الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي

Université Abou Bekr Belkaid
Tlemcen Algérie

جامعة أبي بكر بلقايد
تلمسان الجزائر

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education And Scientific Research

University of Tlemcen - ABOU BEKR BELKAID - Algeria

FACULTY OF SCIENCE

DEPARTMENT of COMPUTER SCIENCE

Graduation Project

To obtain a Master's degree in Computer Science

Specialty: Intelligent Models and Decision (M.I.D)

Thesis title:

A Deep Learning-based Approach for the Views Selection Problem of Graph Data

Presented by:

- BENYAHIA Mohammed Oussama.

Supervised by:

- Mr. MAHFOUD Houari.

Examined by:

- Mr. BENTAALLAH Mohamed Amine.
- Mme. AMGHAR Djazia.

Defended on: 26/06/2024

Academic year: 2023/2024

Acknowledgments

First, I express my profound gratitude to God the Almighty for granting me the strength, courage, and patience to complete this modest work.

I extend my heartfelt gratitude to my supervisor, Mr. MAHFOUD Houari. I am deeply thankful for your guidance, unwavering support, and constructive critiques, which contributed immensely to the successful completion of this work.

I also thank the jury members Mr. BENTAALLAH Mohamed Amine and Mme. AMGHAR Djazia. who graciously agreed to read and evaluate this work.

Furthermore, I express my sincere appreciation to my teachers, whose dedication and expertise have greatly influenced my academic journey. Your commitment to education and your invaluable insights have been instrumental in shaping my understanding and knowledge.

Additionally, I express my sincere thanks to all the individuals who have directly or indirectly contributed to this work. Your contributions, whether big or small, have played a significant role in shaping my academic journey and the successful completion of this research. May God's blessings be upon you all.

Dedication

This work is dedicated to:

My beloved family, whose unwavering love, support, and encouragement have been my constant source of strength throughout this journey. Your sacrifices and belief in me have made this achievement possible.

My teachers, whose dedication to imparting knowledge and wisdom has profoundly shaped my academic and personal growth. Your guidance and mentorship have been invaluable.

My friends and colleagues, for their camaraderie, support, and understanding. Your presence has made this journey more enriching and enjoyable.

And finally, to all those who have inspired and motivated me to strive for excellence. Your influence has been a driving force in my pursuit of knowledge and success.

May this modest work stand as a testament to your enduring impact on my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

VSP	View Selection Problem
AI	Artificial intelligence
ML	Machine Learning
RL	Reinforcement learning
DL	Deep Learning
DRL	Deep Reinforcement learning
SVR	Support Vector Regressor
MLP	Multilayer Perceptron
RBF	Radial Basis Function
GB	Gradient Boosting
XGBoost	Extrem Gradient Boosting
AdaBoost	Adaptive Boosting
ReLu	Rectified Linear Unit
MCMC	Markov Chain Monte Carlo
NLP	Natural Language Processing

General Introduction

1. General context

Nowadays, the amount and complexity of data have reached large volumes, posing considerable obstacles for conventional relational database models. As databases continue to grow exponentially and become more connected, the limitations of the relational model become more evident. Highly interconnected data is a challenge for relational databases, which can result in performance bottlenecks and scalability problems.

To overcome these challenges, graph databases model has come up as potential solution. The graph data model provides several benefits, such as flexibility in data modeling and a more intuitive way to represent complex relationships. Graph database systems outperform relational database systems in managing interconnected data and performing complex data analysis operations.

The graph data model is essential for many real-life applications like fraud detection, social network analysis, community detection, healthcare systems, privacy concern, data science.

Even the graph data model offers more flexibility and scalability than the relational model, lack of efficiency may be remarked when it comes to deal with very large data. Facebook for example represents a graph data having billions of nodes and trillions of edges. Efficiently querying vast amounts of data within a reasonable timeframe is essential for obtaining valuable insights and facilitating decision making.

Our work tackles the problem of querying large data graph and proposes heuristic and intelligent approaches that take into account time and space constraints.

2. Problem Statement

The exponential growth of data in real-life applications has underscored the need for efficient querying techniques to extract valuable insights from large datasets. Views-based querying stands out as a classical solution for overcoming this challenge, particularly in relational databases. However, its application to graph databases remains relatively unexplored.

To answer queries based on views, it is necessary to first extract relevant views that encapsulate frequently queried parts of the data. Next, these views must be materialized, i.e., stored in a form that allows for efficient retrieval and processing. In general, materialized views are very small compared to the original data which makes views-based answering a promising solution.

While the views extraction problem, also known as views selection, has been extensively studied in the context of relational data, graph data has received less attention in this regard. Existing solutions for views selection in graph data primarily rely on heuristic approaches, which may not always yield optimal results. Moreover, some parameters of these heuristics need to be fixed using artificial intelligence. However, no work has studied integrating artificial intelligence in these heuristic solutions.

3. Motivation

The rapid advancement of deep learning has developed various domains within computer science, offering unprecedented capabilities in tasks such as image recognition, natural language processing, and reinforcement learning. Deep learning techniques yield to remarkable success in handling complex data patterns and extracting meaningful insights from large datasets.

We are motivated by the potential of leveraging deep learning techniques to address the views selection problem in graph databases. Despite the widespread adoption of deep learning across various domains, its application to the views selection problem in the context of graph data remains largely unexplored. This presents an exciting opportunity to develop a novel approach that proves the power of deep learning to enhance the efficiency and effectiveness of views extraction in graph databases.

Specifically, we propose to leverage the potential of deep learning to assist heuristic approaches for the views selection problem. This assistance consists in providing good metric when measuring the cost space constraint that is essential for the quality of the heuristic solutions.

This report is organized as follows. **Chapter1** provides background information related to the context of view-based query answering. **Chapter2** explores current solutions for the views selection problem and shows their limitations. **Chapter3** gives an overview of our prediction model which consists in leveraging machine learning and deep learning techniques to predict query space costs accurately. In **Chapter4**, we present a revisited heuristic solution, we explain the modifications done and the integration of our deep learning model. Next, we experiment our solution using a real-life data graph. The report

ends with a general conclusion summarizing all our findings and highlighting the contributions of our work in the field of query optimization.

Chapter 1 Background

1.1 Introduction

We present in this chapter general notions and definitions that are necessary for the rest of this manuscript.

Many systems have been proposed to handle graph databases such as: Neo4j, MemGraph, NebulaGraph, JanusGraph and TigerGraph. We consider Neo4j as it is one of the most used graph database systems both in academia and industry.

Neo4j [1] stands at the forefront of graph database management systems, offering a revolutionary approach to data organization and management. Its APIs (such as APOC, GDS) offer a rich capability for managing graph data. As other systems, data under Neo4j is stored using the graph data model, and it is called graph database.

1.2 Graph Database

A graph database [2] is composed by three entities: node, relationship and property. An example of a Movie graph database is given hereafter.

1.2.1 Nodes

Nodes represent real-life entities such as people, products, or events. For example, in a Movie database, a node may be a Movie or a Person (Actor, Director, Producer, Writer, Follower or Reviewer).

1.2.2 Relationships

Relationships, called also edges, define connections between nodes and represent the associations or interactions between entities. For instance, in a Movie database, relationships could represent connections like "ACTED_IN", "DIRECTED", "WROTE", "FOLLOWS", "REVIEWED" and "PRODUCED". Compared to the relational model, a relationship may represent a foreign key between two relations.

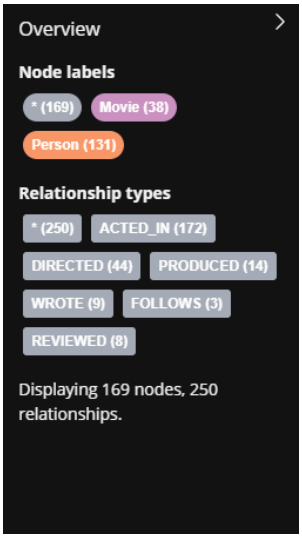
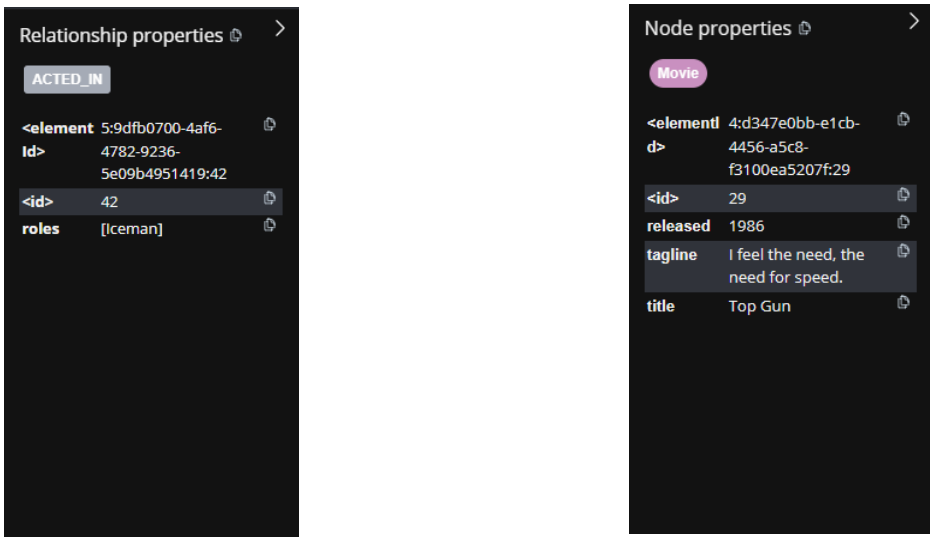


Figure 1.1 List of nodes and relationships composing the Movie graph database

1.2.3 Properties

Properties are key-value pairs associated with nodes and relationships, providing additional information about real-life entities and their connections. For instance, properties associated with a Movie node could include its title, release year, and tagline. Also, the relationship ACTED_IN, relying nodes Person and Movie, may have a property "role".



(a) Properties of the relationship ACTED_IN *(b) Properties of the node Movie*

Figure 1.2 Example of nodes and relationships properties

1.2.4 Data Graphs

Data graphs in Neo4j represents networks of interconnected nodes and relationships. These graphs offer a visual representation of complex data structures, making it easier to understand and analyze the relationships between different entities. **Figure 1.3** shows a part of the Movie graph database, while **Figure 1.4** makes a zoom-in in order to show interactions between nodes (Person and Actors) via several relationships (ACTED_IN, WROTE, DIRECTED).

