NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Tractable View-Based Query Rewriting for Knowledge Graphs

Konstantinos G. Charmantaris

Supervisors: **Dr. Miltiadis Kyriakakos,** Laboratory Teaching Staff
Dr. Theofilos Mailis, Researcher

ATHENS

July 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επανεγγραφή Ερωτημάτων βασισμένη σε όψεις για γράφους γνώσης

Κωνσταντίνος Γ. Χαρμαντάρης

Επιβλέποντες: **Δρ. Μιλτιάδης Κυριακάκος,** Εργαστηριακό Διδακτικό Προσωπικό
Δρ. Θεόφιλος Μαΐλης, Ερευνητής

ΑΘΗΝΑ

Ιούλιος 2024

BSc THESIS

Tractable View-Based Query Rewriting for Knowledge Graphs

Konstantinos G. Charmantaris

S.N.: 1115201900213

SUPERVISORS: **Dr. Miltiadis Kyriakakos,** Laboratory Teaching Staff
Dr. Theofilos Mailis, Researcher

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επανεγγραφή Ερωτημάτων βασισμένη σε όψεις για γράφους γνώσης

Κωνσταντίνος Γ. Χαρμαντάρης
Α.Μ.: 1115201900213

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δρ. Μιλτιάδης Κυριακάκος,** Εργαστηριακό Διδακτικό Προσωπικό
Δρ. Θεόφιλος Μαΐλης, Ερευνητής

ABSTRACT

In this work, we propose an efficient technique for view-based query rewriting for knowledge graphs represented in relational databases. Specifically, we investigate how query rewriting using views can be reduced to the problem of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint* (MNssfKc problem). We show that if we employ the linear cost model for evaluating the execution cost of a query, we can reduce the problem of query rewriting using views to the MNssfKc problem. It should be noted that the latter reduction allows to solve the MNssfKc and thus the view materialization problem in polynomial time in the size of the query with an $(1 - e^{-1})$ approximation of the optimal solution.

SUBJECT AREA: Databases

KEYWORDS: Knowledge graph, RDF graph, Query rewriting, View selection, Knapsack problem

ΠΕΡΙΛΗΨΗ

Σε αυτή την πτυχιακή εργασία, προτείνουμε μια αποδοτική τεχνική για την αναδιατύπωση ερωτημάτων βασισμένη σε όψεις για γραφήματα γνώσης που αναπαρίστανται σε σχεσιακές βάσεις δεδομένων. Συγκεκριμένα, διερευνούμε πώς η αναδιατύπωση ερωτημάτων χρησιμοποιώντας όψεις μπορεί να μειωθεί στο πρόβλημα της *Μέγιστης Υποκανονικής Συνάρτησης με Περιορισμό Σακιδίου* (MNssfKc problem). Δείχνουμε ότι αν χρησιμοποιήσουμε το γραμμικό μοντέλο κόστους για την αξιολόγηση του κόστους εκτέλεσης ενός ερωτήματος, μπορούμε να μειώσουμε το πρόβλημα της αναδιατύπωσης ερωτημάτων χρησιμοποιώντας όψεις στο MNssfKc πρόβλημα. Θα πρέπει να σημειωθεί ότι η συγκεκριμένη μείωση επιτρέπει την επίλυση του MNssfKc και συνεπώς του προβλήματος σε πολυωνυμικό χρόνο σε σχέση με το μέγεθος του ερωτήματος, με προσέγγιση $(1-e^{-1})$ της βέλτιστης λύσης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Βάσεις Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Γράφοι γνώσης, Γράφοι RDF, Επανεγγραφή ερωτημάτων, Επιλογή όψεων, πρόβλημα Knapsack

ACKNOWLEDGEMENTS

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα ήθελα να ευχαριστήσω θερμά τον κύριο Μιλτιάδη Κυριακάκο και τον κύριο Θεόφιλο Μαΐλη για την στήριξη και την καθοδήγηση τους καθ' όλη τη διάρκεια της εκπόνησης της πτυχιακής μου εργασίας. Μου έδωσαν πλήρη ελευθερία προσφέροντάς μου συνεχή γνώση, κίνητρο και όρεξη. Είναι εξαιρετικοί ερευνητές και πάνω από όλα άνθρωποι.

CONTENTS

1. INTRODUCTION

1.1 Introduction

Knowledge Graphs (KGs) are powerful tools for data integration [4, 24], information retrieval [8], and numerous other domains [7], widely used in industrial applications. KGs provide a structured and interconnected representation of information, enabling efficient data retrieval and insightful data analysis. Prominent KGs include DBpedia [21], Yago [26], Google's Knowledge Graph [25], and Microsoft's Satori [21]. These KGs support a wide range of applications, from enhancing search engine capabilities to enabling advanced semantic web technologies and improving data integration platforms.

Knowledge Graphs consist of interconnected entities, stored as triples of the form (s, p, o), where s represents the subject, p represents the predicate, and o represents the object. These triples can be categorized into three main types: (entity, relation, entity), which denotes relationships between entities; (entity, property, value), which describes properties of entities; and (entity, type, class), which classifies entities into types. This structured format allows for rich semantic queries, enabling users to explore complex relationships and derive meaningful insights from the data.

Efficient query answering techniques are critical for ensuring the scalability and performance of KG-driven software systems. One key technique for enhancing query performance is view materialization. View materialization involves selecting specific views to materialize based on query patterns and cost constraints. A view is essentially a stored query, while a materialized view is the result set of the stored query on a specific database instance. By partially materializing recurring computations in our query patterns, we can later exploit these materializations (precomputations) to efficiently answer upcoming queries, thus reducing query execution time and resource consumption.

In their previous work [15], the authors studied the problem of view selection over KG databases. They examined various view selection techniques that focus on identifying recurring computations within a query workload and materializing these computations for future reuse. Their study involved creating a summary of the initial query workload by employing graph-pattern mining techniques, which helped in identifying common substructures within the queries. Additionally, they explored the relationship between the view selection process and the knapsack problem, a classic optimization problem. Their methodology allowed them to solve the view selection problem on the query workload of DBpedia, which consists of more than a million queries. It should be noted that this was the first attempt to study the problem of view materialization on such a large scale, providing valuable insights into the practical challenges and potential solutions for view selection in large-scale KGs.

In this work, we propose an efficient technique for query rewriting for knowledge graphs represented in a relational store. Specifically, we investigate how query rewriting using views can be reduced to the problem of Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint (MNssfKc problem). By employing the linear cost model for evaluating the execution cost of a query, we demonstrate that the query rewriting problem can be effectively reduced to the MNssfKc problem. This reduction is significant as it allows us to solve the MNssfKc problem, and thus the view materialization problem, in polynomial time with an $(1 - e^{-1})$ approximation of the optimal solution.

Our approach leverages advanced optimization techniques to ensure that the query rewriting process is both efficient and scalable. By transforming the problem into a well-understood mathematical framework, we can apply existing algorithms and theoretical results to achieve high-performance query rewriting. This is particularly important for modern knowledge graphs, which are characterized by their large scale and complexity. Effective query rewriting can significantly reduce the computational burden on KG-driven systems, enabling them to handle large volumes of queries with improved response times and reduced resource utilization.

The contributions of this thesis are twofold. First, we provide a theoretical framework for reducing the query rewriting problem to the MNssfKc problem, offering a novel perspective on the relationship between query rewriting and submodular function maximization. Second, we present practical algorithms for implementing this reduction, demonstrating their effectiveness through theoretical analysis and empirical evaluation. These contributions advance the state-of-the-art in query optimization for knowledge graphs, providing a robust foundation for future research and practical applications in this field.

In summary, our work addresses a critical need in the field of knowledge graph management by providing efficient techniques for view-based query rewriting. By leveraging the power of submodular function optimization, we offer a scalable solution that enhances the performance of KG-driven systems, making them more capable of handling the demands of modern data-intensive applications.

