XZ2

FYP Proposal

When Visual Query Interfaces Meet Graph Query Engines

by

LIANG Houdong and YAO Chongchong

XZ2

Advised by

Prof. ZHOU Xiaofang and Dr. HUANG Kai

Submitted in partial fulfillment of the requirements for COMP 4981

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2022-2023

Date of submission: September 16, 2022

Table of Contents

1 Introduction

1.1 Overview

In recent decades, data has exploded exponentially and thus humans are facing the challenge of storing, managing and accessing data [11]. Under this scenario, databases are becoming increasingly crucial because of their strong capacity for storing and managing large amounts of data as well as ensuring data security. A well-performing database is necessary to any company or organization because it stores pertinent details such as employee records, transactional records, salary details, etc. There are many different types of databases including relational databases, graph databases, and cloud databases, each of which has unique strengths in storing and managing data. Among them, graph databases are extremely crucial especially when people care more about the relationships between data points than the individual points themselves [19].

A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store data. The advantage of graph databases is that the relationships between the data points are directly stored and can be queried conveniently. Much real-life information can be seen as a graph database. For instance, social networks (e.g., Twitter) are about connections between people and thus graph databases are a perfect fit for social networks. Each chemical molecule can be seen as a small graph and they can be combined to form a large graph database, which is what PubChem did [20]. Other examples include co-purchase networks (e.g., Amazon.com), and information networks (e.g., DBpedia [7]). According to a recent *Markets and Markets* report [17], the global graph database market size is predicted to increment from USD 1.9 billion in 2021 to USD 5.1 billion by 2026, indicating an exploding demand for the research and analysis of the graph data.

Unfortunately, although there is an increasing need to query graph databases, composing graph queries often demands considerable cognitive effort from users and requires adequate programming skills [1]. To access and research data in a database, one needs to master Query Languages (QL) and Query Engines (QE). Query Languages are computer languages used to make queries in databases. Query Engines are software that lies above the database and provides users with an interactive interface as well as integrated operations. For graph databases, people usually choose Cypher [6] as the Graph Query Language (GQL) and Neo4j [18] as the Graph Query Engine (GQE). A user must be familiar with the syntax of the Cypher and must be able to express his search goal accurately using a syntactically correct form. However, in many real-life domains (e.g., social science, life science, chemical science) it is unrealistic to presume that users are proficient in those skills. If a user has little background in computer science, the time cost of learning Cypher and Neo4j will be huge.

Fortunately, unlike more textual Structured Query Language (SQL), graph queries are closer to human intuition and can be represented graphically [3]. Consequently, Visual Query Interfaces (VQI), an application on top of the QE that enables users to draw the graph query and then convert the graphical pattern into the formal Cypher code, have become a feasible solution. With the help of VQI, non-programmer users do not need to learn the Cypher and Neo4j and can do the query by inputting vertices and edges graphically just like using some painting software. This will greatly reduce the difficulty of doing the query by non-professionals and save an abundance of time for them.

1.2 Objectives

The goal of this project is to design and implement a Visual Query Interface that is compatible with Neo4j, namely VisualNeo, to help non-expert users do graph queries. VisualNeo will include many cutting-edge and useful functions such as data-driven VQI design, action-aware graph query processing, and effective query results visualization, which are novel features of recently published VQIs.

The bottlenecks of this project include the diversity and coverage of recommended patterns provided by data-driven VQI, the responsiveness of action-aware graph query processing, and the aesthetic satisfaction from the users of displaying the results. The construction of data-driven VQI is supported by a powerful framework that generates canned patterns having high diversity and coverage. Data-driven VQI can greatly reduce the cognitive load of users when they build the query. Action-aware graph query processing requires a responsive and acumen design that returns the query results after each step users build the query instead of only after users click the “Run” button. Lastly, according to the aesthetic-usability effect, the aesthetic appearance of a VQI influences its usability as it influences the way users interact with it. Due to the data-driven design of the VQI, the display of the panels on the interface may encounter problems. Therefore aesthetics-aware VQI design will be another challenge in our project.

To implement the data-driven VQI, we introduce modules including a label generator module to generate a set of all labels in the database, a sampler module to reduce the size of the patterns, and a canned pattern selector module to select final patterns, etc. All the modules will combine to streamline the generation of canned patterns. For the action-aware graph query processing, we will adopt a sensitive query processor that utilizes the latency between users' input to evaluate the partially constructed query graph at each step and return the feedback to users to guide users' query formation. Considering aesthetics, we reformulate the data-driven visual layout design problem as an optimization problem and the goal is to find an optimal layout that minimizes query formulation complexity and visual complexity of the interface.

1.3 Literature Survey

1.3.1 Combination of Graph Data Management and HCI

Traditionally, devising efficient techniques in GQE for interactive search and exploration is independent of VQI design. This is because the two key enablers of these efforts, Human-Computer Interaction (HCI) and data management, have evolved into two disjoint scientific fields, rarely making any systematic effort to leverage principles and techniques from each other [2]. The HCI community has focused more on issues such as user task modeling, menu design models, and human factors while data management researchers tend to refrain from those challenges. On the other hand, the HCI researchers are not willing to face the challenges in the data management domain, even though it may provide them with new ways that they build visual interfaces.