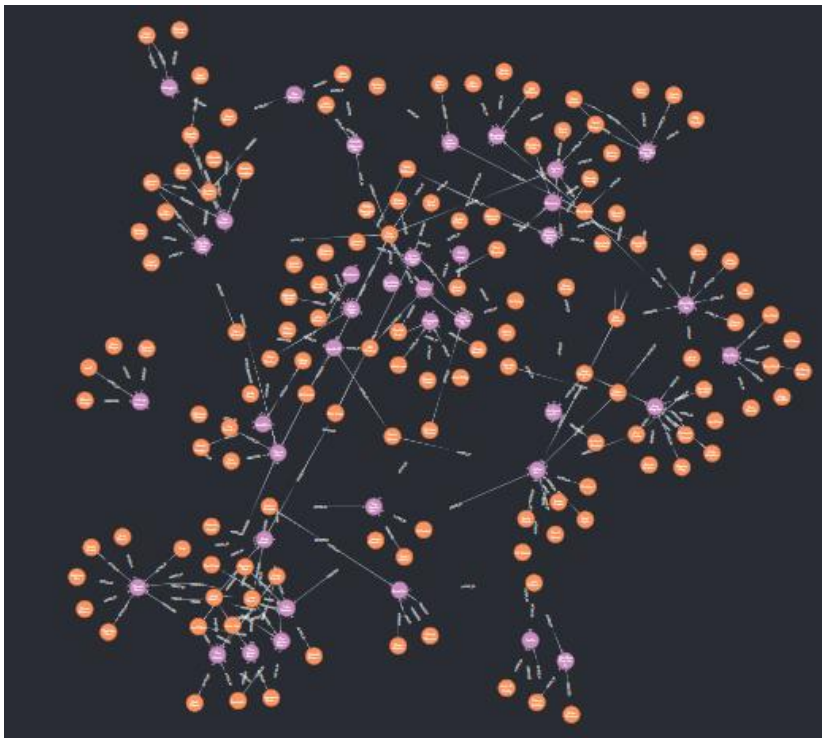


Figure 1.3 Interactions between nodes of the Movie graph database in Neo4j

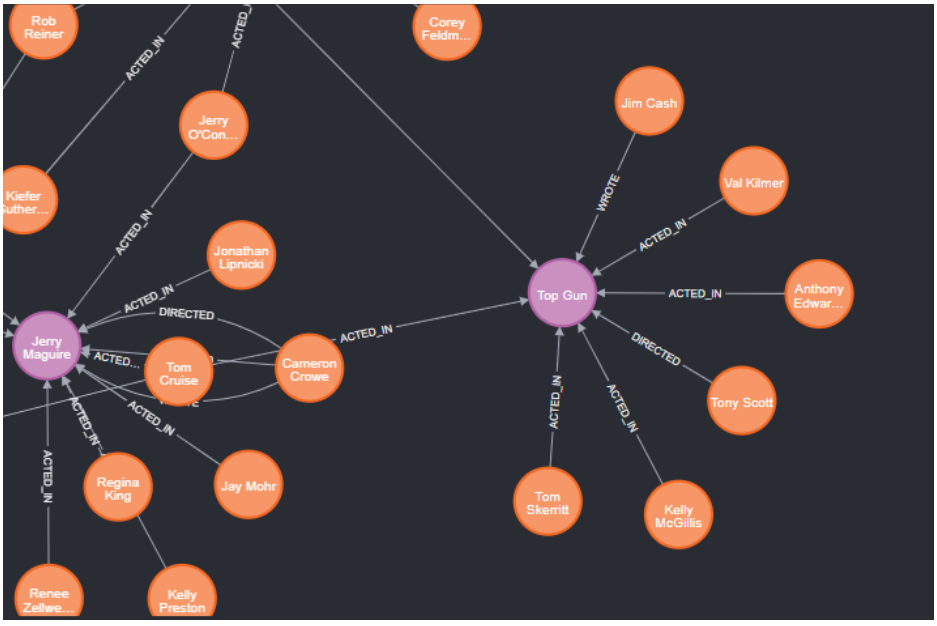


Figure 1.4 Zoom shows the connections between nodes of Movie database

1.3 Cypher Query Language

Cypher [3] [4] is the Neo4j graph query language, specifically crafted for retrieving data from the graph. It's a powerful and declarative language adjusted for graph databases like Neo4j, offering a concise and intuitive syntax for expressing graph patterns and executing operations on nodes and relationships within the database. Inspired by SQL, Cypher offers the possibility to retrieve entities, to define relations between them, to require some arithmetic and Boolean conditions, and finally to return all or only parts of the entities found by the query requirements. As SQL, the result of a Cypher query is a set of lines where each one is composed by entities that fulfill the query requirements.

Example 1.1:

The following Cypher query looks first for all movies that have been released after 2010. Then, it returns title and tagline of each movie found.

```
MATCH (m:Movie)

WHERE m.released>2010

RETURN m.title as Title , m.tagline as Tagline
```

Each movie found is assigned to a variable m, this variable is used to require some conditions over this movie. The result of this Cypher query is given in the following table:

Movies	**Tagline**
"Cloud Atlas"	**"Everything is connected"**

Example 1.2:

The second query below looks first for two actors who have acted together in some movies. Next, for each movie found, the query returns names of these actors as well the title of the movie.

```
MATCH (a1:Person)-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(a2:Person)

RETURN a1.name, m.title, a2.name
```

The execution of this Cypher query returns the following result:

a1.name	m.title	a2.name
Hugo Weaving	**The Matrix**	**Emil Eifrem**
Laurence Fishburne	**The Matrix**	**Emil Eifrem**
Carrie-Anne Moss	**The Matrix**	**Emil Eifrem**
⋮	⋮	⋮

1.4 Cypher Query Answering

Cypher queries are answered within Neo4j using the Graph pattern matching principle. This latter aims to find all subgraphs of the data graph that satisfy all requirements of a given Cypher query. This process enables users to navigate, describe, and extract data from the graph by applying declarative patterns.

In Cypher, graph pattern matching is facilitated through the MATCH clause. This clause allows users to specify the patterns they're seeking within the graph. The WHERE clause specifies conditions that these patterns must satisfy. The graph pattern matching process will look for all subgraphs that have the structure of the pattern specified by MATCH and that satisfy the conditions specified by WHERE. The RETURN clause allows returning all the pattern found or only parts of it.

1.5 Pattern containment

The pattern containment is to determine whether a query A is a part of another query B. For instance, given some materialized views {V1, V2, V3} and a query Q. If each of these views represents a part of Q and their union gives exactly Q, so we say that Q is contained

in the union of these views, and hence, Q can be answered using these views. A detailed example is given hereafter. Therefore, the pattern containment process is an important part of the views-based answering and also the views selection problem as we should show later.

Example 1.3:

This is an example of tree views **{V1, V2, V3}** from the Stack Overflow database [22], which includes nodes labeled "User" and three types of edges: “COMMENT_TO_QUESTION”, “ANSWER_TO_QUESTION”, and “COMMENT_TO_ANSWER”.

V1:

```
MATCH (p1:User)-[:COMMENT_TO_QUESTION]->(p2:User)<-[:ANSWER_TO_QUESTION]-(p3:User)
RETURN *
```

V2:

```
MATCH (p1:User)-[:ANSWER_TO_QUESTION]->(p2:User)<-[:COMMENT_TO_QUESTION]-
(p3:User)-[:ANSWER_TO_QUESTION]->(p4:User) RETURN *
```

V3:

```
MATCH (p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)-[:COMMENT_TO_ANSWER]->(p3:User)
RETURN *
```

Given a new query **Q**:

```
MATCH (p1:User)<-[:ANSWER_TO_QUESTION]-(p2:User)-[:COMMENT_TO_QUESTION]-
>(p3:User)<-[:ANSWER_TO_QUESTION]-(p4:User)-[:COMMENT_TO_ANSWER]->(p5:User)-
[:COMMENT_TO_ANSWER]->(p6:User) RETURN *
```

The pattern containment analysis shows:

- View 1 (V1) is contained in Q and covers 40% of it.
- View 2 (V2) is contained in Q and covers 60% of it.
- View 3 (V3) is contained in Q and covers the remaining 40%.

After pattern containment analysis, one can detect that Q can be answered using only V2 and V3.

This demonstrates how pattern containment is an essential process when working with views

1.6 Views selection problem

In database systems, materialized views are precomputed result sets derived from the underlying data. They work as a way to reduce the time execution of the query processing by storing the results of frequently executed queries. However, maintaining materialized views for all possible queries within a workload can not be feasible due to resource constraints and the computational cost of view maintenance. To deal with this problem, the views selection problem (VSP) [5] aims to detect views that cover a large percentage of the query workload with the least amount of processing cost management.

The key objectives of the Views Selection Problem include:

Maximizing Query Coverage: The selected views should cover a wide range of queries within the workload to reduce the need for executing queries directly on the original graph data.

Minimizing Materialization Costs: The views should be selected in such a way that the overall computational cost of maintaining the views is minimized, considering factors such as storage requirements.

Optimizing Query Performance: By selecting views strategically that address commonly executed queries, the Views Selection Problem aims to improve query performance and reduce response times for end-users.

Example 1.4:
Given a query workload **QW** composed by 7 Cypher queries, **QW= {Q1, Q2, Q3, Q4, Q5, Q6, Q7}**, defined over the Stack Overflow database. A views selection process gives {V1, V2, V3} as the views that cover all queries in **QW**. Details of the workload as well as the selected views are given hereafter.

Query Workload

Q1:

(p1:User)-[:COMMENT_TO_QUESTION]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)<-[:COMMENT_TO_QUESTION]-(p5:User)-[:COMMENT_TO_ANSWER]->(p6:User)

Q2:

(p1:User)<-[:COMMENT_TO_QUESTION]-(p2:User)-[:ANSWER_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_QUESTION]-(p4:User)-[:ANSWER_TO_QUESTION]->(p5:User)

Q3:

(p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)-[:COMMENT_TO_QUESTION]->(p5:User)<-[:COMMENT_TO_QUESTION]-(p6:User)

Q4:

(p1:User)<-[:ANSWER_TO_QUESTION]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)<-[:COMMENT_TO_QUESTION]-(p5:User)-[:COMMENT_TO_ANSWER]->(p6:User)

Q5:

(p1:User)<-[:COMMENT_TO_ANSWER]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_ANSWER]-(p4:User)-[:COMMENT_TO_QUESTION]->(p5:User)<-[:COMMENT_TO_QUESTION]-(p6:User)

Q6:

(p1:User)<-[:COMMENT_TO_QUESTION]-(p2:User)-[:ANSWER_TO_QUESTION]->(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)<-[:COMMENT_TO_QUESTION]-(p5:User)-[:COMMENT_TO_ANSWER]->(p6:User)

Q7:

(p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)

Selected Views

V1:

- `MATCH (p1:User)-[:COMMENT_TO_QUESTION]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User) RETURN *`

V2:

- `MATCH (p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User) RETURN *`

V3:

- `MATCH (p1:User)-[:COMMENT_TO_QUESTION]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User) RETURN *`

These views are selected to maximize the coverage of the workload while minimizing the space cost.

1.7 Views materialization

Views materialization [6] is an essential database management procedure that precomputes and stores the answers to frequently run queries **{Q1..., Qn}** in order to enhance query performance. The system creates and saves the result set of these queries, **{R1..., Rn}**. We call these queries views and their results materialized views. These materialized views are used to speed up access to the results of subsequent queries and cut down on computational overhead.

However, maintaining materialized views is essential to ensure their accuracy and consistency with the underlying data. This often involves periodic refreshes to incorporate changes in the dataset and uphold data integrity.

Overall, views materialization plays a pivotal role in database optimization, contributing to improved query performance and system efficiency.

1.8 Views-based answering

Views-based answering [7] is a query optimization technique in database management, aiming to efficiently answer queries by leveraging precomputed materialized views. The goal is to identify the minimal set of materialized views required to accurately answer a given query. This involves analyzing the query structure and requirements, selecting relevant materialized views, and minimizing redundancy. By utilizing materialized views, database systems optimize query performance and enhance overall efficiency.

In the context of optimal views-based answering, the goal is to find the minimal number of materialized views that can answer the new query.

1.9 Conclusion

We have detailed all necessary notions and definitions that are useful to understand the rest of this manuscript. In this work, we consider only the views selection problem while views materialization and views-based answering are left for future extensions of this work.

Chapter 2 State of the Art

2.1 Introduction

As already said before, the views selection problem has received less attention in the field of graph databases compared to that of relational databases. Only few works have addressed this problem for graph databases. We next present these works and discuss their limitations.

2.2 A Heuristic Approach

2.2.1 Description

Xin Wang proposed a heuristic solution [5] to select views to materialize from a large workload composed by formal graph queries (not real queries as Cypher queries). The pseudo code is given at **Figure 2.1**.

Input: Workload $\mathcal{Q} = \{\mathsf{Q}_1, \cdots, \mathsf{Q}_n\}$ with weight on each Q_i, and a bound $\mathcal{B}$.
Output: A set of view definitions $\mathcal{V}$.