2. PRELIMINARIES

We present some preliminary definitions to formalize the view selection problem on a KG.

2.1 Knowledge Graphs

We first provide a proper definition of a KG and its corresponding queries.

2.2 Knowledge Graph

A **knowledge graph** G is a set of *tripes* of the form (s, p, o), where s stands for *subject*, p for *predicate*, and o for *object*.Triples (s, p, o) are of three kinds: *(entity, relation, entity)*, or *(entity, property, value)*, or *(entity,* `type`*, class)*.

2.3 Conjunctive Query

For

(i) X being a set of variables disjoint from the constants appearing in a graph G (entities, relations, properties, values, and classes);

(ii) $t_1, t_2, \ldots, t_n$ being triple patterns, i.e., extensions of triples that may contain variables in the subject, predicate, or object position,

(iii) $\vec{x}$ being a vector of variables also appearing in the $t_1, t_2, \ldots, t_n$ triple patterns,

a conjunctive query Q on G has the corresponding form:

$$Q\colon q(\vec{x}) \leftarrow t_1, t_2, \ldots, t_n.$$

$q(\vec{x})$ is called the head of the query, while the set $t_1, t_2, \ldots, t_n$ of triple patterns is its body. The variables in the head are called *distinguished variables*, while variables appearing only in the body are called *undistinguished variables*.

2.4 Query answering

A solution to a conjunctive query Q is a mapping m : vars$(Q) \rightarrow C$ from the variables in Q to the constants in G such that the substitution of variables would yield a subgraph of G. The substitutions of distinguished variables constitute the answers to the query.

2.5 View Selection

We now provide some definitions related to the view-selection problem.

2.6 Materialized View

A *view* is a stored query, while a *materialized view* is the result set of the stored query on a specific database instance.

2.7 Query Rewriting

Two queries are equivalent if they have the same answer set for every possible database. A query Q' is a *rewriting* of Q that uses the views $\mathcal{V} = \{V_1, \ldots, V_m\}$ if Q and Q' are equivalent and Q' contains one or more occurrences of materialized views in $\mathcal{V}$. A *rewriting function* Rwrt$(Q, \mathcal{V})$ takes as input the query Q and rewrites it to an equivalent query $Q' =$ Rwrt$(Q, \mathcal{V})$ using views from $\mathcal{V}$. A rewriting function Rwrt is optimal when there exists no other rewriting Q'' of Q such that Cost$^{\epsilon}(Q'') <$ Cost$^{\epsilon}(Q')$, with Cost$^{\epsilon}$ being the function that maps a query to its estimated execution cost.

2.8 Linear Cost Model

In our work, we employ the linear cost model for evaluating the execution cost of a query. The linear cost model assumes that the cost of evaluating a query Q, i.e. Cost$^{\epsilon}$(Q), is proportional to the size of the relational tables appearing in Q. The linear cost model is manifested in [10] while its *linear independence* property is crucial for most of the proofs in this paper.

2.9 Rewriting Benefit

The *degree of benefit* of a rewriting function to a query Q w.r.t. to a set of views $\mathcal{V}$ is defined as

$$\text{Bnft}(Q, \mathcal{V}) = \text{Cost}^{\epsilon}(Q) - \text{Cost}^{\epsilon}(\text{Rwrt}(Q, \mathcal{V})). \tag{2.1}$$

We also denote with Bnft$(\mathcal{Q}, \mathcal{V})$ the benefit of a set of views $\mathcal{V}$ to a query workload $\mathcal{Q}$. It is obvious that the benefit depends on the adopted cost model.

Levy et al. [14] prove that for the conjunctive queries Q and W, there is a rewriting of Q using W iff $\pi_{\emptyset}(Q) \sqsubseteq \pi_{\emptyset}(W)$ i.e., the projection of Q onto the empty set of columns is contained in the projection of W onto the empty set of columns (the projections $\pi_{\emptyset}(Q)$, $\pi_{\emptyset}(W)$ are actually Boolean conjunctive queries). Additionally, they provide the methodology for finding the rewritings of Q based on every containment mapping $\sigma : \pi_{\emptyset}(W) \to \pi_{\emptyset}(Q)$ with $W \in \mathcal{V}$. Given a query Q, a set of views $\mathcal{V}$, and their corresponding materializations, a query optimizer that utilizes the existing view materializations has to:

(i) identify the available rewritings of Q;

(ii) determine the rewriting Q' that is less costly w.r.t. the adopted cost model;

(iii) decide whether it is beneficial to execute Q' instead of Q.

3. SYSTEM ARCHITECTURE

In this chapter we present the architecture of our overall system and its corresponding modules. Figure 3.1 displays the overall architecture of our system for answering queries using views. It should be noted that for the problem of query rewriting, only the *Query Containment Module* is used, but we will present every module of our subsystem for completeness purposes. Our system constitutes of the following modules:

3.1 Query Summarizer Module

The *query summarizer* is responsible for the compact representation of a set of queries. Its main function is that, given a workload of, even millions, queries, it creates a compact representation of the workload by eliminating non-frequent query patterns and adjusting the multiplicities of frequent ones. The later allows for the resolution of various tasks much faster and efficiently, without sacrificing performance. The summaries are created using frequent-pattern mining techniques, which reduce a set of queries to a set of patterns with their corresponding multiplicities. It should be noted that the summarizer normalizes multiplicities in such a way that ensures the same pattern appears in the summary with a multiplicity exactly as many times as it does in the original query workload. The later guarantees that the benefit of a view to the summary equals the view's benefit on the corresponding query workload. A dedicated description of our query summarizer is presented in [15].

3.2 View Selection Module

The view selection module is responsible for, given the summary of a query workload $\mathcal{Q}$, to select the most beneficial views for the corresponding query workload. The view selection module operates in the following steps: I. It translates frequent patterns in the summary to their corresponding equivalent candidate view V; II. For each candidate, it selects the most beneficial one to the summary of the workload $\mathcal{Q}$; III. Since the view V is not selected in isolation, but will augment an existing set of views $\mathcal{V}$, we need to take into account the preexisting views in $\mathcal{V}$ when making our selection. The selection is performed by reduction to the MNssfKc problem. A dedicated description of our query summarizer is presented in [15].

3.3 Query Containment Module

Query containment is a fundamental operation used to expedite query processing in view materialisation and query caching techniques. Our query containment module is bases on the mv-index structure that allows for fast containment checking between a single query and an arbitrary number of stored views. I.e, provided the query of Q and the set of view in $\mathcal{V}$, the containment module performs fast containment checking in order to find the views such that there exists an homomorphism (query containment) from the body of the view to the corresponding query. For the special class of f-graph queries, containment checking is performed in polynomial time. The query containment module can find the corresponding containments in microseconds or less for the containment test against hundreds of

