Periscope/GQ: A Graph Querying Toolkit

Yuanyuan Tian * Jignesh M. Patel * Viji Nair + Sebastian Martini + Matthias Kretzler +
* Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI
{ytian, jignesh}@eecs.umich.edu
+ Department of Internal Medicine, University of Michigan, Ann Arbor, MI
{vijin, sebmarti, kretzler}@med.umich.edu

ABSTRACT

Real life data can often be modeled as graphs, in which nodes represent objects and edges between nodes indicate their relationships. Large graph datasets are common in many emerging applications. Examples span from social networks, biological networks to computer networks. To fully exploit the wealth of information encoded in graphs, systems for managing and analyzing graph data are critical. To address this need, we have designed and developed a graph querying toolkit, called Periscope/GQ. This toolkit is built on top of a traditional RDBMS. It provides a uniform schema for storing graphs in the database and supports various graph query operations, especially sophisticated operations, such as approximate graph matching, large graph alignment and graph summarization. Users can easily combine several operations to perform complex analysis on graphs. In addition, Periscope/GQ employs several novel indexing techniques to speed up query execution. This demonstration will highlight the use of Periscope/GQ in two application domains: life science and social networking.

1. INTRODUCTION

Graphs provide a natural way to model data in a variety of applications. Nodes in graphs usually represent real world objects and edges indicate relationships between objects. Examples of data modeled as graphs include social networks, biological networks, and computer networks. Many graph databases are large and growing rapidly in size. For example, the number of pathways (a pathway is a graph of cellular entities and their interactions) in the well-known KEGG pathway database [6] has increased from 2,706 in 1999 to 29,921 in 2005, then to 66,407 in 2007. The social networking site Facebook contains a large network of registered users and their friendships. The number of Facebook users has grown from less than 5 million in September 2005 to close to 10 million in September 2006, then to 50 million in September 2007. There is a critical need for efficient and effective graph querying systems to query and mine these growing graph databases. This need is especially critical for life science applications [1, 5].

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

Previous graph querying systems [2, 8] have largely focused on relatively *simple* graph operations, such as retrieving nodes, edges and paths. Systems for querying RDF data, such as Jena2 [3], also supports graph pattern queries. However, none of these systems support sophisticated query operations like approximate graph matching, large graph alignment, or graph summarization (see Table 1 for the descriptions of these operations). On the other hand, tools for individual query operations, such as GraphGrep [7], GIndex [12] and Grafil [13], have been developed. These tools are useful, but the power of individual query operations is limited. Complex analysis on graphs usually requires more than one query operation. Users have to combine these individual tools together, going through the complication of resolving the differences in execution platforms and data formats. Therefore, it is crucial to develop graph querying systems that include sophisticated graph operations as well as simple ones.

In this proposal, we introduce a graph querying toolkit, called Periscope/GQ. This toolkit is built on top of a traditional RDBMS. It provides a uniform schema for storing graphs in the database. The key feature of Periscope/GQ is that it supports various simple and complex graph query operations. Users can easily combine several operations to perform complex analysis on graphs. To speed up query operations, Periscope/GQ employs novel indexing techniques that make use of the existing robust index structures in a typical RDBMS, which makes adoption and implementation easy. By applying Periscope/GQ to life science and social networking applications, we demonstrate the power of Periscope/GQ in performing complex analysis on graph databases.

2. SYSTEM ARCHITECTURE

Periscope/GQ is built on top of the RDBMS PostgreSQL (`http://www.postgresql.org/`). Graph data are stored and indexed in the RDBMS, while graph query algorithms are implemented as applications on top of the RDBMS. This design allows us to easily port the implementation to other RDBMSs. Figure 1 shows the architecture of Periscope/GQ.

