Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Optimizing Reformulated RDF Queries

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Damián Alexis Bursztyn

Directores: Cecilia Ana Ruz (*FCEyN, UBA*), Ioana Manolescu, François Goasdoué, Dario Colazzo (*Inria Saclay and Université Paris Sud*)

Buenos Aires, 2013

OPTIMIZACIÓN DE CONSULTAS RDF REFORMULADAS

El desarrollo de la Web Semántica y la creciente popularidad de su principal formato de datos, RDF, trae aparejada la necesidad de técnicas eficientes y escalables de gestión de datos para responder consultas RDF sobre grandes volúmenes de datos heterogéneos.

Una opción popular consiste en la traducción de consultas RDF en consultas SQL a ser ejecutadas en los maduros y eficientes sistemas de gestión de bases de datos relacionales tradicionales (RDBMS). Sin embargo, las bases de datos para Web Semántica plantean retos específicos a las tecnologías clásicas de gestión de datos debido a la presencia de datos implícitos, a los cuales los RDBMS no tienen en cuenta durante la evaluación de consultas. Para achicar la brecha entre las bases de datos de Web Semántica que contienen datos implícitos y la evaluación de consultas proporcionada por los RDBMS actuales, una opción es *reformular* la consulta entrante para luego traducirla en una consulta SQL que, al ser evaluada por el RDBMS, devuelve las respuestas completas .

Si bien este enfoque es conceptualmente suficiente para garantizar el eficaz procesamiento de las consultas de Web Semántica en un RDBMS, en la práctica aparecen problemas significativos de rendimiento debido a la longitud sintáctica de las consultas SQL que resultan de reformulación. Reconocidos y eficientes RDBMSs no son capaces de optimizar eficientemente estas consultas. En algunos casos los RDBMSs simplemente fallan al intentar responder estas consultas, mientras en otros casos se registran tiempos de evaluación muy elevados [1].

En este trabajo, hemos identificado y explotado dos grados de libertad que pueden ser aprovechados para realizar la evaluación de las consultas reformuladas de forma más eficiente. En primer lugar, se enumeran un espectro de consultas alternativas equivalentes, obtenidas mediante el agrupamiento de los átomos de la consulta original y la reformulación de *fragmentos (de átomos) de la consulta*; las reformulaciones de los fragmentos resultantes son envíadas en forma individual al RDBMS para su evaluación, para luego realizar la operación de join entre los resultados intermedios a fines de obtener la respuesta completa a la consulta original. En segundo lugar, SQL proporciona distintas alternativas sintácticas para expresar este tipo de consultas (que consiste en subconsultas cuyos resultados deben ser unidos); detectamos que esta elección sintáctica también permite mejoras en el rendimiento. Diseñamos un modelo de costos que refleja el impacto de las decisiones tomadas dentro de los dos grados de libertad descritos anteriormente. Basado en este modelo de costos, proponemos distintos algoritmos heurísticos que, dada una consulta inicial y las reglas semánticas que aplican en la base de datos (e implican los datos implícitos), realizan las elecciones necesarias en forma automática a fines de producir una consulta SQL cuya evaluación genera la respuesta completa a la consulta de forma eficiente.

Por último, presentamos una amplia gama de experimentos basados en un DBMS off-the-shelf, que dan soporte a los beneficios de los algoritmos propuestos.

Palabras claves: Procesamiento de consultas RDF, SPARQL, reformulación de consultas, gestión de datos semánticos en la Web, optimización de consultas, algoritmos heurísticos.

OPTIMIZING REFORMULATED RDF QUERIES

The development of the Semantic Web and the increasing popularity of its main data format RDF, efficient and scalable data management techniques are needed to handle RDF query answering on large volumes of heterogeneous data. A popular option consists thus of translating RDF queries into SQL queries to be handled by mature and efficient relational database management systems (RDBMSs). However, Semantic Web databases pose specific challenges to classical data management technologies through the presence of implicit data, which RDBMS query evaluation fails to account for. To bridge the gap between the Semantic Web databases containing implicit data and the simple query evaluation provided by RDBMSs, one can *reformulate* the incoming query into an SQL query which, when evaluated by the RDBMS, returns complete answers.

While this approach is conceptually sufficient to ensure efficient processing of Semantic Web queries within RDBMSs, in practice it raises significant performance problems due to the syntactic size of the SQL queries resulting from reformulation. It turns out that many efficient RDBMSs are unable to efficiently optimize such queries. In some cases RDBMS evaluation simply fails, while in other cases very high evaluation times are recorded [1].

In this work, we have identified and exploited two degrees of freedom that could be exploited to make the evaluation of reformulated queries more efficient. First, we enumerate a space of alternative equivalent queries, obtained by grouping atoms from the original query and reformulating *query fragments*; the resulting reformulated fragments are sent individually for evaluation to the RDBMS before joining their results to obtain the complete query answer. Second, SQL provides alternative syntaxes for expressing such queries (consisting of subqueries whose results must be joined); we found that this syntactic choice leads to performance improvements, too. We have devised a cost model capturing the impact of the choices made within the two freedom degrees described above. Based on this cost model, we propose several heuristic algorithms which, given an initial query and the semantic rules holding on the database (and entailing implicit data), automatically makes the necessary choices in order to produce an SQL query whose evaluation computes the query answers more efficiently.

Finally, we present an extensive experimental evaluation based on an off-the-shelf DBMS, which validates the benefits of our proposed algorithms.

Keywords: *RDF query answering, SPARQL, query reformulation, Semantic Web Data Management, query optimization, heuristic algorithms.*

ACKNOWLEDGMENTS

I would like to thanks to all those who helped me during both the Computer Science degree and the development of my thesis.

To mom, dad, Michelle, Gisela and Jaco, whom encouraged and helped me all these past years to close this chapter of my life.
To Nico and Laura, whom treated me like a brother and were there for everything I could need.
To my grandparents, aunts and uncles for their support.
To Rumi, whose smiles and joy always helped.
To Alan, for his advices and guidance.

To Gutes, Nacho and Fav, with whom I had the pleasure of sharing most of the courses. I wouldn't be here today without them.
To all those mates that shared courses with me and made them more enjoyable.
To all the teachers who guided and helped me during these years.