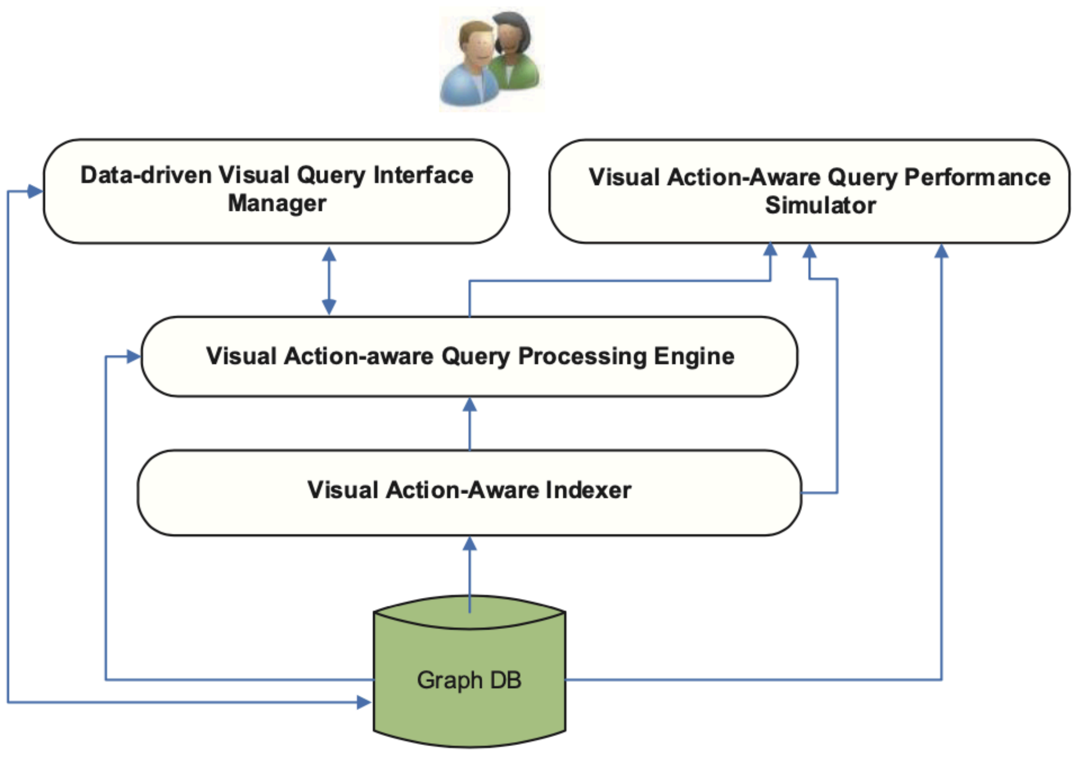
Data-driven Visual Query Interface Manager
Visual Action-Aware Query Performance Simulator
Visual Action-aware Query Processing Engine
Visual Action-Aware Indexer
Graph DB

Figure 1. The architecture of an HCI-aware visual graph query framework [2, Fig. 2]

In recent decades, the chasm between traditional graph data management and HCI has been bridged. In a relatively recent study, an HCI-aware visual graph querying framework is proposed [2]. Figure 1 depicts the generic architecture of the framework. The interface manager is responsible for constructing several panels of the VQI in a data-driven manner. The query processing engine can process the queries during query formulation while the query performance simulator provides an inclusive framework for large-scale empirical study. Lastly, the indexer is used to support the query processing engine by assigning action-aware indexes.

