View Selection over Knowledge Graphs in Triple Stores

Theofilos Mailis
Athena Research Centre
National & Kapodistrian University of Athens, Greece
tmailis@di.uoa.gr

Yannis Kotidis
Athens University of Economics and Business
Greece
kotidis@aueb.gr

Stamatis Christoforidis
National & Kapodistrian University of Athens
Greece
stachris@di.uoa.gr

Evgeny Kharlamov
Bosch Center for AI, Germany
University of Oslo, Norway
evgeny.kharlamov@de.bosch.com

Yannis Ioannidis
Athena Research Centre
National & Kapodistrian University of Athens, Greece
yannis@di.uoa.gr

ABSTRACT

Knowledge Graphs (KGs) are collections of interconnected and annotated entities that have become powerful assets for data integration, search enhancement, and other industrial applications. Knowledge Graphs such as DbPedia may contain billion of triple relations and are intensively queried with millions of queries per day. A prominent approach to enhance query answering on Knowledge Graph databases is View Materialization, ie., the materialization of an appropriate set of computations that will improve query performance.

We study the problem of view materialization and propose a view selection methodology for processing query workloads with more than a million queries. Our approach heavily relies on subgraph pattern mining techniques that allow to create efficient summarizations of massive query workloads while also identifying the candidate views for materialization. In the core of our work is the correspondence between the *view selection* problem to that of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint*. The latter leads to a tractable view-selection process for native triple stores that allows a $(1 - e^{-1})$-approximation of the optimal selection of views. Our experimental evaluation shows that all the steps of the view-selection process are completed in a few minutes, while the corresponding rewritings accelerate 67.68% of the queries in the DbPedia query workload. Those queries are executed in 2.19% of their initial time on average.

PVLDB Reference Format:
Theofilos Mailis, Yannis Kotidis, Stamatis Christoforidis, Evgeny Kharlamov, and Yannis Ioannidis. View Selection over Knowledge Graphs in Triple Stores. PVLDB, 14(13): 3281 - 3294, 2021.
doi:10.14778/3484224.3484227

Proceedings of the VLDB Endowment, Vol. 14, No. 13 ISSN 2150-8097.
doi:10.14778/3484224.3484227

1 INTRODUCTION

Knowledge Graphs (KGs) are collections of interconnected and annotated entities that have become powerful assets for data integration [12, 53], search enhancement [19], and other applications [18]. KGs are now widely used in both academia and industry where prominent KGs such as DBpedia [17], Yago [56], Google's KG [55], and Microsoft's Satori [50] have already reached tremendous scale. Indeed, DBpedia alone currently consists of more than 1 billion triples. Thus, efficient query answering techniques are critical for ensuring scalability of KG driven software systems.

Knowledge Graphs. KGs are typically stored as sets 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)*.

View Selection & Materialization. A prominent approach to enhance query answering is *View Materialization*, ie., given a database $\mathcal{D}$ and a query workload $\mathcal{Q}$ materialize an appropriate set of computations to improve query performance. The problem of *View Selection*, the selection of the appropriate views to materialize, is achieved by finding commonalities across queries in $\mathcal{Q}$ the precomputation of which minimizes the execution of existing or future queries w.r.t. to a cost function (e.g., query evaluation, storage, and subexpression maintenance costs), under a set of constraints (e.g., space budget).

Complexity of the View Selection Problem. Chirkova et al. [15] show that the problem of view materialization for conjunctive queries is EXPTime-hard, while it belongs to the 3EXPTime complexity class. The latter indicates that exhaustive solutions to the aforementioned problem cannot be practically applied for large query workloads. This is the reason that many of the existing approaches have targeted workloads with tens or hundreds of queries [5, 23, 51, 66] and do not scale for workloads containing millions of queries.

Goal. Our goal is given a KG G and a query workload $\mathcal{Q}$ containing million of queries, to select and materialize the views that will improve execution of existing or future queries.

Detrimental to the view selection process is the underlying mechanism for storage and retrieval of the KG G. We focus on relational solutions for the storage of the corresponding KG. Specifically, we

study *native triple stores*, i.e., purpose-built databases for the storage and retrieval of triples, and *column-oriented databases* that store data tables by column rather than by row. Our methodology cannot be directly applied to NoSQL structures for the storage of the underlying graph.

To tackle the *View Selection* problem for KGs, we exploit the explicit graph-nature of the underlying data and queries. The latter allows to employ frequent subgraph-mining techniques fixating on query patterns that appear with a high frequency in $\mathcal{Q}$. A frequent query pattern P in $\mathcal{Q}$ employs a dual role in our view selection methodology: *I.* Frequent query patterns in $\mathcal{P}$ provide a summarization of the workload $\mathcal{Q}$. Instead of examining the query workload in $\mathcal{Q}$ we examine a multiset of patterns that summarize the basic characteristics of the query workload. E.g., the query Q

$$Q : q(?\text{alb}) \leftarrow (?\text{sg}, \texttt{name}, ?\,sN), (?\text{sg}, \texttt{fromAlbum}, ?\text{alb})$$

with a frequency of 3, summarizes the queries Q_1, Q_2, Q_3:

$$\begin{aligned} &Q_1 : q_1(?\text{alb}) \leftarrow (?\text{sg}, \texttt{name}, \text{“}Sail\text{”}), (?\text{sg}, \texttt{fromAlbum}, ?\text{alb}) \\ (1)\quad &Q_2 : q_2(?\text{alb}) \leftarrow (?\text{sg}, \texttt{name}, \text{“}Hello\text{”}), (?\text{sg}, \texttt{fromAlbum}, ?\text{alb}) \\ &Q_3 : q_3(?\text{alb}) \leftarrow (?\text{sg}, \texttt{name}, \text{“1977”}), (?\text{sg}, \texttt{fromAlbum}, ?\text{alb}) \end{aligned}$$

II. Additionally, each frequent query pattern $p \in \mathcal{P}$ spawns a set of view candidates of the form $V : v(\vec{x}) \leftarrow p$, with p constituting the body of the view, while its head corresponds to a subset of the variables appearing in p. E.g., if the query pattern (?sg, name, ? sN), (?sg, fromAlbum, ?alb) appears with a high frequency in $\mathcal{Q}$, it would be beneficial to consider view materializations whose body constitutes of the specific pattern. Therefore the pattern mining process allows to summarize our query workload $\mathcal{Q}$ and identify candidate views for materialization. Furthermore, to ensure the tractability of our approach, we investigate approximation algorithms for the view-selection problem on KGs and its reduction to a variation of the Knapsack problem.

Contributions. Our contributions are epitomized as follows:

► **Workload Summarization.** In the core of our methodology for solving the view selection problem are data mining techniques that allow to represent a query workload $\mathcal{Q}$ via a multiset $\mathcal{P}$ of the most frequent query patterns, the multiplicity of each pattern being its corresponding frequency. The corresponding approach allows our algorithm to represent millions of queries via the most frequent patterns. Theorem 3.2 shows that a set of materialized views being beneficial for the summarization of a query workload $\mathcal{Q}$ is also beneficial for the initial workload $\mathcal{Q}$. Our approach can be further extended by taking into account the temporal evolution of the query workload to employ more refined data mining techniques for forecasting the query patterns that have a high possibility of appearing in future queries [22].

► **View Candidates.** By definition, every frequent pattern $p \in \mathcal{P}$ corresponds to a reappearing sub-pattern within the workload $\mathcal{Q}$, thus it can generate a set of views of the form $v(\vec{x}) \leftarrow p$ with $\vec{x}$ being a subset of the variables appearing in p. We initially study *primordial views* (*select-project* views in database terminology [66]), i.e., views such that all the variables in $\vec{x}$ appear in one atomic formulae (triple pattern) in the body of p. Primordial views are of primary importance for native triple stores, i.e., relational stores that only admit for triple relations, and therefore views can only be introduced trough the usage of “fresh” triple predicates. We also examine the case of arbitrary views that are derived from combining multiple primordial views.

► **View Selection with Quality Guarantees.** In the core of our work is the correspondence between the *view selection* problem to that of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint* (MNSSFKC problem). We prove that for primordial views there exists a reduction of the view selection to the MNSSFKC problem. The latter allows to employ the existing $1 - e^{-1}$-approximation algorithm for solving the *view selection* problem in $O(n^5)$ steps, n being the number of candidate patterns. Additionally, we extend the view selection methodology for arbitrary views based on a variation of the aforementioned approximation algorithm, but with no guarantees.

Implementation & Evaluation. We have implemented our novel structures and algorithms for view selection and tested their efficiency in the KG and query workload of DBPedia. This workload is described in detail in our evaluation Section and contains 1,287,711 conjunctive queries. All the steps of the view-selection process are completed in a few minutes, while the corresponding rewritings accelerate 67.68% of the queries in the workload. Those queries on average are executed in 2.19% of their initial time.

Structure. The rest of the paper is structured as follows: In Section 2 we provide some preliminary definitions. In Section 3 we examine frequent subgraph mining techniques that will accelerate the view selection process by creating summarizations and introducing the appropriate candidates for view materialization. A special form of candidates are primordial views, i.e., corresponding to filtered predicate relations. We prove that primordial views are an optimal materialization choice for native triple stores. In Section 4 we present a tractable view-selection algorithm that is based on the reduction to the MNSSFKC problem and we study its quality guarantees for primordial and arbitrary views. In Section 5 we perform an experimental evaluation of our view-selection techniques. Finally, Section 6 presents the current literature on view materialization, while Section 7 summarizes the paper and mentions directions for future work.

2 PRELIMINARIES

Initially, we will 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.

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)*.

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*.

Query answering. A solution to a conjunctive query Q is a mapping $m : \text{vars}(Q) \rightarrow C$ 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.