To my friends Esteban, Migue, Rafa, Fran, Maria and Diego for being always there when needed.
To Guada, who joined me and been there with a smile while I was doing my internship.

Thanks to Cecilia, who helped me to conclude this thesis.
Finally, special thanks to Ioana, François and Dario for sharing their enthusiasm with me and for their help and dedication during my Internship at Paris, which achieve in this work.

To my dad.

CONTENTS

1. INTRODUCTION

The "Web of Data" vision behind the initial World Wide Web project has found its most recent incarnation through the Semantic Web. More and more data sources are being exported or produced as *triples*, using the Resource Description Format (or RDF, in short) model standardized by the W3C [2]. To exploit this wealth of data, the SPARQL query language has been defined [3].

An RDF dataset consists of both explicitly declared data, and of data *implicitly* present into the database, due to semantic constraints that may hold on it. Such data is obtained by a process termed *entailment* [2], whereas constraints are used to infer from the existing base and the dataset constraints, all the possible consequences. A famous example includes a dataset holding explicitly the fact that *Socrates is a human*, and on which the constraint *Any human is mortal* has been stated to hold. Then, an implicit triple that entailment produces is: *Socrates is mortal.* This example illustrates one of the most natural classes of constraints (or entailment rules) that an RDF dataset may hold, while, as we discuss later on, many more such rules exist.

Query answering in the presence of implicit data must take into account entailment, in order to avoid returning incomplete answers [4]. Two query answering strategies have been devised so far. First, *saturation* consists of making all implicit data explicit; subsequently, query evaluation on the resultant database thus increased is sufficient to compute query answers. In our example, this would amount to adding *Socrates is mortal* to the database. A second strategy consists of *reformulating* the user queries based on the known constraints before evaluating it. Observe that this strategy is applied on a per-query basis, as opposed to saturation which can be applied once and for all on the database. In our example, assuming that a query asks for *all the mortals*, reformulation would turn it into: find *all the mortals as well as all the humans.* The second part of the query is added to reflect the constraint that any answer to the *human* query is also an answer to the *mortals* one. Observe that reformulation does not affect the database, but the query alone.

The trade-offs between saturation and reformulation are as follows. Saturation is straightforward and easy to implement but requires storage space, computation time and must be recomputed upon updates. Moreover, the recursive nature of entailment makes saturation costly (in time and space) and the method not suitable for datasets with a high rate of updates. Reformulation is robust upon updates and made at query runtime. However, reformulated RDF queries may end up being syntactically very large unions of a high number of conjunctive queries, while exhibiting numerous repeated sub-expressions. This makes query evaluation very inefficient.

The increasing importance of Web data and in particular RDF raised interest within the data management scientific community. Many techniques and algorithms have been proposed in recent years for the processing of SPARQL queries, based on vertical partitioning [5], indexing [6], efficient join processing [7], query selectivity estimation (join order optimization of triple patterns performed before query evaluation) [8], SPARQL Multi-Query Optimization (or MQO, in short)-exploiting the possibility of reusing (sharing) results of common subexpressions [9], RDF management systems and optimized triple stores [10, 11, 12, 13], to name a few. However, these approaches does not take entailment into account and as such, they return incomplete result in the presence of implicit data.

Recent work [4] introduced novel BGP query answering techniques for the DB fragment of RDF, designed to work on top of any relational database management system (RDBMS, in short), and studied saturation- and reformulation-based query answering for this fragment. In particular, a novel reformulation algorithm is presented, but the performance of the SQL queries resulting from the invocation of this algorithm still shows the need for optimizations.

A careful analysis of the experimental evaluation performed in [4] highlighted two opportunities to improve reformulated query performance:

1. Given that a reformulated query is a union of many conjunctive queries sharing some common sub-expressions, one could imagine alternative forms of expressing the reformulated query, by *pushing some of the union operators below some joins.* Clearly, there is a large search space for such equivalent ways to state a given reformulated query.

2. Further, the expressive power of the SQL language enables many syntactically different ways to express each query statement thus obtained, and it turns out that the choice among such SQL syntax options does have an impact on the performance of its evaluation, even in recent well-optimized database systems. Thus, a *reformulation optimization* approach is needed, based on *performance (cost)* considerations.

In this thesis, we introduce an effective cost model to compare candidate execution plans and present techniques for optimizing RDF reformulated queries for answering *Basic Graph Pattern (or BGP, in short)* queries within the *database (DB) fragment of RDF.* In particular, we devise naive and efficient heuristic algorithms for BGP queries nodes clusterization (with and without clusters overlapping) w.r.t. a given threshold over the number of reformulations. We present also a naïve technique for clustered query execution (against a RDBMS engine) and introduce optimization opportunities.

Our techniques are designed to be deployed on top of any RDBMS(-style) engine. Further, the parametrization of the threshold in our clusterization algorithms makes them easy to adapt, as the threshold may vary within different RDBMS engines.

To make this thesis self-contained, we provide in Chapter 2 all the necessary background in order to understand the problem we consider; in particular, we introduce RDF data and queries, as well as the fundamental notions behind the optimization of the reformulated queries which is the main topic of this work.

Contributions Our contributions can be summarized as follows.

1. We propose a novel and practical cost model for BGP queries taking into account both, the number of results and the number of reformulations.

2. We introduce algorithms to find a clusterization that accelerates the execution for a given BGP query.

3. We implemented the above algorithms and deployed them on top of PostgreSQL [14]. Extensive experiments with large RDF data performed on different RDF stores confirm the efficiency and effectiveness of our approach over the baseline techniques, presented in previous work [1].

4. We introduce techniques for improving the execution of clustered queries.

The thesis is organized as follows. Chapter 2 introduces RDF graphs and entailment, BGP queries, SPARQL and query reformulation. Chapter 3 defines our cost model, through whom Section 3.5 introduce BGP query clusterization algorithms. Clustered BGP query execution techniques are presented in Section 3.6. These algorithms and techniques are then experimentally compared and studied in Chapters 4 and 5. We discuss related work in Chapter 6, then we conclude.