1.3.2 Data-driven Visual Query Interface Design

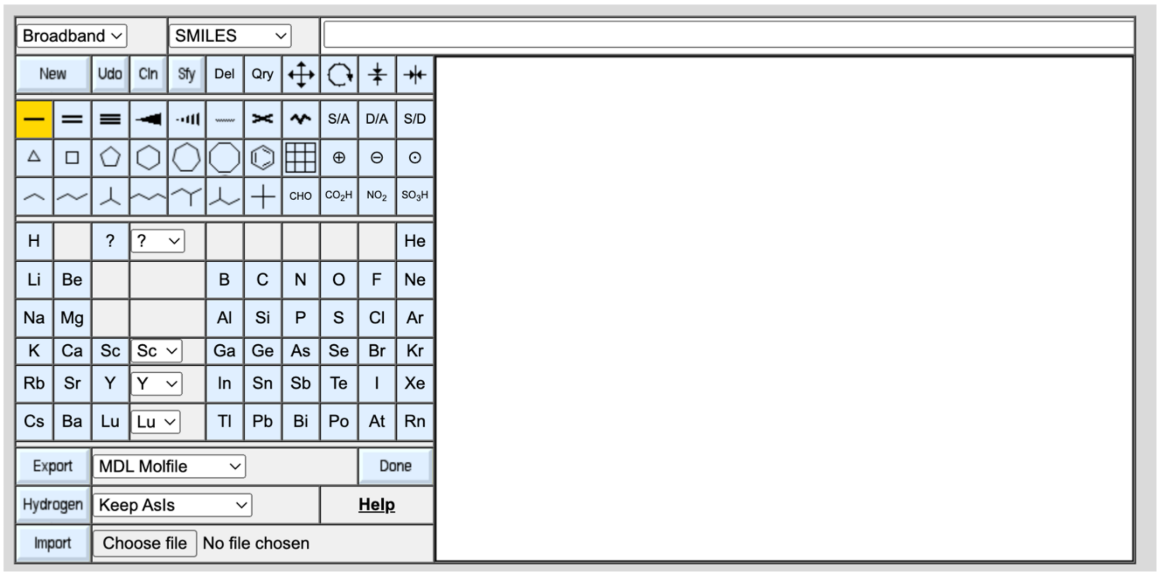

Figure 2. GUI for substructure search in PubChem

Current VQIs such as PubChem (shown in Figure 2) and eMolecules [8] usually have a clear and easy-to-use interface. The GUI provides a set of chemical symbols that users can choose from to assign labels to vertices in a graph query. It also lists dozens of predefined patterns (e.g., benzene ring) that can guide and help users during visual query construction. However, this visual query interface construction process is traditionally data-unaware, meaning that the query interface is not flexible. The chemical elements and predefined patterns are not data-driven but are chosen manually by domain experts. It has two drawbacks. Firstly, it is unrealistic that a domain expert has comprehensive knowledge of the topology of the entire graph database. Consequently, a user may not be able to find the desired patterns on the GUI in formulating certain graph queries. Secondly, the frequent update of the underlying database may lead to obsolete predefined patterns and the emergence of new patterns. Furthermore, such an importable visual query interface cannot be integrated into a different graph database such as computer vision and protein structure. When the domain changes, the GUI needs to be reconstructed from scratch.

In response to this challenge, the AURORA provides a novel way of data-driven construction of the VQI [5]. It contains a bunch of modules including a small graph clustering module, sampler module, CSG generator module, label generator module, canned pattern selector module, etc. Given a graph database containing a collection of small- or medium-sized data graphs, AURORA automatically generates the GUI by filling various panels of the interface. PLAYPEN and DAVINCI similarly propose corresponding data-driven VQI designs [21], [23]. PLAYPEN describes the design philosophy and gives an overview of the system while DAVINCI emphasizes more on the program definition and implementation method. However, according to [4], the maintenance of VQI, data-driven VQIs for massive graphs, and data-driven layout design are still novel research challenges.

1.3.3 Canned Pattern Selection Algorithm

Canned patterns, which are small and frequent subgraph patterns, facilitate efficient query formulation by allowing *pattern-at-a-time* construction. Traditionally, the patterns are selected manually based on domain knowledge. However, this approach is not only arduous but also cannot cover a high diversity of subgraph queries. In response, a research group presented a generic and extensible framework called Tattoo that can automatically generate and select canned patterns from large networks [22]. It first decomposes the network into two regions, namely truss-infested and truss-oblivious regions, and then generates candidate patterns. The final canned patterns are chosen from these candidates by maximizing coverage and diversity and by minimizing the cognitive load of the users.

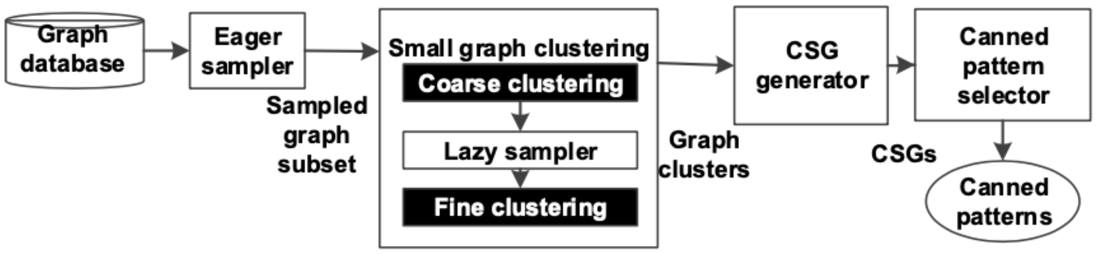

Figure 3. The Framework of CATAPULT [9, Fig. 3]

Another framework of data-driven canned pattern selection called CATAPULT, shown in Figure 3, has a similar design [9]. Based on their topological similarities, CATAPULT clusters the data graphs and then summarize each cluster to create a cluster summary graph (CSG). The canned patterns are generated from these CSGs using the same criteria as Tattoo.

1.3.4 Action-Aware Query Processing

Existing approaches for subgraph matching queries, based on the assumption that the whole graph query has been completely built, are designed to optimize the time required in retrieving the result. However, this assumption may not be necessary. GBLENDER is a new graph query processing algorithm developed recently [16]. Instead of only processing a graph query after the user clicks the "run" button, it handles the incomplete query after each step the user builds the graph. The partial matching results will be fetched to further guide the users' graph query formation. By doing this, it fully exploits the latency provided by the visual query formation. In detail, a novel action-aware indexing scheme is employed to support efficient retrieval so that users' interaction features can be used and explored.

However, GBLENDER cannot be used in subgraph similarity queries and does not support visual query modification. These two drawbacks limit its usage in a practical environment. To solve the problem, the same research team proposed another algorithm called PRAGUE [15]. PRAGUE fix the problems by using a different data structure called spindle-shaped graphs (SPIG). Given a newly added edge in a partial graph query, its set of supergraphs will be recorded in a SPIG in a concise form. Concisely, PRAGUE provides a unified framework to support SPIG-based processing. It is efficient in the modification of subgraph queries and supports similarity queries as well.

