XZ2

FYP Final Report

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

Abstract

Visual Graph Query Interfaces (VQIs) empower non-programmers to query graph data by constructing visual queries intuitively. Devising efficient technologies in Graph Query Engines (GQEs) for interactive search and exploration has also been studied for years. However, these two vibrant scientific fields are traditionally independent of each other, causing a vast barrier for users who wish to explore the full-stack operations of graph querying. In this final year project, we propose a novel VQI system built upon Neo4j called VisualNeo that facilities an efficient subgraph query in large graph databases. VisualNeo inherits several advanced features from recent advanced VQIs, which include the data-driven GUI design and canned pattern generation. Additionally, it embodies a database manager module in order that users can connect to generic Neo4j databases. It performs query processing through the Neo4j driver and provides an aesthetic query result exploration.

Table of Contents

1 Introduction

1.1 Overview

In recent years, the exponential growth of data has presented significant challenges for data storage, management, and accessibility [14]. As a result, databases have become increasingly essential due to their capability to store and manage vast amounts of data while ensuring data security. For any company or organization, a well-performing database is indispensable because it stores vital information such as employee records, transactional records, salary details, and more. There are several types of databases available, including relational databases, graph databases, and cloud databases, each with unique strengths in storing and managing data. Graph databases have become increasingly important, particularly as data relationships become more critical than individual data points [22].

A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store data. The advantage of using graph databases is that the relationships between data points are directly stored, making it convenient to query them. Many real-life scenarios can be modeled as a graph database, such as social networks (e.g., Twitter), where the connections between people are crucial. Each chemical molecule can also be represented as a small graph, and molecules can be combined to form a large graph database, as is the case with PubChem [24]. Co-purchase networks (e.g., Amazon.com) and information networks (e.g., DBpedia [7]) are other examples. According to a recent *Markets and Markets* report [20], the global graph database market size is predicted to increase from USD 1.9 billion in 2021 to USD 5.1 billion by 2026, indicating a significant demand for the research and analysis of graph data.

Unfortunately, although there is an increasing need for 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 a Query Language (QL) and a Query Engine (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 [21] 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, a Visual Query Interface (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, is now a feasible solution. With the help of a VQI, non-programmer users do not need to learn Cypher and Neo4j and can make queries by inputting vertices and edges graphically, just like using some painting software. This can greatly reduce the difficulty for non-professionals to learn how to make queries and save them a lot of time.

1.2 Objectives

1.2.1 Provide a Visual Interface for Graph Querying

The goal of this project was to design and implement a Visual Query Interface that is compatible with Neo4j, called VisualNeo, to help non-expert users make graph queries. VisualNeo includes many cutting-edge and useful functions, such as data-driven VQI design, top-k diversified patterns generation, action-aware graph query processing, and effective query result visualization, which are novel features of recently published VQIs.

The bottlenecks of this project included 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.

1.2.2 Enable Data-driven Design and Pattern Generation

The construction of our 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 queries.

To implement it, we introduce a set of diversified patterns called TED patterns (i.e., Top-k Edge-Diversified patterns) [13], which summarize the characteristics of the underlying database and thus facilitate the query formulation. In particular, TED patterns have a theoretical guarantee of the edge coverage approximation ratio, and its generation process requires limited memory.

1.2.3 Provide Action-aware Query Processing

Action-aware graph query processing requires a responsive and clear design that returns the query results after each step users build the query instead of only after users click the “Run”

button. We adopted 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. This guides users during the query construction process. VisualNeo also provides a real-time query translation module, by which users can know the formal Cypher code of the current graphical query.

1.2.4 Provide Result Exploration and Visualization

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, an improper or user-unfriendly display of the panels on the interface may cause problems. Therefore, aesthetics-aware VQI design was another challenge in our project.

To address this problem, we reformulated the data-driven visual layout design problem as an optimization problem. The goal was to find an optimal layout that minimizes query formulation complexity and the visual complexity of the interface.

1.3 Literature Survey

1.3.1 Combination of Graph Data Management and HCI

Traditionally, devising efficient techniques in a GQE for interactive searching 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, HCI researchers are not willing to face the challenges in the data management domain, even though it may provide them with new ways to build visual interfaces.

In recent decades, the chasm between traditional graph data management and HCI has been bridged. In 2014, an HCI-aware visual graph querying framework was proposed [2]. Figure 1 depicts the generic architecture of this 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.

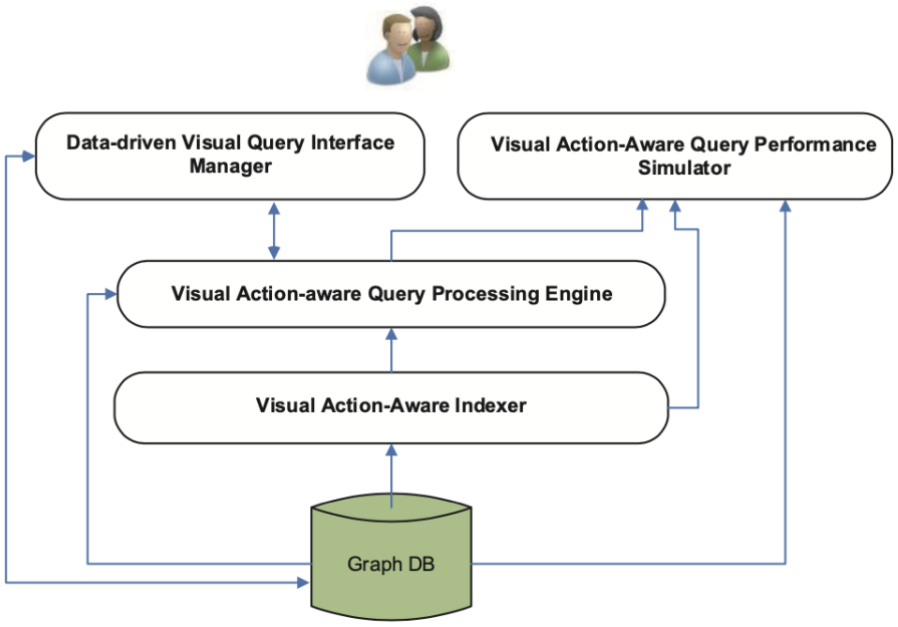

Figure 1. The architecture of an HCI-aware visual graph query framework [2].

