Visual Query Formulation: A Review of Visual Query Systems

Bradley Malgas
Computer Science Department
University of Cape Town
South Africa
mlgbra002@myuct.ac.za

ABSTRACT

Data access has always been limited by a user's technical expertise. Professionals who may be considered experts in their domain but may not be skilled in structured query languages require IT professionals to assist with their query construction needs. The ability to provide end users with easy and simple query formulation is a topic that still needs to be explored in depth. We will be reviewing some approaches taken as well as the systems which utilize these approaches.

KEYWORDS

Visual query systems, Visual Query Formulation, Ontology-based data access, Graph database

1 Introduction

Relational databases are the standard when it comes to storing large amounts of data. Accessing this information becomes increasingly difficult as the size of the database increases. Making useful and powerful queries of the database without understanding the internal structure and the relationships that exist between the data proves very difficult. Ontology-based data access (OBDA) makes this easier by hiding these details from the user and allowing them to use vocabulary from the ontology to create queries. The KnowID architecture [7] that the topic of this literature review was inspired from, makes use of OBDA to allow querying at the conceptual layer. The users' interactions with the querying environment are further simplified using visual queries.

Visual queries make use of graphical elements to allow users with less technical experience in traditional query languages like SQL (Structured Query Language) to formulate queries. Visual queries are achieved by implementing a Visual Query Language that will be used in conjunction with the query environment. These two components together form a Visual Query System (VQS) which we will be exploring in more detail.

2 Visual Query Systems

Visual query systems (VQS) are systems that query databases and visually represent the data as well as any requests. They are a resulting of applying HCI principles in the design of query languages. VQSs are aimed at users with limited technical knowledge about the internal working of a database.

[3] Sets the foundations for early classifications of VQSs and more specifically those that work with relational databases. Query languages are analysed based on two key criteria: functional capabilities and usability. Additionally, criteria can also be introduced. [1] Introduces a taxonomy that can be used to classify VQSs. The taxonomy is based on 3 criteria namely the limitations on what can be expressed, ease-of-use and types of users it can accommodate. We will use this in the Discussion section of the review paper to analyse the approaches taken by VQSs.

In a VQS, the visual representations used for the query can differ from that used for the query result. Both of these can be visually represented in a number of ways. We will explore some of the approaches identified in [3]. The first of which is form-based. This method is meant to resemble a paper-based form. Users input values to specify constraints. This approach is limited as there is no way to understand the relationship between the data. The form-based query result is a table. The advantage of using this structure is that it the same as what's used to show data in the relational model.

The second approach is diagram based and this will be the focus of our paper. In VQSs, the most common diagram components used to display information are geometric objects with lines being used to display the logical relationships between data components. This approach allows the user to have a better understanding of the data. Querying can be done using a range of approaches one of which is used in the X-VIQU system. Users can select objects to begin the query and add constraints as necessary. At each step, a "tree" representing the query is formulated and is added onto as the query becomes more complex. As with the query, the query result can also be represented using different diagrams although it is common to see the same representation used in query and query result.

The strategies used in the query formulation are also classified into classes. One strategy class proposed is that of schema navigation where the user can select which components will be used in the query by choosing a starting concept and building a "path" to other related concepts whilst specifying constraints. Other mentioned strategies include subqueries and pattern matching.

2.1 Graph Based Query Systems

Graphs are a very common mechanism used to represented data. A graph database is simply a collection of graphs where the nodes represent data and edges represent the relationship between

the nodes. This style of data representation is common in the scientific research fields of biology and chemistry.

Queries can be applied to explore and retrieve information from these graphs. The query is represented differently in different graph query languages. The most common approach is to represent the query as a subgraph although graph queries can also be represented as a regular expression over the alphabet of edges [13].