2 Methodology

2.1 Design

2.1.1 System Architecture

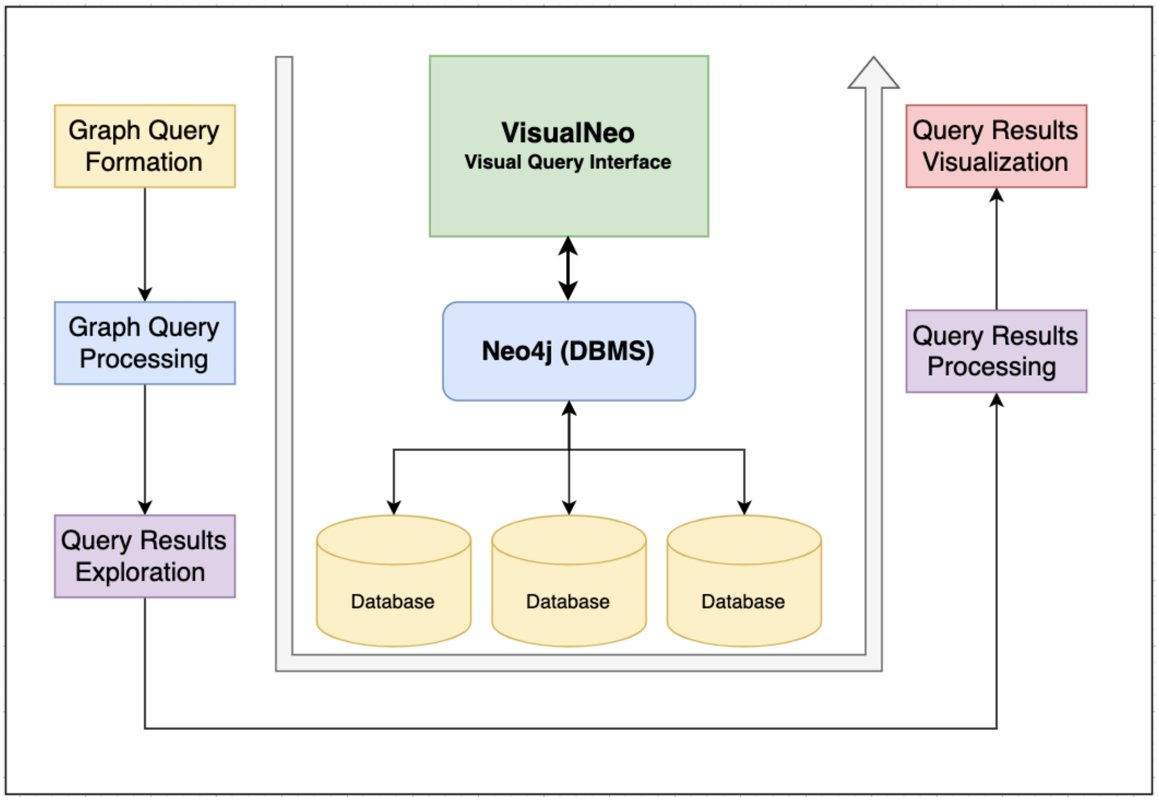

Figure 4. System Architecture

The VisualNeo system is a VQI application on the upper layer of Neo4j, which is the Database Management System (DBMS) we choose for this project. Neo4j connects multiple databases, indicating that the system is compatible with different databases and users can load them manually. The pipeline of system operations is shown in Figure 4: Users first form their graph query through the drawing panel in the VQI, then VisualNeo will parse the graph query and pass the formal Cypher code to Neo4j to do the query. After the query results are explored and processed, they will be returned to the VQI and displayed to the users.

2.1.2 Visual Interface for Graph Querying

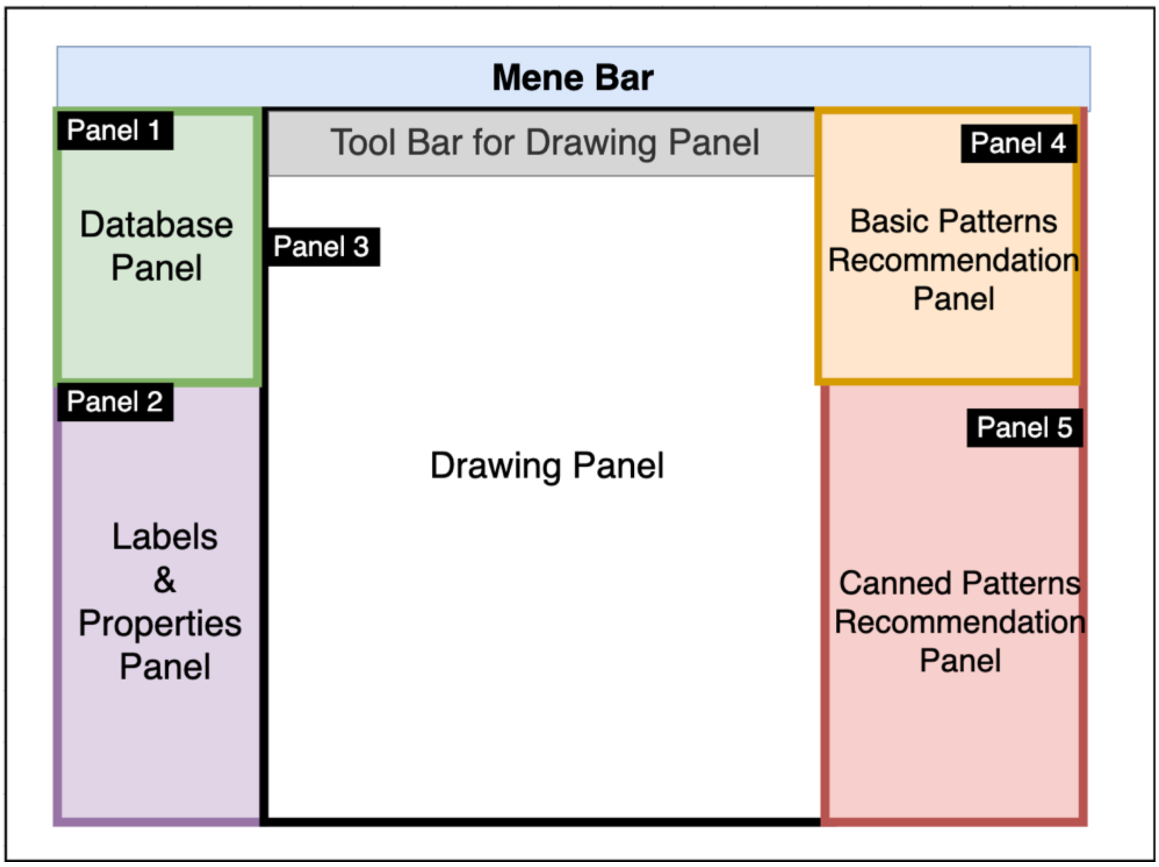
Mene Bar
Panel 1
Database Panel
Tool Bar for Drawing Panel
Panel 3
Panel 4
Basic Patterns Recommendation Panel
Panel 2
Drawing Panel
Panel 5
Labels & Properties Panel
Canned Patterns Recommendation Panel