1.3.2 Data-driven Visual Query Interface Design

Current VQIs such as PubChem [24] (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.

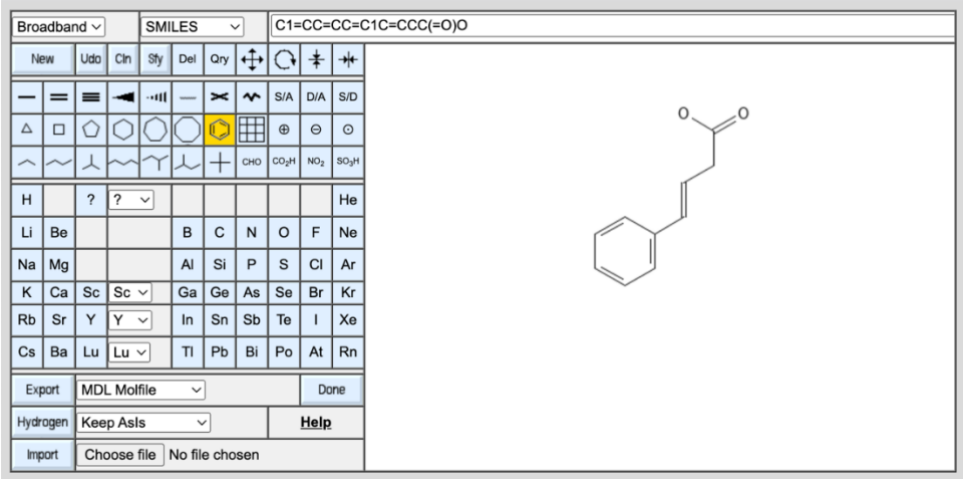

Figure 2. GUI for substructure search in PubChem [24].

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. This has two drawbacks. First, 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. Second, the frequent update of the underlying database may lead to obsolete predefined patterns and the emergence of new patterns. Lastly, 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, AURORA provides a novel way of data-driven construction of a 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 [26] and DAVINCI [28] similarly propose corresponding data-driven VQI designs. 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, can effectively facilitate query formulation by enabling *pattern-at-a-time* construction. However, the traditional approach of manually selecting patterns based on domain knowledge is not only laborious but also unable to cover a broad range of subgraph queries. To address this issue, a research group proposed a generic and extensible framework named TATTOO for automatically generating and selecting canned patterns from large networks [27]. This framework 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 while minimizing the cognitive load of users. Another data-driven framework for canned pattern selection called CATAPULT, illustrated in Figure 3, follows a similar design [11]. CATAPULT clusters the data graphs based on their topological similarities and then summarizes each cluster to create a cluster summary graph (CSG). The canned patterns are generated from these CSGs using the same criteria as TATTOO [27].

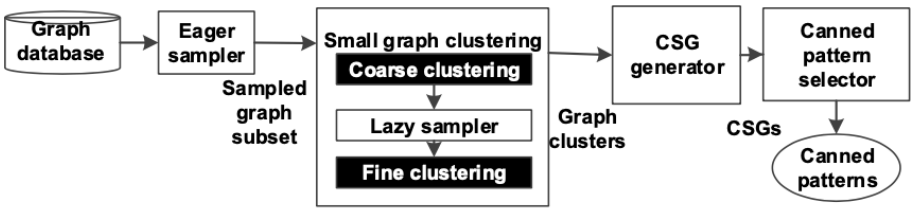

Figure 3. The Framework of CATAPULT [11, Fig. 3].

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 in 2010 [19]. Instead of only processing a graph query after the user clicks the "run" button, it handles the incomplete query after each step when 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 it 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 [18]. PRAGUE fixes the problems by using a different data structure called a spindle-shaped graph (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 shows the architecture of VisualNeo. It consists of five modules, Database Manager, Pattern Recommender, Query Constructor, Query Handler, and Result Explorer.

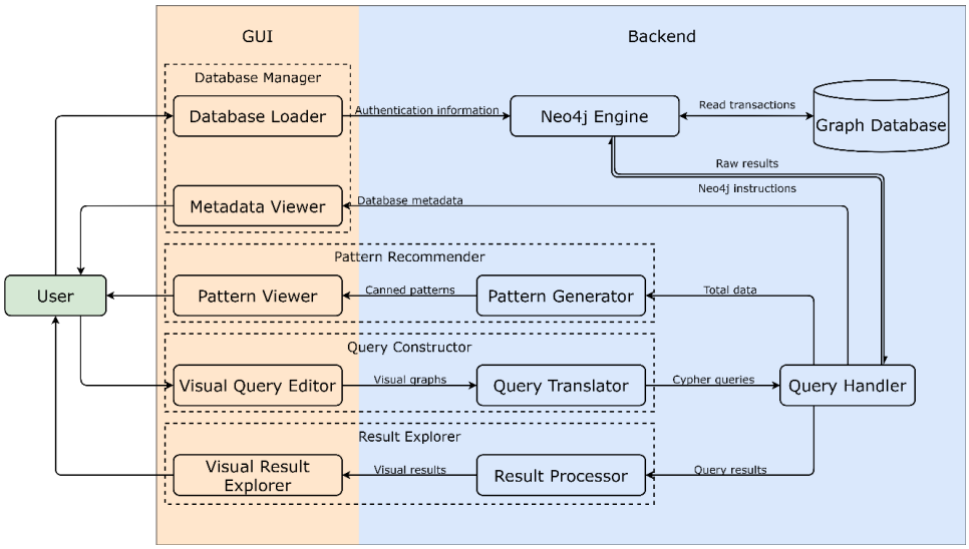

Figure 4. The architecture of VisualNeo.