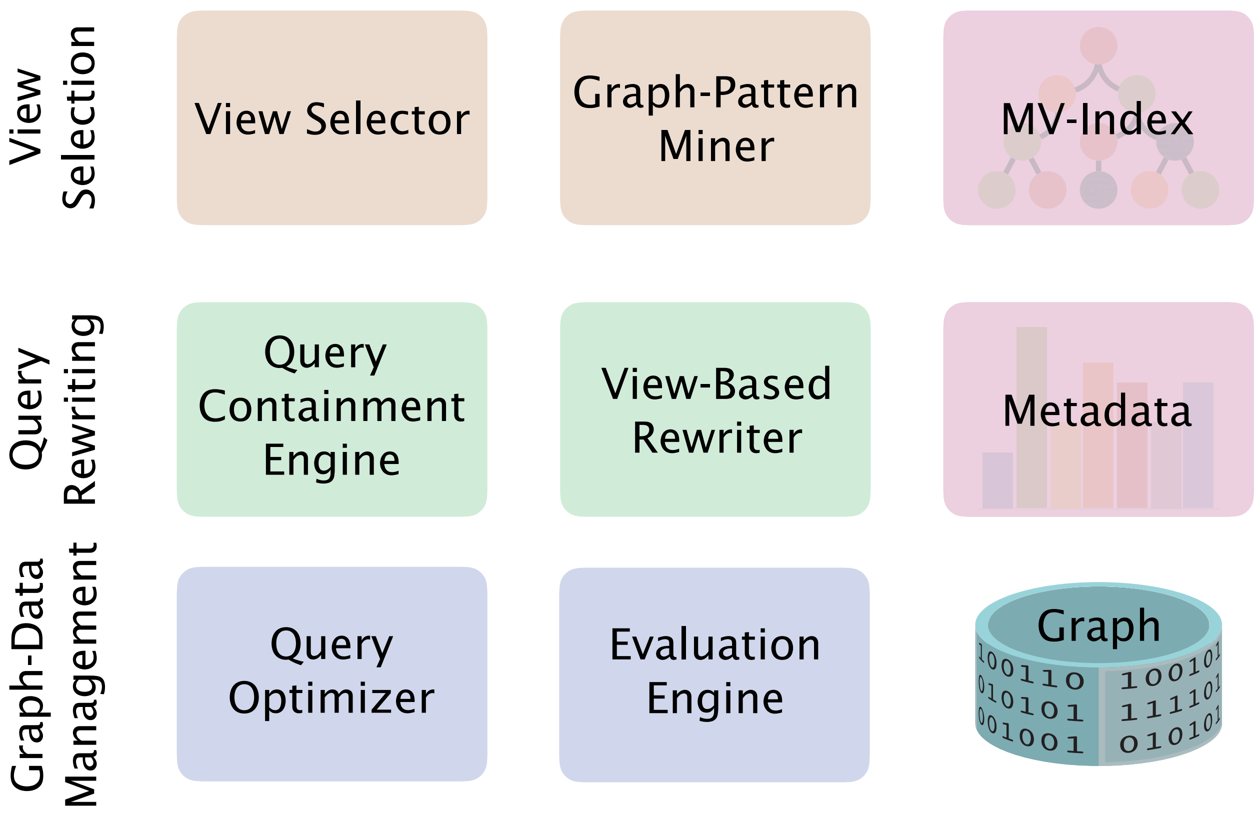

Figure 3.1: Frequent Graph-Pattern Mining with Template Databases

thousands of queries that are indexed within our structure. A dedicated description of our *query containment module* is presented in [17].

4. QUERY REWRITING WITH QUALITY GUARANTEES

In this section we examine the problem of view-based query rewriting.

Definition 1. Given the conjunctive query of Q, where t_is are the triples in Q:

$$Q : q(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} t_i \tag{4.1}$$

and the set of views $\mathcal{V}_Q \in \mathcal{V}$ such that there exists an homomorphism from all $V \in \mathcal{V}_Q$ to $\mathcal{V}$, find the rewriting Q' of Q using the views in $\mathcal{V}_Q$ that minimizes it's corresponding execution cost.

In our work, we employ the linear cost model for evaluating the execution cost of a query. It should be noted that our work focuses on the overall size of the relations and properties appearing in Q. For example, a triple $(?\,x, r, ?\,y)$ has a size that is equivalent to the triples appearing in our knowledge graph with r as their predicate.

4.1 MNssfKc Problem

We will reduce query rewriting to the problem of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint* (MNssfKc problem) presented in [27], i.e., we will identify the parameters of the MNssfKc problem to solve the view selection problem. The latter is a NP-problem, but there exists a $(1-e^{-1})$-approximation polynomial algorithm for solving it.

Problem 1 (MNssfKc [27]). *Let $I = \{1, \ldots, n\}$; $i \in I$ and b be nonnegative integers; and $f(\cdot)$ be a nonnegative, nondecreasing, submodular, polynomially computable set function*[1]*.* MNssfKc *is the following optimization problem:*

$$\max_{S \subseteq I} \left\{ f(S) : \sum_{i \in S} c_i \leqslant b \right\}. \tag{4.2}$$

4.2 Reduction

We first identify the parameters of the reduction from the problem of query rewriting to the MNssfKc problem.

I. For the set $\mathcal{V}_Q := \{V_1, \ldots, V_n\}$ such that $|\mathcal{V}_Q| = n$, we define an arbitrary bijection Bi : $\mathcal{V}_Q \leftrightarrow \{1, \ldots, n\}$ and set $I := \{1, 2, \ldots, n\}$.

II. For each $i \in \{1, \ldots, n\}$ we define $c_i =$ Cost$^{\epsilon}$(Bi$^{-1}(i))$ to be the corresponding view's size, depicting the cost of storing the specific view.

III. Each subset $S \subseteq I$ is mapped via the Bi^{-1} function to a subset of the candidate views $\mathcal{V}'_Q \subseteq \mathcal{V}_Q$ (by mapping each $i \in S$ to Bi$^{-1}(i) \in \mathcal{V}'_Q$). For the specific $\mathcal{V}_Q$', we define $f(S)$ to be the benefit of the materialized views in $\mathcal{V}_Q$' to the rewriting of Q, i.e., Bnft$(Q, \mathcal{V}'_Q)$.

[1]*A set function is (i) submodular if $f(S)+f(T) \geqslant f(S \cup T)+f(S \cap T)$ for all $S, T \in I$, and (ii) nondecreasing if $f(S) \leqslant f(T)$ for all $S \subseteq T$.

IV. Finally, b is the initial execution-cost of the query, i.e., the overall size of the triples appearing in the query.

To complete our reduction, it remains to define the Bnft$(\cdot,\cdot)$ function which obviously depends on the query rewriting algorithm. Our query rewriting algorithm, which is sound and complete for set (and not multiset) semantics, is based on the following observations: For some $V \in \mathcal{V}'_Q$, of the form of:

$$V(\vec{x}') \leftarrow t'_1, t'_2, \ldots, t'_k \tag{4.3}$$

we say that the view V covers a set of query triples in the body of the query of Q in Formula 4.1, denoted with Cov(Q,V), when:

- there exists a homomorphisms h from the body of V to the body of Q ;
- for each variable y appearing in the body but not in the head of V, it's corresponding mapping $h(y)$ needs to be either:
  - (a) an existentially quantified variable that does not appear in any triple in the body of Q outside Cov(Q,V)
  - (b) or a constant.

In that case, the benefit of the views in $\mathcal{V}'_Q$ to the query of Q equals the cost that we would have if we had to read all the covered triples by a set of views $\mathcal{V}'_Q$, minus the cost of reading instead all the corresponding views in $\mathcal{V}$.

$$\text{Bnft}(Q,\mathcal{V}) = \text{Cost}^{\epsilon}(\bigcup_{V\in\mathcal{V}} \text{Cov}(Q,V)) - \sum_{V\in\mathcal{V}} \text{Cost}^{\epsilon}(V) \tag{4.4}$$

In order to solve the previous problem, we first need to prove the submodularity of the benefit function.