2. PRELIMINARIES

2.1 Semantic Web

I have a dream for the Web [in which computers]
become capable of analyzing all the data on the Web. [15]
–Tim Berners-Lee

The “Web of Data” vision behind the initial World Wide Web project has found its most recent incarnation through the Semantic Web. Unlike traditional knowledge-representation systems, typically centralized [16], the Semantic Web is meant to be a world-wide distributed architecture, where data and services easily interoperate, by publishing semantic descriptions of Web resources.

The Semantic Web is an extension of the current World Wide Web, in which the content is given structure, thus enabling a reliable way for the computers to process the semantics and therefore manipulate the data in a meaningful way. This vision, however, is not yet a reality in the current Web [17, 18], in which most content (currently dominated by unstructured and semi-structured documents) is intended for humans to read [16]. As stated in [18], given a particular need, it is difficult to find a resource that is appropriate to it. Also, given a relevant resource, it is not easy to understand what it provides and how to use it.

A key idea to solve such limitations, enabling the Semantic Web vision, is to also publish semantic descriptions of Web resources. These descriptions rely on logical assertions that are used to relate resources to terms in predefined ontologies [18]. Therefore, a fundamental ingredient for the Semantic Web vision is the set of data models and formats for describing items from both the physical and digital world, in a machine exploitable way, providing semantics to applications that use them and facilitating interoperability.

The Semantic Web stack builds on the Resource Description Framework (or RDF, in short) model standardized by the W3C [2]. The RDF Schema (or RDFS, in short) [19] and the Web Ontology Language (or OWL, in short) [20] were proposed to enhance the descriptive power of RDF data sets. More specifically, RDF provides a format to make statements about resources (in particular Web resources) in the form of subject-predicate-object expressions, while RDF Schema (which is a superset of RDF) and OWL (which in turn is a superset of RDFS) provide ontological data models, to state semantic constraints between the classes and the properties used.

The term ontology has been used in different disciplines multifariously. Each community co-opted the term for their own jargon. We use the term to refer to a formal description that provides a shared understanding of a given domain, based on:

- a set of *individuals* (entities or objects),
- *classes* of individuals, and
- the *relationships* that exists between these individuals.

The logical statements on memberships of individuals and in classes or relationships between individuals, form a base of facts, i.e., *knowledge base.*

In a nutshell, current popularity and usage of ontologies in the Web is due to four major reasons:

- Their flexible and natural way of structuring documents in multidimensional ways, allowing to find relevant information through very large documents collections.
- The logical formal semantics of ontologies provide means of inference, enabling reasoning. Therefore, it is possible for an ontology to be interpreted and processed by machines.
- Ontologies allow making concepts precise, improving Web search. For example, when searching for the word “jaguar”, we could specialize the concept in an ontology, *car:jaguar*, avoiding unwanted answers where the term is used with a different meaning (like those referring to the animal with the same name); searching for the concept *country:USA*, instead of the word “USA”, allow us to get also documents in which synonyms (or translations), like “United States”, “United States of America” or “Estados Unidos”, are used.
- Ontologies serve as local join between heterogeneous information sources. Moreover, their inference potential helps to automatically integrate different data sources.

Despite the critics that have questioned its feasibility, the need for data integration grew. The need for shared semantics and a “Web of Data” and information has been increasing, with e-science being its major driver [17]. The amount of data being exported or produced as *triples* using RDF has been expanding due to increased uptake of its principles in scientific research, industry, government and other communities [21]. Today, we see RDF datasets containing billions of triples (3-tuples). For example, the UniProt collection [22] is a huge biological dataset which aims to provide all the UniProt protein sequence and annotation data in RDF format, combining rich annotations from various other collections, with a total of 845 million triples [7]. Further, the Semantic-Web community has addressed a Billion Triples Challenge [23], using a heterogeneous collection that includes DBpedia [24] and Freebase [25], with a total of more than 1.1 billion triples (taking around 88 GB in N-Triples format) [7].

To exploit this wealth of data, the SPARQL query language has been defined [3]; subsequently, novel techniques and algorithms have been proposed for the processing of SPARQL queries, based on vertical partitioning [5], indexing [6], efficient join processing [7], view selection [26], RDF management systems and optimized triple stores [10, 11, 12, 13], to name a few.

2.2 Close World Assumption vs. Open World Assumption

Your assumptions are your windows on the world.
Scrub them off every once in a while, or the light won't come in
–Isaac Asimov

Traditionally, database constraints can be interpreted in two ways [27]: under the Closed World Assumption (CWA) or under the Open World Assumption (OWA). The Close World Assumption states that any fact that is not present in the database is assumed not to hold. Using this approach, if the set of database facts does not respect a constraint, then the database is *inconsistent.* For example, the CWA interpretation of a constraint of the form $R_1 \subseteq R_2$ is: any tuple in the relation R_1 *must* also be in the relation R_2 *in the database*, otherwise the database is inconsistent. On the contrary, under the Open World Assumption, some facts may hold even though they are *not in the database.* For instance, the OWA interpretation of the same example is: any tuple t in the relation R_1 *is considered as also being in the relation* R_2 (t *is propagated to* R_2 because of the inclusion constraint) [1].

2.3 RDF and RDF Schema

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [16]
–Tim Berners-Lee

RDF provides a simple language allowing to express facts about Web resources. A fact describes some metadata of a resource, that is identified by a URI, as a *triplet* of the form `s p o`. A triple states that the value of the *property* `p` for the *subject* `s` is the *object* `o`. In a triplet, the subject and the property are URIs pointing to Web resources, whereas the object may be either a URI or a literal representing a value.

Blank nodes are an essential feature of RDF, allowing to support unknown constants or URIs. A blank node (a.k.a. anonymous resource) is a subject or object in an RDF triplet that is not identified by a URI and is not a literal. Its referred by the notation $\_:b$, where b is a local name that can be used in several triples for staging several properties of the corresponding blank node [18]. An RDF graph may contain several blank nodes, since many of them can co-exist within a graph. For example, we can use a blank node $\_:b_0$ to state that $\_:b_0$ is a professor at the *ComputerScienceDept*, and teaches *Databases*, while the number of students enrolled in the *Databases* class is an unspecified value $\_:b_1$.