Example 2.1. An KG G contains information related to songs and albums:

($s1$, name, "*Masquerade*"), ($s1$, fromAlbum, $al1$),

($al1$, name, "*The Phantom of the Opera*"), ($al1$, artist, $ar3$),

($ar3$, name, "*Andrew L. Webber*"), ($ar3$, type, MusicalArtist)

In the corresponding graph, we use quotation marks to represent String values.

For our running example, we will ask for information related to a specific song. We ask for the name and the album name of a song that is contained within an album in which a musical artist participates. In the following query, elements with a question mark correspond to variables in X:

$$\begin{aligned} Q : q(?sN, ?aN) \leftarrow & (?\text{sNg}, \texttt{name}, ?sN), \\ & (?\text{sNg}, \texttt{fromAlbum}, ?\,alb), (?\,alb, \texttt{name}, ?aN) \end{aligned} \tag{2}$$

The answer to the query if applied on the sample graph database will be the pair (*"Masquerade", "The Phantom of the Opera"*).

2.2 View Selection

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

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.

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* $\textsc{Rwrt}(Q, \mathcal{V})$ takes as input the query Q and rewrites it to an equivalent query $Q' = \textsc{Rwrt}(Q, \mathcal{V})$ using views from $\mathcal{V}$. A rewriting function Rwrt is optimal when there exists no other rewriting Q'' of Q such that $\textsc{Cost}^{\epsilon}(Q'') < \textsc{Cost}^{\epsilon}(Q')$, with $\textsc{Cost}^{\epsilon}$ being the function that maps a query to its estimated execution cost.

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. $\textsc{Cost}^{\epsilon}(Q)$, is proportional to the size of the relational tables appearing in Q. The linear cost model is manifested in [26] while its *linear independence* property is crucial for most of the proofs in this paper.

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

$$\textsc{Bnft}(Q, \mathcal{V}) = \textsc{Cost}^{\epsilon}(Q) - \textsc{Cost}^{\epsilon}(\textsc{Rwrt}(Q, \mathcal{V})). \tag{3}$$

We also denote with $\textsc{Bnft}(\mathcal{Q}, \mathcal{V})$ 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. [37] 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 $\sigma : \pi_{\emptyset}(W) \rightarrow \pi_{\emptyset}(Q)$ 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.

Example 2.2. For the query Q appearing in Formula 2 and the materialized view V appearing in 4a, the query Q' in 4b is a valid rewriting of Q:

$$V : v(?y, ?z) \leftarrow (?x, \texttt{name}, ?y), (?x, \texttt{fromAlbum}, ?z) \tag{4a}$$

$$Q' : q(?sN, ?aN) \leftarrow v(?sN, ?\,alb), (?\,alb, \texttt{name}, ?aN) \tag{4b}$$

2.3 Frequent Subgraph Mining

Finally we identify the problem of frequent subgraph mining.

Frequent Subgraph Mining [64]. Given a graph dataset, $\mathcal{G} = \{G_0, \ldots, G_n\}$, $\textsc{Support}(g)$ denotes the number of graphs in $\mathcal{G}$ in which g is a subgraph. The problem of frequent subgraph mining is to find any subgraph g such that $\textsc{Support}(g) \geqslant \text{minSup}$ (a minimum support threshold).

In our case, frequent subgraph patterns correspond to subgraph patterns appearing in the body of queries within a query workload $\mathcal{Q}$.

3 CANDIDATES FOR VIEW MATERIALIZATION

The focus of our work is building a system that given a query workload $\mathcal{Q}$ selects the views that would be most beneficial to materialize for subsequent queries. The process for selecting the appropriate views is resolved into the following tasks: *(i)* forecasting the characteristics of future queries based on the up-till-now query workload $\mathcal{Q}$; *(ii)* defining the appropriate set of candidate views for materialization $\mathcal{V}_C$ based on our forecasting; *(iii)* selecting the views $\mathcal{V} \subseteq \mathcal{V}_C$ that will be materialized.

Subgraph Mining. This section focuses on the first two tasks and is base on mining frequent subgraph patterns in the workload of $\mathcal{Q}$. Our implementation is based on the existing work for the problem of *frequent subgraph mining* assuming that a subgraph pattern that has a high frequency within $\mathcal{Q}$ also has a high probability of appearing in future queries. Our work can be generalized to more sophisticated methods that take into account information such as

the temporal evolution of the query workload Q for forecasting graph patterns [22].

The section is structured as follows: Paragraph 3.1 describes the problem of frequent subgraph mining and introduces a preprocessing phase that allows to discover invaluable information for the view-materialization problem. Paragraph 3.2 focuses on the hierarchy of frequent subgraph patterns, the notion of closed frequent graph patterns, and why a pruning step needs to be performed to reduce the number of mined patterns, without affecting the quality of the selected views. As stated in the introduction, a frequent subgraph pattern p employs a dual role in our view selection methodology: it allows to summarize our query workload Q and identify candidate views for materialization. Paragraph 3.3 describes how to construct a summarization of Q based on the frequent subgraph patterns, while Paragraph 3.4 describes the construction of view candidates based on frequent patterns.

3.1 Mining Frequent Subgraph Patterns

The first step to identify the candidate views for materialization is discovering the subgraph patterns that appear with a high frequency within a query workload Q. Intuitively, a frequent pattern p that appears as a subgraph within n queries in Q can be used for the rewriting of at most n queries within Q. From a probabilistic standpoint, a view based on a pattern p that has a high probability of appearing in future queries, is more probable to be beneficial compared to a view based on a pattern of lower probability. To identify the frequent subgraph patterns within the workload of Q we employ frequent subgraph mining techniques.

3.1.1 Graph Representation Queries. To find frequent patterns within the query workload Q we employ the GSpan algorithm for pattern-mining which is described in [64], while using the implementation available in [20, 21]. Since the GSpan algorithm is intended for undirected graphs, we have to apply a transformation step that will convert a graph-query Q to its undirected graph representation.

We represent a query in the workload of Q using a variation of the standard methodology, described in [13], for representing relation-database queries using an undirected graphs. In the undirected representation, each triple-pattern (s, p, o) is represented by a node n_p labeled with p. Node n_p is connected via an edge labeled $subject$ to the node n_s and an edge labeled $object$ to the node n_o, where subject and object are fresh labels that do not appear within the workload of Q. The nodes n_s and n_o correspond to the subject and object of the initial triple pattern and are labeled with the identifier var that is used for identifying variables. In case that either s (or t) is actually a constant we connect the node n_s (or n_t) to itself via an edge labeled s (or t). It should be noted that after the GSPan algorithm is applied to the transformed query workload, the acquired frequent patterns are reverted to their initial form.

Example 3.1. This example illustrates the transformation process and how it is employed to unveil homomorphism-based frequent patterns. Suppose that we have the sequence of queries appearing in Formula 1, the queries Q_1, Q_2, Q_3 ask for the albums of various songs. For a minSup of 3, we want our mining algorithm to identify

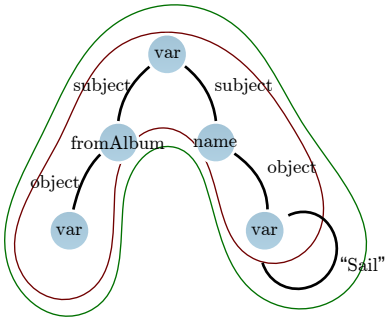

Figure 1: The undirected representation of the graph in Formula 3.1

as frequent the pattern:

$$(?sg, \texttt{name}, ?sN), (?sg, \texttt{fromAlbum}, ?alb). \tag{5}$$

We first show the transformation of the pattern

$$(?sg, \texttt{name}, \textit{"Sail"}), (?sg, \texttt{fromAlbum}, ?alb)$$

that will be represented in the mining process by the undirected graph of Figure 1.

If the part of the graph surrounded with the red line is a frequent pattern, then the inversion process will return as frequent the query pattern of the form (?x, name, ?y), (?z, fromAlbum, ?w), if on the other hand the subgraph surrounded with green line is frequent the pattern (?x, name, *"Sail"*), (?z, fromAlbum, ?w) will be returned.

3.2 The Frequent Pattern Hierarchy

During the mining process, the subgraph patterns that have a support greater than minSup are discovered. A frequent pattern g that has a support of n indicates that there exist exactly n queries in the workload of Q that have a subgraph in their body that is isomorphic to g. We will use $g' \mapsto g$ to indicate that g' is isomorphic to a subgraph of g. Because of the mining process, if the graph pattern g (of a support of n) is frequent, each of its subgraphs g' is also frequent (with a support of at least n). The latter is attributed to the fact that subgraph isomorphism is a transitive relation. Therefore an hierarchy between the frequent subgraph patterns is created.

3.2.1 Pruning the Frequent-Pattern Hierarchy. In order to reduce the search space of view candidates and at the same time minimize the summarization of the query workload Q, we first need to reduce the number of frequent subgraph patterns. A frequent subgraph pattern g is *closed* if there exists no supergraph of g with the same support in the graph database, i.e., for every g' such that $g \mapsto g'$ it applies that $\textsc{Support}(g) > \textsc{Support}(g')$. It can be proved that only closed patterns need to be considered for view materialization. Intuitively, a view V' based on the non-closed frequent pattern g' can be eliminated in favor of a view V based on its closed extension g that has the same support with g'. This is because the size of V is at most the size of V' (the additional edges will remove rows that satisfy V' but not V).

In our implementation, we consider a weaker form of a closure. For some $\epsilon \in [0, 1)$ and the graph pattern g, we say that g is approximately ϵ-closed when for every subgraph g' of g it applies that $(\textsc{Support}(g')-\textsc{Support}(g))/\text{minSup} < \epsilon$. By considering approximately

closed graph patterns for view selection, we manage to further reduce our search space without seriously affecting the effectiveness of our views.