1. initialize $\mathcal{I}_u$, $\mathcal{I}_e$, $\mathcal{V}$ and uncvd; set $\mathsf{V}_m := \emptyset$; max:=0;
2. **while** uncvd > 0 **do**
3. pick a view V_e from $\mathcal{I}_e$ with largest $g(\mathsf{V}_e)$; max:=0;
4. **for each** V_j in $\mathcal{V}$ **do**
5. generate V'_j by enlarging V_j with V_k in $\mathcal{I}_e$;
6. compute $g(\mathsf{V}'_j)$, $\Delta g(\mathsf{V}'_j) = g(\mathsf{V}'_j) - g(\mathsf{V}_j)$;
7. **if** max $< \Delta g(\mathsf{V}_j)$ **then** max $:= g(\mathsf{V}_j)$; $\mathsf{V}_m := \mathsf{V}_j$;
8. **if** $g(\mathsf{V}_e) >$ max **then**
9. $\mathcal{V} := \mathcal{V} \cup \{\mathsf{V}_e\}$; update $\mathcal{I}_u$ and uncvd;
10. **else**
11. update $\mathcal{V}$ with V_m; update $\mathcal{I}_u$, $\mathcal{I}_e$ and uncvd;
12. **if** $\gamma(\mathcal{V}) > \mathcal{B}$ **then** restore $\mathcal{V}$; **break** ;
13. **return** $\mathcal{V}$;

Figure 2.1 Xin's heuristic algorithm [5]

The key idea behind the heuristic is to analyse all parts of the query workload and to determine, at each step of the algorithm, a view to select which has a high goodness. The goodness metric is defined as follows:

$$G(v) = \frac{\delta(v, \omega)}{\gamma(v)}$$

$\boldsymbol{\delta(v, \omega)}$: $coverage\ of\ the\ view\ V\ in\ teh\ workload\ W$
$\boldsymbol{\gamma(v)}$:$Cost\ of\ the\ view\ V$

The algorithm begins parsing all queries at the workload and by considering the edge of each query as a simple possible view. I_e is the set of all single edges from the query workload that are initially considered as possible views. At the first iteration, it determines a view from I_e that has a high goodness and it adds it to the result V. Next, it examines again the views in I_e and determines either a new view to add into V or a view to enlarge from V. Each time a view is added to V or enlarged, the number $uncvd$ is updated, which

specifies the number of edges from the workload that are covered by the views found in V. The heuristic continues this process until the space cost of the chosen views surpasses a given bound β, or until all edges in the workload are covered. The heuristic considers space restrictions and iteratively selects views based on goodness values in order to maximize query efficiency while minimizing materialization costs.

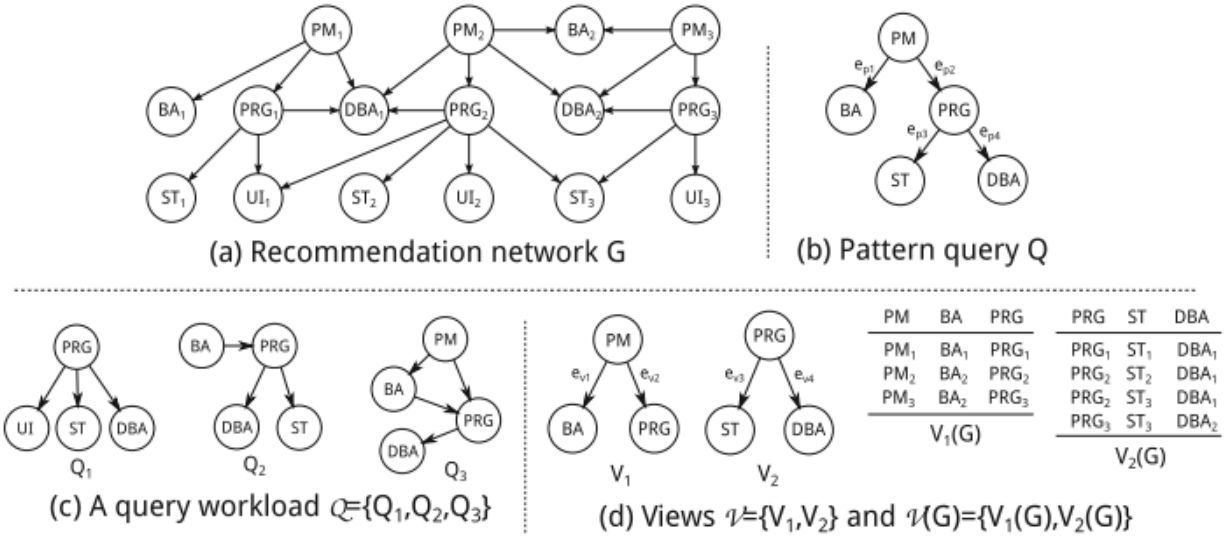


Figure 2.2 Example of Xin's VSP heuristic [5]

Example 2.1:

Figure 2.2 illustrates an application example of the Xin Wang heuristic. **Figure 2.2(a)** gives a portion of a recommendation network illustrated where each node represents a person with a specific job title (e.g., project manager (PM), business analyst (BA), database administrator (DBA), programmer (PRG), user interface designer (UI), and software tester (ST)), and each edge represents collaboration between individuals (e.g., (PM1, PRG1) indicates collaboration between PM1 and PRG1 on a project led by PM1).

Figure 2.2(c) gives a query workload **Q** composed by three queries **{Q1, Q2, Q3}**. The application of the Xin Wang heuristic to this query workload gives the views **{V1, V2}** that are depicted in **Figure 2.2(d)**. Remark that these two views cover all edges of the workload. Therefore, if a new query arrives, likes that of **Figure 2.2(b),** then this query can be answered using only the selected views **{V1, V2}** without accessing the underlying data graph.

2.2.2 Limitations

While Xin Wang's heuristic solution offers promising advancements in view-based query optimization for selecting views to materialize, it may encounter limitations. Here are a few limitations worth noting:

a) Lack of real-life language (Cypher): One significant drawback of Xin's heuristic is that it does not handle real-world query languages such as Cypher. Its weakness makes it less useful in real-world situations where these kinds of languages are usually used for expressing inquiries. Consequently, the heuristic might not adequately represent the variety and complexity present in real-world query workloads, potentially leading to suboptimal view selection and materialization decisions.

b) Limited applicability to certain graph structures: The heuristic does not take into account edges type and it is not adequate for graph databases having few numbers of nodes label.

c) Theoretical parameters for storage and time costs: The heuristic utilizes theoretical parameters for storage and time costs, but it does not provide a clear methodology for determining or retrieving these values from the database. More precisely, the view estimation cost is not detailed while it is the kernel of the heuristic. This lack of clarity can lead to uncertainties and inaccuracies in the estimation of view goodness, affecting the overall effectiveness of the heuristic.

d) Overlapping Views Problem: The selection of a new view is done independently regarded to the views already selected. This may lead to overlapped views where some edges of the query workload are covered by many views. This drawback can result in overlooking more optimal view combinations, leading to suboptimal materialization decisions.

e) No Upper Bound for Cost Storage: The algorithm allows any cost bound as input, but it may not terminate if this cost bound exceeds a certain upper limit. Consequently, the heuristic might select views with a combined cost that surpasses the storage cost of the original database (single edges), leading to inefficiencies and impractical storage requirements.

2.3 Deep Learning-based Approaches

Some works [8] [9] have investigated deep learning approaches for the VSP in the context of relational databases. A new technique, called Deep Reinforcement Learning (DRL), has been also investigated too [25] which combines the power of deep learning and

reinforcement learning. The technique is based on: environment, actions, states, rewards, and policy. It uses neural networks to approximate complex functions, allowing agents to learn from experience and make intelligent decisions in environments. Here's a brief explanation of DRL and its parameters:

- **Environment**: The external system or space in which the agent operates.
- **State**: The current situation or configuration of the environment.
- **Action**: The set of possible moves or decisions the agent can make.
- **Reward**: The feedback signal received after an action, quantifying the immediate benefit.
- **Policy**: A strategy or set of rules that the agent follows to decide on actions based on the current state, aiming to maximize cumulative rewards over time

As the technique has been investigated for relational databases, we think that it will still feasible for graph databases. Here is how we think DRL can be adapted for graph databases:

- **Environment:** The workload of graph queries.
- **State:** The current selected views.
- **Action:** The process of enlarging one of the already selected view or adding a new one.
- **Reward:** A score that aims to maximize the coverage of the workload while minimizing the materialization cost.
- **Policy**: A strategy to select actions that balance the trade-off between exploration (trying new views) and exploitation (optimizing current views).

By defining these elements, reinforcement learning can be effectively applied for the VSP of graph databases. However, due to time limitation, we did not investigate this option in our work.

2.4 Discussion

The experimental study done in [5] shows that the Xin Wang solution is effective for the views selection problem of graph queries. Our goal is to first revise this solution in order to deal with its algorithmic drawbacks and to make it suitable for real-life queries and

general graph structures. Secondly, we will use deep learning to deal with the problem of views cost prediction.

On the other side, reinforcement learning can be used to address the problem of views selection problem. However, due to time constraint, we did not study this approach.

2.5 Conclusion

In this chapter, we explored existing approaches to the view selection problem in the context of graph databases, focusing on Xin Wang's heuristic solution and its limitations. While Xin's heuristic presents promising advancements, it faces challenges such as limited applicability to certain graph structures, the absence of real-life language support, theoretical parameter usage for storage and time costs, the overlapping views problem, and the absence of an upper bound for cost storage. Despite these limitations, Xin's heuristic has demonstrated significant results, prompting further investigation. The next chapter will address these limitations and propose solutions.

Chapter 3 : Our Prediction Model

3.1 Introduction

As explained before, existing heuristic solution selects some views to materialize whose sizes do not exceed a given space cost budget. This budget can be defined according to the limited space accorded by the database administrator for the materialization process. During the views selection process, the challenging task is, how to predict the space cost of some view? It is clear that some views may cover more parts of the query workload while they may exceed the accorded space budget. So, a compromise must be done between covering aspect and space aspect of the view to select. However, no intelligent model is given to predict the space cost of views.

Graph	Size (GB)	Nodes	Edges
Stack Overflow	3.62	2601977	63497050

Table 3.1 Stack Overflow database size

To address this challenge and optimize Xin Wang heuristic solution, we propose leveraging machine learning or deep learning techniques to develop a prediction model. Specifically, our goal is to train a model capable of predicting the space used to materialize a view (Cypher query) in a Neo4j database. This prediction model would enable more informed decisions when calculating the goodness of views, ultimately leading to more accurate selections for materialization and reducing their time.

However, the first obstacle encountered when doing prediction is the large size of real-life data graphs. For example, **Table 3.1** shows details about the real-life Stack Overflow graph database. As mentioned, the graph contains millions of nodes and edges which makes any prediction process time-consuming. One of the primary obstacles we encounter is the significant time required to execute queries and collect relevant data. This challenge underscores the need for efficient data collection strategies to overcome potential delays and ensure the timely development of our prediction model.

In the rest of this chapter, we delve into our approach for addressing these challenges, outlining our methodology for dataset collection, model training, and the development of a predictive model for view space estimation.

3.2 Graph sampling

Due to the complexity of our database and the time-consuming nature of data collection, we used graph data sampling as a solution to reduce the collection process delays.

3.2.1 Definition of Data Sampling

Graph data sampling [10] [11] is a process of selecting a subset of nodes and edges from a larger graph dataset in order to create a representative sample for the data collection process. This sampling technique aims to capture the essential characteristics and structural properties of the original graph according to our needs while reducing the overall size of the dataset. The goal is to create a sample that accurately reflects the distribution and relationships present in the complete graph.

3.2.2 Criteria of Data sampling

While each sampling technique may have its specific principles, there are some common principles that often apply across different techniques:

a) Representativeness: The sampled data should accurately represent the characteristics of the original dataset. This ensures that any analysis or modeling performed on the sample reflects the most properties of the entire dataset.

b) Randomness: Random sampling techniques ensure that each element in the dataset has an equal chance of being selected for inclusion in the sample. This helps reduce bias and ensures that the sample is unbiased and statistically valid.

c) Efficiency and Scalability: Sampling techniques should be efficient in terms of time and resources. They should also be scalable to handle large datasets efficiently, capable of generating representative samples from datasets of varying sizes without sacrificing accuracy.

d) Bias Reduction: Sampling techniques should minimize bias introduced during the sampling process. Biased samples can lead to inaccurate conclusions, so efforts should be made to minimize or eliminate sources of bias.

f) Statistical Validity: Sampled data should be statistically valid, meaning that any inferences or conclusions drawn from the sample can be extrapolated to the population with a certain level of confidence.