The database (DB) fragment of RDF Introduced in [1], the DB fragment extends previously studied fragments of RDF [26, 28, 29] by adding support for blank nodes. Moreover, the authors analyzed and compared the two established techniques for handling RDF entailment, namely *saturation* and *reformulation.*

Constructor	Triple	Relational notation
Class assertion	s τ o	o(s)
Property assertion	s p o	p(s, o)

Fig. 2.1: RDF statements.

Given a set of URIs U, a set of literals (constants) L, and a set of blank nodes (unknown URIs or literals) B, such that U, B and L are pairwise disjoint, we say a triple is *well-formed* when it satisfies (all) the following statements:

- its subject belongs to $U \cup B$;
- its property belongs to U;

- its object belongs to $U \cup B \cup L$.

Notationally, we use `s`, `p`, `o` and $\_{:}b$ in triples (possibly with subscripts) as placeholders. That is, `s` stands for values in $U \cup B$, `p` stands for values in U, `o` represents values from $U \cup B \cup L$, and $\_{:}b$ denotes values in B. Finally, we use strings between quotes as in "*string*" to denote literals. The built-in property `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`, denoted τ from now on, is used to specify to which *classes* a resource belongs (a form of resource typing). Figure 2.1 shows how to use triples to describe resources.

Example 1 (Movies). *The fact that Stanley Kubrick has directed Jack Nicholson in the movie The Shining can be represented by the following* triples*:*

$$\begin{array}{l}\langle doi_0\ hasName\ \text{“}Stanley\ Kubrick\text{''}\rangle,\\ \langle doi_0\ hasDirected\ doi_1\rangle,\\ \langle doi_1\ \tau\ Movie\rangle,\\ \langle doi_1\ hasName\ \text{“}The\ Shining\text{''}\rangle,\\ \langle doi_2\ hasStarred\ doi_1\rangle,\\ \langle doi_2\ hasName\ \text{“}Jack\ Nicholson\text{''}\rangle\end{array}$$

An alternative, and sometimes more intuitive, way to visualize and represent all the triples information is using a graph, where there is a node for each (distinct) subject or object, labeled with its value; a triplet is represented as a directed edge, labeled with the property value, between the subject node and the object node.

$$\begin{array}{l}\langle doi_0\ \tau\ Book\rangle,\\ \langle doi_0\ hasTitle\ \text{“}Around\ the\ World\ in\ Eighty\ Days\text{''}\rangle,\\ \langle doi_0\ hasGender\ Novel\rangle,\\ \langle doi_0\ publishedIn\ 1873\rangle,\\ \langle \_{:}b_0\ hasWritten\ doi_0\rangle,\\ \langle \_{:}b_0\ hasName\ \text{“}Jules\ Verne\text{''}\rangle,\\ \langle \_{:}b_0\ hasNationality\ \text{“}French\text{''}\rangle\end{array}$$

Subject	**Property**	**Object**
doi_0	τ	*Book*
doi_0	*hasTitle*	"*Around the World in Eighty Days*"
doi_0	*hasGender*	*Novel*
doi_0	*publishedIn*	1873
$\_{:}b_0$	*hasWritten*	doi_0
$\_{:}b_0$	*hasName*	"*Jules Verne*"
$\_{:}b_0$	*hasNationality*	"*French*"

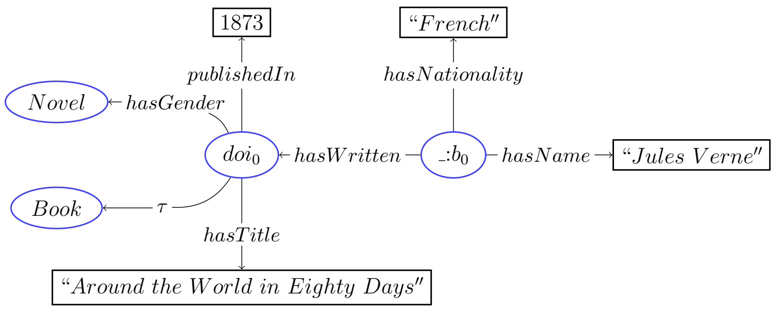


Fig. 2.2: Alternative RDF representations

Example 2 (Alternative RDF representations). *The Figure 2.2 presents three equivalent representations. The graph characterizes a resource* doi_0 *that belongs to the class Book, whose title is "Around the World in Eighty Days", was published in the year* 1873*, and belongs to the gender Novel. The author of the book is the unknown resource* $\_{:}b_0$*, whose name is "Jules Verne", while the nationality of* $\_{:}b_0$ *is "French".*

Constructor	Triple	Relational notation
Subclass constraint	$\mathtt{s} \prec_{sc} \mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Subproperty constraint	$\mathtt{s} \prec_{sp} \mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Domain typing constraint	$\mathtt{s} \hookleftarrow_{d} \mathtt{o}$	$\Pi_{\text{domain}}(\mathtt{s}) \subseteq \mathtt{o}$
Range typing constraint	$\mathtt{s} \hookrightarrow_{r} \mathtt{o}$	$\Pi_{\text{range}}(\mathtt{s}) \subseteq \mathtt{o}$

Fig. 2.3: RDFS statements

RDFS is an extension of RDF that enhances the descriptive power of an RDF dataset trough the declaration of facts (constraints) in particular domains. An RDF Schema specifies constraints on the individuals (classes) and the relations (properties) used in the RDF triplets, enabling entailment (one of the most valuable features of the Semantic Web). For instance, given a RDF dataset containing the fact that "$Jules\ Verne''$ $hasWritten$ the $Book$ "$Around\ the\ World\ in\ Eighty\ Days''$, whereas an RDFS states that those who wrote books are $writers$; the fact "$Jules\ Verne''$ is a $writer$ is implicitly present in the dataset. Figure 2.3 shows the allowed constraints and how to express them. In this figure, as we can see, $\mathtt{s}, \mathtt{o} \in U \cup B$, while domain and range denote respectively the first and second attribute of every property.