2.1 Data Model and Data Storage

Periscope/GQ supports a general graph model. Under this model, graphs can be directed or undirected. Nodes and edges are allowed to have arbitrary labels and attributes. In fact, node and edge labels can be viewed as special at-

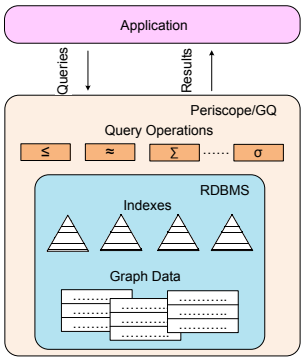

Figure 1: Periscope/GQ architecture

Graph Table

graphID	attrName	attrType	attrValue
1	name	string	wnt pathway
1	source	string	KEGG Database
.....	.....	.....	.....

Node Table

graphID	nodeID	attrName	attrType	attrValue
1	1	label	string	wnt
1	1	familyID	string	K00182
.....	.....	.....	.....	.....

Edge Table

graphID	node1ID	node2ID	attrName	attrType	attrValue
1	1	2	directed	boolean	false
1	1	2	link type	string	protein interaction
.....	.....	.....	.....	.....	.....

Figure 2: Examples of the graph table, the node table and the edge table

tributes. Furthermore, attributes can be of arbitrary types.

Graphs are stored in a graph table, a node table and an edge table using the following schema:

```
Graph(graphID, attrName, attrType, attrValue)
Node(graphID, nodeID, attrName, attrType, attrValue)
Edge(graphID, node1ID, node2ID, attrName, attrType,
attrValue)
```

Each graph is uniquely identified by a `graphID` in the graph table. A graph can have attributes associated with it. For example, in Figure 2, the graph with `graphID=1` has a string attribute called `name`. The value of this attribute is `wnt pathway`. This graph has another string attribute describing the `source` of the graph data. Within each graph, nodes are uniquely identified by their `nodeIDs`. Similarly, nodes can have attributes associated with them. In Figure 2, the node with `nodeID=1` in the graph with `graphID=1` has two attributes `label` and `familyID`. Edges within a graph are identified by the IDs of the end nodes. Again, edges can have attributes associated with them. In Figure 2, the edge in graph 1 with end nodes 1 and 2 is undirected, which is indicated by setting the `directed` attribute to the value `false`. A directed edge is represented by setting this `directed` attribute to the value `true` and the direction is from `node1ID` to `node2ID`. This edge has another attribute indicating its `link type`. Graphs, nodes or edges with multiple attributes have multiple entries in the corresponding tables. In the current implementation, all graphs in the system must have a `name` attribute with non-null values; all nodes must have a `label` attribute with non-null values; and all edges must have a `directed` attribute with non-null values.

2.2 Graph Query Operations

The graph query operations supported in Periscope/GQ are listed in Table 1. The first six operations in this table are relatively "simple" and have been extensively studied. The last three operations are more complex and play crucial roles in complex analyses, hence are described below.

Approximate Graph Matching: Analogous to the keyword search in a sequence/text database, graph matching finds graphs or subgraphs in the database similar to the query graph. It is an important operation to analyze graphs in complex ways. Due to the noisy and incomplete nature of most real graph datasets, *approximate* matching plays a more critical role than exact matching in practice. Approximate matching allows node/edge insertions and deletions, and node/edge mismatches.

Periscope/GQ incorporates the novel approximate graph matching method, called SAGA [11]. SAGA employs a flexible model for computing graph similarity and utilizes an index-based matching technique that allows it to efficiently evaluate queries even against large graph datasets.

Large Graph Alignment: Most graph matching methods, including SAGA, are designed to query graphs that are small in size (tens of nodes and edges). However, some applications require matching *large* graphs. One such example is to align protein interaction networks (graphs with thousands of nodes and edges usually) of two species to study evolutionary conservation.

To address the need for approximate matching of large graphs, Periscope/GQ incorporates a novel technique, called TALE [10]. TALE employs an indexing technique, which achieves high pruning power and scales linearly with database sizes. The innovative matching algorithm utilizing this index is orders of magnitude faster than the state-of-the-art graph alignment methods.