3.2.3 Main Algorithms of Data Sampling

a) Random Walk Sampling [10]**:** This technique involves starting at a random node in the graph and iteratively traversing edges to neighboring nodes based on certain probabilities. It is effective for exploring large graphs and capturing local graph structures.

b) Node Sampling [10]**:** Node sampling involves selecting a subset of nodes from the graph based on predefined criteria, such as node degree, centrality measures, or node attributes. This technique is useful for studying specific subsets of nodes or identifying important nodes in the graph.

c) Edge Sampling [10]**:** Edge sampling focuses on selecting a subset of edges from the graph, often based on edge attributes, edge weights, or structural properties. Edge sampling can help identify important relationships or patterns in the graph.

d) Snowball Sampling [10]**:** Snowball sampling starts with a small set of seed nodes and iteratively expands the sample by adding neighboring nodes or edges. This technique is useful for capturing local neighborhoods around specific nodes or exploring the graph's connectivity.

e) Stratified Sampling [10]**:** Stratified sampling divides the graph into strata or partitions based on certain criteria (e.g., node attributes, communities) and then samples from each stratum independently. This technique ensures that each stratum is adequately represented in the sample.

f) Metropolis-Hastings Sampling [10]**:** Metropolis-Hastings sampling is a Markov Chain Monte Carlo (MCMC) method that iteratively generates samples by proposing moves based on a transition probability distribution and accepting or rejecting these moves based on a specified criterion. It is useful for sampling from complex probability distributions and exploring the entire graph space.

3.2.4 Noe4j Data Sampling Algorithms

Neo4j's GDS (Graph Data Science) library [12] provides predefined techniques for sampling graph data. Two commonly used techniques are:

a) Random Walk with Restart [13]**:** This technique involves initiating random walks from a set of starting nodes and allowing them to traverse the graph. After a certain number of steps, the walks are "restarted" from the original starting nodes, ensuring exploration of different regions of the graph. RWR is useful for capturing the local structure of the graph and identifying important nodes or communities.

b) Common Neighbor Aware Random Walk [14] [15]**:** a graph exploration technique that enhances traditional random walk algorithms by considering common neighbors of nodes. It aims to prevent local loops during traversal, ensuring efficient exploration, particularly in dense graph regions. By prioritizing nodes with high degrees and fewer common neighbors with previously visited nodes, it accelerates convergence and optimizes graph sampling, especially in graphs with solid dense regions or high clustering coefficients.

We notice that these Neo4j data sampling algorithms are still in their infancy and some limitations have been remarked:

a) Efficiency: It may be difficult for Neo4j methods to sample data from large, complex graphs in an effective manner. It can become more difficult and time-consuming to sample a graph as it gets bigger, which makes it harder to gather a representative dataset for model training.

Using methods such as CNARW on huge datasets may lead to serious performance problems. For instance, on a local PC with an i7 7th gen processor and 16GB of RAM, we tried to apply the CNARW technique on the Stack Overflow dataset (63M Edges, 2M Nodes). The technique took more than 7 days and didn't terminate the sampling.

b) Structure Preservation: Certain Neo4j sampling techniques may fail to preserve the structural properties of the original graph during the sampling process. This can lead to sampled datasets that fail to capture important graph characteristics, resulting in suboptimal model performance.

c) Inadequate Coverage: Neo4j methods might not fully cover every area of the graph, particularly when there are isolated nodes or detached components. The reliability of prediction models trained on the data may be impacted by sampled datasets that do not accurately represent the graph as a result of this insufficient coverage**.**

As existing Neo4j sampling algorithms are time-consuming, we were obliged to propose our own solution in order to get a sample subgraph of our large Stack Overflow dataset.

3.2.5 Our algorithm for Data Sampling

Based on the random-walk principle, we propose an algorithm that enables the extraction of representative subgraph while overcoming time limitations and ensuring the efficiency of the sampled data. Our algorithm is given in **Figure 3.1.**

Algorithm:

$\boldsymbol{Cnode}$: $current\ node$

$\boldsymbol{Nnode}$: $next\ node$

$\boldsymbol{CovrN}$: $set\ \ of\ covered\ nodes$

$\boldsymbol{Nbors}$: $set\ of\ node\ neighbors$

$\boldsymbol{Rtype}$: $relation(edge)\ type$

$Input: (graph: \boldsymbol{G}, sampling\ ratio: \boldsymbol{r})$

$Output: sampled\ graph: \boldsymbol{SG}$

1. $Initialize : CovrN = [\,], Nbors = [\,], Cnode = NULL, Nnode = NULL, Rtype = NULL$
2. $Cnode = random\_node(G).$
3. $While\ size(SG) < r * size(G) :$
4. $\quad Nbors = neighbors(Cnode)$
5. $\quad if\ Nbors = [\,]\ then:$
6. $\quad\quad Cnode = random\_node(SG)$
7. $\quad\quad continue$
8. $\quad Nnode = random\_node(Nbors, weight = common\_neighbors(neighbor, Cnode))$
9. $\quad if\ Nnode\ in\ CovrN\ then:$
10. $\quad\quad Cnode = random\_node(SG)$
11. $\quad\quad continue$
12. $\quad CovrN.append(Nnode)$
13. $\quad Rtype = relationS(Cnode, Nnode)$
14. $\quad SG.append((Cnode, Rtype, Nnode))$
15. $Return\ SG$

Figure 3.1 Our data sampling algorithm

The algorithm begins by inputting the graph, represented as a CSV file containing the start and end nodes of each edge, and the ratio of sampling size. We create a dictionary to store each node and its neighbors along with the type of edge. The main process involves selecting a start node randomly as the starting point and then choosing the next node by prioritizing those with the most common neighbors shared with the current node. If the next node is already visited, the current node is selected randomly from visited nodes. Otherwise, the algorithm appends the current node and the next node with their relation type as edges to the sampled graph. This process continues until the sampled graph reaches or exceeds the specified ratio size. Finally, the algorithm returns the sampled graph as visited edges.

To check the performance of our algorithm, we applied it over the Stack Overflow database, targeting a reduction in size to just 0.5% of the original. This reduction ratio was selected to substantially decrease the database size, facilitating accelerated data collection, while simultaneously retaining its fundamental structural properties. By presenting the results, we aim to evaluate the algorithm's success in achieving this reduction objective while upholding the integrity of the original graph's connectivity and complexity. Hereafter are some details of the original graph dataset which are necessary to analyse the results of our data sampling solution:

- **Nodes Label:** "User"
- **Edges Types**:
  - **C2A:** "COMMENT _TO_ ANSWER";
  - **A2Q:** "ANSWER_TO_QUESTION";
  - **C2Q:** "COMMENT_TO_QUESTION"

The details of our data sampling result are given in **Table 3.2**.

Sampled Database			Original Database		
49633 Nodes			2601977 Nodes		
Edges			Edges		
C2Q	A2Q	C2A	C2Q	A2Q	C2A
95513	86103	129358	20268151	17823525	25405374
All edges: 310974			All edges: 63497050		

Table 3.2 Comparison between Original and Simplified Databases

Enhancement Results:

we summarize here after the enhancements given by our algo compared to the limitations of the Neo4j algorithms:

- **Time Limitations:** Our algorithm effectively addresses time constraints by employing an efficient randomized exploration process for graph sampling, while also leveraging CSV file operations to expedite data retrieval and graph representation. However, it's important to acknowledge that execution time may

vary depending on factors such as the graph's size and the specified sampling ratio. In our experiments, the algorithm typically terminates within 24 minutes. We recall that when using the CNARW technique of Neo4j, the execution time extended to 7 days, and the process did not terminate within the Neo4j environment. So, our algorithm gives a significant improvement.

- **Structure Preservation:** Our algorithm guarantees the preservation of the sampled subgraph's full connectivity and structural integrity. By leveraging a random walk approach based on common neighbors, it ensures that crucial relationships between nodes and the overall complexity of connectivity are maintained, reflecting the inherent relationships and preserving the original graph's structure within the sampled dataset.

3.3 Data collection

To collect a dataset comprising Cypher queries and their respective space costs, we started by extracting information from a simplified graph database. The main idea is to perform the Cypher query and to obtain all nodes and edges making parts of its result. After that, a space cost is attributed to each node and edge which represents its real space required by the system to store it. We are using Neo4j hardware sizing calculator [16] to extract the real space cost attributed by Neo4j to store each entity, the **Figure 3.2** shows these costs:

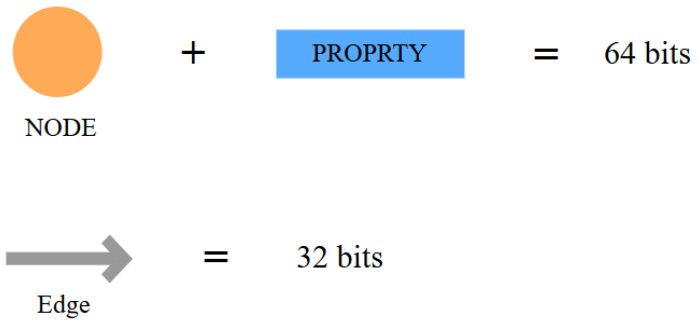


Figure 3.2 Real space cost for nodes and relationships under Neo4j

3.4 Features extraction

To extract features from Cypher queries for use as input, we employ the following process:

a) Counting the Number of Each Type of Edges:

For each Cypher query and each edge type, we count the number of outgoing and incoming edges having this type. This involves analyzing the structure of the query and identifying patterns representing different edge types.

b) Encoding Cypher Queries:

We utilize one-hot encoding to represent Cypher queries. Since each Cypher query typically involves interactions between specific node labels and edge types, we construct a binary vector to represent each query.

For example, in the Stack Overflow dataset with a single node label (User) and three edge types (COMMENT_TO_QUESTION, ANSWER_TO_QUESTION, and COMMENT_TO_ANSWER), we create a binary vector of length 6 for each edge encountered in the query with its two possible directions.

Each position in the vector corresponds to a specific edge type and direction. For instance, we have in **figure 3.3** a detailed example of cypher query encoding:

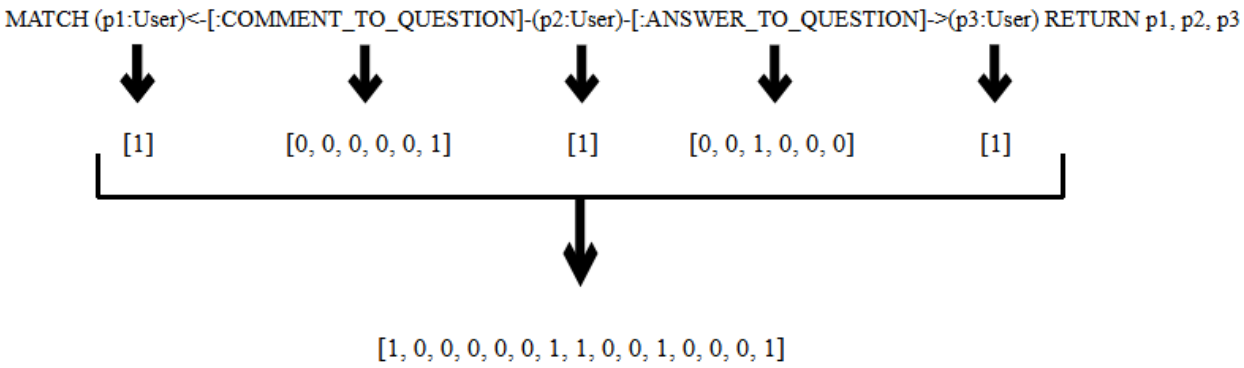


Figure 3.3 Query encoding example

By extracting these features from Cypher queries, we can effectively represent the query structure and edge interactions, enabling their utilization as input for deep learning or machine learning models.

3.5 Prediction model

To predict query space cost, we trained several machine learning and deep learning models.

3.5.1 Machine Learning Models

1. Linear Regression [17] [18]**:** A basic regression algorithm that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation.

2. DecisionTreeRegressor [17] [18]**:** A decision tree algorithm that recursively splits the data based on feature thresholds to predict the target variable. It's interpretable and can handle both numerical and categorical data.

3. RandomForest Regressor [17] [18]**:** An ensemble learning method that constructs multiple decision trees and averages their predictions to improve accuracy and reduce overfitting.

4. AdaBoost Regressor [17] [18]**:** Another ensemble learning method that combines multiple weak learners sequentially, with each subsequent learner focusing on the mistakes made by the previous ones.

5. Gradient Boosting Regressor [19]**:** A machine learning model that iteratively corrects errors made by previous weak learners (usually decision trees) to enhance overall predictive accuracy. It employs gradient descent optimization to minimize the loss function during training.