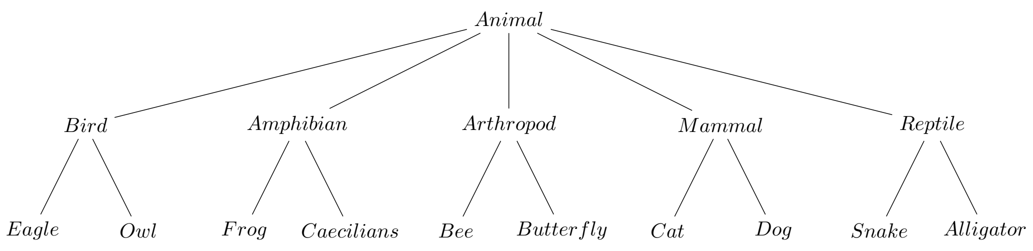


Fig. 2.4: Animal class hierarchy.

Example 3 (Continued). *For instance, the animal hierarchy characterized in Figure 2.4 and the properties hasSkeleton, hasEndoSkeleton, hasGestationTime and hasFelineLeukemia are represented by the RDF Schema shown in Figure 2.5.*

$$
\begin{array}{ll}
\langle Amphibian \prec_{sc} Animal\rangle, & \langle Butterfly \prec_{sc} Arthropod\rangle, \\
\langle Bird \prec_{sc} Animal\rangle, & \langle Cat \prec_{sc} Mammal\rangle, \\
\langle Arthropod \prec_{sc} Animal\rangle, & \langle Dog \prec_{sc} Mammal\rangle, \\
\langle Mammal \prec_{sc} Animal\rangle, & \langle Snake \prec_{sc} Reptile\rangle, \\
\langle Reptile \prec_{sc} Animal\rangle, & \langle Alligator \prec_{sc} Reptile\rangle, \\
\langle Frog \prec_{sc} Amphibian\rangle, & \langle hasSkeleton \hookleftarrow_{d} Animal\rangle, \\
\langle Caecilians \prec_{sc} Amphibian\rangle, & \langle hasEndoSkeleton \prec_{sp} hasSkeleton\rangle, \\
\langle Eagle \prec_{sc} Bird\rangle, & \langle hasGestationTime \hookleftarrow_{d} Mammal\rangle, \\
\langle Owl \prec_{sc} Bird\rangle, & \langle hasFelineLeukemia \hookleftarrow_{d} Cat\rangle \\
\langle Bee \prec_{sc} Arthropod\rangle, &
\end{array}
$$

Fig. 2.5: Animal RDF Schema.

One consistent difference between the Semantic Web and many data models for programming languages is the CWA [30]. The RDF data model uses the OWA, therefore constraints, such as the ones in Figure 2.3 are interpreted under the OWA instead of the CWA. This may lead to added information [27], a key feature of RDF. Implicit triples holds as part of the dataset, although they are not explicitly present in it.

Example 4 (Movies continued). *Consider an extension, using RDFS constraints, of the triples set used in Example 1:*

Data (facts)	***RDF Schema (constraints)***
$\langle doi_0\ hasName\ “Stanley\ Kubrick”\rangle$,	$\langle hasName\ \hookleftarrow_d\ Person\rangle$,
$\langle doi_0\ hasDirected\ doi_1\rangle$,	$\langle hasName\ \hookrightarrow_r$ rdfs:Literal$\rangle$,
$\langle doi_1\ \tau\ Movie\rangle$,	$\langle hasDirected\ \hookleftarrow_d\ Director\rangle$,
$\langle doi_1\ hasName\ “The\ Shining”\rangle$,	$\langle hasDirected\ \hookrightarrow_r\ Movie\rangle$,
$\langle doi_2\ hasStarred\ doi_1\rangle$,	$\langle hasStarred\ \hookleftarrow_d\ Actor\rangle$,
$\langle doi_2\ hasName\ “Jack\ Nicholson”\rangle$,	$\langle hasStarred\ \hookrightarrow_r\ Movie\rangle$,
	$\langle Actor\ \prec_{sc}\ Person\rangle$,
	$\langle Director\ \prec_{sc}\ Person\rangle$

The triples $\langle doi_2\ \tau\ Actor\rangle$ and $\langle doi_0\ \tau\ Director\rangle$ holds in the dataset because the pairs of triples $\langle doi_2\ hasStarred\ doi_1\rangle$, $\langle hasStarred\ \hookleftarrow_d\ Actor\rangle$ and $\langle doi_0\ hasDirected\ doi_1\rangle$, $\langle hasDirected\ \hookleftarrow_d\ Director\rangle$ respectively, are present in the dataset. Moreover, the triples $\langle doi_2\ \tau\ Actor\rangle$ and $\langle doi_0\ \tau\ Director\rangle$ also holds in the dataset.

2.4 RDF entailment

Vision is the art of seeing what is invisible to others.
–Jonathan Swift

The W3C [31] refers as entailment to the RDF feature that allows modeling *implicit data*. Implicit RDF triples (triples that are considered to hold beyond that are not explicitly present in the dataset) are derived based on the explicit set of RDF triples in the dataset by applying the entailment rules. The process of applying an entailment rule, to a given dataset, is known as *immediate entailment* and its denoted as $\vdash^i_{\text{RDF}}$ [1]. A triple `s p o` is said to be entailed by a dataset D, denoted $D \vdash_{\text{RDF}}$ `s p o`, if and only if there is a sequence of immediate entailments that leads from D to `s p o`. Please note that implicit triples entailed in previous steps are also taken into account during the single application of an entailment rule.