Graph Summarization: As graphs in many applications, especially large-scale social networking applications, grow larger and larger, it becomes almost impossible for users to understand the information encoded in large graphs by mere visual inspection. Therefore, graph summarization methods are required to help users understand the underlying characteristics of large graphs.

Periscope/GQ employs the k-SNAP method [9] to summarize graphs. k-SNAP allows users to freely choose the attributes and relationships that are of interest, and then makes use of these features to produce small and informative summary graphs. Furthermore, users can control the resolution of the resulting summaries and "drill down" or "roll up" the information, just like the OLAP-style aggregation methods in traditional database systems.

2.3 Efficient Query Evaluation using Indices

To efficiently evaluate queries, Periscope/GQ employs a variety of indexing techniques. For simple operations, such as graph selection, node selection and edge selection (see Table 1), traditional indexing methods are sufficient. However, designing indexing mechanisms for the more complex operations, such as approximate graph matching and large graph alignment, is more challenging. Rather than designing new index structures, which makes adoption and implementation hard, Periscope/GQ makes use of existing index structures already provided inside the RDBMS in interesting ways.

Operation	Description
Graph Selection	Select graphs based on conditions of graph attributes.
Node Selection	Select nodes based on conditions of node attributes and/or graphID.
Edge Selection	Select edges based on conditions of edge attributes and/or graphID, and/or nodeIDs.
Node Similarity	Given a node, find nodes that have similar attribute values and similar neighbors.
Path Existence	Decide whether two given nodes are connected by a path.
Shortest Path	Find the shortest path between two given nodes.
Approximate Graph Matching	Find graphs or subgraphs in the database that are similar to the query graph.
Large Graph Alignment	Align two or more large graphs to find conserved subgraphs.
Graph Summarization	Produce summaries capturing the characteristics of the original graphs.

Table 1: Graph operations supported in Periscope/GQ

In [11], we proposed the *Fragment Index* to speed up the approximate graph matching in SAGA. The indexing units of the *Fragment Index* are small subgraphs in the database. We used a B+-tree to implement the *Fragment Index* (see [11] for details). The large graph alignment method TALE [10] employs the *Neighborhood Index* to expedite the query processing. The indexing units of the *Neighborhood Index* are the neighborhoods of all the nodes in the database. A *neighborhood* is defined as the induced subgraph of a node and its neighbors (adjacent nodes). This *Neighborhood Index* is implemented as a hybrid index structure, which has two levels. The first level of this index structure is a B+-tree. Each leaf entry of this B+-tree points to a second-level bitmap index (see [10] for details). Both the *Fragment Index* and the *Neighborhood Index* are easily implemented inside a typical RDBMS, and result in orders of magnitude speedup for the corresponding query operations in most cases.

3. DESCRIPTION OF DEMONSTRATION

In this section, we use two real example applications: one life science application and one social networking application, to demonstrate the power of Periscope/GQ in performing complex analysis on graphs.

3.1 Example 1: Gene Regulatory Networks

Life science is experiencing a transition from focusing on the function of a single molecule to analyzing biological systems and their behavior as regulatory networks. Genome wide microarray analysis with pathway mappings and scientific literature searches can generate gene regulatory networks of different species under different biomedical conditions. These gene regulatory networks can be naturally modeled as graphs, where nodes represent genes and edges indicate their interactions. The size of an individual gene regulatory network can be as large as several thousands of nodes and tens of thousands of edges. These networks serve as a rich source of information to be analyzed for discoveries that can lead to the cure of human diseases. Graph querying systems plays a critical role in helping life scientists analyze large gene regulatory network datasets.

In this section, we show an example of how different graph query operations in Periscope/GQ can be combined to help a group of life scientists find the key drugable pathways to validate therapeutic targets for Type 1 Diabetic Nephropathy (DN). Figure 3 shows the workflow of the analysis.