3.3 Frequent Patterns as Query Workload Summarization

Frequent subgraph patterns $\mathcal{P}$ play a dual role in the view selection process: they provide the appropriate candidates for view materialization but also allow our algorithm to represent in compact form a query workload $\mathcal{Q}$ via a smaller multiset of pattern-based queries $\mathcal{Q}_{\mathcal{P}}$. In Theorem 3.2 we show that, for the linear cost model, a view V being beneficial for a query Q_p based on the pattern p is also beneficial for every query Q that contains the specific pattern, i.e., $p \mapsto \textsc{Body}(Q)$.

Theorem 3.2. *For the linear cost model, if p is a frequent pattern that is isomorphic to a subgraph of Q; $\vec{x}$ is a vector of all variables appearing in p; Q_p is a query of the form*

$$Q_p : q_p(\vec{x}) \leftarrow p \tag{6}$$

and there exists a rewriting $\textsc{Rwrt}(Q_p, \mathcal{V})$ with a benefit of d: then there exists a rewriting of Q w.r.t. $\mathcal{V}$ with $\textsc{Bnft}(Q, \mathcal{V}) \geqslant d$.

(Proof in Appendix A)

It should be noted that when creating the summarization of the query workload, we should adjust the multiplicity of queries in $\mathcal{Q}_{\mathcal{P}}$ to ensure that each query $Q \in \mathcal{Q}$ is not represented by multiple pattern-based queries in $\mathcal{Q}_{\mathcal{P}}$.

3.4 Identifying the Head of Candidate Views

Paragraph 3.1 was dedicated to finding the frequent patterns within a query workload $\mathcal{Q}$. Each frequent pattern p gives birth to a family of views of the form:

$$V : v(\vec{x}) \leftarrow t_1, t_2, \ldots, t_m \tag{7}$$

with the body of the view corresponding to the triples within the pattern p. If the pattern p has n variables that can be combined in 2^n different subsets, there exist 2^n candidates for view materialization.

3.4.1 Primordial Views. Considering all the different materialization choices is not a scalable solution. In order to reduce the aforementioned search space, we consider a syntactically restricted form of candidates for view materialization that we call *primordial views*. The body of a primordial view corresponds to a frequent pattern, while its head corresponds to all the variables appearing in some t_i in formula 7. E.g., if t_i is a triple pattern of the form $(?x, p, ?y)$, with $?x$, $?y$ being variables, then the head of the corresponding view in formula 7 becomes $v(?x, ?y)$. As long as it contains at least one constant, each primordial materialized view V_i can be represented in a native triple store by introducing a fresh predicate name p_{v_i} to identify the view.

Example 3.3. For the frequent pattern in Formula 5, the candidate primordial views for materialization are the following:

$$V_1 : v_1(?sg, ?sN) \leftarrow (?sg, \texttt{name}, ?sN), (?sg, \texttt{fromAlbum}, ?alb)$$
$$V_2 : v_2(?sg, ?alb) \leftarrow (?sg, \texttt{name}, ?sN), (?sg, \texttt{fromAlbum}, ?alb)$$

corresponding to the predicates `fromAlbum` and `name`. These views can be represented in the native RDF system by adding the new predicates p_{v_1} and p_{v_2}. It should be noted that if every song has a corresponding album, materializing p_{v_1} would not be a good materialization choice since it does not filter out any triples from the predicate `name`. If on the other hand, only 10% of the songs belong to a certain album, introducing p_{v_1} is a beneficial materialization choice.

In Proposition 3.4 we show that for native triple stores and the linear cost model, a view selection strategy that allows only for primordial views is an optimal solution for a query workload $\mathcal{Q}$ with no undistinguished variables.

Proposition 3.4. *Suppose that we have a KG that is stored in a native triple store, for every query workload $\mathcal{Q}$ containing conjunctive queries with no undistinguished variables, only primordial views are beneficial for query execution w.r.t. the linear cost model.*

(Proof in Appendix A)

3.4.2 Abritrary Views. In paragraph 3.4.1 we introduced primordial views and discussed their importance for the case that the underlying data and views are stored in a native triple store where information can only be represented as triples. We will now examine materialization options in the case that our data and views are stored within a relational store, allowing for relations of an arbitrary arity.

Combining Primordial Views. It's straightforward that a frequent pattern p having n variables can generate 2^n candidates for view materialization (corresponding to the different combinations of head variables). Nevertheless, based on Proposition 3.5, the variables within the head of a candidate view should form a connected subgraph within p, i.e., be the result of joining two or more primordial views on their common head variables (for primordial views having the same pattern p as their body). E.g., the view V_1 and V_2 in Example 3.3 can be combined to the non-primordial view:

$$V : v(?sg, ?sN, ?alb) \leftarrow (?sg, \texttt{name}, ?sN), (?sg, \texttt{fromAlbum}, ?alb)$$

Therefore, our view-selection strategy is based on joining primordial views that have the same pattern as their body and head variables in common. It should be noted that the number of such candidate views in some cases remains exponential w.r.t. the number of variables appearing in a frequent pattern p.

Proposition 3.5. *For a query Q and its optimal rewriting Q' with the same distinguished variables, if a set of distinguished variables appear together in some atomic formulae in Q, then they need to appear together in some atomic formulae in Q'.*

(Proof in Appendix A)

4 VIEW SELECTION WITH QUALITY GUARANTEES

In this section we focus on efficient and tractable algorithms for selecting the views that will be materialized.

Query Rewriting. Interrelated with the view selection process is the query rewriting that determines an equivalent rewriting of the initial query using the appropriate materialized views to decrease the execution cost of a query. Paragraph 4.1 examines query rewriting algorithms for both primordial and non-primordial views. Of primary importance for the view selection process is

Algorithm 1 Query Rewriting for Primordial Views

1: **function** RWRT(Query Q, PrimordialViews $\mathcal{V}$)
2: **for** each $t_i \in Q$ **do** $S_i := \{t_i\}$
3: **for** each containment mapping $m : V \rightarrow Q$ **do**
4: **if** $m(t'_j) = t_i$ and VARS(t'_j) $\in$ HEAD(V) **then**
5: $S_i := S_i \cup \{v(m(t'_j))\}$
6: **Create** rewritten query Q' **with**
7: HEAD(Q') := HEAD(Q)
8: BODY(Q') := $\emptyset$
9: **for** each $t_i \in Q$ **do**
10: $\theta := \min_{R \in S_i} \text{COST}^{\epsilon}(R)$
11: R_θ := the triple rewriting corresponding to θ
12: BODY(Q') := BODY(Q') $\wedge R_\theta$
13: **return** Q'

its algorithmic tractability, therefore we examine rewriting algorithms that have specific attributes to ensure the corresponding tractability. Specifically, we investigate tractable rewriting algorithms whose underlying function for benefit is submodular and also provide a good approximation of the optimal rewriting of a query. It should be noted that the corresponding rewriting algorithms are specific for the view selection process, while during query execution other algorithms can be employed (possibly non-tractable and non-submodular).

View Selection and the MNSSFKC problem. The aforementioned properties for tractability and submodularity are related to the problem of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint* (MNSSFKC). In paragraph 4.2 we outline the MNSSFKC problem and study the reduction from the view selection to the MNSSFKC problem. Based on the approximate solution for the MNSSFKC problem, we adapt our algorithm for view selection and study its quality guarantees for primordial and non-primordial views.

View Selection & Multi-Query Optimization. In paragraph 4.3, we propose a different approach to solve the view-selection problem by combining existing multi-query optimization techniques with the query-workload summarization algorithm.

4.1 Query Rewriting

To find an optimal query rewriting procedure, Levy et al. [37] consider rewritings that reduce the number of conjuncts in a query. We will follow an alternative approach considering optimality w.r.t. to the linear cost model assumption. Intuitively, our approach tries to reduce the total disk access cost that is proportional to the view sizes, while the optimality proposed in [37] aims at reducing the number of joins that will be executed during query evaluation. This means that our optimization goal is data driven, while the optimization goal presented in [37] is query driven. We consider query rewritings for the case of primordial and non-primordial views, focusing on tractable algorithms that induce a submodular benefit function.

4.1.1 Query Rewriting for Primordial Views. For primordial views and the linear cost model, the algorithm for query rewriting is pretty straightforward. By construction of primordial views, a query Q can be rewritten by replacing each triple $t_i \in Q$ with the corresponding primordial view of the minimum cost, if such a view exists.

The rewriting process for a query Q is detailed in Algorithm 1, with Formula 8 illustrating the form of a query Q and a view V:

$$Q : q(\vec{x}) \leftarrow t_1 \wedge \ldots \wedge t_n \tag{8a}$$
$$V : v(\vec{y}) \leftarrow t'_1 \wedge \ldots \wedge t'_m \tag{8b}$$

The rewriting process is completed in the following steps: I. For the i^{th} triple pattern t_i appearing in the body of Q we create the set S_i of candidate rewritings of it. Initially, the corresponding set contains the triple pattern itself, corresponding to the case that the triple pattern remains unchanged in its rewritten form (line 2). II. Then for each containment mapping $m : V \rightarrow Q$ from a materialized view $V \in \mathcal{V}$ to Q (line 3); that maps the triple t'_j in the body of V to the triple t_i in Q; such that the variables in t'_i also appear in the head of V (line 4): we add $v(m(\vec{y}))$ to the candidate rewritings of the t_i triple. It should be noted that, according to Section 3.4.1, the view v is actually represented within the knowledge graph by the newly-introduced predicate p_v. III. The algorithm then initializes the rewriting Q' of Q that has the same head variables as Q, while its body is initially empty (lines 6-8). IV. The rewriting of each triple pattern with the minimum cost will be selected to represent it in the body of the rewritten query Q' (lines 9-12).

PROPOSITION 4.1. *For a set $\mathcal{V}$ of materialized views consisting exclusively of primordial views, query rewriting can be decided in linear time w.r.t. the linear cost model.*

The proof is an immediate consequence of the fact that for each rewriting of Q, we need to replace each of its triples with the primordial view V that is mapped to the corresponding triple and has a minimum cost.