The Database Manager module first establishes a connection to a graph database server with user authentications and displays its metadata obtained by the Query Handler module. The Query Handler module also exports the whole database such that the Pattern Recommender module can then utilize METIS [10] (for graph partitioning) and TED [13] (for pattern generation) to produce diversified and high-coverage patterns. With the aid of these two modules, the Query Constructor module enables the user to form visual query graphs effortlessly. To integrate with the GQE (i.e., Neo4j), the query graphs are further translated into formal query languages and fed to the Query Handler module. Next, the Query Handler module

instructs the GQE to execute read transactions and retrieve the desired results. Finally, the Result Explorer module converts the unprocessed results into navigable visual graphs for the user to investigate. More details are discussed below.

2.1.2 Database Manager

This module establishes a connection to a graph database server and displays its metadata. When the user clicks the "Load Database" button shown in Figure 5 and specifies the authentication information, the Database Manager module sends a connection request to the server. To achieve high universality, this module is compatible with all standard Neo4j servers and only requires read authority.

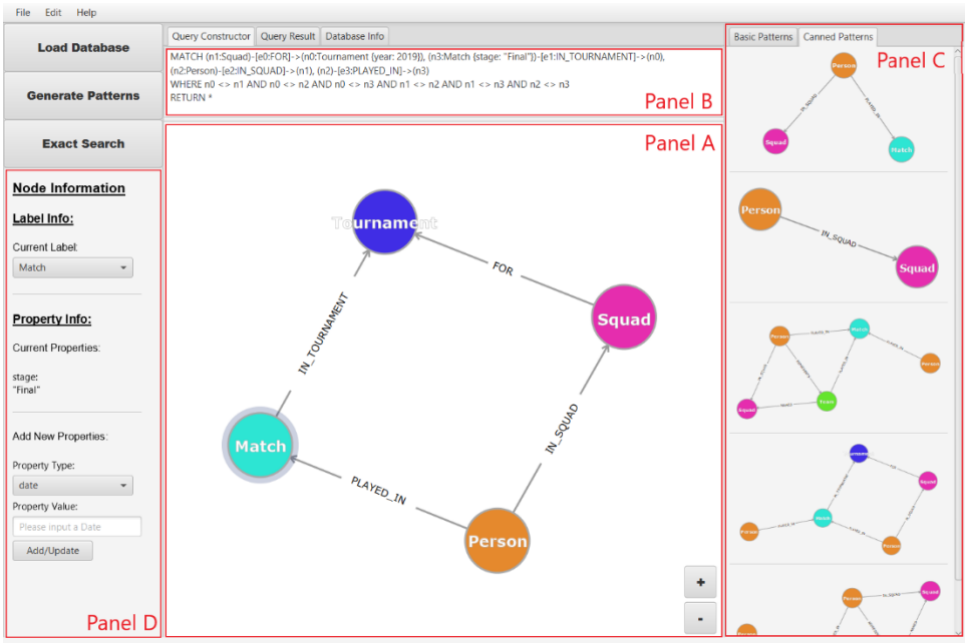

Figure 5. The Query Constructor panel.

After connection, the Query Handler module automatically executes a sequence of queries to retrieve database metadata. The metadata is then displayed in the Database Information panel (Panel G) shown in Figure 6 and is also used to facilitate query formulation by providing label constraints and property constraints in the Query Constructor module.

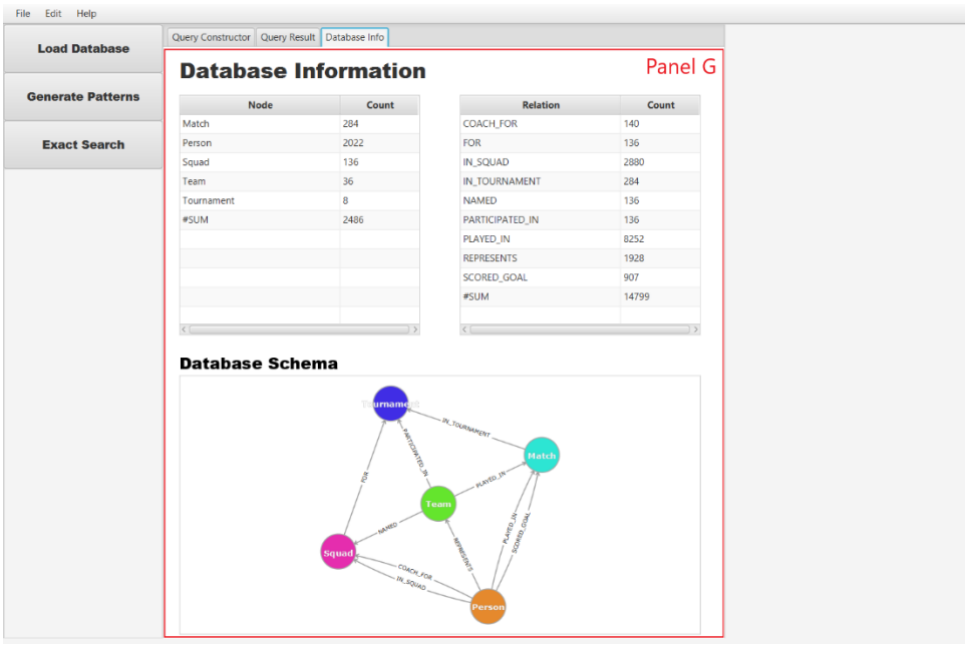

Figure 6. The Database Info panel.

2.1.3 Pattern Recommender

This module generates TED patterns and populates the canned pattern panel when the user clicks the “Generate Pattern” button shown in Figure 5 and specifies the constraints of desired patterns. These patterns should maximally cover the characteristics of the graph database while preserving a high degree of diversity so that they can later facilitate the query construction for the user.