Figure 5. VQI Design

Figure 5 depicts a rough design of the VisualNeo VQI. *Panel 1* enables a user to complete the database operations such as loading and pattern generation, and the searching operations such as exact search and similarity search. *Panel 2* contains two sub-panels: Labels Panel and Properties Panel. Labels Panel contains a list of distinct vertex labels in the user-selected database. Properties Panel enables users to add property constraints to the vertices when building graph queries. Note that Labels Panel is only useful when the graphs in the database are heterogeneous. *Panel 3* is used for graph query formation, providing a toolbar containing multiple drawing tools. VisualNeo supports two types of patterns, namely basic patterns, and canned patterns. The formal is small-sized predefined patterns (e.g., an edge, triangle, hexagon) chosen by the Basic Pattern Recommendation Algorithm and do not change with a different database. The latter is those of larger size defined through the Canned Pattern Recommendation

Algorithm. *Panels 5* and *6* recommend the basic and canned patterns, respectively. In terms of the VQI design principle, Panel 5 is a paradigm of manual design while Panel 6 represents data-driven design.

2.1.3 Visual Query Formation

Based on the user's purpose of using VisualNeo, we can divide the way of graph construction into two categories: bottom-up search and top-down search. Bottom-up search is goal-driven, meaning that a user has the desired graph query in his mind. The user will try to leverage the basic or canned patterns to form the target query. In contrast, top-down search is stimulus-driven, indicating that a user does not have a predefined target. Rather, he learns the representative subgraphs provided in the basic and canned pattern recommendation panels to gain some inspiration. The design of graph query formation in our project should satisfy both needs. Users should be able to effortlessly find the desired patterns to speed up the query formation process, get inspired by the recommended patterns, and even learn some key features of the underlying database.

To formulate a graph query, users have two approaches. The *edge-at-a-time* approach refers to constructing the graph query by adding one edge at a time. By contrast, in the *pattern-at-a-time* approach, users drop basic patterns or canned patterns on the VQI to the drawing panel. As one would expect, the latter is more efficient because it takes less time to construct a graph query. The goal of our project is to first build a powerful *edge-at-a-time* approach so that at least the user can build the target graph query step-by-step without too much trouble. After that, we will build an accurate and efficient Pattern Recommendation Algorithm that can shorten the *pattern-at-a-time* building process.

2.1.4 Action-Aware Graph Query Processing

In our project, we will first implement action-unaware graph query processing, meaning that the query process will remain idle when the graph query is being built. Only after the user finishes composing the complete graph query and clicks the "Run" button, the query processing will start. Thus, we can adopt the traditional graph query processing techniques to realize that.

On the other hand, the action-aware graph query processing requires an efficient query processing module to retrieve matching results immediately, which needs an abundance of effort. This part will be one of our last goals to achieve.

2.1.5 Query Results Exploration and Visualization

In this part, we adopt different techniques of query results exploration and visualization depending on the type of the graph. When queries are posted on a large collection of small- or medium-sized graphs, subgraphs that match the visual query will be displayed page-by-page. The results can be sorted through some distance metrics we define. For large graphs, on the other hand, we use *supergraphlet-at-a-time* mode, meaning that only a small portion of the large graph containing the matched results is displayed. Users can switch the supergraphlet and view the matches in other parts of the graph.

2.2 Implementation

2.2.1 Visual Interface for Graph Querying

In this project, the GUI will be implemented using JavaFX [14]. JavaFX is the mainstream of Java [13] GUI design and has all the fundamental components we need for the VQI. The layout of the GUI is described in *Section 2.1.2*.