Example 4.2. For the query Q and the view V in Formulas 9, as well as the predicate p_v used to represent the view V,

$$Q : q(?sg, ?alb) \leftarrow (?sg, \texttt{name}, \textit{"Sail"}), (?sg, \texttt{fromAlbum}, ?alb) \tag{9a}$$
$$V : v(?x, ?y) \leftarrow (?\text{x}, \texttt{name}, ?\text{y}), (?x, \texttt{fromAlbum}, ?w) \tag{9b}$$

the triple $(?sg, \texttt{name}, \textit{"Sail"})$ can either remain unchanged, or rewritten to $(?sg, p_v, \textit{"Sail"})$. Suppose that $\text{COST}^{\epsilon}(?sg, \textit{name}, \textit{"Sail"}) = 1000$ and $\text{COST}^{\epsilon}(?sg, p_v, \textit{"Sail"}) = 100$ are the corresponding costs of reading the two triple patterns. Then, according to the linear cost model, the rewriting of the initial query using the triple pattern $(?sg, p_v, \textit{"Sail"})$ would benefit the query execution by 900 reads. Therefore, the initial query would be rewritten to Q':

$$Q' : q'(?sg, ?alb) \leftarrow (?sg, p_v, \textit{"Sail"}), (?sg, \texttt{fromAlbum}, ?alb).$$

4.1.2 Query Rewriting for Non-Primordial Views. The problem of query rewriting for non-primordial views on a column store is a NP-optimization problem for the linear cost model. This can be shown by considering all the feasible alternatives for the mapping of triples to views, computing the actual cost of each alternative, and selecting the rewriting with the lowest estimated cost. The NP-completeness of the problem remains open for our employed cost model. We will provide an approximation solution to the problem that greedily selects the best rewriting for each triple in Q. Our algorithm is performed in linear time, while for a conjunctive query of n variables and k conjuncts it is a factor $3 \cdot k/n$ approximation.

Algorithm 2 Query Rewriting for Non-Primordial Views

1: **function** ApproxRwrt(Query Q, Views $\mathcal{V}$)
2: **for** each $t_i \in Q$ **do** $S_i \leftarrow \{t_i\}$
3: **for** each containment mapping $m : V \rightarrow Q$ **do**
4: **if** $m(t'_j) = t_i$ and Vars$(t'_j) \in$ Head(V) **then**
5: $\vec{y}' :=$ Vars(t'_j)
6: $V' : v'(\vec{y}') \leftarrow v(\vec{y})$
7: $S_i := S_i \cup \{v'(m(\vec{y}'))\}$

▷ *The remaining algorithm is identical to lines 6-13 of Algorithm 1.*

Query Rewriting Algorithm. Our algorithm for rewriting approximates an optimal solution by greedily finding for each triple $t_i \in Q$ its most beneficial rewriting w.r.t. the existing views. The latter is achieved by ignoring the fact that a view V can be used to rewrite multiple triples into a single conjunct in the rewritten query Q'. The rewriting process for a query Q is a variation of the one presented in Algorithm 1. In Algorithm 2 we illustrate only the first step of the rewriting process, i.e., creating the candidate rewritings for each triple in Q: I. For the i^{th} triple pattern t_i appearing in the body of Q we create the set S_i of candidate rewritings for the specific triple. Initially S_i contains the triple pattern itself, corresponding to the case that the triple pattern remains unchanged in its rewritten form (line 2). II. Then for each containment mapping $m : V \rightarrow Q$ from a materialized view $V \in \mathcal{V}_C$ to Q (line 3); that maps the triple t'_j in the body of V to the triple t_i in Q; such that the variables in t'_j also appear in the rewritten head of V (line 4): we create the projection V' of the view V to the attributes that appear in t'_j (lines 5,6). Finally, we add to S_i a candidate rewriting for the triple pattern t_i that is based on the projection of the view V (line 7). It is straightforward that the rewriting process is performed in linear time w.r.t. the number of existing containment mappings.

Intuitively, Algorithm 1 performs rewritings by decomposing each view to its corresponding *primordial projections*, i.e., projections that correspond to the definition of a primordial view in Section 3.4.1. E.g., in Formula 10, the view V is decomposed to the primordial projections V'_1 and V'_2:

$$\begin{aligned} V &: v(?x, ?y, ?w) \leftarrow (?x, \texttt{name}, ?y), (?x, \texttt{fromAlbum}, ?w) \\ V'_1 &: v'_1(?x, ?y) \leftarrow (?x, \texttt{name}, ?y), (?x, \texttt{fromAlbum}, ?w) \\ V'_2 &: v'_2(?x, ?w) \leftarrow (?x, \texttt{name}, ?y), (?x, \texttt{fromAlbum}, ?w) \end{aligned} \tag{10}$$

It should be noted that when computing the cost of rewriting the triple t_i based on some primordial projection V' of a view V, the cost of reading V is reduced to the cost of reading the projection V'. This is due to the ability of column stores to only access the appropriate attributes and not an entire relation. Proposition 4.3 studies the quality guarantees of such an approximation.

Proposition 4.3. *For a query containing n variable vertices and k edges, Algorithm 1 will produce a rewriting whose cost is, in the worst case, a $\frac{3 \cdot k}{n}$-approximation of the cost of the optimal rewriting. (Proof in Appendix A)*

Optimizations. In our implementation, we have devised improved versions of Algorithm 2 that are still being executed in linear time. One straightforward improvement is to merge multiple primordial projections that refer to the same view whenever they have some term in common. E.g., suppose that the query Q' in Formula 11 is a rewriting that uses the primordial projections V'_1, V'_2 in Formula 10:

$$Q' : q'(?sg, ?alb) \leftarrow v'_1(?sg, \text{“Sail”}), v'_2(?sg, ?alb) \tag{11}$$

By re-combining the two projections, query Q' can be further rewritten to its more efficient form that uses the initial non-primordial view V:

$$Q'' : q''(?sg, ?alb) \leftarrow v(?sg, \text{“Sail”}, ?\, alb).$$

In our implementation, we have further improved the performance of Algorithm 2 by tweaking the cost estimation function Cost$^{\epsilon}$ in order to encourage merging of primordial projections to the corresponding initial views. Intuitively, if an attribute is shared between multiple primordial projections of the same originating view, the cost of reading the corresponding attribute will be distributed between the primordial projections. That way, the choice of primordial projections with shared attributes is encouraged throughout the rewriting process, which will also encourage merging to the initial non-primordial views in later steps.

4.2 View Selection

Having defined the query rewriting process, we proceed to designate the view selection methodology.

The *degree of benefit* of a rewriting function to a query Q (a set of queries $\mathcal{Q}$) w.r.t. to a set of views $\mathcal{V}$, defined in Section 2.2, is

$$\begin{aligned} \text{Bnft}(Q, \mathcal{V}) &= \text{Cost}^{\epsilon}(Q) - \text{Cost}^{\epsilon}(\text{Rwrt}(Q, \mathcal{V})) \\ \text{Bnft}(\mathcal{Q}, \mathcal{V}) &= \sum_{Q \in \mathcal{Q}} \text{Bnft}(Q, \mathcal{V}). \end{aligned}$$

A set of views $\mathcal{V}$ is beneficial for a query $Q \in \mathcal{Q}$ when there exists a rewriting $Q' = \text{Rwrt}(Q, \mathcal{V})$ such that $\text{Cost}^{\epsilon}(Q') < \text{Cost}^{\epsilon}(Q)$, or equivalently, when $\text{Bnft}(Q, \mathcal{V}) > 0$. The objective of the *view selection process* is to identify the views in $\mathcal{V}$ that are the most beneficial to materialize w.r.t. the query workload $\mathcal{Q}$, i.e., that maximize $\text{Bnft}(\mathcal{Q}, \mathcal{V})$. In our implementation, when considering the benefit of a view V (set of views $\mathcal{V}$) to a query workload $\mathcal{Q}$, we employ the summarization $\mathcal{Q}_{\mathcal{P}}$ of the query workload based on the frequent patterns appearing in it (Section 3.3). Based on Theorem 3.2 the summarization provides a lower bound of the actual benefit.

Problem 1 (View Selection Problem). *Given a set of candidate views $\mathcal{V}_C$, a query workload $\mathcal{Q}$, and a storage capacity of b: which subset $\mathcal{V} \subseteq \mathcal{V}_C$ to materialize such that the size of $\mathcal{V}$ is less than b and the query workload of $\mathcal{Q}$ w.r.t. $\mathcal{V}$ is benefited the most.*

MNssfKc Problem. We will reduce view selection to the problem of *Maximizing a Nondecreasing Submodular Set Function Subject to a Knapsack Constraint* (MNssfKc problem) presented in [58], 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 2 (MNssfKc [58]). *Let $I = \{1, \ldots, n\}$; $i \in I$ and b be non-negative integers; and $f(\cdot)$ be a nonnegative, nondecreasing, submodular, polynomially computable set function*[1]*. Consider the following*

[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 $S \subseteq T$.