Entailment rules are grouped by type. A first kind, known as simple entailment rules, use blank nodes to produce generalizations. For example, using the dataset shown in Example 4:

$$doi_0\ hasDirected\ doi_1 \vdash^i_{\text{RDF}} doi_0\ hasDirected\ \_{:}b$$

A second group of rules use a particular case of rule named se_1 [31] that applies only to literals, producing generalizations using blank nodes in the presence of literals. For instance,

$$doi_0\ hasName\ “Stanley\ Kubrick” \vdash^i_{\text{RDF}} doi_0\ hasName\ \_{:}b$$

The third kind of rules, known as RDF entailment rules, generates triples that type properties and, similarly to the second group, produces generalizations using blank nodes but in the presence of XML Literals:

$$doi_2\ hasStarred\ doi_1 \vdash^i_{\text{RDF}} hasStarred\ \tau\ rdf{:}Property$$

Another group of rules, named RDFS rules, derives entailed triples from the constraints in the schema and the semantics of built-in classes and properties, for instance:

$$\mathtt{p}_1 \prec_{sp} \mathtt{p}_2, \mathtt{p}_2 \prec_{sp} \mathtt{p}_3 \vdash^i_{\text{RDF}} \mathtt{p}_1 \prec_{sp} \mathtt{p}_3$$

Finally, the last kind of entailment rules, the extensional ones, produces generalizations that are described in the semantics of RDF but not covered by the RDFS rules. For instance,

$$\mathit{hasStarred} \hookleftarrow_d \mathit{Actor}, \mathit{Actor} \prec_{sc} \mathit{Person} \vdash^i_{\text{RDF}} \mathit{hasStarred} \hookleftarrow_d \mathit{Person}$$

Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
$\mathtt{s}\ \tau\ \mathtt{o}$	$\mathtt{o} \prec_{sc} \mathtt{o}$
$\mathtt{s\ p\ o}$	$\mathtt{p} \prec_{sp} \mathtt{p}$

Fig. 2.6: Schema triple entailment rules from a single instance-level triple.

Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
$\mathtt{s}_1 \prec_{sc} \mathtt{s}_2$	$\mathtt{s}_1 \prec_{sc} \mathtt{s}_1$
$\mathtt{s}_1 \prec_{sc} \mathtt{s}_2$	$\mathtt{s}_2 \prec_{sc} \mathtt{s}_2$
$\mathtt{p}_1 \prec_{sp} \mathtt{p}_2$	$\mathtt{p}_1 \prec_{sp} \mathtt{p}_1$
$\mathtt{p}_1 \prec_{sp} \mathtt{p}_2$	$\mathtt{p}_2 \prec_{sp} \mathtt{p}_2$
$\mathtt{p} \hookleftarrow_d \mathtt{s}$	$\mathtt{p} \prec_{sp} \mathtt{p}$
$\mathtt{p} \hookleftarrow_d \mathtt{s}$	$\mathtt{s} \prec_{sc} \mathtt{s}$
$\mathtt{p} \hookleftarrow_d$ rdfs:Literal	$\mathtt{p} \prec_{sp} \mathtt{p}$
$\mathtt{p} \hookrightarrow_r \mathtt{s}$	$\mathtt{p} \prec_{sp} \mathtt{p}$
$\mathtt{p} \hookrightarrow_r \mathtt{s}$	$\mathtt{s} \prec_{sc} \mathtt{s}$
$\mathtt{p} \hookrightarrow_r$ rdfs:Literal	$\mathtt{p} \prec_{sp} \mathtt{p}$

Fig. 2.7: Schema triple entailment rules from a single schema-level triple.

Rule name [31]	Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
$rdfs_2$	$\mathtt{p} \hookleftarrow_d \mathtt{s}, \mathtt{s}_1\ \mathtt{p}\ \mathtt{o}_1$	$\mathtt{s}_1\ \tau\ \mathtt{s}$
$rdfs_3$	$\mathtt{p} \hookrightarrow_r \mathtt{s}, \mathtt{s}_1\ \mathtt{p}\ \mathtt{o}_1$	$\mathtt{o}_1\ \tau\ \mathtt{s}$
$rdfs_7$	$\mathtt{p}_1 \prec_{sp} \mathtt{p}_2, \mathtt{s}\ \mathtt{p}_1\ \mathtt{o}$	$\mathtt{s}\ \mathtt{p}_2\ \mathtt{o}$
$rdfs_9$	$\mathtt{s}_1 \prec_{sc} \mathtt{s}_2, \mathtt{s}\ \tau\ \mathtt{s}_1$	$\mathtt{s}\ \tau\ \mathtt{s}_2$

Fig. 2.8: Instance triple entailment rules from instance and schema triples combined.

Rule name [31]	Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
$rdfs_5$	$\mathtt{p} \prec_{sp} \mathtt{p}_1, \mathtt{p}_1 \prec_{sp} \mathtt{p}_2$	$\mathtt{p} \prec_{sp} \mathtt{p}_2$
$rdfs_{11}$	$\mathtt{s} \prec_{sc} \mathtt{s}_1, \mathtt{s}_1 \prec_{sc} \mathtt{s}_2$	$\mathtt{s} \prec_{sc} \mathtt{s}_2$
ext_1	$\mathtt{p} \hookleftarrow_d \mathtt{s}_1, \mathtt{s}_1 \prec_{sc} \mathtt{s}$	$\mathtt{p} \hookleftarrow_d \mathtt{s}$
ext_2	$\mathtt{p} \hookrightarrow_r \mathtt{s}_1, \mathtt{s}_1 \prec_{sc} \mathtt{s}$	$\mathtt{p} \hookrightarrow_r \mathtt{s}$
ext_3	$\mathtt{p} \prec_{sp} \mathtt{p}_1, \mathtt{p}_1 \hookleftarrow_d \mathtt{s}$	$\mathtt{p} \hookleftarrow_d \mathtt{s}$
ext_4	$\mathtt{p} \prec_{sp} \mathtt{p}_1, \mathtt{p}_1 \hookrightarrow_r \mathtt{s}$	$\mathtt{p} \hookrightarrow_r \mathtt{s}$

Fig. 2.9: Schema triple entailment rules from two schema-level triples.

The first three sets of rules are not of a great interest for us since we consider it is more interesting to know that doi_0 *hasAbbreviation* "DL'' than to know that some unknown resource _:*b* has the abbreviation "DL". Similarly, the fact that *Airline* $\prec_{sc}$ *Airline* is not very relevant for us. In this work we focus on a useful subset of the entailment rules

specified by the W3C [31]. In particular, our approach consider the subset of instance-level entailment rules shown in Figure 2.8, and the schema-level entailment rules characterized in Figure 2.9. The set of rules described in Figure 2.8, produces instance RDF triples when applied to a instance triple and a schema triple. The group of entailment rules in Figure 2.9 generates schema triples (constraints) instead, using two schema triples as input.
The complete set of entailment rules specified by the W3C [31] can be found in Appendix B, Section 6. Moreover, Section 6 contains the RDF and RDFS axiomatic triples.

