be used for constructing new graphs.
Outline

• Introduction
• **Graph Data Model**
  • Query Language for Biochemical Network (bcnQL)
  • Examples
  • Implementation
• Conclusion
Object-Oriented Graph Data Model

- **Basic** classes in the graph data model to represent graph structure:
  - **Node** class: represents nodes in graph;
  - **Edge** class: represents edges in graph;
  - **Path** class: represents paths in graph;
  - **Hyper-path** class: represents sub-graphs of larger graph;
  - **Graph** class: represents the graphs which may present in the database or new graphs composed by hyper-paths;

- Application-specific **extended** classes in the graph data model are used to reflect the features of different type networks:
  - The nodes in the graph may be different data types, and each data type could be represented by an **extended subclass of Node** class.
  - The edges in the graph may be different data types as well, and each data type could be represented by an **extended subclass of Edge** class.
  - **Complementary** classes can be designed to help users to define objects referred by the Node subclasses or Edge subclasses.
Object-Oriented Graph Data Model (cont.)

Basic class schema:

```java
class Node:
    (name: string);

class Edge:
    (from: Node;
     to: Node);

class Path:
    (start: Node;
     end: Node;
     nodes: sequence<Node>;
     edges: sequence<Edge>;
     numberofnodes: integer;
     length: integer);

class Hyper-path:
    (nodes: set<Node>;
     edges: set<Edge>;
     paths: set<Path>;
     numberofnodes: integer;
     numberofedges: integer);

class Graph:
    (nodes: set<Node>;
     edges: set<Edge>);
```
Extended Subclasses for Biochemical Network

The nodes in the graph may be biochemical entities or interactions, and using arcs to connect interactions to their inputs and outputs entities.

Node class has extended subclasses, such as:
- BiochemicalEntity_Gene
- BiochemicalEntity_RNA
- BiochemicalEntity_Protein
- BiochemicalEntity_Compound
- BiochemicalInteraction

Representation of the reactions in the upper right corner of the KEGG pathway using graph data model (Entities are boxes, interactions are oval boxes)
Biochemical network query language (bcnQL)

- bcnQL queries consist of five elements:
  - **Output** clause: specifies query outputs
  - **Define** clause: specifies variables denoting collection of node objects, edge objects, path objects, hyper-path objects and graph objects.
  - **Where** clause: specifies query condition
  - **Group by**: allows the objects to be grouped based on certain attribute
  - **Order by**: allows the objects to be ordered by certain attribute
- The simple bcnQL can be combined through the set operators like:
  - **UNION**
  - **INTERSECT**
  - **MINUS**
Outline

• Introduction
• Proposed Data Model
• Proposed Graph Query Language
• Examples
• Implementation
• Conclusion
Biochemical network query language (bcnQL)

- General Form of Query:

  Output: <Output-list>
  Define: <Node-formulas> | <Edge-formulas> | <Path-formulas> | <Hyper-path-formulas> | <Graph-formulas>
  Where: <selection-predicate>
  Group By: <grouping-objects-attributes>
  Order By: <ordering-attributes>

- The define clause allows the definition of one or more variables using node, edge, path, and graph formulas.
- Most of the “expressibility” of the language comes from the define clause.
- The output clause may output objects, their properties (dot-path expressions), or aggregates.
Formulas in Define Clause

**Node/Edge formula:**

- Nodes: \( n[(subclass) \ <predicate>] \)
- Edges: \( e[(subclass) \ <predicate>] \)

**where**

- \( n,e: \) are variables; the bracketed expression is optional.
- \( (subclass) \) is optional and provides the ability to work with application specific classes.
- \( <predicate> \) is a selection condition on the attributes of the Node/Edge objects.