2.2.2 Visual Query Formation

The *edge-at-a-time* visual query formation approach will be implemented through the basic tools provided in the drawing panel. To make the *edge-at-a-time* approach convenient, the design of basic tools will be aesthetic as well as intuitive. The *pattern-at-a-time* approach is supported by Basic Pattern Recommendation Algorithm and Canned Pattern Recommendation Algorithm. The Canned Pattern Recommendation Algorithm will contain several smaller modules such as the canned pattern selector module, sampler module, and CSG generator module.

2.2.3 Action-Aware Graph Query Processing

For the action-unaware graph query processing, our Query Builder and Query Handler can satisfy the need. For the action-aware graph query processing, we introduce an action-aware indexing scheme. As the size of a query graph increases, the size of candidate data graphs decreases, and corresponding information will be returned to users to guide its further construction.

2.2.4 Query Results Exploration and Visualization

Same as VQI design, Query Results Exploration and Visualization will be supported by JavaFX. To have the desired layout, we need to implement an additional flexible panel that enables users to view or switch or even further process the match results.

2.3 Testing

In the testing phase, we perform several tests to check various functions. These tests involve checking a list of prepared questions regarding the functions and the answers to them should be either apparent or quantitative, requiring no subjective opinions.

2.3.1 GUI Functionality

To test the GUI functionality, we need to perform an exhaustive query task that covers all modules and check that the interface is operating properly (e.g., presenting clear results, giving timely feedback) for each module. For example, we can perform both exact and similarity queries on small and large graph databases separately. Then during the query process, we record the answers to a list of questions such as:

1. Can the query graphs be successfully drawn?
2. Are the recommended patterns and the query results successfully and clearly presented?
3. Are the query construction notifications shown clearly and in time?

2.3.2 Handler-Driver Connection

To test the handler-driver connection, we need to perform a query task that involves transactions between the query handler and the Neo4j driver. Then during the query process, we record the answers to a list of questions such as:

1. Is the database successfully loaded?
2. Are the query statements correct and efficient?
3. Are the query results correctly retrieved?

2.3.3 Results Exploration and Visualization

To test the exploration and visualization of the results, we need to perform a query task that involves post-processing and graphical presentation of the query results. Then during the query process, we record the answers to a list of questions such as:

1. Are the unwanted items and properties eliminated?
2. Are the query results sorted in the desired order?
3. Is the structure of the query results correctly preserved and presented through visualization?

2.3.4 Pattern Recommendation Modules

To test the pattern recommendation modules, we need to perform a query task that involves canned pattern recommendation. Then during the query process, we record the answers to a list of questions such as:

1. Is the canned pattern generation process efficient?
2. Is the canned pattern generation process consistent among distinct databases?
3. Are the canned patterns diverse and exhaustive?

2.3.5 Action-Aware Query Processing

To test the action-aware query processing, we need to perform a query task that involves basic query construction. Then during the query process, we record the answers to a list of questions such as:

1. Is the intermediate query result generation efficient?
2. Are enough intermediate query results generated?
3. Is the query indexing well utilized in final query result generation?

2.4 Evaluation

2.4.1 Learnability

Learnability measures whether new users can interact effectively with the VQI and achieve optimal performance. Learnability can be reflected by the easiness of the VQI design, the intuitiveness of the functionalities, and the effectiveness of the constant information exchange. In this project, learnability is extremely crucial because the target users of the application are non-expert users who are not familiar with the database.

2.4.2 Flexibility

Flexibility is reflected in the number of channels through which users exchange information with the system. A flexible system can realize multi-channel information exchange with users so that users can get the information returned by the system in time during use, to standardize their operations and gain more accurate results. In our project, we define the rate of information exchange to measure flexibility. Users will report their feeling about whether they can communicate with the system and receive the returned information conveniently and quickly.

2.4.3 Robustness

Robustness is the level of support provided to a user during the achievement of goals. In the project, we adopt data-driven VQI to provide the set of labels and high-coverage and high-diversity canned patterns so that users can do the top-down and bottom-up searches with low cognitive load. Furthermore, we implement action-aware graph query processing so that users can receive guidance and hints after each step they build the graph. Users will provide feedback on whether they receive adequate and instructive information to support their further query building. Based on the users' feedback, we will decide the further improvement to the project.

2.4.4 Efficiency

Efficiency represents the speed with which a user can perform the tasks after he learns about the system. In this project, efficiency can be measured through the total amount of time or steps saved by using canned patterns or other tools compared to *edge-at-a-time* construction. High efficiency relies on the lucid and easy-to-use user interface design as well as the accuracy of the selection of the canned patterns.

2.4.5 Memorability

Memorability refers to the extent that a user remembers the system's functions after not using it for some time. To achieve high memorability, the functionalities of the VQI must be concise as well as powerful. This examines our skills of concentrating more functions into a single interactive element (e.g., buttons) and providing a natural and memorable arrangement of these elements. The evaluation of memorability will be done through some beta tests.

2.4.6 Errors

Errors represent three criteria: the number of errors made by users, the severity of the errors, and whether these errors can be recovered easily. Users inevitably make some operative errors due to their unfamiliarity or misunderstanding of the system, making it paramount to provide some error-fixing message by either displaying the error or suggesting a remedy. Providing an assembly line of operations, including a capsulated VQI and independent backend modules, is necessary as well because it can avoid severe errors that crash the application or confuse users.