In [9], we begin with a query subgraph. The graph database is then searched to find all graphs in the database that contain the query subgraph. When a subgraph exists in the database it is said to be a *fragment* of the database. Certain fragments appear in graphs more often than others. Based on the frequency of its appearance, a fragment can either be *frequent* or *infrequent*. The proposed GBLENDER system uses fragments and graph matching for graph queries with the use of *action-aware* indexing. The two types of indexing methods, action-aware frequent indexing (A^2F) and action-aware infrequent indexing (A^2I) match frequent and infrequent query fragments. The system increases performance by allowing the graph query construction and processing to happen simultaneously. This is achieved by using the indexing mechanisms to retrieve partial results while the query is being formulated.

[10] Introduces project ARIANE - a conceptual model of querying medical databases using a graph query. The ARIANE system is aimed at producing results that satisfy users' needs and minimizing any extra or unnecessary information. The project uses the Unified Medical Language System (UMLS) as an ontology as it contains much of the terms used in the medical domain. This works well since knowledge in the UMLS is organized in a data structure. The UMLS structure is rather complex and is modelled conceptually using a hierarchy. The Semantic network is on the top layer, the context in the middle and the concepts are at the lowest layer. Unlike GBLENDER which uses indexing, ARIANE uses projection in order to match queries. Once a user has formulated the query graph, it needs to be able to be 'projected' onto the data graph for there to be a match in the data structure.

In order to formulate a query, the user needs to interact with an environment that will be used to construct the query. The user interface of GBLENDER allows users to drag labels from a label pane into a graph query pane. Edges can be added to link together different labels. While the user adds more and more labels, the current query gets classified as being either frequent or infrequent. The query is processed using the GBLENDER algorithm which builds on Ullman's algorithm. The user interface for ARIANE functions in a similar way of GBLENDER. In order to build a query graph, a user will firstly have to select a concept and a context for node, a relationship that will be involved and then a context and concept of a secondary node. This will then build a conceptual graph that links these two nodes. This process can be repeated to create bigger queries.

[6] Introduces an aspect of databases that is not seen with the two VQSs mentioned above, the idea of temporality being a part of the database. A temporal database stores information about time instances. This includes when data is considered valid, when this validity was established and when the data was stored. The existing graph model can be extended to include temporal aspects of data structures.

Queries are constructed in the same way as with the previous Graph Based VQSs with the added difference being the ability to use temporal operators namely *Snapshot* which is used for temporal selection and *Slice* which is used for temporal projection. Temporal selection is a logical condition that must be true of the time associated with the data. The result of applying this temporal selection, is the actual time values returned - the temporal projection.

Queries of this type are formulated using the entire graph database shown in the 'Schema Window'. This window shows the user all the various classes and the relationships between them. The user will then select one schema to focus on. Once this has been obtained, the user selects a classes and relationship types that will be used in the query. The query generation process is handled similar to those seen before, but users can now create Snapshot or Slice queries where Snapshot queries will return results pertaining to one time-interval and Slice queries return results pertaining to more than one time-interval.

2.2 Ontology Based Query Systems (using SPARQL)

Ontology-based data access is based on the concept of data virtualization [11]. This means all data remains where it is an ontology is an added layer to provide a knowledge base. In this system, the database is used for the retrieving of data and the ontology is used for the retrieval of implicit facts about the data.

OptiqueVQS [12] is an ontology based visual query system. The focus of this system is to make querying simpler for end-users who have minimal technical experience. This system does not have any formal syntax for the query representation and instead opts to project the ontology onto a graph and use this to assist with querying.

In OptiqueVQS, user queries are constructed in the form of a tree-like structure and the user is able to link classes and properties. In order to create suggestions, the system uses a technique call ed 'graph navigation' which involves projecting an OWL2 ontology onto a graph and suggesting properties that are accessible within one step.

The WONDER system [2] uses similar graph properties. In the WONDER system, queries are comprised of a query graph as well as a constraint expression. The query graph, which is almost identical to the ontology graph, relies on users dragging nodes from the ontology graph and then joining them as desired in a query pane. In order to specify the constraints, the system runs the query using DL-Lite and then applies constraints on top of this using an EQL-Lite query. In order to have an interactive environment to display the visual queries, the WONDER system uses Scalable Vector Graphics (SVG).