Algorithm 3 View Selection

1: **function** ViewSelection(StorageSpace b, CandidateViews $\mathcal{V}_C$, MaterializedViews $\mathcal{V}$)
2: $\quad\mathcal{V}_C := \{V \in \mathcal{V}_C : \text{Cost}^{\epsilon}(V) \leqslant b\}$
3: $\quad$**if** $\mathcal{V} = \emptyset$ **then return** $\mathcal{V}$
4: $\quad\theta_{\max} := 0$
5: $\quad$**for all** $V \in \mathcal{V}_C$ **do**
6: $\quad\quad\theta := \frac{\text{Bnft}(Q,\mathcal{V}\cup\{V\})-\text{Bnft}(Q,\mathcal{V})}{\text{Size}^{\epsilon}(V)}$
7: $\quad\quad$**if** $\theta > \theta_{\max}$ **then**
8: $\quad\quad\quad V_t := V$
9: $\quad\quad\quad\theta_{\max} := \theta$
10: $\quad$**return** ViewSelection($b - \text{Cost}^{\epsilon}(V), \mathcal{V}_C \setminus \{V_t\}, \mathcal{V} \cup \{V_t\}$)

optimization problem:

$$\max_{S \subseteq I} \left\{ f(S) : \sum_{i \in S} c_i \leqslant b \right\}. \tag{12}$$

Reduction. We first identify the parameters of the reduction from Problem 1 to Problem 2 for primordial and non-primordial views. I. For the set $\mathcal{V}_C \leftarrow \{V_1, \ldots, V_n\}$ such that $|\mathcal{V}_C| = n$, we define an arbitrary bijection $\text{Bi} : \mathcal{V}_C \leftrightarrow [\![1, n]\!]$ and set $I := [\![1, n]\!]$. II. For each $i \in [\![1, n]\!]$ we define $c_i = \text{Cost}^{\epsilon}(\text{Bi}^{-1}(i))$ 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 $\mathcal{V} \subseteq \mathcal{V}_C$ (by mapping each $i \in S$ to $\text{Bi}^{-1}(i) \in \mathcal{V}$). For the specific $\mathcal{V}$, we define $f(S)$ to be the benefit of the materialized views in $\mathcal{V}$ to the query workload in $\mathcal{Q}$, i.e., $\text{Bnft}(\mathcal{Q}, \mathcal{V})$. IV. Finally, b is the available storage space for materialization.

Approximation Algorithm. It is straightforward to show that by solving the formula in 12 we acquire an optimal solution for the view selection problem. What is interesting about Problem 2 is that there exist a $(1 - e^{-1})$-approximation algorithm for maximizing formula 12 as long as $\text{Bnft}(\mathcal{Q}, \mathcal{V})$ and consequently $f(S)$ are non-negative, nondecreasing, polynomially computable, submodular functions [58]. $\text{Bnft}(\mathcal{Q}, \mathcal{V})$ is: I. non-negative since every query Q is a valid rewriting of itself; II. non-decreasing since for the sets of views $\mathcal{V} \subseteq \mathcal{V}'$, each rewriting of Q using $\mathcal{V}$ is also a valid rewriting of Q using $\mathcal{V}'$; III. while the benefit functions for the rewriting Algorithms are polynomial for primordial and non-primordial views (Algorithms 1,2). IV. In Proposition 4.4 we prove that the Bnft function is submodular for the linear cost model and the rewriting algorithm intended for primordial views (Algorithm 1). The Bnft function for the rewriting algorithm of non-primordial views is not submodular, nevertheless, as presented in the experimental evaluation section (Section 5), is efficiently used in combination with the suggested view-selection algorithm.

Proposition 4.4. *For $\mathcal{Q}$ being a set of queries, the $\text{Bnft}(\mathcal{Q},\mathcal{V})$ function for primordial views, based on the rewritings of Algorithm 1 and the linear cost model Cost^{ϵ}, is a nondecreasing and submodular set function.*

(Proof in Appendix A)

View Selection Algorithm. An adjustment of the approximation algorithm appearing in [58] is presented in Algorithm 3 for the view selection problem with $\mathcal{V}_C$ being the candidates views for materialization, $\mathcal{V}$ the selected views, and b the available storage. I. The first step of the algorithm is to remove the candidate views that do not fit in the available storage space (line 2). II. If the set of remaining candidates is empty, then the set $\mathcal{V}$ of the already selected views for materialization is returned (line 3). III. Else, the algorithm finds the view $V_t \in \mathcal{V}_C$ whose addition to $\mathcal{V}$ produces the maximum benefit to storage cost ratio (lines 6,8). IV. The corresponding view is removed from the set $\mathcal{V}_C$ of candidates and added to the set $\mathcal{V}$ of views that are selected for materialization (lines 10). V. The algorithm is then executed for the updated $\mathcal{V}_C$ and $\mathcal{V}$ (line 10). For our study, we assume that the cost estimation function Cost^{ϵ} has a $O(1)$ complexity, thus the estimated cost of a query can be computed immediately. The latter ensures that the Bnft function is actually linear. In practice, we need to consider approximation counters to ensure a tractable complexity of the Cost^{ϵ} function. The latter is out of the scope of the paper and will be considered in future work.

Algorithmic Guarantees. We now examine the algorithmic guarantees of the proposed methodology. When restricting the selection process to primordial views, Algorithm 3 provides the same guarantees as the MNssfKc problem, i.e., a $(1 - e^{-1})$-approximation of the optimal selection of views w.r.t. the linear cost model assumption. For non-primordial views, the selection process involves two approximation steps that contribute to the final result: the approximation step related to the minimization of the ApproxRwrt and the approximation step related to the maximization of the ViewSelection process. Unfortunately, we can find the degree of approximation only during the view selection process, by combining the $\frac{3 \cdot k}{n}$-approximation degree of the view selection process that affects the Bnft function with the $(1 - e^{-1})$-approximation degree related to the view selection process. Therefore, guarantees can be computing only during the view-selection process.

Optimizations. Finding a locally optimal view $V_t \in \mathcal{V}$ is the most demanding step of Algorithm 3 (lines 6,8). To prune the search space, we keep a memoization table for the benefit to storage cost ratio. For updating the memoization table, we consider a variation of the technique presented in [51]. The updated benefit ratio is computed only for the view that has the optimal benefit ratio based on previous computations. If the updated ratio agrees with the previously computed ratio, or is greater than all benefit ratios of other views, the corresponding view will be selected. Otherwise, the view with the next highest benefit ratio is examined.

4.3 View Selection & Multi-Query Optimization Algorithms

In this paragraph we propose a different approach to solve the view-selection problem (Problem 1) by combining existing multi-query optimization techniques with the query-workload summarization algorithm that was described in Section 3.3.

Under the assumption that each frequent subgraph pattern has a high probability of appearing in subsequent queries, we employ existing multi-query optimization techniques on the query-workload summaries that are derived from frequent-pattern mining. Our selection process is performed in two simple steps: I. We first construct a summary based on the query workload summarization

process described in Section 3.3. The query-workload summary assigns a corresponding multiplicity to each pattern-derived query according to Section 3. II. Then the multi-query optimization algorithm selects the most appropriate views for materialization. The idea is that the summarization step allows multi-query optimization algorithms, designed for the processing of hundreds/thousands of queries, to be applied to million of queries.

Implementation. In our implementation we have employed the multi-query optimization algorithm described in [31] that also provides guarantees regarding the quality of the views that will be selected. The corresponding approach is evaluated in Section 5 and compared with the view-selection strategies for primordial and non-primordial views.

5 EXPERIMENTAL EVALUATION

The aim of our evaluation section is to examine the performance of the view selection methodology as well as the quality of the views that get selected throughout the view-selection process. For our testing scenarios, our application takes as input a knowledge graph G; a query workload Q, corresponding to past queries; a query workload Q_T corresponding to future queries; and produces the views $\mathcal{V}$ that will be materialized for future query execution. The quality of the views in $\mathcal{V}$ is later tested for rewritings w.r.t. to the query workload Q_T. We consider two different scenarios regarding the relational storage of the knowledge graph G: *(i)* storing the knowledge graph G as a set of triples within a native triple store; *(ii)* storing the knowledge graph G as a set of triples within a relational column store. In both cases, we selected MonetDb as the underlying database. For the case of MonetDb emulating a native triple store, i.e., a relational storage that allows to store only triples, our views are limited to relations containing two attributes (Section 3.4.1).

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.

Implementation Setup. We have implemented our algorithm in Java 8 using the Apache Jena 3.6.0 open source Semantic Web framework [28] 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 [38].

Benchmark. For benchmarking our methodology, we employed the DbPedia *semantic knowledge graph* [2, 7] 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* Q, containing 1,277,711 queries that will be used for selecting the appropriate views for materialization and the DbPedia *testing query workload* 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.

The Pattern-Mining Algorithm. The subgraph pattern-mining algorithm is detrimental to the view selection process since it allows to generate the corresponding candidate views as well as an effective summarization of the query workload. Figure 2 illustrates the correspondence between the minSup parameter of the pattern-mining process and its overall execution time. The frequent-subgraph patterns are mined from the DbPedia training query-workload Q. The x-axis displays the selected minimum support of a frequent pattern, while the y-axis displays the time to discover all frequent patterns. It should be noted that our implementation involves a linear preprocessing optimization step that dominates the process. Nevertheless, we observe that the lesser the minSup is, the greater is the corresponding mining time. This is because a higher minSup results to a lesser number of mined patterns. The latter is displayed in Figure 3 that illustrates the number of discovered patterns with respect to different minSup values.

Subgraph Hierarchy. Next our algorithm creates an hierarchy between the different patterns which is crucial for acquiring the query-workload summary $Q_{\mathcal{P}}$ and generating the candidate views (Section 3.2). During the process, approximately non-closed patterns and patterns containing automorphisms are removed. In Figure 3, we observe that a great portion of the initial subgraph patterns are either non-closed or contain automorphisms and thus are removed. Figure 4 illustrates the different phases of the hierarchy creation and their execution time. It should be noted that the hierarchy creation dominates the process, though computed in a few seconds.

Scalability of the View-Selection process. In order to showcase our algorithm's capability to efficiently select the appropriate views for materialization, we study its execution time on varying query-workload sizes. Figure 5 illustrates the various stages of our algorithm for training w.r.t. query-workload samples ranging from 183,958 to 1,287,706 queries, while the available storage for materialization is 1000 records in all cases. For the different training query-workload samples, the mining algorithm searches for patterns that appear in 0.7‰ of the queries in the workload. We observe that our algorithm behaves well for augmenting workload sizes, this can be attributed to the pattern-mining step that effectively represents each workload by a corresponding summarization. Therefore, the view-selection process depends on the size of the summarization and not on the actual size of the query-workload.