2.4.7 Pleasantness

Pleasantness measures the users' satisfaction with the application when using it. It is more subjective and can be seen as a combined factor of the aforementioned six criteria. Users will give feedback on whether it is enjoyable to work with the system and whether they will continue to use it in the future.

3 Project Planning

3.1 Distribution of Work

Task	LIANG Houdong	YAO Chongchong
Exploration of Neo4j and Cypher	●	○
Literature Survey	●	○
Drawing Panel	●	○
Query Builder	○	●
Query Handler	○	●
Database Loader	○	●
Homogeneous Graph Query Pipeline	○	●
Label and Property Panels	●	○
Heterogeneous Graph Query Pipeline	●	○
Constrained Graph Query Pipeline	○	●
Query Results Processing and Exploration	○	●
Query Results Visualization	●	○
Pattern Recommendation Panel	●	○
Basic Pattern Recommendation Algorithm	○	●
Canned Pattern Recommendation Algorithm	○	●
Subgraph Similarity Query Module	●	○
Query Formation Notification Module	○	●
Responsive Query Formation Guidance Module	○	●
GUI Functionality Testing	●	○
Handler-Driver Connection Testing	○	●
Results Exploration and Visualization Testing	●	○
Pattern Recommendation Modules Testing	○	●
Action-Aware Query Processing Testing	○	●
Proposal	●	○
Monthly Reports	●	○
Progress Report	○	●
Final Report	●	○
Oral Presentation and Demo	●	○
Video Trailer	○	●

● Leader ○ Assistant

3.2 GANTT Chart

Task	July	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May
Exploration of Neo4j and Cypher	■	■									
Literature Survey	■	■	■								
Drawing Panel		■	■								
Query Builder			■	■							
Query Handler			■	■							
Database Loader			■	■							
Homogeneous Graph Query Pipeline				■							
Label and Property Panels				■	■						
Heterogeneous Graph Query Pipeline				■	■						
Constrained Graph Query Pipeline				■	■						
Query Results Processing and Exploration				■	■						
Query Results Visualization				■	■						
Pattern Recommendation Panel						■	■				
Basic Pattern Recommendation Algorithm						■	■				
Canned Pattern Recommendation Algorithm						■	■	■			
Subgraph Similarity Query Module							■	■	■		
Query Formation Notification Module								■	■	■	
Responsive Query Formation Guidance Module								■	■	■	
GUI Functionality Testing									■	■	
Handler-Driver Connection Testing									■	■	
Results Exploration and Visualization Testing									■	■	
Pattern Recommendation Modules Testing									■	■	
Action-Aware Query Processing Testing									■	■	
Proposal			■								
Monthly Reports				■	■		■				
Progress Report								■			
Final Report									■	■	
Oral Presentation and Demo										■	■
Video Trailer											■

4 Required Hardware and Software

4.1 Hardware

Development and Deployment PC:	PC with MS Windows 10 or later

4.2 Software

Java	Programming language
JavaFX	Software Platform
Cypher	Graph query language
IntelliJ IDEA [12]	IDE
Neo4j	Graph query engine

5 References

[1] S. Abiteboul, et al, “The Lowell Database Research Self-Assessment,” in *Commun. ACM*, 2005, vol. 48, no. 5, pp. 111–118.

[2] S. S. Bhowmick, “DB ⋈ HCI: Towards Bridging the ChasmBetween Graph Data Management and HCI,” in *DEXA*, 2014, pp. 1-11.

[3] S. S. Bhowmick and B. Choi, “Data-driven visual query interfaces for graphs: Past, present, and (near) future,” in *SIGMOD*, 2022, pp. 2441-2447.

[4] S. S. Bhowmick, B. Choi, and C. Dyreson, “Data-driven visual graph query interface construction and maintenance: challenges and opportunities,” in *Proc. VLDB*, 2016, vol. 9, no. 12, pp. 984–992.

[5] S. S. Bhowmick, K. Huang, H. E. Chua, Z. Yuan, B. Choi, and S. Zhou, “AURORA: Data-driven construction of visual graph query interfaces for graph databases,” in *SIGMOD*, 2020, pp. 2689–2692.

[6] Cypher. (2022). *Neo4j Inc.*. Accessed: Sept. 11, 2022. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/.

[7] DBpedia. (2022). *DBpedia Association*. Accessed: Sept. 11, 2022. [Online]. Available: https://www.dbpedia.org/.

[8] Emolecules. (2022). *Emolecules*. Accessed: Sept. 11, 2022. [Online]. Available: https://www.emolecules.com/.

[9] K. Huang, H. E. Chua, S. S. Bhowmick, B. Choi, and S. Zhou, “CATAPULT: Data-driven Selection of Canned Patterns for Efficient Visual Graph Query Formulation,” in *SIGMOD*, 2019, pp. 900-917.

[10] K. Huang, S. S. Bhowmick, S. Zhou, and B. Choi, “PICASSO: Exploratory Search of Connected Subgraph Substructures in Graph Databases,” in *Proc. VLDB*, 2017, vol. 10, no. 12, pp. 1861–1864.

[11] InsideBIGDATA. “The Exponential Growth of Data.” Accessed: Sept. 11, 2022. [Online]. Available: https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/.

[12] IntelliJ IDEA. (2022). *JetBrains*. Accessed: Sept. 11, 2022. [Online]. Available: https://www.jetbrains.com/idea/.