Proof. To prove submodularity, for the set of views $\mathcal{V}_1$, $\mathcal{V}_2$, it suffices to show that:

$$\text{Bnft}(Q,\mathcal{V}_1\cup\mathcal{V}_2) + \text{Bnft}(Q,\mathcal{V}_1\cap\mathcal{V}_2) \leq \text{Bnft}(Q,\mathcal{V}_1) + \text{Bnft}(Q,\mathcal{V}_2) \tag{4.5}$$

Let's assume the following notation: $\mathcal{V}_\cap = \mathcal{V}_1 \cap \mathcal{V}_2$, $V'_1 = \mathcal{V}_1 \setminus \mathcal{V}_\cap$, and $V'_2 = \mathcal{V}_2 \setminus \mathcal{V}_\cap$. The left hand size of Inequality 4.5 is rewritten as follows:

$$\text{Bnft}(Q,\mathcal{V}_1\cup\mathcal{V}'_2) + \text{Bnft}(Q,\mathcal{V}_\cap) \tag{4.6}$$

since $\mathcal{V}_1\cup\mathcal{V}_2 = \mathcal{V}_1\cup\mathcal{V}'_2$ applies. By expanding Formula 4.6 , based on the definition of benefit in Formula 4.4 we have that

$$\text{Cost}^{\epsilon}(\bigcup_{V\in\mathcal{V}_1\cup\mathcal{V}'_2} \text{Cov}(Q,V)) - \sum_{V\in\mathcal{V}_1\cup\mathcal{V}'_2} \text{Cost}^{\epsilon}(V) + \text{Cost}^{\epsilon}(\bigcup_{V\in\mathcal{V}_\cap} \text{Cov}(Q,V)) - \sum_{V\in\mathcal{V}_\cap} \text{Cost}^{\epsilon}(V) \tag{4.7}$$

The latter, based on the properties of the $\sum$ aggregate becomes:

$$\text{Cost}^{\epsilon}(\bigcup_{V\in\mathcal{V}_1\cup\mathcal{V}'_2} \text{Cov}(Q,V)) - \sum_{V\in\mathcal{V}_1} \text{Cost}^{\epsilon}(V) + \text{Cost}^{\epsilon}(\bigcup_{V\in\mathcal{V}_\cap} \text{Cov}(Q,V)) - \sum_{V\in\mathcal{V}_2} \text{Cost}^{\epsilon}(V) \tag{4.8}$$

Finally, based on the properties of the union operator, we have that:

$$\begin{aligned}\text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_1\cup\mathcal{V}_2'}\text{Cov}(Q,V))+\text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_\cap}\text{Cov}(Q,V))\leq\\ \text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_1}\text{Cov}(Q,V))+\text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_2'}\text{Cov}(Q,V))+\text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_\cap}\text{Cov}(Q,V))=\\ \text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_1}\text{Cov}(Q,V))+\text{Cost}^\epsilon(\bigcup_{V\in\mathcal{V}_2}\text{Cov}(Q,V))\end{aligned} \tag{4.9}$$

By following the inequalities and equalities it is straightforward that the Inequality 4.6 applies as we wanted to show.

The submodularity property of the benefit function, allows the reduction of the query rewriting problem to the MNssfKc problem as already stated. The later allows to employ an approximate solution for the query rewriting problem that operates in polynomial time in the size of the query and guarantees a $(1-e^{-1})$ approximation of the optimal solution. □

4.2.1 MNssfKc-Based Query Rewriting Algorithm

Based on the Theory of MNssfKc, we propose a $(1-e^{-1})$-approximation polynomial algorithm for solving it.

Our algorithm works as follows: For the set of views $\mathcal{V}_Q$ that can be used to rewrite the query Q, our algorithm works in two stages. Let's assume that Q' is the current state of rewriting for the query Q.

- In the first stage, we start by considering all rewritings Q' of Q that employ two or fewer views.
- In the second stage, starting from all the previously-identified rewritings, we find the view V that can be used to rewrite Q' and maximizes the $\frac{\text{Bnft}(Q,V)}{\text{Cost}^\epsilon(V)}$ ratio. It should be noted that each time Q' is rewritten by employing, in a greedy fashion, the algorithm needs to update the corresponding benefit and cost functions for all remaining views.

Example 1. Let's assume that we have the query in Fig.]4.2 defined as:

$$\begin{aligned}q(x_1,x_5)\leftarrow&(x_2,orange,x_1),(x_2,orange,x_3),(x_3,green,x_4),(x_4,green,x_1),\\&(x_4,purple,x_5),(x_4,cyan,x_6),(x_7,purple,x_6),(x_8,orange,x_7)\end{aligned}$$

and the corresponding views:

$$V_1(x_1,x_2,x_3)\leftarrow(x_2,orange,x_1),(x_2,orange,x_3) \tag{4.10}$$
$$V_2(x_4,x_6,x_7,x_8)\leftarrow(x_4,cyan,x_6),(x_7,purple,x_6),(x_8,orange,x_7) \tag{4.11}$$
$$V_3(x_3,x_6)\leftarrow(x_3,green,x_4),(x_4,cyan,x_6) \tag{4.12}$$
$$V_4(x_1,x_4,x_5)\leftarrow(x_4,green,x_1),(x_4,purple,x_5) \tag{4.13}$$

with initial benefit-to-cost ratios

$$\frac{\text{Bnft}(Q,V_1)}{\text{Cost}^\epsilon(V_1)}=0.8;\frac{\text{Bnft}(Q,V_2)}{\text{Cost}^\epsilon(V_2)}=0.5;\frac{\text{Bnft}(Q,V_3)}{\text{Cost}^\epsilon(V_3)}=0.5;\frac{\text{Bnft}(Q,V_4)}{\text{Cost}^\epsilon(V_4)}=0.0. \tag{4.14}$$

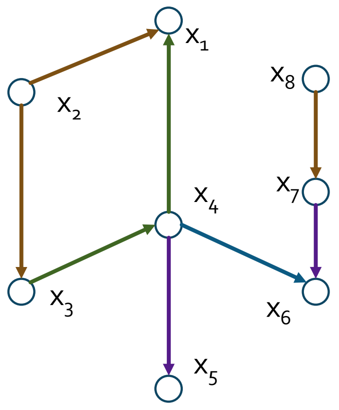

Figure 4.1: Initial Query Q

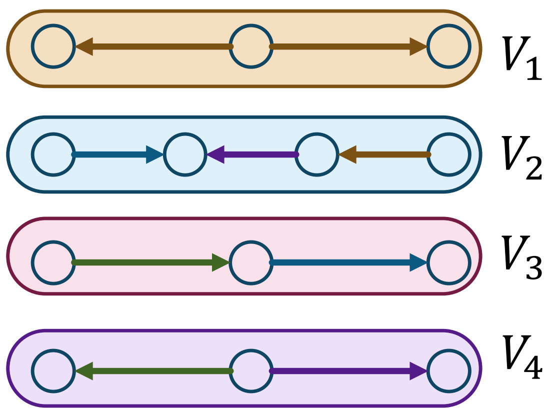

Figure 4.2: Views for Rewriting $\mathcal{V}_Q$

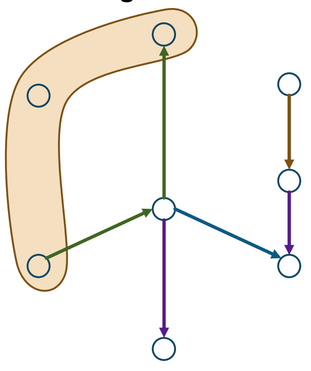

Figure 4.3: Rewriting Q'

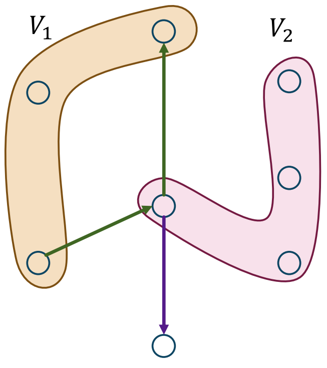

Figure 4.4: Rewriting Q''

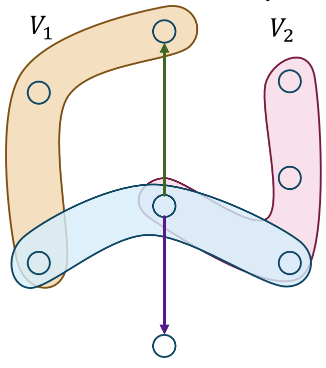

Figure 4.5: Rewriting Q'''

Initially, our algorithm chooses for the rewriting the view that maximises the benefit-to-cost ratio and the initial query is rewritten to:

$$q'(x_1, x_5) \leftarrow V_1(x_1, x_2, x_3), (x_3, green, x_4), (x_4, green, x_1),$$
$$(x_4, purple, x_5), (x_4, cyan, x_6), (x_7, purple, x_6), (x_8, orange, x_7)$$

while displayed in Fig. 4.3. The corresponding rewriting does not change the benefit-to-cost ratio for any of the remaining views, while the benefit-to-cost ratio for V_1 becomes 0, since the view cannot be further reused for the rewriting.