If \( (subclass) \) presents, objects of that type satisfying the predicate are returned, otherwise, Node or Edge objects satisfying the predicate are returned.
Formulas in Define Clause

Simple Path Formula:

Paths

\[ n_1[(\text{subclass}) \ <\text{predicate}>] \]
\[ e_1[(\text{subclass}) \ <\text{predicate}>] \]
\[ n_2[(\text{subclass}) \ <\text{predicate}>] \]

where

\( n_1, e_1, n_2 \) are variables
Formulas in Define Clause

Simple Path Formula With Regular Expression:

Paths
n1[(subclass) <predicate>] _* n2[(subclass) <predicate>]

Paths
n1[(subclass) <predicate>] _N n2[(subclass) <predicate>]

Paths
n1[(subclass) <predicate>] _S n2[(subclass) <predicate>]

where
n1, n2: variables
*: one or more edges; N: N edges, S: shortest path
Formulas in Define Clause

General Path Formula:

Concatenate one or more simple path formulas by introducing edges between them:

\[ P_1 \; e_1[(\text{subclass}) \; <\text{predicate}>] \]
\[ P_2 \; e_2[(\text{subclass}) \; <\text{predicate}>] \]
\[ \ldots \]
\[ \ldots \]
\[ P_{n-1} \; e_{n-1}[(\text{subclass}) \; <\text{predicate}>] \]
\[ P_n \]
Formulas in Define Clause

Hyper-path formula:

Hyper-paths
<path-formula> p₁, …, <path-formula> pₙ,
[graph-predicate]

A hyper-path object is constructed from a collection of paths based on the graph-predicate that specifies connecting points in the paths.

One path from each pᵢ is chosen and connected using the graph-predicate.
Formulas in Define Clause

Graph formula:

Union (<hyper-path-formula> as h1, <hyper-path-formula> as h2)

Intersect (<hyper-path-formula> as h1, <hyper-path-formula> as h2)
The informal semantics of a bcnQL query is described as follows:

- Each formula in the “define clause” produces a collection of objects. A **Cartesian product** of the various collections of objects is obtained.

- The tuple of objects that **satisfy** the where-predicate are **retained**.

- If the “group by” clause is present, the results are **split into partitions** based on partition attribute(s).

- If the “order by” clause is present, the results are **sorted** by the attribute defined in the “order by”.

- Output clause expressions are **evaluated** as query results.
Outline

• Introduction
• Proposed Data Model
• Proposed Graph Query Language
• **Examples**
• Implementation
• Conclusion
Query Example 1

- Get all interactions involving a certain substance (says, gamma-glutamyl kinase).

Output `p1.nodes[1]`

Define paths

```
n1[(BiochemicalInteractions)]_n2[(BiochemicalEntities _protein)]n1.name = 'Gamma-glutamyl kinase' as p1
```
Query Example 2

Find all processes that lead from node A to node B in less than 5 steps, and more than 3 steps in the Biochemical Network.

Output p1
Define
Paths n1[n1.name='A'] _* n2[n2.name='B'] as p1
Where p1.length >= 3 and p1.length <=5;
Query Example 3

User specifies a set of nodes in the network – the ‘seed’ nodes – and prompts the system to extract the portions of the network or sub-graphs that interconnect each pair of seed nodes via the smallest number of individual links (Notes: here we use three nodes (A, B, C) as the ‘seed’ nodes):

Output h1
Define hyper-paths

\[
n1[n1\text{.name}='A'] \_s n2[n2\text{.name}='B'] \text{ as } p1,
n1[n1\text{.name}='A'] \_s n2[n2\text{.name}='C'] \text{ as } p2,
n1[n1\text{.name}='B'] \_s n2[n2\text{.name}='C'] \text{ as } p3
\]
\[
[p1.n1=p2.n1 \text{ and } p1.n2 = p3.n1 \text{ and } p2.n2 = p3.n2] \text{ as } h1
\]
Implementation

- Db4o: Object-oriented database
- The query is sent to the bcnQL parser (using JFlex/JCUP) in which the syntax and semantic if query are verified.
- The formulas in the define clause are evaluated by the bcnQL evaluator (each type of formula is evaluated by the corresponding algorithm)
- The outputs of the evaluator are sent to OODB (db4o).

Architecture of the query system
Conclusion

• Since the inept of the available tools for querying pathway database, there is a need for a query language to specify queries needed in biochemical networks.

• A object – oriented graph data model:
  – Could represent biochemical network integrating metabolic, regulatory and signal transduction networks.
  – Could represent the node, edge, path, and graph objects and their properties.

• Biochemical network query language (bcnQL)
  – Empowers users to query entities, interactions, processes and pathways with conditions specified on them.
  – Could be used for composing interested graph with defined graph operations
Questions
Thanks