Dataset closure The *closure* of a dataset D, denoted D^{∞}, is unique (up to blank node renaming) and contains explicitly all the (implicit) triples that can be entailed from the original dataset D.

Given a dataset D, its closure is defined as the fixed point function:

- $D^0 = D$
- $D^{\alpha} = D^{\alpha-1} \cup \{\texttt{s p o} \mid D^{\alpha-1} \vdash^{i}_{\text{RDF}} \texttt{s p o}\}$

Moreover, a triple $\texttt{s p o} \in D^{\infty} \iff (\texttt{s p o} \in D) \vee (D \vdash_{\text{RDF}} \texttt{s p o})$; any triple entailed by a dataset D will be entailed by its closure and vice versa: $D \vdash_{\text{RDF}} \texttt{s p o} \iff D^{\infty} \vdash_{\text{RDF}} \texttt{s p o}$. As entailment is specified within the RDF W3C standard [2], D and D^{∞} are equivalents, given that the semantic of a dataset is its closure.

Example 5 (**Dataset closure**). *For instance, if we call D the dataset composed by the Schema in Figure 2.5 and the data shown in Figure 2.10, its closure is defined as:*

$$\begin{aligned}
D^0 = \;& D \\
D^1 = \;& D^0 \cup \{doi_1\ \tau\ Cat, doi_0\ \tau\ Mammal, doi_0\ hasSkeleton\ True, doi_0\ \tau\ Animal, Frog\ \prec_{sc}\ Animal, Caecilians\ \prec_{sc}\ Animal, \\
& Eagle\ \prec_{sc}\ Animal, Owl\ \prec_{sc}\ Animal, Bee\ \prec_{sc}\ Animal, Butterfly\ \prec_{sc}\ Animal, Cat\ \prec_{sc}\ Animal, Dog\ \prec_{sc}\ Animal, \\
& Snake\ \prec_{sc}\ Animal, Alligator\ \prec_{sc}\ Animal, hasEndoSkeleton\ \hookleftarrow_d\ Animal, hasGestationTime\ \hookleftarrow_d\ Animal, \\
& hasFelineLeukemia\ \hookleftarrow_d\ Mammal\} \\
D^2 = \;& D^1 \cup \{hasFelineLeukemia\ \hookleftarrow_d\ Animal, doi_1\ \tau\ Mammal, doi_1\ \tau\ Animal\} \\
D^3 = \;& D^2 \therefore D^{\infty} = D^2
\end{aligned}$$

$$\begin{array}{l}
\langle doi_0\ \tau\ Cat \rangle, \\
\langle doi_0\ hasEndoSkeleton\ True \rangle, \\
\langle doi_0\ hasGestationTime\ 63 - 65 days \rangle, \\
\langle doi_1\ hasFelineLeukemia\ False \rangle,
\end{array}$$

Fig. 2.10: RDF dataset for Example 5

2.5 BGP queries and SPARQL

Getting information off the Internet is like taking a drink from a fire hydrant.
–Mitchell Kapor

Basic graph pattern (or BGP, in short) queries are a well-known subset of the W3C SPARQL [3] query language for RDF, that corresponds to the familiar class of conjunctive queries from relational databases. Subject of several works recently [1, 8, 26, 32], BGP queries (a.k.a. SPARQL join queries) are the most used SPARQL queries in real-world [32].

BGP queries are composed by a set of *distinguished variables*, the head, and a set of *triple patterns*, the BGP. Each triple pattern has a *subject*, a *property*, and a *object*. Subjects and properties can be URIs, blank nodes or variables, whereas objects can also be literals. BGP queries can be characterized as Boolean (those with empty head) or non-Boolean queries, and their expressions are of the form `ASK WHERE` G, and `SELECT` $\bar{x}$ `WHERE` G respectively, where G is a BGP and $\bar{x}$ is a subset of the variables occurring in G.

Example 6 (BGP queries). *As an example, the following Boolean BGP query returns a positive answer (i.e.,* true*) if there exists, in the dataset, a director that also starred the movie,* false *otherwise.*

$$\begin{array}{lll} q_0() & \text{:- } x\ hasDirected\ y, & (0) \\ & \phantom{\text{:- }} x\ hasStarred\ y & (1) \end{array}$$

Instead, the following non-Boolean query retrieves the actors that worked in a movie directed by a French.

$$\begin{array}{lll} q_1(x) & \text{:- } x\ hasStarred\ y, & (0) \\ & \phantom{\text{:- }} w\ hasDirected\ y, & (1) \\ & \phantom{\text{:- }} w\ hasNationality\ \text{“}French''& (2) \end{array}$$

Finally, q_2 *retrieves the pairs of French actors and American directors that worked together.*

$$\begin{array}{lll} q_2(x, w) & \text{:- } x\ hasStarred\ y, & (0) \\ & \phantom{\text{:- }} w\ hasDirected\ y, & (1) \\ & \phantom{\text{:- }} x\ hasNationality\ \text{“}French'', & (2) \\ & \phantom{\text{:- }} w\ hasNationality\ \text{“}American'' & (3) \end{array}$$

We can also characterize BGP queries as a directed graph, where subjects and objects of the triples are represented by the vertices, labeled with their values (i.e., URIs, literals and variables), whereas the predicates are represented by the edges, labeled with their values. Figure 2.11 shows the illustrated representation of the queries in Example 6, whereas Figure 2.12 illustrates well-known basic query patterns as the ones characterized in recent works [9].