6. XGBoost Regressor [19]**:** An optimized implementation of gradient boosting that uses an ensemble of weak decision trees (often shallow trees) to create a strong predictive model. XGBoost efficiently minimizes the loss function using a tailored gradient descent optimization algorithm.

7. SVR [17] [18]**:** A regression algorithm that uses support vector machines to find the optimal hyperplane that best separates the data into different classes while minimizing the error.

3.5.2 Deep Learning Models

1. MLP Regressor [20]**:** A type of feedforward neural network with multiple layers of nodes (neurons) that can learn complex patterns in the data by passing information through nonlinear activation functions.

2. RBF Regressor Network [21]**:** a neural network architecture utilizing radial basis functions in its hidden layer. With an input layer, hidden layer employing radial basis functions, and an output layer, it learns to approximate complex relationships between

input and output variables. Effective for nonlinear data relationships, it excels in regression tasks.

We evaluated all the mentioned machine learning and deep learning models based on three techniques:

Mean Absolute Error (MAE) [17]**:** This metric calculates the average absolute difference between the predicted values and the actual values. It provides a straightforward measure of the model's accuracy, where lower MAE values indicate better performance.

Mean Absolute Percentage Error (MAPE) [17]**:** MAPE measures the percentage difference between the predicted values and the actual values. It's particularly useful for understanding the relative accuracy of the model across different ranges of values. Similar to MAE, lower MAPE values indicate better performance.

Mean Squared Error (MSE) [17]**:** MSE calculates the average of the squared differences between the predicted values and the actual values. It penalizes larger errors more heavily than smaller ones, making it sensitive to outliers. Like MAE, lower MSE values indicate better performance, but it's influenced more by large errors.

After modifying and optimizing the parameters of each model to give her best, we obtained the results given in **Figures 3.(4-6).**

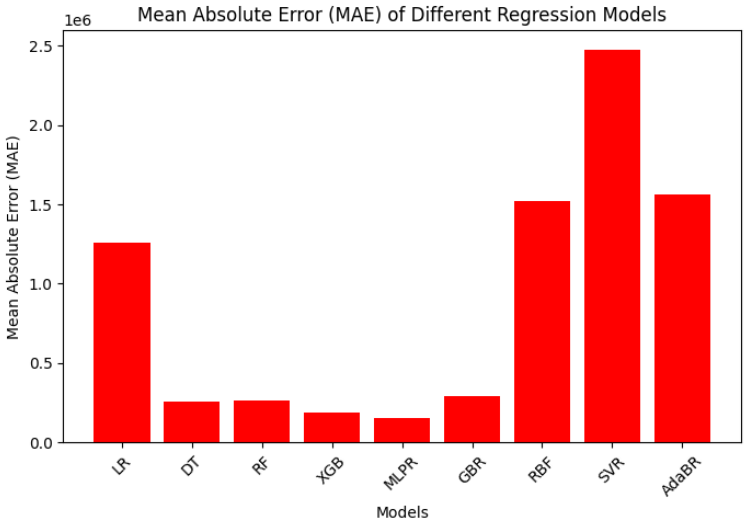


Figure 3.4 Mean Absolute Error (MAE) of Different Regression Models

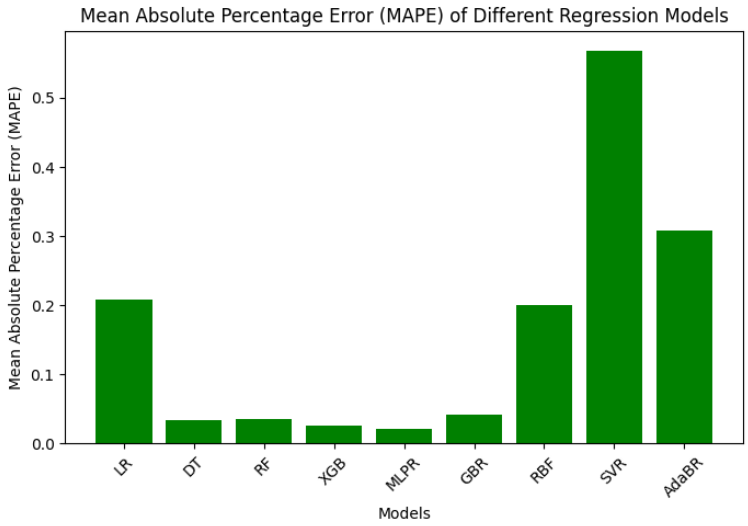


Figure 3.5 Mean Absolute Percentage Error (MAPE) of Different Regression Models

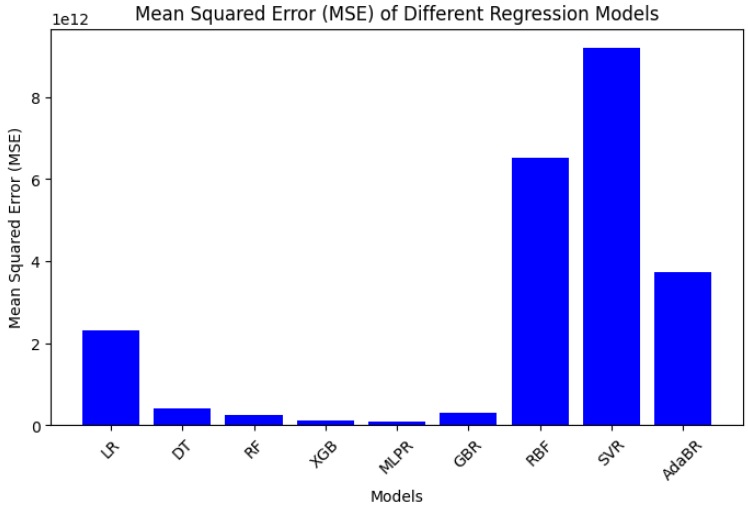


Figure 3.6 Mean Squared Error (MSE) of Different Regression Models

From the histograms presented, it's evident that the MLPRegressor consistently achieves the most significant reduction in prediction errors compared to other models. This remarkable performance can be attributed to its underlying architecture and training parameters. Specifically, the MLPRegressor utilizes a neural network model with a sequential architecture comprising several dense layers. For instance, the model configuration includes dense layers with varying activation functions, such as Sigmoid and Relu, along with linear activation for the output layer. The architecture of the model is given in **Table 3.3**.

Model: "sequential_4"			
Input Shape: (None, 107)			
Layer (type)	Output Shape	Param #	Activation Function
dense_12 (Dense)	(None, 2000)	216000	Sigmoid
dense_13 (Dense)	(None, 1000)	2001000	Relu
dense_14 (Dense)	(None, 1)	1001	Linear
• Total params: 2,218,001 • Trainable params: 2,218,001 • Non-trainable params: 0			

Table 3.3 MLPRegressor architecture

The MLPRegressor is trained over 100 epochs, allowing it to effectively learn and adapt to the complexities present in the data. This extensive training period enables the model to capture intricate patterns and relationships, resulting in superior prediction accuracy.

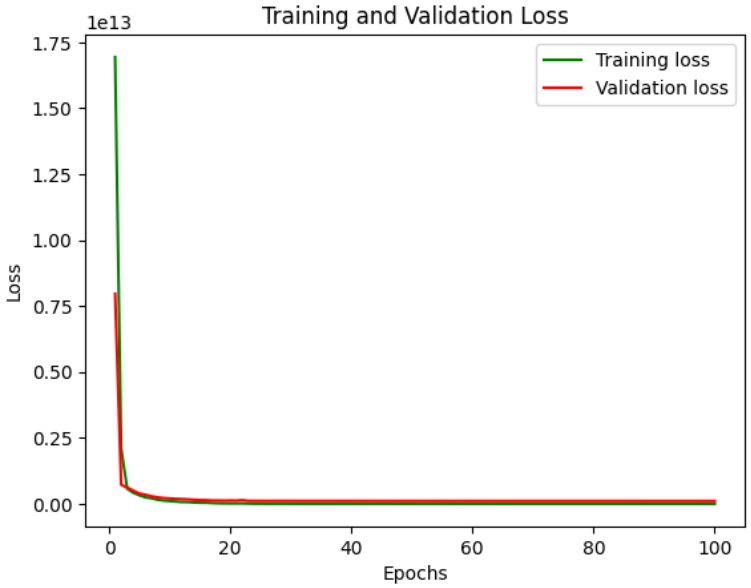


Figure 3.7 Training and Validation Loss curve

The success of the MLPRegressor can also be observed in the curve of loss given in **Figure 3.7**, where both the training loss and validation loss exhibit a consistent decrease over the epochs. This trend signifies that the model effectively learns from the training data while generalizing well to unseen validation data, indicating robust performance.

Additionally, the architecture of the MLPRegressor neural network, with its multi-layered configuration and specific activation functions, contributes to its exceptional predictive capabilities. The dense layers in the network enable the extraction of complex features from the data, while the choice of activation functions facilitates non-linear transformations, allowing the model to capture intricate relationships between input features and target outputs.

Overall, the MLPRegressor's advantages lie in its ability to leverage the expressive power of deep neural networks to accurately predict query space costs. Its multi-layered architecture and prolonged training duration facilitate the extraction of intricate features from the data, leading to more precise predictions compared to other models in the evaluation.

3.6 Conclusion

In this chapter, we tackled the challenge of predicting space costs for materialized views in graph databases. We developed a custom random walk-based graph sampling algorithm to efficiently handle large datasets like Stack Overflow, preserving structural integrity while reducing size. By extracting features from Cypher queries, we trained and evaluated several prediction models, finding that the Multilayer Perceptron (MLP) Regressor performed best. In the next chapter, we will focus on integrating and optimizing this model within heuristic solution and evaluate its efficiency in query optimization and resource management in graph databases

Chapter 4 Our Deep-Learning based Views Selection Solution

4.1 Introduction

In this chapter, we explain how to address the limitations identified in the original heuristic proposed in Chapter 2 by integrating our deep learning approach from the previous chapter and adding some refinements. We then evaluate the results of the enhanced heuristic.

4.2 Heuristic Solution Revisited

1. Enhanced Pattern Containment: We developed a new pattern containment checking algorithm that takes into account Cypher queries (not formal queries) and considers edge types. This allows the heuristic to work with real-life query languages and real-life graph structures.

2. Deep Learning for Space Cost Prediction: Our deep learning model allows to predict space cost of any Cypher query. By integrating this model to the heuristic solution, we can choose more effectively views to materialize, which enhances the quality of the selected views.

3. Revised Goodness Calculation Formula:

Given a query workload composed by Cypher queries. The goodness $G(v)$ of a view v is calculated using our following formula:

$$G(v,QW) = \frac{C(v,QW)}{average\ of\ space\ cost\ of\ each\ edge\ e\ \in\ v} \times Coverage\ factor$$

where:

- $C(v, QW)$: the number of covered edges by $v \in QW$.
- The average of space cost of each edge $e \in v$ is calculated with:

$$\frac{(space\ cost\ of\ the\ v) \times 200}{total\ number\ of\ edges\ of\ v}$$

Here, the factor of 200 is utilized to scale the space cost, reflecting the estimation from the sampled database, which is 0.5% of the original database.

- The coverage factor is calculated with:

$$\frac{coverage\ of\ all\ views\ selected\ with\ the\ current\ view}{number\ of\ all\ edges\ in\ the\ QW}$$

which normalizes the sum of coverage for all views selected with the current view by the total number of edges in the workload.

This formulation integrates the coverage factor into the goodness calculation, ensuring that views are evaluated in the context of their collective impact on coverage and cost, thereby addressing the limitation of isolated computation and avoiding biases in view selection. Specifically, this approach prevents the heuristic from selecting views that have maximum coverage of the workload without considering the edges already covered by previously selected views.

4. Fixing Upper Bound for Cost Storage: For any query workload QW, the worst case consists in materializing all single and distinct edges that compose QW. For each edge we can estimate its space cost using our deep learning model. Thus, the sum of all space costs related to these single and distinct edges represents the maximum space cost that can be used for the VSP. This sum specifies our upper bound cost. That is, if the user determines a space bound that exceeds our upper bound cost, then the heuristic will be run using our upper bound cost to avoid an infinite execution as done by the original heuristic. This improvement could not be done by the original version of Xin Wang as no deep learning model has been proposed for view cost estimation.