2.1.4 Query Constructor

This module provides the user with a Visual Query Editor for visual query formulation and translates the visual graphs into formal Cypher queries. It has four components as shown in Figure 5, the Graph Editor panel (Panel A), the Query View panel (Panel B), the Pattern View panel (Panel C), and the Element Constraint panel (Panel D). By interacting with this editor, the user can construct his desired queries, receive query formulation responses, and perform exact or similarity searching. Detailed operations are discussed in the implementation part.

2.1.5 Query Handler

This module executes Cypher queries and processes query results. It acts as an agent between the Neo4j driver and the VQI and is responsible for executing all Neo4j instructions.

2.1.6 Result Explorer

This module processes query results and displays them in an aesthetic and navigable manner. As shown in Figure 7, the Result Explorer has two components: Panel E displays a large graph composing all results, and Panel F shows the list of matching records.

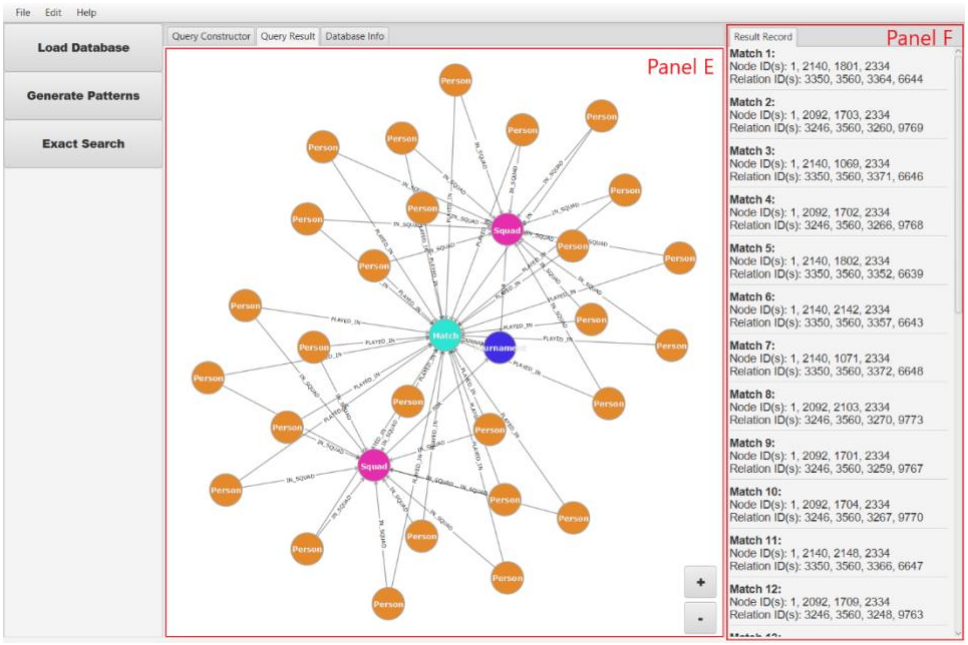

Figure 7. The Query Result panel.

Similar to the Query Constructor module, the user may navigate through the result graph by dragging and scrolling. Additionally, Panel E immediately navigates to the corresponding pattern and highlights it when a user clicks an item in Panel F, which considerably reduces the effort for finding a specific match in a large graph.

2.2 Implementation

2.2.1 Database Manager

To assist query construction with action-aware responses, we selected a set of metadata queried after each database connection, including node/relationship counts, node/relationship labels, node/relationship property keys & data types, and the schema graph that defines the topology among different classes of nodes and relationships. These queries either directly access the Neo4j count store or invoke database procedures. Therefore, such searches have $O(1)$ time complexity for each class/property, which ensures fast initialization regardless of the database size.

2.2.2 Pattern Recommender

Most existing TED pattern generation techniques are dedicated to small-graph databases, where a database consists of numerous small graphs [13]. However, our target databases are generally large. Therefore, we divided the Pattern Recommender module into two parts: METIS graph partitioning [10] and TED pattern generation [13]. First, METIS partitions the large network into a large collection of smaller medium-sized graphs (e.g., tens of nodes per graph). Next, the TED algorithm greedily searches for a set of patterns with maximum edge coverage. Given a set of partitions $D = \{G_1, G_2, \dots G_n\}$ and an integer k, the TED algorithm first enumerates all 1-sized subgraphs (i.e., edges) and appends them into the set $\mathcal{P}$. Then, an iterative process is performed to enumerate τ-sized ($\tau \geq 2$) subgraphs by performing the right-most extension [25] and appending them into the set $\mathcal{P}$. When the size of $\mathcal{P}$ is k and a new subgraph g is generated, a swapping-based process is developed to determine if g should be swapped into $\mathcal{P}$. In particular, it first calculates the loss score (i.e., decreased coverage if p is swapped out) for $p \in \mathcal{P}$ and then records the pattern p_t and its pattern score S_L such that p_t has the minimum loss score. The benefit score (i.e., increased coverage if g is swapped in) S_B of g is also recorded. The subgraph

g is considered a promising candidate and swapped into $\mathcal{P}$ if $S_B > (1+\alpha)S_L + (1-\alpha)|Cov(\mathcal{P}, D)|/k$ is satisfied, where $\alpha \in [0,1]$ is a swapping threshold for balancing the loss score S_L and the average coverage of patterns in $\mathcal{P}$. The approximation ratio (w.r.t., total coverage) of patterns $\mathcal{P}$ is bounded by $|Cov(\mathcal{P}, D)|/\left|Cov\left(\mathcal{P}_{opt}, D\right)\right| \geq 1/4$, where $\mathcal{P}_{opt}$ is the optimal solution, which can be obtained by enumerating all subgraphs and generating all possible combinations of k subgraphs.

Both METIS and TED have been proven to be efficient so we can safely integrate them into VisualNeo and perform pattern generation on the fly. In our case, pattern generation generally takes less than one second and does not noticeably slow query construction.

2.2.3 Query Constructor

To formulate a graph query, the user can choose from one of 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. In our project, we first implemented the edge-at-a-time approach, so that the user can build the target graph query step-by-step effortlessly. After that, we built an accurate and efficient pattern generation algorithm that could considerably boost the speed of query construction. The detailed interaction rules for the query editor are given below.