In the next step, the query of q' is further rewritten to employ view V_2 as part of its rewriting:

$$q''(x_1, x_5) \leftarrow V_1(x_1, x_2, x_3), (x_3, green, x_4), (x_4, green, x_1),$$
$$(x_4, purple, x_5), V_2(x_4, x_6, x_7, x_8)$$

while displayed in Fig. 4.4. By employing the view V_2 in the rewriting, the benefit of the view V_3 is reduced, since the cover the same edge, and its benefit-to-cost ratio becomes 0.2. The final rewriting of the query is

$$q'''(x_1, x_5) \leftarrow V_1(x_1, x_2, x_3), V_3(x_3, x_6), (x_4, green, x_1),$$
$$(x_4, purple, x_5), V_2(x_4, x_6, x_7, x_8)$$

and displayed in Fig. 4.5.

5. EXPERIMENTAL EVALUATION

The aim of our evaluation section is to examine the performance of the query-rewriting strategy based on a view-selection process, part of existing work [16, 15]. For our testing scenarios, our application takes as input a knowledge graph G; a query workload $\mathcal{Q}$, corresponding to past queries; a query workload $\mathcal{Q}_T$ corresponding to future queries; and produces the views $\mathcal{V}$ that will be materialized for future query execution. The rewritings w.r.t. the views in $\mathcal{V}$ are later tested w.r.t. to the query workload $\mathcal{Q}_T$. The knowledge graph G is stored as a set of triples within a relational column store. We selected MonetDb as the underlying database.

5.1 Hardware and memory

We deployed our implementation on a server of 2 Intel(R) Xeon(R) CPUs @2.2GHz each with 10 cores/20 threads per CPU and 128GB of main memory. The data are stored in a MonetDb v11.37.11 database running on the same server.

5.2 Implementation Setup

We have implemented our algorithm in Java 8 using the Apache Jena 3.6.0 open source Semantic Web framework [11] to parse SPARQL query workloads. For efficiently, computing containment mappings from a set of views $\mathcal{V}$ to an examined query Q, we have employed the mv-index structure introduced in our previous work [17].

5.3 Benchmark

For benchmarking our methodology, we employed the DbPedia *semantic knowledge graph* [3] that has $189{,}511{,}679$ triples and its corresponding size on disk is $133.93\,GB$. The corresponding real-world query workload [1], originating from queries on the DbPedia knowledge graph, contains $1{,}287{,}711$ queries. We have randomly partitioned the query workload into the DbPedia *training query workload* $\mathcal{Q}$, containing $1{,}277{,}711$ queries that will be used for selecting the appropriate views for materialization and the DbPedia *testing query workload* $\mathcal{Q}_T$, containing 1000 queries that will be used for testing the efficiency of the selected materialized views. We proceed with each step of the view-selection process.

5.4 Effectiveness of the Query-Rewriting of Views

We now examine the quality of the selected views by rewriting the queries within the testing-query workload Q_T containing 1000 queries. We will consider the following parameterization for our problem: minSup value of 500; available storage capacities of 100, 1000, 5000, $10{,}000$, $15{,}000$, $20{,}000$ records; the view selection process described in [15] and the query rewriting methodology described in Chapter 4. We should point out that for the view-selection methodology we also employ the linear cost model assumption and not a more complicated cost estimation function.

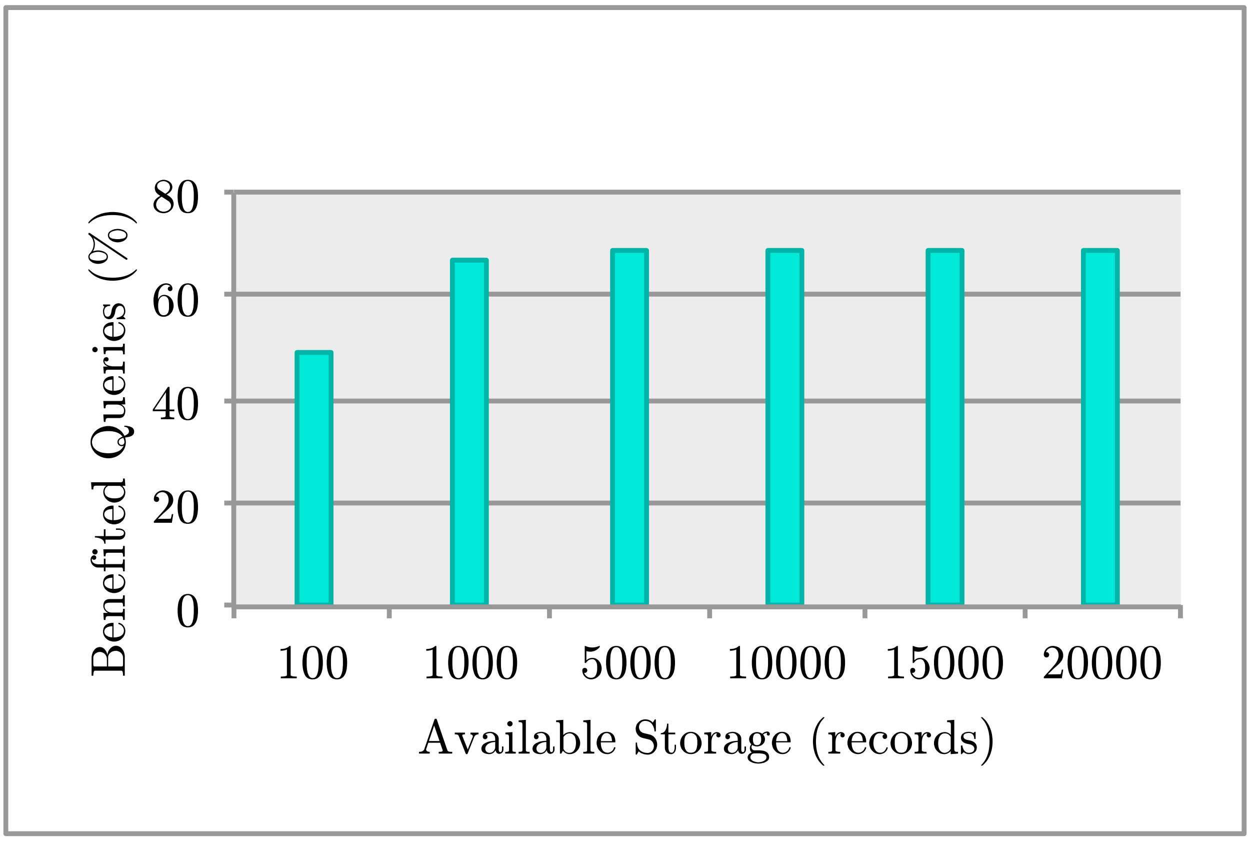

Figure 5.1: Benefited Queries (%)

Figure 5.1 illustrates the percentage of the queries that are benefited from our query-rewriting methodology. The x-axis represents the available storage for materialization–measured in terms of records used for materialization–, while the y-axis the percentage of benefited queries.

Figure 5.2 illustrates the overall execution time for the queries in the testing workload Q_T; for varying capacities for materialization. The x-axis in Figure 5.2 illustrates the available storage for materialization , while the y-axis illustrates the overall execution time for the testing workload Q_T w.r.t. the suggested query-rewriting methodology. We observe that the query workload is insignificantly benefited for more than 1000 records of available storage.