Effectiveness of Selected 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; view selection considering primordial, non-primordial (combined) views, or views that were employed by extending the Volcano-based multi-query optimization methodology described in Section 4.3. We should point out that for the view-selection methodology described in Section 4.3, we also employ the linear cost model assumption and not a more complicated cost estimation function.

Figure 6 illustrates the percentage of the queries that are benefited from each view-selection methodology. The x-axis represents the available storage for materialization–measured in terms

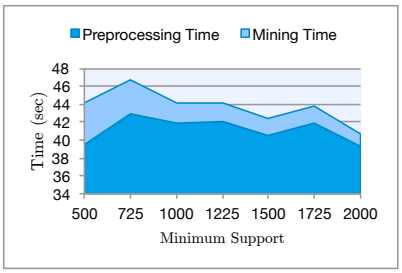

Figure 2: Mining Time

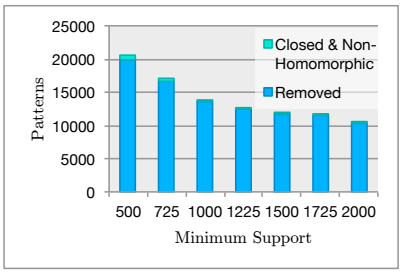

Figure 3: Number of Patterns

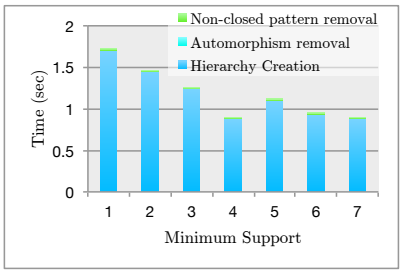

Figure 4: Hierarchy Creation Phases

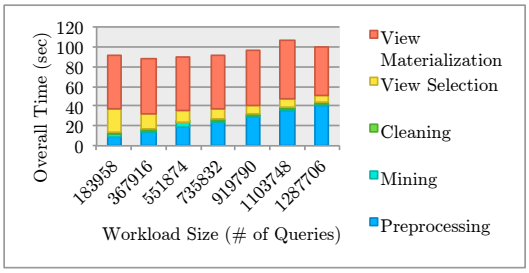

Figure 5: The scalability of the view selection process for various query-workload sizes.

of records used for materialization–, while the y-axis the percentage of benefited queries. The two algorithms based on primordial and non-primordial views have similar results, while they benefit more queries compared to the Volcano-based view-selection algorithm.

Figure 7 illustrates the overall execution time for the queries in the testing workload Q_T; for the different view-selection methodologies; and varying capacities for materialization. The x-axis in Figure 7 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 various view-selection alternatives. We observe that the primordial and non-primordial techniques are more beneficial for the overall execution time compared to the proposed Volcano-based technique. Also, the query workload is insignificantly benefited for more than 1000 records of available storage.

Figure 8 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 8 is in logarithmic scale and all the three algorithms manage to substantially reduce the overall execution time for the benefited queries.

6 RELATED WORK

View materialization techniques have been extensively studied by the data-management community in the context of multiple-query optimization, Semantic Web & graph data systems, and data warehouses that are used to accelerate On-Line Analytical Processing.

► **Multiple-Query Optimization.** The view-selection process for the multiple-query optimization problem identifies the appropriate views that will be used for answering to a given set of queries. Sellis [52] studies the problem of multiple-query optimization providing its systematic analysis and considering global access plans that access subqueries. Mistry et al. [44], Roy et al. [51] examine algorithms for multi-query optimization by selecting materialized views and indexes based on the Directed-Acyclic-Graph representation of the query plan to identify common subexpressions. Agarawal et al. [4] describe as system for view and index selection that incorporates several heuristics for pruning the space of possible view configurations. Zhou et al. [66] present an efficient solution for the problem of common subexpression identification by introducing a light-weight mechanism, called tables signatures, for identifying sharable subexpressions.

Chirkova et al. [15] formalize the view selection problem and provide a lower Exp and an upper 3Exp bound for it. Kathuria and Sudarshan [31] devised an approximation algorithm that runs in time quadratic to the number of common subexpressions and provide theoretical guarantees on the quality of the solution obtained. Jindal et al. [29] focus on the problem of subexpression selection, i.e., computing the subexpressions of a query that are most beneficial to be materialized and reduce it to the bipartite graph labeling problem, and integrate their implementation into the Cloudviews system [30].

A different methodology for solving the multiple-query and the view selection problem has been presented by Bayir et al. [8], Chaves et al. [14] that employ evolutionary techniques such as genetic algorithms. An overall analysis of the view selection problem has been presented by Mami and Bellahsene [40].

Our approach differs from previews view-materialization approaches since it allows to plug in various subgraph mining & forecasting solutions in order to predict the graph-patterns that will appear in future queries. It takes advantage of the graph-nature of knowledge-graph queries that allows to employ pattern-mining and forecasting techniques.

► **Semantic Web & Graph Data Systems.** Much research effort has been invested in the development of scalable centralised or distributed triple stores, techniques for indexing KGs and for processing queries. Among the centralised approaches, native triple stores like Jena [41], Sesame [11], HexaStore [63], SW-Store [3], MonetDB-RDF [54], RDF-3X [47], and BitMat [6] have been carefully designed to keep up pace with the growing scale of RDF collections. Systems like TriAD [25], RDFox [46], H-RDF-3X [27], EAGRE [65] implement various optimizations for the distributed execution of joins.

View materialization techniques have recently gained attention by the Semantic Web community and graph data systems. In [16], an approach for the materialization of shortcuts that reduces the execution cost of path queries is suggested. In [23], a different materialization strategy where an initial query workload $\mathcal{W}$ is transformed to

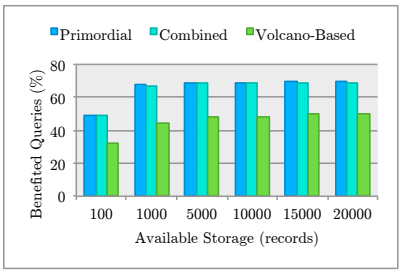

Figure 6: Benefited Queries (%)

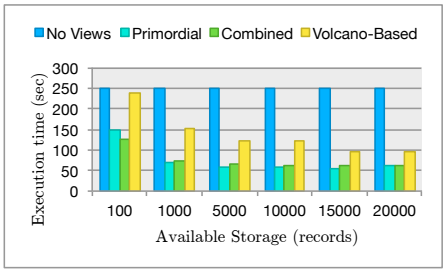

Figure 7: Execution times: Q_T

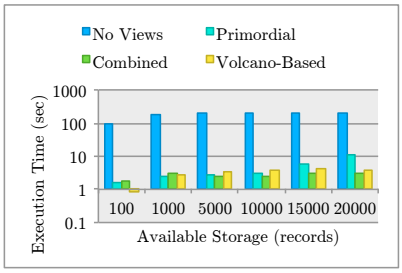

Figure 8: Execution times: benefited Q_T

a set of simpler views $\mathcal{V}$ along with a set of rewritings is presented. In [48], a strategy that caches SPARQL-query results and uses them to rewrite queries is studied. Caching strategies for graph query processing have been studied in [61, 62]. The caching algorithms in [48] and [61, 62] are based on finding subgraph-isomorphisms between incoming and cached queries. The approach in [48] is based on a canonical labelling algorithm, while [61, 62] adopt a *filter then verify* strategy where candidate graphs for isomorphism are filtered out based on certain features and then the actual test for isomorphism is performed. Finally, [42] studies the creation of an indexing structure that classifies triples based on the properties of their subjects and objects.

For a detailed analysis of Knowledge Graphs such that of DbPedia, the reader may refer to the existing bibliography. The query workload of DBPedia is studied in [49] and an analysis of the different operators that appear within DBPedia queries is performed. For various workloads, the structural characteristics related to the graph representation of queries are studied in [10], along with the evolution of queries over time. Finally, a study of the Wikidata knowledge graph is presented in [39].

► **Data Warehouses.** View-selection techniques have been studied for data warehouses and problem of online analytical processing. Several early techniques were proposed including AND/OR graphs [24], modeling the problem as a state optimization [59], and lattices to represent data cube operations [26, 35, 45], while the problem of view management has been also studied for decentralized OLAP applications using blockchains [43]. It should be noted that the problem of view materialization for data warehouses has different objectives targeting the improvement of Roll-up, Drill-down, and Slicing & Dicing operations.

7 CONCLUSIONS AND FUTURE WORK

In our work we studied the problem of view materialization and examined subgraph pattern mining techniques that allow to create efficient summarizations of our workloads and to identify the candidate views for materialization. We have also studied the correspondence between the view selection and the MNssfKc problem and proved that there exists a tractable view-selection process for native triple stores that allows a $(1 - e^{-1})$-approximation of the optimal selection of views. Our experimental evaluation showed that all the steps of the view-selection process are completed in a few minutes, while the corresponding rewritings accelerate 67.68% of the queries in the DbPedia query workload and those queries are executed in 2.19% of their initial time.

In future work we intend to study view-selection techniques for streaming data [57], focusing on stream processing for Semantic Web applications [32–34]; as well as complex event processing [36]. Additionally, we intend to integrate approximate counters [60] into our view-selection methodology that will be used by our cost-estimation function and examine entropy-based techniques when computing the benefit of different view alternatives [9]. Finally, we intend to generalize our work towards more sophisticated pattern-mining methods that take into account information such as the temporal evolution of the query workload Q for forecasting graph patterns [22].

ACKNOWLEDGMENTS

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project "Reinforcement of Postdoctoral Researchers - 2nd Cycle" (MIS-5033021), implemented by the State Scholarships Foundation (IKY).