[5] Introduces a different way to represent the user queries. ViziQuer uses a UML styled notation to represent the visual queries. Each node represents a data instance. Each data instance

contains instance attributes and can also contain additional conditions. Between each node there is a connecting edge which represents the relationship between the two instances. The resulting "UML graph" query must always contain a main query node that serves as the root of each query.

Each VQS also provides users the ability of saving queries. The ViziQuer system in particular, makes use of projects to save a user's queries. Each project can contain multiple query graphs which themselves can contain multiple queries within them. To start a project, a user needs to supply a data schema with a SPARQL endpoint. Each project also needs SPARQL engine to assist with the query translation. Functionality is made available to extract the scheme from an OWL ontology.

Visual query systems will always stay behind SPARQL in terms of expressive abilities due to the lack of certain selection queries [5]. The ViziQuer tool helps facilitate the creation of visual queries on SPARQL data. It builds upon previous work [4] and extends basic visual query generation to include more complex tasks like subqueries and aggregation.

2.3 Conceptual Queries using Visual Query Languages

A Visual Query System differs from a Visual Query Language (VQL). A VQS is a system that users interact with to generate queries, a VQL is a language that has pre-defined syntax and formal semantics [11]. Visual Query Systems are not as expressive as the Visual Query Languages that they are based on [5]. This trade-off is a result of keeping them usable. Developing an expressive visual query language will result in a system that is too complicated for end users and less efficient than regular structured query languages for IT experts.

To assist with this dilemma, we can make use of a conceptual schema. A conceptual schema is expressed using concepts that may be familiar to end-users. It is different from SQL in that in these queries don't need to be updated if there is a change made to the underlying schema.

The Query by Diagram System [10] makes use of this idea. Before query formulation, the user is able to browse the conceptual schema. Users will then extract any subschema which contains the information that will be used in the query. The system makes use of graphical primitives which are functions that map the database to the required structure.

While some systems use existing VQLs to assist with queries, others create a new language entirely. The ConQuer-II [5] language is a conceptual query language allowing users to formulate queries in terms of relationships using operators such as "and", "or" and even "not" (negation) avoiding the need for understanding the details of relational databases. Unlike the previously mentioned VQS systems, ConQuer-II is based on Object-role Modelling (ORM).

ORM is a modelling approach that revolves around the concept of objects whereby every component in the model is object that has a specific role. For example, while reading this review, you are assuming the role of reader while this review is assuming the role of being read. ORM makes not explicit use of attributes [8]. ORM models result in queries that are more stable and also more expressive than Entity-Relationship Modelling.

In ORM, object types are shown as labelled ovals. Entity types has a solid outline while value types have a dotted outline. Each role is represented as a box. Each box forms part of a relationship that can an n-ary relationship type. This forms the basis for ConQuer queries, but users need not be familiar with this schema although it differs from the ER schema used in the QBD system.

3 Discussion

After understanding the approaches taken by the various systems in order to create Visual queries, we need to analyse the, The criteria we will use to analyse the discussed VQSs are: limitations on what can be expressed, ease-of-use and types of users it can accommodate.

3.1 Limitations

Some of the problems that VQSs are faced with are the ability to address more complex problems such as cyclic and disjunctive queries as well as negation. Apart from query formulation other related concerns are being able to efficiently deal with large ontologies.

Certain VQSs also have limitations that may not be present in others. OptiqueVQS does not provide the ability to generate complex queries since it heavily focused on end-users and their typical use cases. Users are able to delete nodes in the event of a mistake during the query process, a feature which is not present in the graph-based querying in GBLENDER.

3.2 Ease of Use

One of the best ways to demonstrate the ease-of-use of VQSs is to compare one with a great user interface and one with one that isn't that user friendly.

The user interface of OptiqueVQS contains multiple UI elements that have been linked together. The benefits if this approach provides modularity and leaves room for flexibility [12]. The first widget element is the used for representing queries. Users add nodes and edges, which represent objects and object properties respectively, onto this pane. The second widget contains a list of all objects and the third, a list of all properties. These widgets are updated dynamically as the query is formulated providing the user with suggestions based on the current query. We were able to see this system in practice and the time taken to learn the system is relatively low.