Figure 5.3 illustrates the reduction in execution time exclusively for the queries in Q_T that are benefited from the materialization. We should note that the y-axis in Figure 5.3 is in logarithmic scale.

The experimental section practically demonstrates the significant improvement of query performance using the rewriting techniques we proposed. Specifically, we observe that, on average, there is a several orders of magnitude improvement in the execution of queries. These improvements are not only theoretical but are validated through extensive testing and benchmarking. By employing these rewriting techniques, we are able to optimize the query process, leading to faster and more efficient data retrieval. The results clearly indicate that our approach can handle complex query workloads effectively, making it a valuable contribution to the field of query optimization. This substantial enhancement in query performance underscores the practical applicability and potential impact of our proposed methods in real-world scenarios.

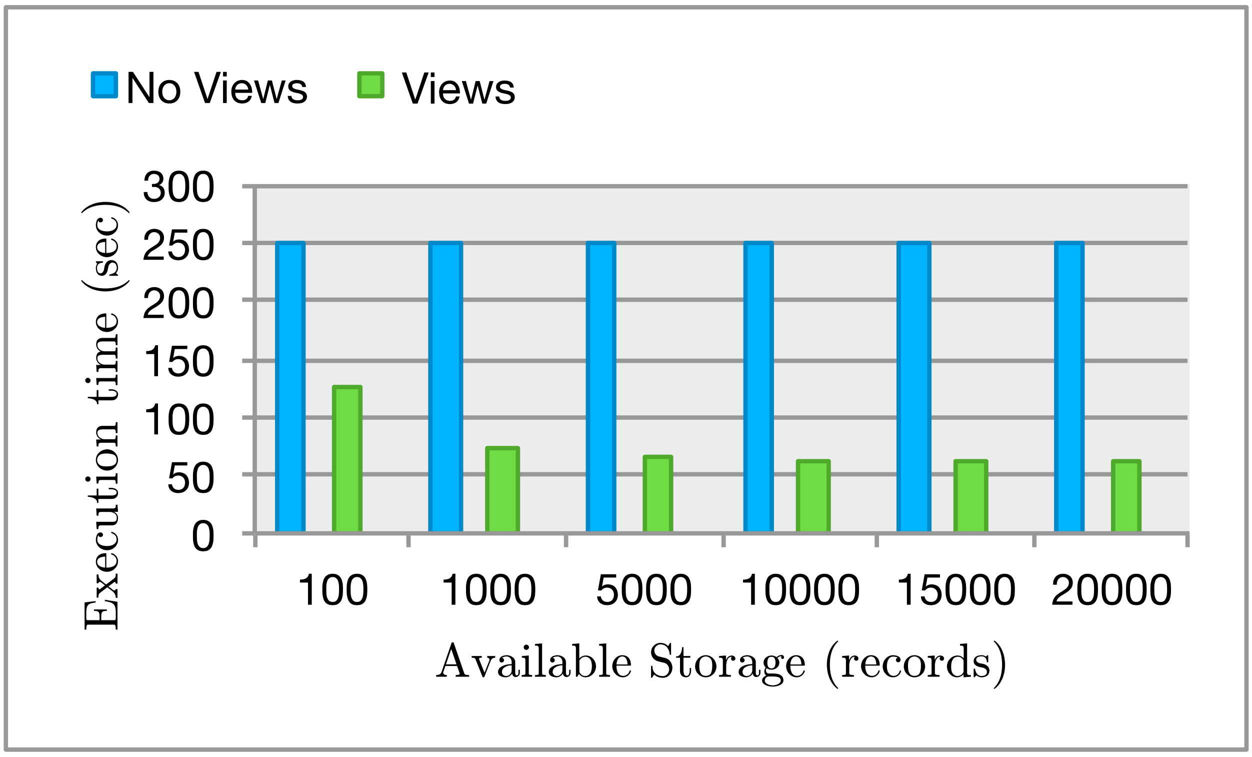

Figure 5.2: Execution times: $\mathcal{Q}_T$

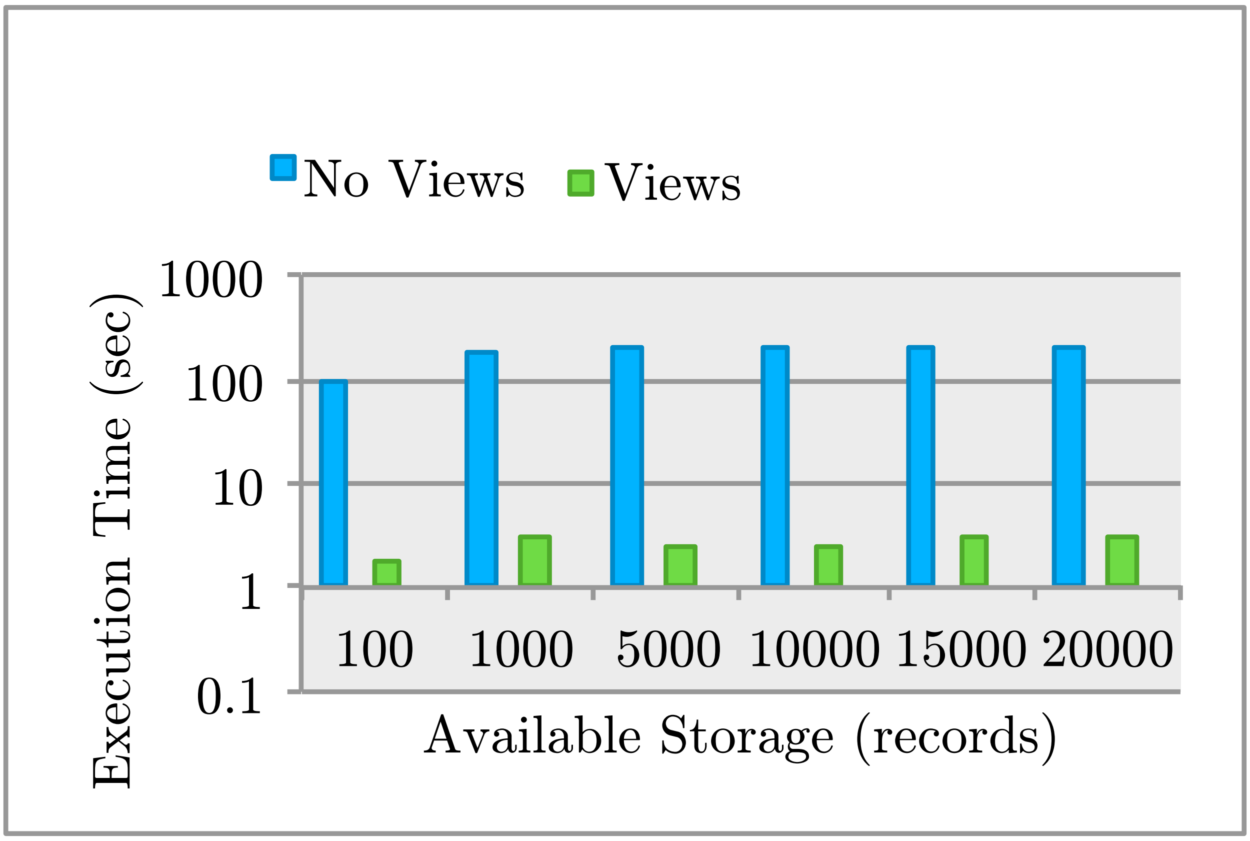

Figure 5.3: Execution times: benefited $\mathcal{Q}_T$

6. RELATED WORK

Query rewriting and view materialization play crucial roles in multiple-query optimization, a field dedicated to improving the efficiency of database systems by optimizing the execution of several queries at once. Early research in this area introduced fundamental techniques for reducing computational costs by reusing common subexpressions across queries. Subsequent developments advanced these methods by creating efficient algorithms and addressing the challenges of materialized view selection and maintenance. These studies also explored automation in view selection and the exploitation of similar subexpressions to further optimize query processing. More recent contributions have refined these techniques to address scalability issues in large-scale systems, enhancing both theoretical understanding and practical applications. Collectively, this body of work has significantly advanced the field, laying the groundwork for current approaches to query optimization and view management.