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements No. 945539 (Human Brain Project SGA3) and No. 777413 (Delivering Agile Research Excellence on European e-Infrastructures).

A THEORETICAL PROOFS

Proof of Theorem 3.2. Based on the subgraph isomorphism $p \mapsto \text{Body}(Q)$ we know that there is a bijection $f : \text{Vars}(\vec{x}) \to \text{Vars}(Q)$ such that $f(p)$ is a subgraph of Q. Therefore, Q has the form:

$$Q : q(\vec{y}) \leftarrow \Psi \wedge f(p) \tag{13}$$

with Ψ representing the remaining part of the query, i.e., the part that is not isomorphic to p. Since the vector $\vec{x}$ in Formula 6 contains all the variables in the body of p, query $Q' : q'(\vec{y}) \leftarrow \Psi \wedge q_p(f(\vec{x}))$ is equivalent to Q (Point ♣).

Suppose that Q'_p is the rewriting $\text{Rwrt}(Q_p, \mathcal{V})$ of Q_p with a benefit of d: $Q'_p : q'_p(\vec{x}) \leftarrow p'$. By definition of a rewriting, Q_p and Q'_p are equivalent queries (Point ♠). We consider the query Q'' that we will show is a valid rewriting of Q with a benefit of d:

$$Q'' : q''(\vec{y}) \leftarrow \Psi \wedge f(P'). \tag{14}$$

Since $\vec{x}$ contains all the variables in P', Q'' is equivalent to $Q''' : q'''(\vec{y}) \leftarrow \Psi \wedge q'_p(f(\vec{x}))$. Based on Point ♠, Q''' is equivalent to Q'. The latter combined with Point ♣ and the transitivity of query

equivalence, implies that Q'' is an equivalent rewriting of Q as we wanted to show. Based on the linear independence of the linear cost model, it applies that

$$\begin{aligned}\textsc{Cost}^{\epsilon}(Q) - \textsc{Cost}^{\epsilon}(Q'') &= \textsc{Cost}^{\epsilon}(\Psi) + \textsc{Cost}^{\epsilon}(f(P)) \\ &\quad - \textsc{Cost}^{\epsilon}(\Psi) - \textsc{Cost}^{\epsilon}(f(P')) \\ &= \textsc{Cost}^{\epsilon}(P) - \textsc{Cost}^{\epsilon}(P') = \textsc{Bnft}(Q, W).\end{aligned}$$

Therefore, we know that there exists a rewriting of Q with a benefit of d which completes our proof. □

Proof of Proposition 3.4. We first show the next proposition (Proposition 3.5) that will be used throughout the proof.

For a graph query Q containing two variables in each triple pattern, suppose that there exists a rewriting Q' that uses a view V which is not a primordial view. Based on Proposition 3.5 and since Q contains only distinguished variables, for each triple t_i in Q the variables in t_i appear together in some atomic formula in Q'. Therefore, by removing the atomic formulae of V from Q' we get the query Q'' that is also a rewriting of Q. For the linear cost model it applies that $\textsc{Cost}^{\epsilon}(Q'') < \textsc{Cost}^{\epsilon}(Q')$ and therefore a non-primordial view V won't be beneficial for any of the queries. □

Proof of Proposition 3.5. We first rewrite Q and Q' such that they both have the same head variables. The rewritten query Q' is of the form:

$$Q' : q'(\vec{x}) \leftarrow \bigwedge_{i=1}^{k} v_i(\vec{\omega}_i') \wedge \bigwedge_{i=k+1}^{l} t_i' \tag{15}$$

with each predicate v_i corresponding to a view and t_is corresponding triples with variables. We create a query Q'' by replacing each view atom in Q' with the view's corresponding body. E.g., the head of the view $v(\vec{x})$ in Formula 7 will be replaced with its body $t_1, \ldots, t_m$. For the non-distinguished variables in V_i, we use fresh names, such that each atomic formulae in Q' has its dedicated set of undistinguished variables in Q''.

By construction of Q'', we know that Q'' and Q' are equivalent and since Q' is equivalent to Q—being its rewriting—, Q'' and Q are also equivalent queries. The latter implies that there exists the containment mappings $m : Q \rightarrow Q''$ and $m' : Q'' \rightarrow Q$. Since Q and Q'' have the same distinguished variables in there heads, it applies that each distinguished variable in Q is mapped to itself in Q'' and vice versa. Since there exists a containment mapping, for every triple t_i appearing in the body of Q, there exists a corresponding triple t_j' in the body of Q'' such that $m(\textsc{Vars}(t_i)) = \textsc{Vars}(t_j')$. The latter and since Q and Q'' have the same distinguished variables, implies that if a set of distinguished variables appears in t_i, it should also appear in t_j'. By construction of Q'' from Q', we have that the distinguished variables of t_j' also appear in the head of V —in any other case, fresh variables are introduced— which concludes our proof. □

Proof of Proposition 4.3. Suppose that the query Q containing k triples and n variables has the form:

$$Q : q(\vec{x}) \leftarrow t_1 \wedge \ldots \wedge t_k \tag{16}$$

and its corresponding rewriting $Q' := \textsc{ApproxRwrt}(Q, \mathcal{V}_C)$ using the views $V_1, \ldots, V_k$ to represent the triples $t_1, \ldots, t_k$ is:

$$Q' : q'(\vec{x}) \leftarrow v_1(\vec{x}_1') \wedge \ldots \wedge v_k(\vec{x}_k').$$

The worst case will occur when there exists a view V of exactly k columns and V is isomorphic to Q, thus there exists a rewriting Q'' $Q'' : q''(\vec{x}) \leftarrow v(\vec{x}')$.

Since Algorithm 2, selects a rewriting based on the view V_i to replace the triple pattern t_i, according to Algorithm 2 there exists a triple pattern t_j' in the body of V_i and a containment mapping m such that $m(t_j') = t_i$. Suppose that t_i and t_j' contain m_i and l_i variables respectively. Then the cost of using V_i to replace the corresponding triple would be $l_i \times \textsc{Length}(V_i)$, while the cost of using V to replace the corresponding triple would be $m_i \times \textsc{Length}(V)$. Algorithm 2 indicates that:

$$l_i \times \textsc{Length}(V_i) \leq m_i \times \textsc{Length}(V) \tag{17}$$

otherwise V would be selected for the triple pattern's rewriting instead of V_i. The ratio between the approximation and the actual cost of the optimal rewriting is:

$$\frac{\sum_{i=1}^{k}(l_i \times \textsc{Length}(V_i))}{n \times \textsc{Length}(V)} \leq \frac{\sum_{i=1}^{k} m_i}{n} \tag{18}$$

the inequality is an immediate result of applying formula 17 to the initial ratio. The ratio $\sum_{i=1}^{k} m_i/n$ is maximized when the quantity m_i gets maximized, i.e., when each triple in Q has 3 variables. Therefore the ratio is less or equal than $\frac{3 \cdot k}{n}$ and our algorithm is a $\frac{3 \cdot k}{n}$ approximation of the optimal rewriting. In real-world queries, that have at most 2 variables in each of their triple patterns, the approximation ratio becomes $\frac{2 \cdot k}{n}$. □

Proof of Proposition 4.4. For $\boldsymbol{Q}$ being a set of queries, $Q \in \boldsymbol{Q}$, and $\mathcal{V}$ being the, selected according to Algorithm 1, set of views to materialize : to show that $\textsc{Bnft}(\boldsymbol{Q}, \mathcal{V})$ is submodular, it suffices to show that $\textsc{Bnft}(Q, \mathcal{V})$ is submodular, i.e., to prove submodularity for a single query. To prove the latter, we construct the rewritings of Q w.r.t. $\mathcal{V}_1$ and $\mathcal{V}_2$ based on the optimal rewritings for $\mathcal{V}_1 \cup \mathcal{V}_2$ and $\mathcal{V}_1 \cap \mathcal{V}_2$. The corresponding rewritings provide a lower value for $\textsc{Bnft}(Q, \mathcal{V}_1) + \textsc{Bnft}(Q, \mathcal{V}_2)$ and we will show that this value equals $\textsc{Bnft}(Q, \mathcal{V}_1 + \mathcal{V}_2) + \textsc{Bnft}(Q, \mathcal{V}_1 \cap \mathcal{V}_2)$ which will complete our proof since it shows that the Bnft function is submodular.

Suppose that the rewriting of the triple t_i in the query Q w.r.t. a set of materialized views $\mathcal{V}$ is $\textsc{Rwrt}(t_i, \mathcal{V})$ according to Algorithm 4.1, we will define a candidate rewriting for $\mathcal{V}_1$ and $\mathcal{V}_2$ based on the optimal rewritings for $\mathcal{V}_1 \cup \mathcal{V}_2$ and $\mathcal{V}_1 \cap \mathcal{V}_2$. In the next rewriting, we denote with ♣ the cases when the rewriting $\textsc{Rwrt}(t_i, \mathcal{V}_1 \cup \mathcal{V}_2)$ utilized a view $V \in \mathcal{V}_1 \setminus \mathcal{V}_2$ to rewrite t_i, i.e., a view appearing exclusively in $\mathcal{V}_1$ and not $\mathcal{V}_2$:

$$\textsc{Rwrt}(t(\vec{\omega}_i), \mathcal{V}_1) \leftarrow \begin{cases} \textsc{Rwrt}(t_i, \mathcal{V}_1 \cup \mathcal{V}_2) & \text{Case ♣} \\ \textsc{Rwrt}(t_i, \mathcal{V}_1 \cap \mathcal{V}_2) & \text{otherwise} \end{cases}$$

$$\textsc{Rwrt}(t(\vec{\omega}_i), \mathcal{V}_2) \leftarrow \begin{cases} \textsc{Rwrt}(t_i, \mathcal{V}_1 \cap \mathcal{V}_2) & \text{Case ♣} \\ \textsc{Rwrt}(t_i, \mathcal{V}_1 \cup \mathcal{V}_2) & \text{otherwise} \end{cases}$$

By construction of the new rewriting, it can be checked that each triple $t_i \in Q$ is rewritten in two different ways in $\textsc{Rwrt}(t_i, \mathcal{V}_1)$ and $\textsc{Rwrt}(t_i, \mathcal{V}_2)$. One of the rewritings is based on $\mathcal{V}_1 \cap \mathcal{V}_2$ and the other is based on $\mathcal{V}_1 \cup \mathcal{V}_2$. Since the $\textsc{Cost}^{\epsilon}$ function is linear independent (for the linear cost model), it is straightforward that the necessary equality between the rewritings applies which completes our proof. □

REFERENCES

[1] 2012. DbPedia log. https://github.com/AKSW/SPARQL2NL/tree/master/resources/dbpediaLog [Online; accessed 16-September-2021].