[13] Java. (2022). *Oracle*. Accessed: Sept. 11, 2022. [Online]. Available:

https://www.java.com/.

[14] JavaFX. (2022). *JavaFX.* Accessed: Sept. 11, 2022. [Online]. Available: https://openjfx.io/.

[15] C. Jin, S. S. Bhowmick, B. Choi, and S. Zhou, "PRAGUE: Towards Blending Practical Visual Subgraph Query Formulation and Query Processing," in *ICDE*, 2012, pp. 222-233.

[16] C. Jin, S. S. Bhowmick, X. Xiao, J. Cheng, and B. Choi, "GBLENDER: Towards blending visual query formulation and query processing in graph databases," in *SIGMOD*, 2010, pp. 111-122.

[17] Markets and Markets. "Graph Database Market." Accessed: Sept. 11, 2022. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/graph-database-market-126230231.html?gclid=Cj0KCQiAxc6PBhCEARIsAH8Hff1pUb5PI2peZmHQa-AvoPd2MRWXyPwGfEKYFu6I86Z-SgGyQ2a8G88aAmgmEALw_wcB.

[18] Neo4j. (2022). *Neo4j Graph Data Platform.* Accessed: Sept. 11, 2022. [Online]. Available: https://neo4j.com/.

[19] B. Platz. "Why Graph Databases Are an Essential Choice for Master Data Management." DATAVERSITY. Accessed: Sept. 11, 2022. [Online]. Available: https://www.dataversity.net/why-graph-databases-are-an-essential-choice-for-master-data-management/.

[20] PubChem. (2022). *PubChem Sketcher.* Accessed: Sept. 11, 2022. [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/edit3/index.html.

[21] Z. Yuan, H. E. Chua, S. S. Bhowmick, Z. Ye, B. Choi, and W. S. Han, "PLAYPEN: Plug-and-play Visual Graph Query Interfaces for Top-down and Bottom-up Search on Large Networks," in *SIGMOD*, 2022, pp. 2381-2384.

[22] Z. Yuan, H. E. Chua, S. S. Bhowmick, Z. Ye, W. S. Han, and B. Choi, "Towards Plug-and-Play Visual Graph Query Interfaces: Data-driven Canned Pattern Selection for Large Networks," in *Proc. VLDB*, 2021, vol. 14, no. 11, pp. 1979–1991.

[23] J. Zhang, S. S. Bhowmick, H. H. Nguyen, B. Choi, and F. Zhu, "DaVinci: Data-driven visual interface construction for subgraph search in graphdatabases," in *ICDE*, 2015, pp. 1500-1503.

6 Appendix A: Meeting Minutes

6.1 Minutes of the 1st Project Meeting

Date:	June 3, 2022
Time:	7:00 PM
Place:	Online ZOOM Meeting
Present:	YAO Chongchong, LIANG Houdong, HUANG Kai (a postdoc of Prof. Zhou)
Absent:	None
Recorder:	YAO Chongchong

1. Approval of minutes

This was the first formal group meeting, so there were no minutes to approve.

2. Report on progress

2.1 All team members have read the FYP instructions and schedule.

3. Discussion items

3.1 All team members checked the requirements and milestones of FYP and discussed the overall schedule with Dr. HUANG.

3.2 Dr. HUANG addressed the main objectives and focus of the team.

3.3 Dr. HUANG shared some tasks done by others on the same topic and self-learning materials on Neo4j.

4. Goals for the next phase

4.1 All team members will install and learn Neo4j.

6.2 Minutes of the 2nd Project Meeting

Date:	Aug 18, 2022
Time:	7:30 PM
Place:	Online ZOOM Meeting
Present:	YAO Chongchong, LIANG Houdong, HUANG Kai
Absent:	None
Recorder:	LIANG Houdong

1. Approval of minutes

The minutes of the last meeting were approved without amendment.

2. Report on progress

2.1 All team members have studied the basics of Cypher Query Language and the use of Neo4j.

2.2 All team members read the research papers about PICASSO, TATTO systems, and so on.

2.3 All team members watched the demonstration videos about the above systems.

2.4 All team members have started building the application using the official Java driver of Neo4j

3. Discussion items

3.1 Dr. HUANG restated the common classification of the graph. As for large graph databases like interpersonal relationship networks, they are more general and popular in today's research. As for small graph databases like PubChem, they can be seen as a special case of the large graph because the combination of the small graph can be seen as a not completely connected large graph.

3.2 Team members discussed what level of complexity the queries should have. We consider three levels of complexity: only nodes and edges, nodes and relationships with labels and nodes, and relationships with labels and other constraints. Dr. HUANG addressed the problem by suggesting that we should first try to build the most general one because if the general one works, the query with less information can also be done.

3.3 All team members discussed the choice of the platform to build the project and the form of the application. Dr. HUANG suggested that either a web or desktop application is viable and we could choose any platform suitable for the project.

3.4 Dr. HUANG pointed out a potential weakness of the web application. If the large graph is loaded, the website may be lagging or crash.

3.5 Dr. HUANG gave some suggestions about the process to start the process: we should first build the basic line of operation and make it work for the simplest query. Then we can build the more complex functions above the basic layer.

4. Goals for the next phase

4.1 All team members will complete the most basic function of the application: load the database, build the visual patterns for the query, do the query and return the correct results and output the results visually.