6.1 Query Rewriting and View Materialization in Multiple-Query Optimization

Query rewriting and view materialization have been extensively explored within the context of multiple-query optimization, which aims to improve the efficiency of database systems by optimizing the execution of multiple queries simultaneously. Early foundational work by Sellis (1988) in ”Multiple-query optimization” [23] proposed multiple-query optimization techniques that laid the groundwork for subsequent research in this area. Sellis’s approach involved identifying common subexpressions among multiple queries and computing them once, which significantly reduces the overall computation cost and improves system performance.

Roy et al. (2000) in ”Efficient and extensible algorithms for multi query optimization” [22] expanded on this by presenting efficient and extensible algorithms that further optimized multiple queries, demonstrating significant performance improvements. Their work introduced new heuristics for query decomposition and rewriting, which could adapt to various types of database systems, making it a versatile solution for multi-query optimization.

Mistry et al. (2001) focused on ”Materialized view selection and maintenance using multi-query optimization” [19] addressing the challenge of choosing which views to materialize for optimal query performance. This study highlighted the importance of balancing the maintenance cost of materialized views with the query performance benefits they offer. Their proposed algorithms aimed to find an optimal set of views that minimized the total cost, including both the query execution and view maintenance costs.

Agarwal et al. (2000) in ”Automated selection of materialized views and indexes for SQL databases” [2] automated the selection of materialized views and indexes in SQL databases, providing a comprehensive solution for optimizing database performance through strategic view materialization and indexing. This work is notable for its practical approach, incorporating real-world constraints and performance metrics to guide the selection process, making it highly applicable to commercial database systems.

Zhou et al. (2007) introduced techniques in ”Efficient exploitation of similar subexpressions for query processing” [30] for efficiently exploiting similar subexpressions in query processing, thereby reducing redundancy and improving execution times. Their approach involved detecting overlapping subqueries and optimizing their execution through shared computation, which is particularly effective in environments with high query load and complex query structures.

Chirkova, Halevy, and Suciu (2002) provided a formal perspective in ”A formal perspective on the view selection problem” [5] establishing theoretical foundations that guide practical view selection strategies. They explored the complexity of the view selection problem and proposed approximation algorithms that offer near-optimal solutions with provable performance guarantees, bridging the gap between theory and practice.

More recently, Kathuria and Sudarshan (2017) in ”Efficient and provable multi-query optimization” [13] developed efficient and provable multi-query optimization methods that ensure both correctness and performance gains. Their work introduced new optimization techniques that are not only efficient but also come with formal correctness proofs, enhancing the reliability of multi-query optimization in critical applications.

Jindal et al. (2018) explored subexpression selection at datacenter scale in ”Selecting subexpressions to materialize at datacenter scale” [12] addressing the challenges of optimizing query processing in large-scale distributed systems. Their study focused on the unique requirements of datacenter environments, such as scalability and fault tolerance, and proposed novel techniques for selecting and materializing subexpressions to improve query performance across distributed nodes.

6.2 Semantic Web and Graph Data Systems

In the domain of the Semantic Web and graph data systems, view materialization and query rewriting techniques have been pivotal for optimizing RDF (Resource Description Framework) databases and SPARQL queries. Dritsou et al. (2011) proposed optimization strategies for query shortcuts in ”Optimizing query shortcuts in RDF databases” [6] which enhance query performance by precomputing and reusing frequently accessed query results. Their approach leverages the hierarchical nature of RDF data to identify and materialize query shortcuts that can be reused across multiple queries, significantly reducing query evaluation time.

Goasdoué et al. (2011) tackled view selection in semantic web databases in ”View selection in semantic web databases” [9] focusing on efficient view materialization to support complex query workloads. This work addressed the challenge of selecting views in a dynamic environment where query patterns may change over time, proposing adaptive algorithms that can adjust the set of materialized views based on evolving query workloads.

Papailiou et al. (2015) developed graph-aware, workload-adaptive SPARQL query caching techniques in ”Graph-aware, workload-adaptive SPARQL query caching” [20] that dynamically adapt to query patterns, significantly improving query response times in graph databases. Their system continuously monitors query patterns and adapts the caching strategy accordingly, ensuring that frequently queried subgraphs are cached and reused, which improves overall system performance.

Wang, Ntarmos, and Triantafillou (2017) introduced GraphCache in ”GraphCache: A caching system for graph queries” [29] a caching system specifically designed for graph queries, which leverages the structure of graph data to enhance caching efficiency. Their system is designed to handle the unique challenges of graph data, such as the need to cache not just individual query results but also intermediate query states and subgraphs, which are critical for efficient graph query processing.

Their earlier work in 2016, ”Indexing query graphs to speedup graph query processing” [28] also addressed indexing query graphs to speed up graph query processing, providing methods for indexing that facilitate faster query evaluation. By creating specialized indexes that capture the structural properties of query graphs, their techniques enable rapid identification and retrieval of relevant subgraphs, which accelerates query execution.

Meimaris et al. (2017) extended characteristic sets for graph indexing in ”Extended characteristic sets: Graph indexing for SPARQL query optimization” [18] offering advanced techniques for SPARQL query optimization by capturing and utilizing the structural properties of graph data. Their work introduced novel indexing structures that enhance the performance of SPARQL queries by reducing the number of joins and intermediate results, which are typically expensive in graph query processing.

6.3 Connections and Contributions

The body of work in multiple-query optimization and the Semantic Web demonstrates the broad applicability and critical importance of query rewriting and view materialization across different data management paradigms. Our work builds on these foundational principles by addressing the problem of view-based query rewriting within the context of knowledge graphs represented in relational databases. By leveraging the reduction of query rewriting to the problem of Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint (MNssfKc), we provide a novel approach that offers polynomial-time solutions with provable approximation guarantees.

The integration of techniques from multiple-query optimization and graph data systems into a unified framework for knowledge graph query rewriting represents a significant advancement. This work not only bridges the gap between these research areas but also introduces new possibilities for efficient query processing in complex data environments. By effectively combining these methodologies, we are able to address several longstanding challenges in the field of knowledge graph management, particularly those related to query performance and scalability.

Our proposed technique extends the state-of-the-art by providing a scalable and theoretically grounded solution to the view materialization problem. The reduction of query rewriting to the MNssfKc problem allows for the application of well-established optimization algorithms, ensuring that our approach is both efficient and reliable.

Furthermore, our work has practical implications for improving the performance of knowledge graph query systems. By optimizing the selection and materialization of views based on recurring query patterns, we can significantly reduce query execution times and resource consumption. This is particularly important for industrial applications where large volumes of data and high query throughput are common. The ability to precompute and reuse query results not only enhances system efficiency but also improves the user experience by providing faster and more reliable query responses.

In addition to its practical benefits, our approach contributes to the theoretical understanding of query optimization in knowledge graphs. By framing the view materialization problem within the context of submodular function optimization, we provide new insights into the nature of query rewriting and its relationship to other optimization problems. This theoretical perspective opens up new avenues for future research, encouraging the exploration of additional cost models and optimization techniques that could further enhance the performance and applicability of our methods.

Moreover, our work highlights the importance of interdisciplinary approaches in advancing the field of data management. By drawing on concepts from multiple-query optimization, graph theory, and submodular function optimization, we demonstrate the value of integrating diverse methodologies to solve complex problems. This interdisciplinary approach not only enriches the field but also fosters collaboration and innovation, paving the way for future breakthroughs in knowledge graph management and query optimization.