1. The user can select/add/remove elements in Panel A through simple mouse-keyboard operations. He can navigate through the panel by dragging or adjusting the zoom by scrolling.
2. Panel B displays a real-time translation from the visual graph in Panel A to the Cypher query to aid the query formulation. The MATCH clause is translated relationship-wise for clearness. Furthermore, since Cypher makes use of relationship isomorphism for path matching where

the same relationship appears at most once for each matching record while the same node possibly has multiple matches, the WHERE clause consists of pair-wise inequalities among nodes to ensure that Neo4j only searches for the exact query graph.

3. Panel C displays the canned patterns generated by the Pattern Recommender module and several basic patterns applicable to universal databases. The user can use these patterns in their query construction through simple drag-over operations.
4. After selecting an element, the user can view and edit its label and properties in Panel D.
5. When the user finishes the graph, he can click the "Exact Search" button in the Query Constructor Panel (Figure 5), and then the Query Constructor module sends the translated Cypher query to the Query Handler module.

In the process of building a visual query, the resulting query is automatically translated into a formal Cypher query by traversing all relevant relationships. This entails generating a corresponding line in the MATCH clause for each relationship in the form of (startNode)-[thisRelation]-(endNode). Nevertheless, this approach is prone to isomorphic issues, whereby the same node or relationship may be returned multiple times for each matching record. For instance, if a user creates a visual query in the form of (n2)-[r1]-(n1)-[r2]-(n3) (Pattern A in Figure 8), it may result in a mismatch with Patterns B or C if the translated query lacks appropriate isomorphic constraints.

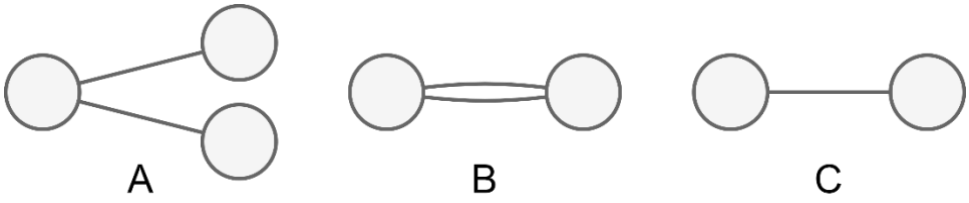

Figure 8. Indistinguishable patterns without node isomorphism (A and B) or relationship isomorphism (A and C).

To avoid this issue, additional measures are necessary to ensure accurate and reliable query

translation. Although Neo4j Cypher employs relationship isomorphism to prevent the same relationship from being returned multiple times within a single result record, it does not guarantee node isomorphism. While this matching mechanism may assist programmers in managing complex queries, it can be misleading for inexperienced users and result in queries that match unexpected patterns, (e.g., Pattern B in Figure 8 is considered a valid match for Pattern A). Moreover, most existing Cypher-oriented VQI (e.g., Popoto.js [23]) do not handle node isomorphism because they do not connect to a GQE as a backend. To address this issue, we introduce additional inequality constraints on each node pair in the WHERE clause to guarantee node isomorphism. Moreover, to improve query efficiency, we eliminate trivial inequalities before incorporating them into the constraints. For instance, if two nodes have different labels or properties, the corresponding inequality can be removed.

2.2.4 Query Handler

This module's features include executing Cypher queries via the Neo4j driver, receiving query results, extracting desired data from returned records, converting raw data into Java-typed data, and boxing data into proper containers for further processing. To ensure the validity of our instructions and optimize their performance, we exploited the Neo4j driver to make all sessions and transactions read-only to avoid modification and undesired authentication errors. Besides, in composite queries (e.g., metadata queries), consecutive transactions are bundled into a single session to enhance efficiency.

2.2.5 Result Explorer

To visualize query results aesthetically and quickly, we adopted several techniques for both result processing and graph arrangement.

In cases where the query involves nodes with high centrality or relationships with high

betweenness, the result likely contains duplicate elements. This occurrence of redundant information can cause high data transfer traffic and memory costs at the local device. To avoid such inefficiencies, we generate an ID reference list with Cypher for each query and only keep the information of distinct elements. This approach ensures that the final result is devoid of any information loss, and simultaneously prevents unnecessary data transfer or memory usage at the local device.

To arrange the results in an orderly manner, we adopt the Fruchterman-Reingold (FR) force-directed graph layout algorithm [9] with slight modifications:

1. In contrast to the simple graphs used in the original FR algorithm, our subject graphs are in general multigraphs which may have self-loops and multiple relationships between the same pair of nodes. Therefore, we always prune self-loops and keep at most one relationship between each (unordered) pair of nodes before the force simulation to avoid unnecessary computations and excessive proximity between heavily connected nodes.
2. Given the navigable and zoomable properties of Panel E, we specify a constant optimal distance between connected node pairs and removed boundaries around the graph, which reduces the computation cost per iteration and produces better visual effects.
3. As we no longer restrict nodes with boundaries, we set a maximum distance on repulsive forces so that disjoint subgraphs do not repel each other to infinity. We also center the centroid of all nodes at the origin after the force simulation to cancel the universal offset. Since we only compute the centroid once for each graph, the additional $O(|V|)$ computation cost is negligible compared to the overall $O\big(K(|V|^2 + |E|)\big)$ time complexity, where $G = (V, E)$ is the graph and K is the number of simulation iterations.

2.3 Testing

We performed several tests to check all the functions. These tests involved checking a list of prepared questions regarding the functions, and the answers to them were either apparent or quantitative, requiring no subjective opinions. The test results can be found in Appendix D: Testing Cases.

2.3.1 Database Manager

To test the database manager, we loaded a few remote Neo4j databases (e.g., Twitch and Women's World Cup 2019). After the loading, we checked whether there is a successful connection between the database and Neo4j driver and whether relevant data is displayed on our interface. During the testing process, we recorded the answers to a list of questions such as:

1. Is the database successfully loaded?
2. Is the metadata of the database correctly displayed on our database metadata panel?
3. Can the user add label and property constraints to nodes and relationships during the query construction process?