[2] 2019. DbPedia 3.9. http://downloads.dbpedia.org/3.9/en/ [Online; accessed 16-September-2021].

[3] Daniel J Abadi, Adam Marcus, Samuel R Madden, and Kate Hollenbach. 2009. SW-Store: a vertically partitioned DBMS for Semantic Web data management. *VLDB J.* 18, 2 (2009), 385–406.

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

[5] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. 2000. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, Vol. 2000. 496–505.

[6] Medha Atre, Vineet Chaoji, Mohammed J Zaki, and James A Hendler. 2010. Matrix Bit loaded: a scalable lightweight join query processor for RDF data. In *WWW*. ACM, 41–50.

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

[8] Murat Ali Bayir, Ismail H Toroslu, and Ahmet Cosar. 2006. Genetic algorithm for the multiple-query optimization problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 1 (2006), 147–153.

[9] Dritan Bleco and Yannis Kotidis. 2019. Using entropy metrics for pruning very large graph cubes. *Information Systems* 81 (2019), 49–62.

[10] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *PVLDB* 11, 2 (2017), 149–161.

[11] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. 2002. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *ISWC*. 54–68.

[12] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. 2020. Obi-Wan: Ontology-Based RDF Integration of Heterogeneous Data. *Proc. VLDB Endow.* 13, 12 (2020), 2933–2936. http://www.vldb.org/pvldb/vol13/p2933-buron.pdf

[13] Ashok K Chandra and Philip M Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *STOC*. ACM.

[14] Leonardo Weiss F Chaves, Erik Buchmann, Fabian Hueske, and Klemens Böhm. 2009. Towards materialized view selection for distributed databases. In *Proceedings of the 12th international conference on extending database technology: advances in database technology*. 1088–1099.

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

[16] Vicky Dritsou, Panos Constantopoulos, Antonios Deligiannakis, and Yannis Kotidis. 2011. Optimizing query shortcuts in RDF databases. *ESWC* (2011), 77–92.

[17] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. 2018. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web* 9, 1 (2018), 77–129.

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

[19] 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. 2010. Building Watson: An overview of the DeepQA project. *AI magazine* 31, 3 (2010), 59–79.

[20] Philippe Fournier-Viger and Chao Cheng. 2019. HUE-Span. http://www.philippe-fournier-viger.com/spmf/tkgtkg/ [Online; accessed 16-September-2021].

[21] Philippe Fournier-Viger, Chao Cheng, Jerry Chun-Wei Lin, Unil Yun, and R Uday Kiran. 2019. Tkg: Efficient mining of top-k frequent subgraphs. In *International Conference on Big Data Analytics*. Springer, 209–226.

[22] Philippe Fournier-Viger, Ganghuan He, Chao Cheng, Jiaxuan Li, Min Zhou, Jerry Chun-Wei Lin, and Unil Yun. 2020. A survey of pattern mining in dynamic graphs. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 6 (2020), e1372.

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

[24] Himanshu Gupta. 1997. Selection of views to materialize in a data warehouse. In *International Conference on Database Theory*. Springer, 98–112.

[25] Sairam Gurajada, Stephan Seufert, Iris Miliaraki, and Martin Theobald. 2014. TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing. In *SIGMOD*. ACM, 289–300.

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

[27] Jiewen Huang, Daniel J Abadi, and Kun Ren. 2011. Scalable SPARQL querying of large RDF graphs. *PVLDB* 4, 11 (2011), 1123–1134.

[28] Apache Jena. 2007. semantic web framework for Java.

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

[30] Alekh Jindal, Shi Qiao, Hiren Patel, Zhicheng Yin, Jieming Di, Malay Bag, Marc Friedman, Yifung Lin, Konstantinos Karanasos, and Sriram Rao. 2018. Computation reuse in analytics job service at microsoft. In *Proceedings of the 2018 International Conference on Management of Data*. 191–203.

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

[32] Evgeny Kharlamov, Yannis Kotidis, Theofilos Mailis, Christian Neuenstadt, Charalampos Nikolaou, Özgür Özçep, Christoforos Svingos, Dmitriy Zheleznyakov, Sebastian Brandt, Ian Horrocks, Yannis E. Ioannidis, Steffen Lamparter, and Ralf Möller. 2016. Towards analytics aware ontology based access to static and streaming data. In *ISWC*. 344–362.

[33] Evgeny Kharlamov, Yannis Kotidis, Theofilos Mailis, Christian Neuenstadt, Charalampos Nikolaou, Özgür Özçep, Christoforos Svingos, Dmitriy Zheleznyakov, Yannis Ioannnidis, Steffen Lamparter, Ralf Möller, and Arild Waaler. 2019. An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data. *J. Web Semant.* (2019).

[34] Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, Özgür L. Özçep, Mikhail Roshchin, Nina Solomakhina, Ahmet Soylu, Christoforos Svingos, Sebastian Brandt, Martin Giese, Yannis E. Ioannidis, Steffen Lamparter, Ralf Möller, Yannis Kotidis, and Arild Waaler. 2017. Semantic access to streaming and static data at Siemens. *J. Web Semant.* 44 (2017), 54–74.

[35] Yannis Kotidis and Nick Roussopoulos. 2001. A case for dynamic view management. *TODS* 26, 4 (2001), 388–423.

[36] Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. Towards creating a generalized complex event processing operator using FlinkCEP: architecture & benchmark. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*. 188–189.

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

[38] Theofilos Mailis, Yannis Kotidis, Vaggelis Nikolopoulos, Evgeny Kharlamov, Ian Horrocks, and Yannis E. Ioannidis. 2019. An Efficient Index for RDF Query Containment. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1499–1516.

[39] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In *ISWC*. 376–394.

[40] Imene Mami and Zohra Bellahsene. 2012. A survey of view selection methods. *Acm Sigmod Record* 41, 1 (2012), 20–29.

[41] Brian McBride. 2001. Jena: Implementing the rdf model and syntax specification. In *ISWC*. CEUR-WS. org, 23–28.

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

[43] Kostas Messanakis, Petros Demetrakopoulos, and Yannis Kotidis. 2021. SmartViews: Decentralized OLAP View Management Using Blockchains. In *Big Data Analytics and Knowledge Discovery (Lecture Notes in Computer Science)*, Vol. 12925. Springer, 216–221.

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

[45] Konstantinos Morfonios, Stratis Konakas, Yannis Ioannidis, and Nikolaos Kotsis. 2007. ROLAP implementations of the data cube. *ACM Computing Surveys (CSUR)* 39, 4 (2007), 12.

[46] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. 2015. RDFox: A highly-scalable RDF store. In *ISWC*. 3–20.

[47] Thomas Neumann and Gerhard Weikum. 2010. x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. *PVLDB* 3, 1-2 (2010), 256–263.

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

[49] Francois Picalausa and Stijn Vansummeren. 2011. What are real SPARQL queries like?. In *SWIM*. ACM, 7.

[50] Richard Qian. 2013. Understand Your World with Bing. https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing [Online; accessed 16-September-2021].

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

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

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

[54] Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, and Stefan Manegold. 2008. Column-store support for RDF data management: not all swans are white. *PVLDB* 1, 2 (2008), 1553–1563.
[55] Amit Singhal. 2012. Introducing the Knowledge Graph: Things, not Strings. https://blog.google/products/search/introducing-knowledge-graph-things-not/ [Online; accessed 16-September-2021].
[56] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. 697–706.
[57] Christoforos Svingos, Theofilos Mailis, Herald Kllapi, Lefteris Stamatogiannakis, Yannis Kotidis, and Yannis Ioannidis. 2016. Real time processing of streaming and static information. In *IEEE Big Data*. 410–415.
[58] Maxim Sviridenko. 2004. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters* 32, 1 (2004), 41–43.
[59] Dimitri Theodoratos, Timos Sellis, et al. 1997. Data warehouse configuration. In *VLDB*, Vol. 97. 126–135.
[60] Daniel Ting. 2019. Approximate distinct counts for billions of datasets. In *Proceedings of the 2019 International Conference on Management of Data*. 69–86.
[61] Jing Wang, Nikos Ntarmos, and Peter Triantafillou. 2016. Indexing Query Graphs to Speedup Graph Query Processing. In *EDBT*. 41–52.
[62] Jing Wang, Nikos Ntarmos, and Peter Triantafillou. 2017. GraphCache: A Caching System for Graph Queries. In *EDBT*. 13–24.
[63] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. 2008. Hexastore: sextuple indexing for semantic web data management. *PVLDB* 1, 1 (2008), 1008–1019.
[64] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-Based Substructure Pattern Mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 721–724.
[65] Xiaofei Zhang, Lei Chen, Yongxin Tong, and Min Wang. 2013. EAGRE: Towards scalable I/O efficient SPARQL query evaluation on the cloud. In *ICDE*.
[66] Jingren Zhou, Per-Ake Larson, Johann-Christoph Freytag, and Wolfgang Lehner. 2007. Efficient exploitation of similar subexpressions for query processing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. 533–544.