5. Final revisited version: After algorithmic and deep-learning enhancements done over the heuristic solution of Xin Wang, the final version of the revisited heuristic is given in **Figure 4.1.**

$\boldsymbol{Ie}$: $the\ different\ views\ with\ single\ edges\ and\ their\ "goodness"\ values$
$\boldsymbol{Ep}$: $number\ of\ edges\ in\ the\ workload$

$\boldsymbol{Y}(\boldsymbol{V})$: *total space cost of views in* V
$\boldsymbol{V}$: *list of selected views*
$\boldsymbol{uncvd}$: *number of uncovered edges of the workload*
$\boldsymbol{g}(\boldsymbol{v})$: *goodness of the view* v
$\boldsymbol{Covr}(\boldsymbol{V})$: *number of edges covered by selected views in* V
$\boldsymbol{upB}$: *upper bound space cost*

Input: Workload $\boldsymbol{Q} = \{\boldsymbol{Qi}, \bullet\bullet\bullet, \boldsymbol{Qn}\}$, *and a space cost bound* $\boldsymbol{B}$.

Output: A set $\boldsymbol{V}$ *of selected views.*

1. *Initialize* $Ep, Ie, uncvd = 0$; *set* $Vm := NULL, upB = 0$
2. *sort* Ie *by the goodness value(descending)*
3. $V = [Ie[0]]$
4. $upB = SpaceCost(Ie)$
5. *if* $upB < B$ *then*
6. $B = upB$
7. *while* $uncvd > 0 \wedge Y(V) \leq B$:
8. *pick a view* $Ve \in Ie$ *with largest* $g(Ve)$ *and which is not in* V;
9. $max := 0$;
10. *for each* $Vj \in V$ *do*
11. *generate* $V'j$ *by enlarging* Vj *with all possible single edges*;
12. *compute* $g(V'j), \Delta g(V'j) = g(V'j) - g(Vj)$;
13. *if* $max < \Delta g(V'j)$ *then*
14. $max := g(V'j)$;
15. $Vm := V'j$;
16. *if* $g(Ve) > max$ *then*
17. $V.append(Ve, g(Ve))$;
18. $uncvd = Ep - C(V)$;
19. *else*
20. *update* V *by replace the view before enlarging with* Vm;
21. $uncvd = Ep - C(V)$;
22. *return* V

The places in code that are revisited

Figure 4.1 Revisited parts of the original heuristic

The algorithm employs a systematic approach to view selection, balancing the goodness of views, their coverage, and the space cost constraints. By iteratively evaluating views based on their goodness values and expanding them strategically to maximize coverage

while minimizing space costs, it aims to optimize materialization decisions effectively. This ensures that the selected views collectively provide comprehensive coverage of the workload while adhering to resource constraints, ultimately enhancing query performance and reducing computational overhead.

4.3 Brute-force Solution

We implemented a brute-force algorithm that extracts best views to materialize by examining an exponential number of all possible views. This algorithm is intractable in practice, but we implemented it only in order to compare its efficiency with that of our deep-learning based heuristic.

$\boldsymbol{Ep}$: $number\ edges\ \in Workload$

$\boldsymbol{Max}$: $number\ of\ edges\ contains \in\ the\ largest\ query\ from\ Workload.$

$\boldsymbol{singles}$: $different\ views\ with\ single\ edges$

$\boldsymbol{Bv}$: $best\ selected\ views$

$\boldsymbol{Vv}$: $set\ of\ \ tables\ of\ combined\ views$

Input: $Workload\ \boldsymbol{Q} = \{\boldsymbol{Qi}, \bullet\bullet\bullet, \boldsymbol{Qn}\}$.

Output: $A\ set\ of\ view\ definitions\ \boldsymbol{Bv}$.

1. $Initialize\ singles, Max, Ep, V = [\square], Ve = [\square], Vv = [[\,]], Bv = NULL.$
2. $singles = extractSingleViews(Q).$
3. $V.append(singles)$
4. $For\ each\ i \in [1, \dots, Max]:$
5. $\quad Ve = enlarge\ each\ view \in V\ by\ all\ possible\ single\ edges.$
6. $\quad V.append(Ve)$
7. $length = ranging(1, len(singles))$
8. $for\ each\ Cv \in Combine\ (V, length)$
9. $\quad Vv.append(Cv)$
10. $min = +\infty$
11. $for\ each\ cv \in Vv\ do:$
12. $\quad if\ Covr(cv, Q) = Ep \wedge Y(cv) < min\ then:$
13. $\quad\quad Bv = cv$
14. $\quad\quad min = Y(cv)$
15. $return\ Bv$

Figure 4.2 Brute-force solution Algorithm

The algorithm begins by extracting all different single edges from the workload and each one represents a single view. Then, it enlarges them to generate all possible views of lengths varying from 1 to Max. It then examines all possible combinations of these views, with lengths ranging from one to the total number of single-edge views. Each combination is stored in the set of lists Vv. Finally, the algorithm identifies the combination of views in Vv that covers all the edges of the workload Q while minimizing the space cost.

As known before, we observed that the algorithm requires considerable time to process even small workloads. Therefore, we decided to use multithreading in order to optimize its execution time.

Input: $Workload\ \boldsymbol{Q} = \{\boldsymbol{Qi}, \bullet\bullet\bullet, \boldsymbol{Qn}\}$.

Output: $A\ set\ of\ view\ definitions\ \boldsymbol{Bv}$.

1. $Initialize\ Ep, V = [\square], Ve = [\square], Vv = [[\,]], Vc = [[\,]], Bv = NULL$.
2. $singles = extractSingleViews(Q)$.
3. $V.append(singles)$
4. $For\ each\ i \in [1, \ldots, Max]$:
5. $\quad Ve = enlarge\ each\ view \in V\ by\ all\ possible\ single\ edges.$
6. $\quad V.append(v\ in\ Ve\ if\ Covr(v, Q) \geq 1)$
7. $length = ranging(1, len(singles))$
8. $for\ each\ Cv \in Combine\ (V, length)$
9. $\quad Vv\ .append(Cv)$
10. $\boldsymbol{Threads}\{$
11. $for\ each\ cv \in Vv\ do$:
12. $\quad if\ Covr(cv, Q) = Ep\ then$:
13. $\quad\quad Vc.append(cv)$
14. $\}$
15. $min = +\infty$
16. $for\ each\ cv \in Vc\ do$:
17. $\quad if\ Y(cv) < min\ then$:
18. $\quad\quad Bv = cv$
19. $\quad\quad min = Y(cv)$
20. $return\ Bv$

Figure 4.3 Brute-force solution Algorithm (With parallel version)

The main process of this algorithm involves adding a condition to combine only those views that cover at least one edge from the workload. Additionally, we use multithreading to extract combinations from all combinations of views that cover all workload edges. As a result, we achieve a significant reduction in execution time compared to previous algorithms.

4.4 Our Implementation

We first summarize all technologies and tools that have been used in this project then we describe our GUI dedicated for the VSP of graph databases.

4.4.1 Technologies and Tools

a) Programming Language

The primary programming language used for this project is Python (3.10.6) [23]. Python was selected for its extensive libraries, ease of use, and strong community support. The following libraries and frameworks have been used:

- **Matplotlib (3.7.1)**
  Used for plotting and visualizing results.

- **Numpy (1.23.1)**
  Employed for numerical computations and handling multi-dimensional arrays.

- **Pandas(1.5.2)**
  Utilized for data manipulation and analysis.

- **Sklearn (1.2.1)**
  Applied for machine learning algorithms and model evaluation.

- **Pytorch (2.2.1)**
  Utilized for one-hot encoding and other deep learning tasks.

- **Tensorflow (2.11.0)**
  Used for building and training deep learning models.

- **Keras (2.11.0)**
  Integrated with TensorFlow for importing and training models.

- **Itertools**
  Used for creating iterators for efficient looping.

- **Gradio (4.7.1)**
  Used to create interactive user interfaces.

- **Joblib (1.3.2)**
  Employed for saving and loading machine/Deep learning models.

- **Os**
  Used for reading files and interacting with the operating system.

- **Re**
  Used for regular expression operations in Natural Language Processing (NLP) tasks.

b) Tools and Environments

The development environment for this project included:

- **Jupyter Notebook**
  Used for interactive development and testing of code.

- **VS Code**
  Utilized for efficient coding and debugging.

- **Google Colab**
  Used for running extensive computations using Google's cloud resources.

- **Neo4j**
  Used as a graph database for storing and querying data.

- **AuraDB**
  A managed Neo4j database service.

- **AuraDS**
  Used for data science and analytics on graph data within the Neo4j ecosystem.

4.4.2 A Simple GUI for The VSP

Using Gradio [24], we developed a simple interface that allows the use of our solution for the VSP of graph databases. The interface allows user to: (a) input a Cypher query workload (as text or CSV file); and (b) specify the space bound in MB. Then, it returns the best views to materialize as well as the coverage rate and the space cost of these views.

Figure 4.4 shows our interface with Example 4.1 given in Section 4.5. The example was run with a space bound fixed to 3000 MB. The result given is composed by the views {V1, V2} that have been detailed in Example 4.1 of Section 4.5.

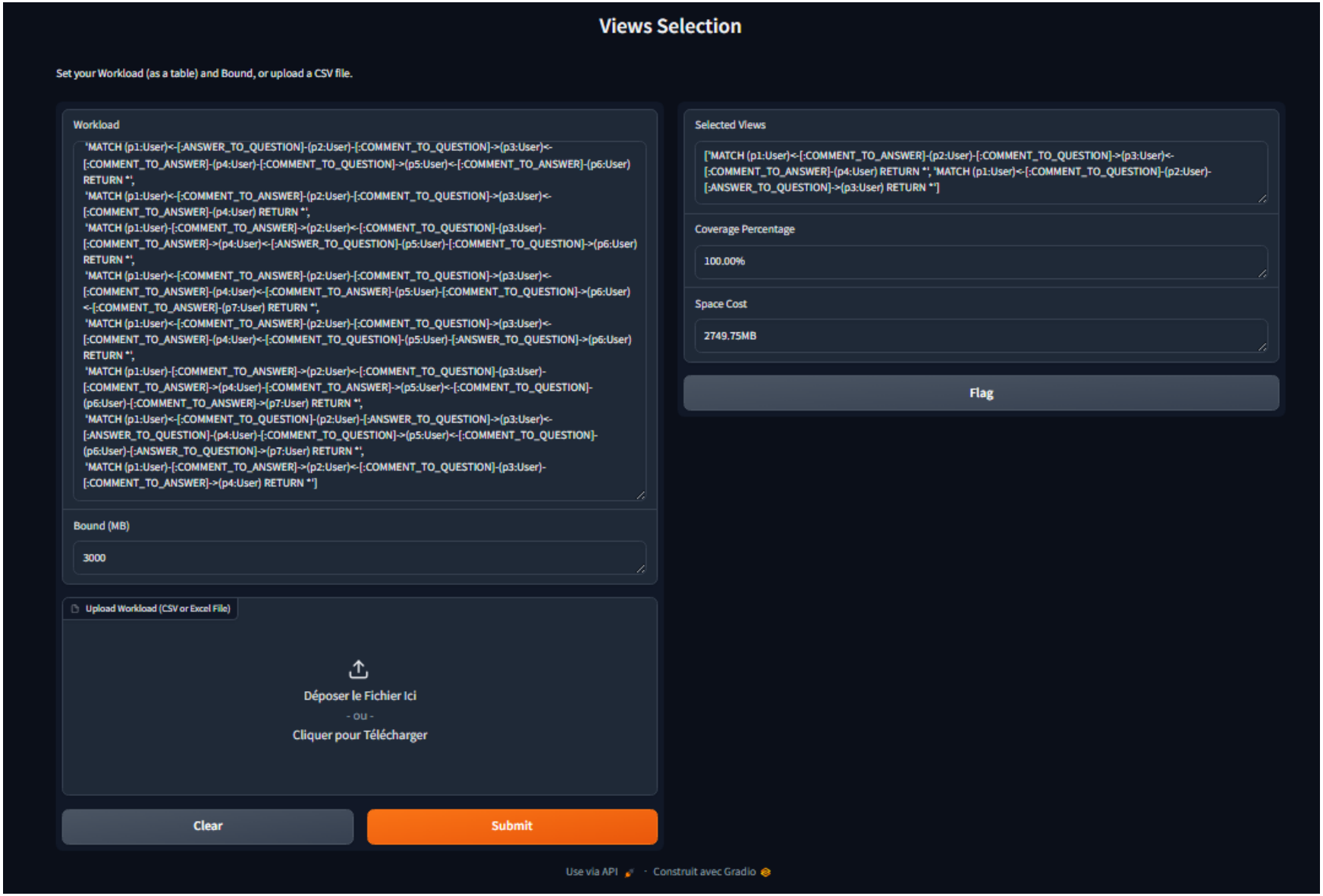


Figure 4.4 Our GUI for VSP

4.5 Experimental Study

In our comparison, we observe that our deep-learning based heuristic provides comparable results to the optimal solution generated by the naive algorithm.