2.3.2 Pattern Recommender

To test the pattern recommender, we performed a query task that utilized TED patterns. During the query process, we recorded the answers to a list of questions such as:

1. Is the TED pattern generation process efficient?
2. Are the generated TED patterns successfully and clearly presented in the pattern panel?
3. Are the generated TED patterns diverse and exhaustive enough to facilitate the query construction process?

4. Can the user drag and drop the TED patterns to the drawing board during the construction process?

2.3.3 Query Constructor

To test the query constructor, which is equivalent to the GUI functionality, we performed an exhaustive query task that covers all types of query constructions. For example, we performed both *edge-at-a-time* construction methods and pattern-at-a-time construction methods. Furthermore, we tried different combinations of label and property constraints. Then during the construction process, we recorded the answers to a list of questions such as:

1. Can the query graphs be successfully drawn?
2. Are the query construction notifications shown clearly and on time?
3. Is the construction panel user-friendly enough so that the user can build the query graph easily?

2.3.4 Query Handler

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

1. Are the translated query statements correct and efficient?
2. Are the query statements successfully passed to and processed by the Neo4j driver?
3. Are the query results correctly retrieved?

2.3.5 Result Explorer

To test the exploration and visualization of the results, we performed a query task that involves

post-processing and a 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 retrieved query results successfully displayed in the result panel?
2. Are the query results sorted in the desired order?
3. Is the structure of the query results correctly preserved and presented through visualization?

2.4 Evaluation

To assess the effectiveness of our VisualNeo system, we conducted a survey by creating a questionnaire and inviting a group of clients to use our system and provide feedback. The questionnaire can be found in Appendix E: Questionnaire, and the results are presented in Appendix F: Evaluation Results. These results provide valuable insights into the user experience and performance of our system. The questionnaire focused on several key metrics and criteria, including:

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. For our VQI, 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. For our VQI, we define the rate of information exchange to measure flexibility. Users will report their feelings 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. Our VQI is data-driven and it provides a set of labels and high-coverage and high-diversity TED patterns so that users can do top-down and bottom-up searches with low cognitive load. Furthermore, we implemented action-aware graph query processing so that users can receive guidance and hints after each step they build the graph. Our volunteer testers provided feedback on whether they received adequate and instructive information to support their further query construction. Based on their feedback, we were able to make some further improvements to the system.

2.4.4 Efficiency

Efficiency represents the speed with which a user can perform the tasks after he learns about the system. In our VQI, 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 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 means that we had to examine 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 was 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 prevent 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 would like to continue to use it in the future.

3 Discussion

The primary goal of our project was to develop VisualNeo, a data-driven visual query system built on the graph query engine Neo4j. Despite successfully incorporating several cutting-edge features such as action-aware query processing and TED patterns, we encountered various challenges during the development process. In the following sections, we will discuss these challenges in detail and explain our efforts to overcome them. It is important to note that not all issues were successfully addressed and some challenges require further research and effort to resolve.

3.1 Node Isomorphism in Graphs without Self-Loops

In Section 2.2.3, we proposed a method to ensure node isomorphism of queries by adding additional inequality constraints in the WHERE clause. To improve query efficiency, we restricted the application of inequality constraints to node pairs that are undistinguishable, i.e., those that could potentially match the same node in the database. We define two nodes in a query to be undistinguishable if, for the label and each property, either the label/property is not specified for at least one of the nodes, or both nodes have the same value for the label/property. For instance, a node labeled “People” and a node with no label specification are possible to match the same node in the database, but two nodes specifying the age property as 20 and 24 separately are guaranteed to match different nodes in the database. Moreover, such constraints cannot be further optimized under general circumstances, because any undistinguishable node pair without an inequality constraint fails to avoid matching the trivial pattern that comprises a single node with a self-loop attached to it.

While this method is necessary for ensuring node isomorphism in the general case, we explored the possibility of further reducing the number of inequality constraints by considering the absence of self-loops in most graph databases, since self-loops can generally be replaced by node properties and usually increase the amount of work involved in data processing. Subsequently, we discovered that undistinguishable node pairs with certain asymmetric local topologies are also guaranteed to match different nodes in the database, so we can further remove their inequality constraints. Unfortunately, identifying these topologies requires solving the subgraph isomorphism problem, which currently lacks efficient solutions. Thus, we had to abandon this approach.

3.2 Advanced Constrained Query

An area where VisualNeo could benefit from further development is in its support for advanced constrained queries. At present, VisualNeo can handle general queries with labels and properties. However, its capabilities are not without limitations. For example, while some databases allow a node to have multiple labels concurrently, VisualNeo operates under the assumption that a node can have only one label at most. Moreover, real Neo4j databases have a total of 11 property types: Integer, Float, String, Boolean, Point, Date, Time, LocalTime, DateTime, LocalDateTime, and Duration. Composite types include List and Map. Not all of these types have convenient conversion functions in Java. For this reason, we do not support all property constraints, particularly for complex types such as Lists. Additionally, mandating that users input Time and Duration using strict syntax rules is not user-friendly. Therefore, we need to devise a more intuitive and straightforward method for users to input these constraints. This is an area that requires further attention and development.

3.3 Performance with Larger Datasets

One challenge we encountered while developing VisualNeo was its performance with larger datasets. At present, the database size is limited to millions of records, which is adequate for most academic projects but may fall short for business applications. Furthermore, there are limited performance optimizations in place to expedite query processing. Consequently, if the database is too large or if there are excessive query results, the query may take an extended amount of time to execute and the system may become unresponsive. To address this issue, we attempted to enhance performance in two ways. Firstly, by optimizing the query string and removing redundant constraints. Secondly, by offering guidance to users when constructing the query graph through real-time query results and an ER diagram of the underlying database. Despite these efforts, much more work is required to improve the system's performance when databases reach the size of billions.