Through genome wide microarray analysis on human and mouse DN samples, large gene regulatory networks of the two species are generated. Each network contains hundreds to thousands of nodes and edges. By issuing graph summarization queries using k-SNAP, summaries based on features of interest are generated to help life scientists understand the underlying characteristics of individual networks.

In addition, cross-species network comparison is an effective way to identify which subnetwork produce the disease in both systems. This operation can be achieved by aligning the networks of the two species using TALE. This conserved subnetwork is a good candidate for therapeutic target validation. This operation can also be pipelined with further queries, such as querying the conserved subnetwork against a database of pathways to find out which biological processes might be involved or affected by the conserved mechanism. A pathway consists of a set of cellular entities interacting to carry out some biological process. The query against a database of pathways can be achieved by an approximate graph matching operation using SAGA. Alternatively, the conserved subnetwork can also be used to query a database of parsed literature graphs to search for papers that may have already studied the conserved mechanism. In [11], we described a way to perform document comparison using the graph matching method SAGA. Through natural language analysis, each biomedical document is represented by a graph in which a node indicates a gene studied in the document and a link is drawn between two genes if they are discussed in the same sentence (indicating a potential association between the two). Matching the conserved subnetwork against these parsed literature graphs can help life scientists find out previous studies with similar interests.

Through the above analysis, the life scientists actually identified several good candidates that they are validating for therapeutic targets.

3.2 Example 2: DBLP Coauthorship Networks

In this section, we demonstrate how Periscope/GQ can be used to analyze coauthorship networks from the DBLP Bibliography data [4]. In a coauthorship network, each node represents an author, and the edge between two authors indicate their coauthorship.

In this example, we are first interested in studying how researchers in the database area coauthor with each other. However, the coauthorship patterns are hidden inside the large DB coauthorship network, which contains over 7 thousand nodes and around 20 thousand edges. Graph summarization operation (using the k-SNAP method) is very critical in understanding the characteristics of this large graph (see Figure 4). k-SNAP generates summaries with the resolutions that the users can understand, and also provides "drill-down" and "roll-up" abilities to navigate summaries with different resolutions. Detailed analysis on coauthorship

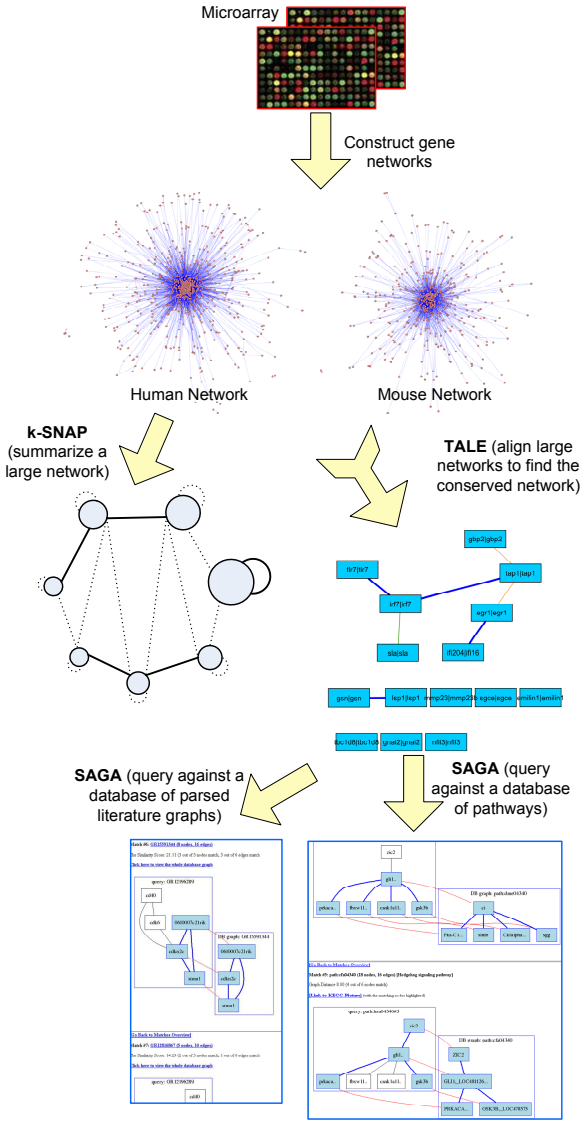

Figure 3: Example application of Periscope/GQ for gene regulatory network analysis