In summary, our contributions to the field are multifaceted. We provide a novel, efficient, and scalable solution to the view-based query rewriting problem in knowledge graphs, grounded in both theoretical rigor and practical applicability. Our work bridges the gap between multiple-query optimization and graph data systems, offering new possibilities for efficient query processing in large-scale and complex data environments. Through this integration of techniques and perspectives, we advance the state-of-the-art and lay the foundation for future research and development in the field of knowledge graph query optimization.

7. CONCLUSION AND FUTURE WORK

In this thesis, we presented a novel and efficient technique for view-based query rewriting specifically designed for knowledge graphs represented in relational databases. Our approach addresses the unique challenges posed by the complexity and scale of modern knowledge graphs, which are increasingly prevalent in various industrial applications such as search engines, semantic web technologies, and data integration platforms.

In this thesis, we presented a novel and efficient technique for view-based query rewriting specifically designed for knowledge graphs represented in relational databases. Our approach addresses the unique challenges posed by the complexity and scale of modern knowledge graphs, which are increasingly prevalent in various industrial applications such as search engines, semantic web technologies, and data integration platforms.

Our theoretical analysis demonstrates the potential of this approach to significantly enhance query performance by minimizing execution costs through optimal view materialization. This is particularly beneficial for applications involving large-scale and complex knowledge graphs, where traditional query processing techniques may fall short due to high computational demands.

In our future work, we intend to perform extensive experimental evaluations based on our theoretical findings. These experiments will be designed to validate the practical effectiveness of our approach in reducing the overall execution cost of queries in real-world scenarios. We plan to implement our technique in a prototype system and benchmark its performance against existing state-of-the-art methods. This will involve:

- Dataset and Query Selection: We will select a diverse set of knowledge graph datasets and a range of query workloads that reflect common use cases in industrial applications.
- Implementation and Optimization: We will implement our view-based query rewriting technique within a relational database management system, ensuring that the algorithms are optimized for practical use.
- Performance Metrics: We will evaluate our approach using various performance metrics, including query execution time, resource utilization, and scalability. Comparisons with baseline methods will be made to highlight the improvements achieved.
- Scalability Analysis: We will investigate the scalability of our approach by varying the size and complexity of the knowledge graphs and the corresponding query workloads, ensuring that our technique can handle large-scale data efficiently.
- Extended Cost Models: While our current work focuses on the linear cost model, future research will explore the applicability of our reduction technique to other cost models, such as logarithmic or polynomial cost functions, to broaden the scope of our approach.
- Integration with Other Optimization Techniques: We aim to explore the integration of our view-based query rewriting technique with other optimization strategies, such as indexing and caching, to further enhance query performance.
- User Feedback and Adaptation: Incorporating user feedback into the optimization process to dynamically adjust view materialization strategies based on evolving query patterns and user requirements.

By pursuing these future research directions, we aim to establish a comprehensive framework for efficient query processing in knowledge graphs, ultimately contributing to the advancement of data management technologies and their application in various industrial domains. The insights gained from our experimental results will provide valuable guidance for further refinement and adoption of our techniques in practical settings.

In conclusion, this thesis makes significant strides in addressing the challenges of view-based query rewriting for knowledge graphs represented in relational databases. Our novel approach not only advances the theoretical understanding of the problem but also paves the way for practical implementations that can enhance the performance of complex query workloads in modern data-driven applications.

BIBLIOGRAPHY

[1] Dbpedia log, 2012. [Online; accessed 16-September-2021].

[2] S Agarawal, S Chaudhuri, and V Narasayya. Automated selection of materialized views and indexes for sql databses. In *Proceedings of 26th International Conference on Very Large Databases, Cairo, Egypt*, pages 191–207, 2000.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *ISWC/ASWC*, pages 722–735, 2007.

[4] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. Obi-wan: Ontology-based RDF integration of heterogeneous data. *Proc. VLDB Endow.*, 13(12):2933–2936, 2020.

[5] Rada Chirkova, Alon Y Halevy, and Dan Suciu. A formal perspective on the view selection problem. *The VLDB Journal—The International Journal on Very Large Data Bases*, 11(3):216–237, 2002.

[6] Vicky Dritsou, Panos Constantopoulos, Antonios Deligiannakis, and Yannis Kotidis. Optimizing query shortcuts in rdf databases. *ESWC*, pages 77–92, 2011.

[7] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. Why we need knowledge graphs: Applications. In *Knowledge Graphs*, pages 95–112. Springer, 2020.

[8] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[9] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *PVLDB*, 5(2):97–108, 2011.

[10] Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. Implementing data cubes efficiently. *ACM SIGMOD Record*, 25:205–216, 1996.

[11] Apache Jena. semantic web framework for java, 2007.

[12] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, and Hiren Patel. Selecting subexpressions to materialize at datacenter scale. *Proceedings of the VLDB Endowment*, 11(7):800–812, 2018.

[13] Tarun Kathuria and S Sudarshan. Efficient and provable multi-query optimization. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 53–67, 2017.

[14] Alon Y Levy, Alberto O Mendelzon, and Yehoshua Sagiv. Answering queries using views. In *PODS*, pages 95–104. ACM, 1995.

[15] Theofilos Mailis, Yannis Kotidis, Stamatis Christoforidis, Evgeny Kharlamov, and Yannis Ioannidis. View selection over knowledge graphs in triple stores. *Proceedings of the VLDB Endowment*, 14(13):3281–3294, 2021.

[16] Theofilos Mailis, Yannis Kotidis, Vaggelis Nikolopoulos, Evgeny Kharlamov, Ian Horrocks, and Yannis Ioannidis. An efficient index for rdf query containment. In *Proceedings of the 2019 international conference on management of data*, pages 1499–1516, 2019.

[17] Theofilos Mailis, Yannis Kotidis, Vaggelis Nikolopoulos, Evgeny Kharlamov, Ian Horrocks, and Yannis E. Ioannidis. An efficient index for RDF query containment. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1499–1516. ACM, 2019.

[18] Marios Meimaris, George Papastefanatos, Nikos Mamoulis, and Ioannis Anagnostopoulos. Extended characteristic sets: graph indexing for sparql query optimization. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 497–508. IEEE, 2017.

[19] Hoshi Mistry, Prasan Roy, S Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *ACM SIGMOD Record*, volume 30, pages 307–318. ACM, 2001.

[20] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, and Nectarios Koziris. Graph-aware, workload-adaptive sparql query caching. In *SIGMOD*, pages 1777–1792. ACM, 2015.

[21] Richard Qian. Understand your world with bing, May 2013. [Online; accessed 16-September-2021].

[22] Prasan Roy, Srinivasan Seshadri, S Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 249–260, 2000.

[23] Timos K Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[24] Longxiang Shi, Shijian Li, Xiaoran Yang, Jiaheng Qi, Gang Pan, and Binbin Zhou. Semantic health knowledge graph: semantic integration of heterogeneous medical knowledge and services. *BioMed research international*, 2017, 2017.

[25] Amit Singhal. Introducing the knowledge graph: Things, not strings, May 2012. [Online; accessed 16-September-2021].

[26] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

[27] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[28] Jing Wang, Nikos Ntarmos, and Peter Triantafillou. Indexing query graphs to speedup graph query processing. In *EDBT*, pages 41–52, 2016.

[29] Jing Wang, Nikos Ntarmos, and Peter Triantafillou. Graphcache: A caching system for graph queries. In *EDBT*, pages 13–24, 2017.

[30] Jingren Zhou, Per-Ake Larson, Johann-Christoph Freytag, and Wolfgang Lehner. Efficient exploitation of similar subexpressions for query processing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 533–544, 2007.