4 Conclusions

4.1 Technical Accomplishments

In this final year project, we present a data-driven visual subgraph query system called VisualNeo. The system is built upon Neo4j, a popular graph query engine. It possesses a Database Manager module with which users can connect to local or remote Neo4j graph databases by providing authentication information. It supports generic Cypher query processing via Neo4j driver, which includes nodes, relationships, labels, and properties. Therefore, VisualNeo builds a bridge between VQI and GQE, introducing a new direction for these two scientific fields.

VisualNeo also inherits several state-of-the-art features from recent data-driven VQI designs. First, it generates a set of diversified patterns called TED patterns (i.e., Top-k Edge-Diversified patterns), which summarizes the characteristics of the underlying database and thus facilitate the query formulation. In particular, TED patterns have a theoretical guarantee of the edge coverage approximation ratio, and its generation process requires limited memory. Second, it embodies an aesthetic query results explorer which adopts the Fruchterman-Reingold algorithm to display the retrieved results.

VisualNeo embraces innovative features as well. To ensure user-friendliness, VisualNeo provides adequate support during the query formulation process. It displays metadata information of the underlying database and provides real-time query translation to guide users to perform exploratory searches where users are unsure about their initial goals or ways to achieve their goals.

4.2 Future Work

4.2.1 Similarity Search

One potential area for future improvement of VisualNeo is to incorporate similarity search functionality. This would involve developing graph metrics to measure the similarity between two graphs and integrating these metrics into the query processing pipeline. Similarity search enables users to find patterns that are similar to their composed query graph, which is important when they do not know the precise patterns they are looking for. To implement similarity search, VisualNeo needs to support more advanced algorithms and data structures, as well as efficient query processing techniques. This would be a valuable addition to the system, enhancing its ability to support exploratory data analysis and knowledge discovery tasks.

4.2.2 Real-time Query Result Feedback and Query Construction Advice

Another potential improvement for VisualNeo is the inclusion of real-time query result feedback and query construction advice. This feature would allow users to receive immediate feedback on the number of query results after each step of building the visual query. This would enable users to stop if there are no matched records and gain a better understanding of the underlying database. Additionally, VisualNeo could provide suggestions for constructing more effective queries based on the user's current query graph. For example, it may suggest adding an additional node in a specific position. This would provide significant assistance to non-expert users in completing their tasks and further enhance the user-friendliness of VisualNeo.

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] Fruchterman, Thomas MJ and Reingold, Edward M. Graph drawing by force-directed placement. Software: Practice and experience, 1991.

[10] Karypis, George and Kumar, Vipin. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[11] 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.

[12] 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.

[13] K. Huang, H. Hu, Q. Ye, K. Tian, B. Zheng, X. Zhou, “TED: Towards Discovering Top-k Edge-Diversified Patterns in a Graph Database,” in *SIGMOD*, 2023.

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

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

[16] Java. (2022). *Oracle*. Accessed: Sept. 11, 2022. [Online]. Available: https://www.java.com/.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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/.

[23] Popoto.js. (2022) *NHOGS Interactive.* Accessed: Sept. 11, 2022. [Online]. Available: https://popotojs.com.

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

[25] X. Yan and J. Han, “Gspan: Graph-based substructure pattern mining,” in *ICDM*, 2002.

[26] 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.

[27] 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.

[28] 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, make queries and return the correct results, and output the results visually.

6.3 Minutes of the 3rd Project Meeting

Date: Oct 3, 2022
Time: 10:15 AM
Place: Room 3208A
Present: Yao Chongchong, Liang Houdong, Prof. Zhou, Dr. Huang Kai
Absent: None
Recorder: Yao Chongchong

1. Report on progress

1.1 All members presented the current project to Prof. Zhou and Dr. Huang. Liang Houdong introduced pattern construction and database loading, and Yao Chongchong introduced metadata fetching and exact searching.

2. Discussion items

2.1 Prof. Zhou gave suggestions for project demonstration on how to make the project functions easy to understand. He also suggested that we should insist on a simple graph database (namely the "Movies" database) for the presentation.

2.2 Prof. Zhou suggested that we should perform explorations on the database right after the connection and present information about the database to help users learn about the database and facilitate their query construction.

2.3 Yao Chongchong posted a question on what data records to return and how to properly present the query results to the user. Prof. Zhou suggested that in the case of small databases like "Movies", we can show the whole database and highlight the matching nodes in the database. Dr. Huang suggested that for large databases, we can expand the matching pattern outwards for one relationship and show the node IDs.

2.4 All members posted a question about how to connect to a local database and how to load other types of data (e.g. csv, txt). Dr. Huang suggested that we can build a custom interface for handling different types of data and we should put constraints on the format of data files. He also suggested that we can utilize the import function provided by Neo4j.

3. Goals for the next phase

3.1 All members will explore the import function of Neo4j and figure out how to connect to a local database and how to load different types of data files.

3.2 All members will decide the types of returned results and implement the query result presentation.

6.4 Minutes of the 4th Project Meeting

Date: Oct. 28, 2022
Time: 3:30 PM
Place: Room 3208A
Present: Yao Chongchong, Liang Houdong, Prof. Zhou, Dr. Huang Kai
Absent: None
Recorder: LIANG Houdong

1. Report on progress

1.1 All members presented the current project to Prof. Zhou and Dr. Huang. Liang Houdong introduced new frontend features including a new method to create and link nodes, a metadata panel to display the table information, and a new layout of edges and their labels. Yao Chongchong introduced metadata fetching, and new debugging information and demonstrated the formal Cypher query generated by exact searching.

2. Discussion items

2.1 Prof. Zhou gave suggestions on the database loading and metadata fetching functions that the time should be restricted to around 2 seconds.

2.2 Yao Chongchong posted a question on the query formation that Neo4j may mismatch the user's input with unwanted patterns due to its inability to recognize different nodes in the graphical structure and thus we need to eliminate these cases manually. The algorithm to do this could be an additional topic of this project.