networks using the k-SNAP method can be found in [9]. The k-SNAP operation can also be used to examine the similarities and differences in the coauthorship relations across communities, such as the DB community and the AI community. Our demonstration will include these examples.

As shown in Figure 4, further analysis can be performed on subnetworks of the DB coauthorship network. Subnetworks can be easily generated by node selection and edge selection operations. For example, one can construct a coauthorship network only about authors who publish in SIGMOD conference from 2001 (when double-blind review was first adopted) to 2007. This SIGMOD coauthorship network can be constructed by selecting authors (nodes) who have at least one SIGMOD paper in year $\geq$ 2001 and $\leq$ 2007 and coauthorships (edges) that appear in these publications. Similarly, one can also construct a VLDB coauthorship network. Further analysis can be performed by aligning the SIGMOD and VLDB coauthorship networks to compute the conserved coauthorships across the two communities.

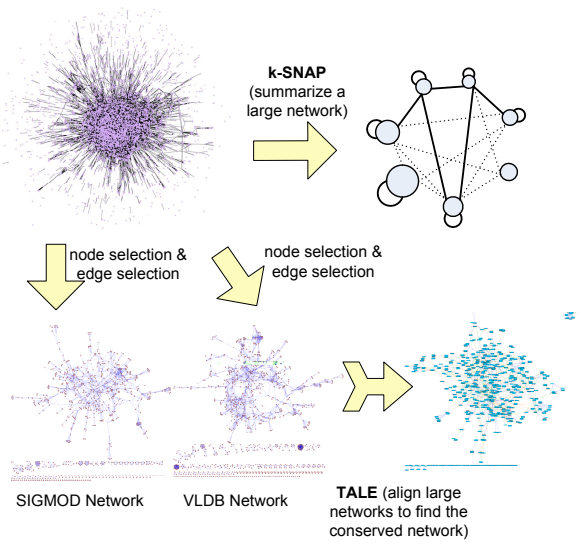

Figure 4: Example application of Periscope/GQ to analyze coauthorship networks

Acknowledgment

This research was primarily supported by the National Science Foundation under grant DBI-0543272, the National Institutes of Health under grant 1-U54-DA021519-01A1 and by an unrestricted research gift from Microsoft Corp.

4. REFERENCES

[1] A. Gupta and B. Ludäscher. The many faces of process interaction graphs: A data management perspective. *OMICS*, 7(1):105–108, 2003.

[2] R. H. Guting. Graphdb: Modeling and querying graphs in databases. In *VLDB*, 1994.

[3] K. Wilkinson et al. Efficient RDF storage and retrieval in Jena2. *Techniqual Report: HPL-2003-266*, 2003.

[4] M. Ley. DBLP Bibliography. `http://www.informatik.uni-trier.de/~ley/db/`.

[5] M. Baitaluk et al. Pathsys: Integrating molecular interaction graphs for systems biology. *BMC Bioinformatics*, 7(55), 2006.

[6] M. Kanehisa et al. The kegg resources for deciphering the genome. *Nucleic Acids Res.*, 32:D277–D280, 2004.

[7] D. Shasha, J. T.-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.

[8] L. Sheng, Z. M. Ozsoyoglu, and G. Ozsoyoglu. A graph query language and its query processing. In *ICDE*, 1999.

[9] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.

[10] Y. Tian and J. M. Patel. TALE: a tool for approximate large graph matching. In *ICDE*, 2008.

[11] Y. Tian et al. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.

[12] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, 2004.

[13] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, 2005.