Example 4.1:

Consider the following workload of Cypher queries:

Query Workload

Q1:

(p1:User)<-[:ANSWER_TO_QUESTION]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:ANSWER_TO_QUESTION]-(p4:User)-[:COMMENT_TO_QUESTION]->(p5:User)<-[:ANSWER_TO_QUESTION]-(p6:User)-[:COMMENT_TO_QUESTION]->(p7:User)

Q2:

(p1:User)<-[:ANSWER_TO_QUESTION]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_ANSWER]-(p4:User)-[:COMMENT_TO_QUESTION]->(p5:User)<-[:COMMENT_TO_ANSWER]-(p6:User)

Q3:

(p1:User)<-[:COMMENT_TO_ANSWER]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_ANSWER]-(p4:User)

Q4:

(p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)<-[:ANSWER_TO_QUESTION]-(p5:User)-[:COMMENT_TO_QUESTION]->(p6:User)

Q5:

(p1:User)<-[:COMMENT_TO_ANSWER]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_ANSWER]-(p4:User)<-[:COMMENT_TO_ANSWER]-(p5:User)-[:COMMENT_TO_QUESTION]->(p6:User)<-[:COMMENT_TO_ANSWER]-(p7:User)

Q6:

(p1:User)<-[:COMMENT_TO_ANSWER]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User)<-[:COMMENT_TO_ANSWER]-(p4:User)<-[:COMMENT_TO_QUESTION]-(p5:User)-[:ANSWER_TO_QUESTION]->(p6:User)

Q7:

(p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)-[:COMMENT_TO_ANSWER]->(p5:User)<-[:COMMENT_TO_QUESTION]-(p6:User)-[:COMMENT_TO_ANSWER]->(p7:User)

Q8:

(p1:User)<-[:COMMENT_TO_QUESTION]-(p2:User)-[:ANSWER_TO_QUESTION]->(p3:User)<-[:ANSWER_TO_QUESTION]-(p4:User)-[:COMMENT_TO_QUESTION]->(p5:User)<-[:COMMENT_TO_QUESTION]-(p6:User)-[:ANSWER_TO_QUESTION]->(p7:User)

Q9:

(p1:User)-[:COMMENT_TO_ANSWER]->(p2:User)<-[:COMMENT_TO_QUESTION]-(p3:User)-[:COMMENT_TO_ANSWER]->(p4:User)

Selected Views

V1:

- MATCH (p1:User)<-[:ANSWER_TO_QUESTION]-(p2:User)-[:COMMENT_TO_QUESTION]->(p3:User) RETURN *

V2:

- MATCH (p4:User)<-[:COMMENT_TO_ANSWER]-(p3:User)-[:COMMENT_TO_QUESTION]->(p2:User)<-[:COMMENT_TO_ANSWER]-(p1:User) RETURN *

Both our solution and brute-force solution yield the same solution **{V1, V2}**. These selected views cover the entire workload with a space cost of 2.7 GB, resulting in a 25% reduction in storage space compared to the original database total cost.

From this example, we observe that our solution provides satisfactory results comparable to those obtained from the brute-force approach. To ensure its effectiveness in view selection and materialization, we must experiment with different workloads.

4.4.3 Experimental Settings

a) Database Description

We conducted experiments using the Stack Overflow database, which comprises 63,497,050 edges and 2,601,977 nodes. The database contains three types of edges: "COMMENT_TO_ANSWER" , "ANSWER_TO_QUESTION" , and "COMMENT_TO_QUESTION." All nodes are labeled as "User" reflecting user interactions within the Stack Overflow platform.

b) Query Workload:

Our experimental study was conducted using three query workloads, each differing in the number of queries:

1. **100 Queries**
2. **1,000 Queries**
3. **1 million Queries**

Each query within these workloads contains between one and six edges. This variability in the number of edges allows for a comprehensive assessment of the performance and scalability of our approach across different query complexities.

c) Our Machine Settings

The experiments were conducted on a machine with the following specifications:

Processor: Intel Core i7, 7th Generation, 2.70 GHz

RAM: 16 GB, 2200 MHz

Storage: 256 GB SSD + 1TB HDD

4.4.4 Implemented Algorithms

The following list contains all algorithms that have been implemented in this work:

- **Data Sampling Algorithm:** This algorithm is responsible for sampling graph databases to create a simplified representation suitable for data collection. It uses a custom random walk process to efficiently sample the graph while preserving its essential properties.

- **Revisited Heuristic:** Our heuristic algorithm builds upon Xin Wang's approach to view selection and materialization. It incorporates enhancements such as pattern containment, trained models for space cost prediction, and upper bound selection to improve the efficiency and effectiveness of view selection decisions.

- **Naive Algorithm:** A straightforward approach that exhaustively explores all possible views to find the optimal solution. It selects views that cover all edges in the workload while minimizing the space cost.

- **Cypher Queries Workload Generator:** This component generates a workload of Cypher queries, simulating real-world query patterns that the view selection algorithms must optimize for. The workload includes queries of varying complexities and patterns, reflecting the diversity of queries typically encountered in practice.

4.4.5 Experimental Sets

The experimentation involved comparing the performance of the heuristic and naive algorithms in terms of execution time and goodness (coverage/space cost) across different numbers of queries in the workload. Three key curves were plotted to visualize these comparisons:

a) Checking efficiency of our solution: by comparing its time with that of the brute-force solution

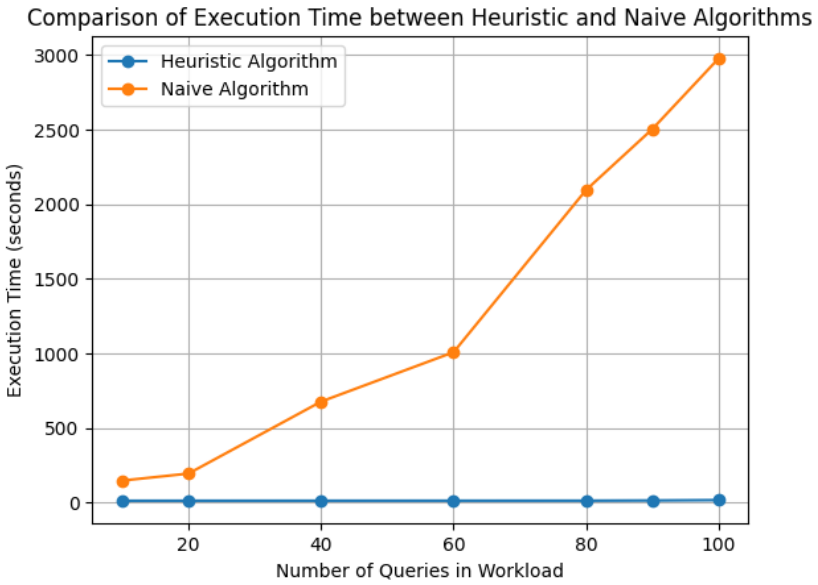


Figure 4.5 Comparison of Execution Time between the Heuristic and the brute-force solution

This curve illustrates the time taken by both algorithms to execute as the number of queries in the workload varies. The heuristic algorithm consistently demonstrated significantly faster execution times compared to the naive algorithm. The curve for the heuristic algorithm shows a rapid response, highlighting its ability to quickly process queries and select views.

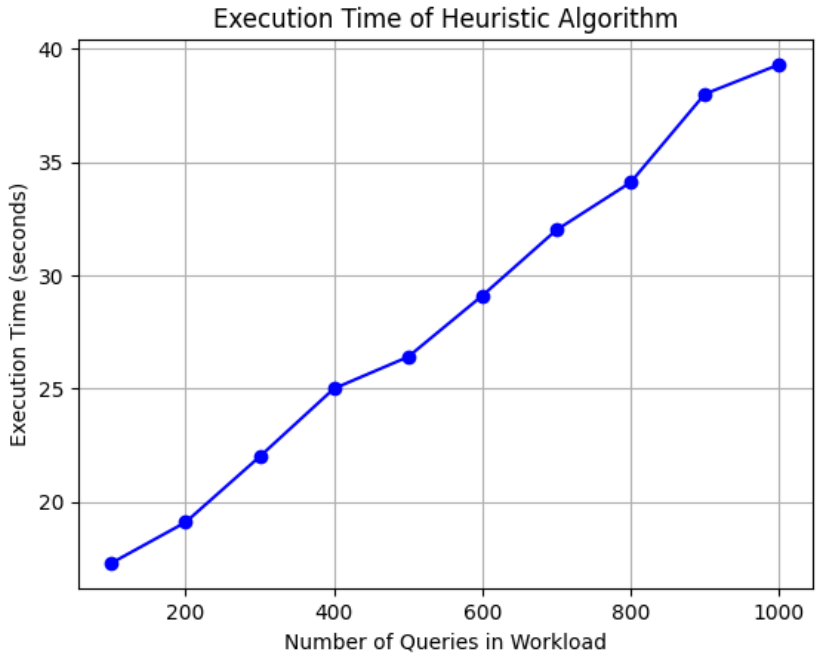


Figure 4.6 Execution Time of Heuristic Algorithm

This curve specifically illustrates the execution time of the heuristic algorithm, highlighting its speed and efficiency as the workload size increases. The curve demonstrates a steady execution pattern, indicating the heuristic algorithm's capability to manage real-life workloads in a practical time. We successfully executed a workload containing 1 million queries, and the algorithm responded in only 178 minutes.

b) Checking effectiveness of our solution: by comparing its goodness with that of the brute-force solution

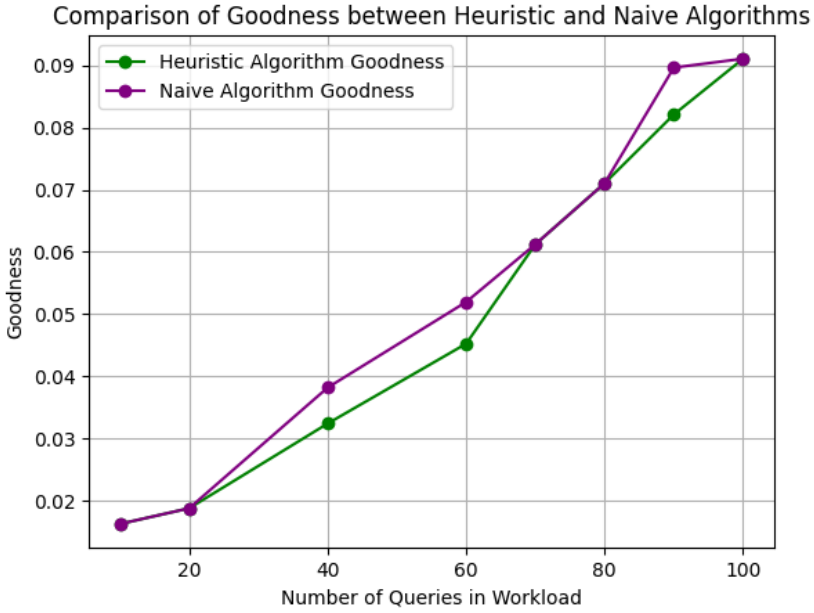


Figure 4.7 Comparison of Goodness between the Heuristic and the brute-force solution

The goodness curve for both algorithms shows their respective performance in terms of coverage and space cost. Despite their different techniques, both algorithms often yielded similar goodness values. There were instances where the goodness values overlapped or were very close, indicating comparable performance in optimizing view selection decisions.