On the other hand, the query formulation in ConQuer is textual based. The user begins by picking object. Once the object is placed in the query pane, the system displays the roles played by that object. The user can then choose a relationship. Object contained within each relationship can also be clicked and the system will then display the roles played by that object. Through

this process, the user can generate a query path. This path is displayed in a few lines as an outline of the internal structure. This means users will need some understanding on the outline in order to generate meaningful queries. Internally the queries are represented as syntax trees that get transformed into SQL, which would have been a better approach for representing the queries visually.

3.3 User Types

ConQuer-II is backend independent and can be used with multiple databases. It can allow the user the ability to express powerful, complex queries. This does however limit the types of users that can use the tool since it is very difficult to understand as an end user without some technical knowledge. Having to be able to differentiate between Location, Navigation or Replacing primitives does not line up with the use of conceptual queries which is essentially to use concept which are familiar to users.

4 Conclusions

In the topic of Visual Query Formulation there many approaches that can be taken to achieve the desired results. Graph-based queries make use of graph databases and attempt to match subgraphs. Ontology-based data access allow the use of ontologies as a knowledge layer data access mechanism that can be leveraged for query formulation. Sometimes it is even common for developers to create their own Visual Query Language to help facilitate the query formulation process. Each approach has produced a number of systems. Each system also has its own flaws and drawbacks. The best approach will be explored in the Project Proposal.

REFERENCES

[1] Batini, C., Catarci, T., Costabile, M. F. and Levialdi, S. *Visual Query Systems: A Taxonomy*. City, 1991.

[2] Calvanese, D., Keet, C. M., Nutt, W., Rodríguez-Muro, M. and Stefanoni, G. *Web-based graphical querying of databases through an ontology: the WONDER system*. City, 2010.

[3] Catarci, T., Costabile, M. F., Levialdi, S. and Batini, C. Visual query systems for databases: A survey. *Journal of Visual Languages & Computing*, 8, 2 (1997), 215-260.

[4] Čerāns, K., Bārzdiņš, J., Šostaks, A., Ovčiņņikova, J., Lāce, L., Grasmanis, M. and Sproģis, A. *Extended UML class diagram constructs for visual SPARQL queries in ViziQuer/web*. Voila, City, 2017.

[5] Čerāns, K., Šostaks, A., Bojārs, U., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Romāne, A., Sproģis, A. and Bārzdiņš, J. *ViziQuer: a web-based tool for visual diagrammatic queries over RDF data*. Springer, City, 2018.

[6] Fernandes, S., Schiel, U. and Catarci, T. *Visual query operators for temporal databases*. IEEE, City, 1997.

[7] Fillottrani, P. R. and Keet, C. M. An architecture for efficient knowledge-driven information and data access (2019).

[8] Halpin, T. *Object-role modeling (ORM/NIAM)*. Springer, City, 1998.

[9] Jin, C., Bhowmick, S. S., Xiao, X., Cheng, J. and Choi, B. *GBLENDER: towards blending visual query formulation and query processing in graph databases*. City, 2010.

[10] Joubert, M., Fieschi, M., Robert, J.-J., Volot, F. and Fieschi, D. UMLS-based conceptual queries to biomedical information databases: an overview of the project ARIANE. *Journal of the American Medical Informatics Association*, 5, 1 (1998), 52-61.

[11] Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D. and Horrocks, I. Ontology-based end-user visual query formulation: Why, what, who, how, and which? *Universal Access in the Information Society*, 16, 2 (2017), 435-467.

[12] Soylu, A., Kharlamov, E., Zheleznyakov, D., Jimenez-Ruiz, E., Giese, M. and Horrocks, I. *Ontology-based visual query formulation: An industry experience*. Springer, City, 2015.

[13] Wood, P. T. Query languages for graph databases. *ACM Sigmod Record*, 41, 1 (2012), 50-60.