2.3 In response to Chongchong's question, Dr. Huang Kai advised that we could research some existing methods and see whether this is a convenient or predefined way to do it.

2.4 Dr. Huang Kai suggested that we could first complete the pipeline of all operations so that users could use the application to do some basic queries.

3. Goals for the next phase

3.1 Implement the query result processing and visualization function and complete the pipeline of all operations.

3.2 Implement a parser to convert the Cypher code to the corresponding graphical structure in the drawing panel.

3.3 Start researching the pattern recommendation algorithms.

3.4 Reduce the time of loading the database and fetching the metadata.

6.5 Minutes of the 5th Project Meeting

Date: Nov. 25, 2022
Time: 4:30 PM
Place: Room 3208A
Present: Yao Chongchong, Liang Houdong, Prof. Zhou, Dr. Huang Kai
Absent: None
Recorder: YAO Chongchong

1. Report on progress

1.1 All members presented the project progress to Prof. Zhou and Dr. Huang. Yao Chongchong introduced the new canvas and camera classes with the updated zoom function and Liang Houdong introduced the new user-friendly control system for query construction.

2. Discussion items

2.1 Dr. Huang briefly taught team members some knowledge about canned pattern generation, including 1) the maximal edge-coverage criteria for pattern assessment, 2) the pattern-generating algorithms on data streams, and 3) the method of mapping large graphs to blocks of small graphs.

2.2 Dr. Huang instructed team members on future objectives and suggested that they can submit a demo paper on visualized graph query construction with Neo4j.

2.3 Prof. Zhou reminded team members to try to keep on track and stick to the schedule.

3. Goals for the next phase

3.1 All members will continue completing the overall query pipeline.

3.2 All members will implement the function of parsing Cypher query statements to visual graphs.

3.3 All members will explore canned pattern generation and start working on that topic.

6.6 Minutes of the 6th Project Meeting

Date: Dec 23, 2022
Time: 3:30 PM
Place: Room 3208A
Present: Yao Chongchong, Liang Houdong, Dr. Huang Kai
Absent: None
Recorder: LIANG Houdong

1. Report on progress

1.1 All team members have presented the project progress to Dr. HUANG.

2. Discussion items

2.1 Dr. HUANG illustrated the main parts of the demo paper we are going to write: In the introduction section, we could introduce concepts such as graph and subgraph query; In the system architecture section, we could make a diagram of our system and highlight our main functions such as query constructor, query mapper, query result explorer, and pattern generation. In the related systems and novelty section, we could discuss some related systems and explain their shortcomings and show how our system overcomes these shortcomings and becomes novel. In the demonstration overview section, we could focus on the users' experience when using our system. We could set up some scenarios and explain what users could explore.

2.2 Team members discussed the pattern generation issue. Dr. HUANG introduced the METIS module to us, which could partition the graph easily by cutting the large graph with minimum loss. We could partition the graph offline and apply the result directly to our system. The hyperparameters are user-specific and can be set to fixed.

2.3 Team members discussed the completion of the pipeline, which is the query processing and result exploration part. Dr. HUANG suggests we use the same way as our graph query formulation module. As for the query processing, the driver can fulfill all the requirements.

2.4 Dr. HUANG gave suggestions on our future progress: we should complete the result exploration function and pattern generation function. After that, we can start working on the demo paper.

3. Goals for the next phase

3.1 All members will continue completing the overall query pipeline.

3.2 All members will implement the function of parsing Cypher query statements to visual graphs.

3.3 All members will explore canned pattern generation and start working on that topic.

7 Appendix B: Project Planning

7.1 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 Result Processing and Exploration	○	●
Query Result 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	○	●
Result 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

7.2 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 Result Processing and Exploration											
Query Result 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											
Result 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											

8 Appendix C: Required Hardware and Software

8.1 Hardware

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

8.2 Software

Java [16]	Programming language
JavaFX [17]	Software Platform
Cypher	Graph query language
IntelliJ IDEA [15]	IDE
Neo4j	Graph query engine

9 Appendix D: Testing Cases

Module	Test Case	Pass/Not Pass
Database Manager	Can the database be successfully loaded?	Pass
	Can the metadata of the database be correctly displayed on our database metadata panel?	Pass
	Can the user add label and property constraints to nodes and relationships during the query construction process?	Pass
Pattern Recommender	Is the TED pattern generation process efficient?	Pass
	Can the generated TED patterns be successfully and clearly presented in the pattern panel?	Pass
	Are the generated TED patterns diverse and exhaustive enough to facilitate the query construction process?	Pass
	Can the user drag and drop the TED patterns to the drawing board during the construction process?	Pass
Query Constructor	Can the query graphs be successfully drawn?	Pass
	Can the query construction notifications be shown clearly and in time?	Pass
	Can the construction panel be user-friendly enough so that the user can build the query graph easily?	Pass
Query Handler	Are the translated query statements correct and efficient?	Pass
	Are the query statements successfully passed to and processed by the Neo4j driver?	Pass
	Are the query results correctly retrieved?	Pass
Result Explorer	Can the retrieved query results be successfully displayed in the result panel?	Pass
	Can the query results be sorted in the desired order?	Pass
	Can the structure of the query results be correctly preserved and presented through visualization?	Pass

10 Appendix E: Questionnaire

No.	Questions
0	Please rate the extent to which you agree/disagree with the following: 1 – Disagree, 2 – Somewhat disagree, 3 – Neither agree nor disagree, 4 – Somewhat agree, 5 – Agree
1	I can interact effectively with the VQI and complete the query process (database loading, query construction, result visualization) smoothly.
2	I can receive adequate support from database panels, TED patterns and query translator during the query construction process so that you can successfully build the target query graph.
3	I feel that VisualNeo's functionality is intuitive and easy to use.
4	I find VisualNeo bug-free and even if I make some operative errors, the system can notify me and recover from the errors.
5	I feel satisfied with the application when using it and would like to contine using it in the future.

11 Appendix F: Evaluation Results