4.6 Conclusion

Our experiments underscored the efficiency of the heuristic algorithm in optimizing view selection decisions, as evidenced by its superior performance in terms of execution time. Moreover, the comparable goodness values obtained by both algorithms proved the heuristic algorithm's capability to provide effective solutions in view-based query

optimization scenarios. These findings validate the effectiveness of the proposed heuristic and highlight its potential for enhancing query performance in real-world database environments.

General Conclusion

In this work, we have addressed the views selection problem for graph databases that consists in determining which parts of user queries are frequently asked in order to materialize them.

The problem has been widely studied for relational databases, while graph databases have received less attention according to this problem. As only one solution has been addressed to this problem, the Xin Wang heuristic solution, our goal consisted first in enhancing algorithmically the Xin's solution by addressing its limitations such as the lack of a real-life language, its limited applicability to certain graph structures, the lack of a model for calculating storage costs, the overlapping views, and the possibility of non-termination execution. Next, to deal with the space cost estimation, our idea was to integrate a machine/deep learning technique to the heuristic solution. However, the large size of real-life data graphs required sampling our data graph first before proceeding to a data collection process. We remarked that existing data sampling algorithms for graph databases are still in their infancy and encounter several limitations like inadequate coverage and high time complexity. To overcome this problem, we proposed our data sampling algorithm that adopts the random walk principle and allows extracting a sampled version of data in an efficient manner. Next, we conducted a data collection process, and we compared several machine learning and deep learning models. We concluded that the MLP model is the best as it allows an accurate space cost prediction with lower computational overhead. Finally, we integrated our deep learning model into the heuristic solution along with some algorithmic enhancements.

Using real-life data graphs, we compared our deep-learning-based solution with the naive approach. The experimental results show that our solution outperforms the naive solution in terms of result accuracy and execution time.

This work is the first one that integrates deep learning into the views selection problem under graph data. The results found are simple but very promising. As future directions, we plan to:

- Develop a Deep reinforcement learning solution and compare it to our solution.

- Propose a complete solution combining views selection, views materialization, and views-based query answering.

- Develop more advanced graph sampling techniques and integrating them with our approach.

In conclusion, this work provides valuable insights and practical solutions for optimizing query performance in modern database systems. The proposed framework not only

advances the current state of view-based query optimization but also pave the way for future research and practical implementations, contributing to the ongoing evolution of database management technologies.

References

[1] NEO4J Graph Database & Analytics | Graph Database Management System. Visited on march 18th ,2024. [Online]. Available at: https://neo4j.com.

[2] Bonifati A, Fletcher G. P, Voigt H, & Yakovets, N. Querying graphs. Synthesis Lectures on Data Management, 10(3), 1–184., N: https://doi.org/10.2200/s00873ed1v01y201808dtm051, 2018.

[3] Database using Cypher.Visited on March 20th,2024[Online]. Available at: https://neo4j.com/docs/getting-started/cypher-intro.

[4] OpenCypher, Visited on March 20th,2024[Online]. Available at: https://opencypher.org.

[5] Wang X, Liu X, Chen Y, Zhong X, Cheng P. View selection for graph pattern matching. DEXAS'2020, (pp. 93–110).

[6] Troullinou G, Kondylakis H, Lissandrini M, & Mottin D. SOFOS: Demonstrating the challenges of materialized view selection on knowledge graphs. arXiv, 2021. Available at: https://doi.org/10.48550/arXiv.2103.06531.

[7] X. Wang.Answering Graph Pattern Matching Using Views: A Revisit. In 30th International Conference on Data Engineering (ICDE). 2017.

[8] Han Y, Li G, Yuan H, & Sun J. An Autonomous Materialized View Management System with Deep Reinforcement Learning .presented at: the 37th International Conference on Data Engineering (ICDE) in 2021.

[9] Yuan H, Li G, Feng L, Sun J, & Han Y. Automatic View Generation with Deep Learning and Reinforcement Learning. presented at the 36th IEEE International Conference on Data Engineering (ICDE) in 2020.

[10] Zhang X, Ni Y, Li S, Gao G, Fang L, Wang Y, Zhao Y, & Zhou Z .A survey of large graph sampling techniques In Journal of Computer-Aided Design & Computer Graphics ,p. 1805–1814. 2022.

[11] L. Zhang. Graph sampling: An introduction. The Survey Statistician, 83, 27–371. 2021.

[12] Graph Data Science Library Manual v2.6 - NEO4J Graph Data Science. (n.d.). Neo4j Graph Data Platform, Visited on April 12th,2024 [Online]. Available at: https://neo4j.com/docs/graph-data-science.

[13] Random walk with restarts sampling - Neo4j Graph Data Science. (n.d.). Neo4j Graph Data Platform., Visited on April 12th,2024 [Online]. Available at: https://neo4j.com/docs/graph-data-science/current/management-ops/graph-creation/sampling/rwr/.

[14] Common Neighbour Aware Random Walk sampling - Neo4j Graph Data Science. (n.d.). Neo4j Graph Data Platform. , Visited on April 12th,2024 [Online]. Available at: https://neo4j.com/docs/graph-data-science/current/management-ops/graph-creation/sampling/cnarw/.

[15] Wang R, Li Y, Lin S, Wu W, Xie H, Xu Y, & Lui.Common Neighbors Matter: Fast Random Walk Sampling with Common Neighbor Awareness. IEEE Transactions on Knowledge and Data Engineering, 1, 2022.

[16] Neo4j - hardware sizing calculator. Visited on March 28th,2024 [Online]. Available at: https://neo4j.com/hardware-sizing-calculator/.

[17] C.-A. Azencott, Introduction au Machine Learning. Paris, Dunod, 2019.

[18] M. J. J. Douglass, Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd edition by Aurélien Géron, 2020.

[19] Bentéjac, Candice & Csörgő, Anna & Martínez-Muñoz, Gonzalo. A Comparative Analysis of XGBoost.In 11th BRICS Summit. 2019.

[20] S. Weidman, Deep Learning from Scratch: Building with Python from First Principles, O'Reilly Media., 2019.

[21] He Z., Yu J., & Guo B. Execution time prediction for cypher queries in the NEO4J database using a learning approach.In 56th Annual Meeting of the National Association of Medical Examiners, 14(1), 55. 2022.

[22] Leskovec J. (2017). SNAP: Network datasets: Stack Overflow temporal network. Visited on February 28th,2024 [Online]. Available at: https://snap.stanford.edu/temporal-motifs/data.html.

[23] Van Rossum G. (2007). Python programming language. In USENIX Annual Technical Conference. Available at: https://www.python.org.

[24] Abid Ab, Abdalla A, Abid Al, Khan D, Alfozan A, Zou J. Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild (2019). Visited on April 20th,2024[Online]. Available at: https://www.gradio.app.

[25] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Belle-mare MG, Graves A, Riedmiller MA, Fidjeland A, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature 518(7540):529–533. 2015.

Abstract

With the exponential growth of Web data, relational databases reach rapidly their limits when it comes to handling highly interconnected data, and lead to inefficiency with complex join-queries. Graph databases, with their flexible data modeling and efficient handling of complex relationships and queries, emerge as a potent solution. When examining user queries, one can remark that some parts of these queries are frequently requested, and then materializing these parts allows for efficient querying of large graph databases. These parts are called views and the process that aims to find which parts (views) to materialize is called the Views selection problem (VSP). Contrary to graph databases, the VSP has been widely studied for relational databases. In this thesis, we studied the VSP for graph databases and using Cypher queries. We first study some heuristic solution for the VSP and revise it by proposing significant algorithmic enhancements. Secondly, we propose a deep learning model that allows estimating space cost of any view to materialize. This model allows to enhance the quality of the heuristic by selecting views that make a compromise between space cost and coverage rate. Through comprehensive experiments based on real-life data, we demonstrated that our enhanced heuristic allows selecting views of good quality in terms of space cost and coverage. This work is the first one that investigates deep learning approaches for the VSP.

KEYWORDS: Graph databases, Views selection problem, Cypher queries, Heuristic algorithms, Deep learning model, Space cost, Coverage rate, Neo4j.

Résumé

Avec la croissance exponentielle des données Web, les bases de données relationnelles atteignent rapidement leurs limites lorsqu'il s'agit de gérer des données hautement interconnectées, conduisant à des inefficacités avec des requêtes de jointure complexes. Les bases de données graphes, avec leur modélisation flexible des données et leur gestion efficace des relations et des requêtes complexes, émergent comme une solution puissante. Lors de l'examen des requêtes des utilisateurs, on remarque que certaines parties de ces requêtes sont fréquemment demandées et la matérialisation de ces parties permet une interrogation efficace des grandes bases de données graphes. Ces parties sont appelées vues et le processus visant à déterminer quelles parties (vues) matérialiser est appelé le problème de la sélection des vues (VSP). Contrairement aux bases de données graphes, le VSP a été largement étudié pour les bases de données relationnelles. Dans cette thèse, nous avons étudié le VSP pour les bases de données graphes en utilisant des requêtes Cypher. Nous avons d'abord étudié une solution heuristique pour le VSP et l'avons révisée en proposant des améliorations algorithmiques significatives. Ensuite, nous proposons un modèle d'apprentissage profond permettant d'estimer le coût spatial de toute vue à matérialiser. Ce modèle permet d'améliorer la qualité de l'heuristique en sélectionnant des vues qui font un compromis entre le coût spatial et le taux de couverture. Grâce à des expériences complètes basées sur des données réelles, nous avons démontré que notre heuristique améliorée permet de sélectionner des vues de bonne qualité en termes de coût spatial et de couverture. Ce travail est le premier à explorer des approches d'apprentissage profond pour le VSP.

MOTS-CLÉS : Bases de données graphes, Problème de la sélection des vues, Requêtes Cypher, Algorithmes heuristiques, Modèle d'apprentissage profond, Coût spatial, Taux de couverture, Neo4j.

ملخص

مع النمو المتسارع للبيانات على الويب، تصل قواعد البيانات الكلاسيكية (BDD relationnelle) بسرعة إلى حدودها عندما يتعلق الأمر بمعالجة البيانات المترابطة بشكل كبير، مما يؤدي إلى تباطئ في إستخراج البيانات. من بين الحلول المتاحة هو تصميم قواعد بيانات على شكل شبكة (BDD orientée graphe) تظهر المعلومات (noeuds)و العلاقات (arcs) التي تربطها. يمكن ملاحظة أن بعض أجزاء هذه المعلومات يتم طلبها بشكل متكرر، وتسمح عملية تخزين هذه الأجزاء على حدى باستعلام فعال لقاعدة البيانات. تُسمى هذه الأجزاء بالعروض، والعملية التي تهدف إلى تحديد أي أجزاء (العروض) يجب تخزينها تُسمى مشكلة اختيار العروض .(VSP) على عكس قواعد البيانات الشبكية، تم دراسة مشكلة اختيار العروض بشكل واسع في قواعد البيانات الكلاسيكية . في هذه الأطروحة، درسنا مشكلة اختيار العروض لقواعد البيانات الشبكية. أولاً، درسنا حلاً تجريبيًا لمشكلة اختيار العروض وقمنا بمراجعته من خلال اقتراح تحسينات خوارزمية. ثانيًا، نقترح نموذجًا للتعلم العميق يسمح بتقدير حجم (mémoire) لأي عرض ليتم تخزينه. يتيح هذا النموذج تحسين جودة الحل التجريبي من خلال اختيار العروض التي تحقق توازنًا بين الحجم ومعدل التغطية. من خلال تجارب شاملة تعتمد على بيانات حقيقية، أظهرنا أن الحل التجريبي المحسن لدينا يتيح اختيار عروض ذات جودة جيدة من حيث الحجم والتغطية. هذا العمل هو الأول الذي يستكشف نهج التعلم العميق لمشكلة اختيار العروض

الكلمات المفتاحية: قواعد البيانات الشبكية، مشكلة اختيار العروض، Cypher، الخوارزميات التجريبية، نموذج التعلم العميق، تقدير الحجم، معدل التغطية، Neo4j.