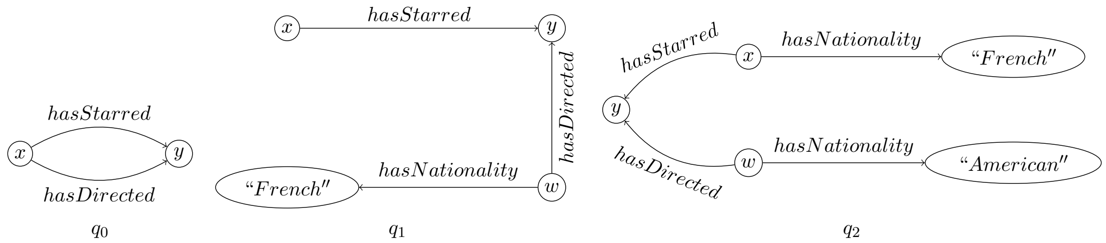


Fig. 2.11: Graphical representation of BGP queries.

Alternatively, BGP queries can be represented as a non-directed graph that illustrate the joins between triple patterns. Vertices represent the set triple patterns in the BGP, and are labeled with their identifiers; there's an edge between two vertices if there's at least one variable in common among them. Moreover, the edges could be labeled with the set of shared variables. Figure 2.13 exhibits an illustrated representation of the triple patterns and the joins between them for the queries in Example 6.

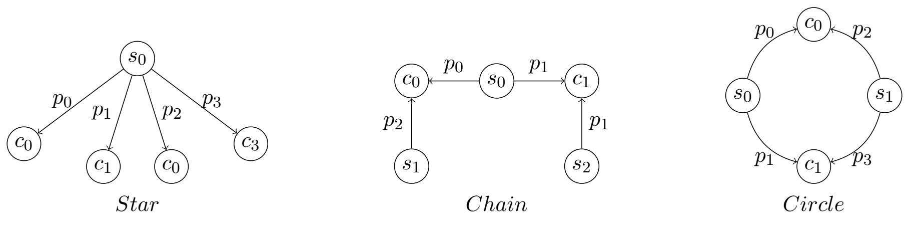


Fig. 2.12: Basic query patterns.

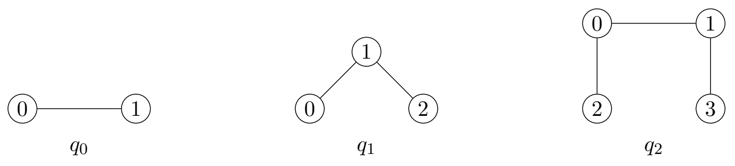


Fig. 2.13: Illustration of queries triple patterns and joins among them.

Summarizing, querying RDF bears some similarity to querying relational data, however, there are quite a few discrepancies in the data model. First, an RDF dataset is comprised of a unique, usually very large (surpassing in some case the billion triples [22, 23]), set of triples, contrasting with traditional relational databases featuring tens or even hundreds of relations with varying numbers of attributes. Moreover, classic relational databases provides the means to specify particular data structures trough the data definition language (or DDL, in short).

Airlines

Airline Id	*Name*	*Abbreviation*
1	*“Delta Air Lines″*	*“DL″*
2	*“Aerolineas Argentinas″*	*“AR″*
...	...	...
3	*“Air France″*	*“AF″*

Airports

Airport Id	*Name*	*Code*	*City*
1	*“Ministro Pistarini International Airport″*	*“EZE″*	*“Buenos Aires″*
2	*“Charles de Gaulle Airport″*	*“CDG″*	*“Paris″*
3	*“John F. Kennedy International Airport′*	*“JFK″*	*“New York″*
4	*“San Paulo – Guarulhos International Airport″*	*“GRU″*	*“San Pablo″*
...	...	...	...
5	*“Madrid – Barajas Airport″*	*“MAD″*	*“Madrid′*

Flights

Flight Id	*Airline Id*	*Flight Number*	*Orig. Airport Id*	*Dest. Airport Id*	*Departure Time*	*Est. Arrival Time*
1	3	*“AF* 394″	2	1	14/06/2013, 23.20	15/06/2013, 08.05
2	3	*“AF* 1400″	2	5	14/06/2013, 20.00	14/06/2013, 22.05
3	2	*“AR* 1132″	1	5	17/06/2013, 23.55	18/06/2013, 17.10
...	...	...	...	...	...	...
4	1	*“DL* 1164″	3	2	17/06/2013, 14.55	18/06/2013, 06.40

Fig. 2.14: The flights relational database

Example 7 (Flights running example). *For instance, suppose we are about to model flight schedules, including airplane company, origin and destination airport, departure time and estimated arrival time. In traditional relational databases, the database schema **FLIGHTS** for the database shown in Figure 2.14 is defined by:*

$$\boldsymbol{FLIGHTS} = \{Airlines, Airports, Flights\}$$

where the relations Airlines, Airports and Flights are composed by the following attributes:

$$\begin{aligned}
Airlines = \quad & \{Airline\ Id, Name, Abbreviation\}\\
Airports = \quad & \{Airport\ Id, Name, Code, City\}\\
Flights = \quad & \{Flight\ Id, Airline\ Id, Flight\ Number, Origin\ Airport\ Id,\\
& Destination\ Airport\ Id, Departure\ Time, Estimated\ Arrival\ Time\}
\end{aligned}$$

Instead, the RDF dataset, shown in Figure 2.15, will be composed by a single large set of triples (including information regarding l airports, k airlines, and n flights).

Subject	**Property**	**Object**
doi_0	τ	*Airline*
doi_0	*hasName*	*“Delta Air Lines″*
doi_0	*hasAbbreviation*	*“DL″*
doi_1	τ	*Airline*
doi_1	*hasName*	*“Aerolineas Argentinas″*
doi_1	*hasAbbreviation*	*“AR″*
...	...	...
doi_k	τ	*Airline*
doi_k	*hasName*	*“Air France″*
doi_k	*hasAbbreviation*	*“AF″*
doi_{k+1}	τ	*Airport*
doi_{k+1}	*hasName*	*“Ministro Pistarini International Airport″*
doi_{k+1}	*hasCode*	*“EZE″*
doi_{k+1}	*hasCity*	*“Buenos Aires″*
doi_{k+2}	τ	*Airport*
doi_{k+2}	*hasName*	*“Charles de Gaulle Airport″*
doi_{k+2}	*hasCode*	*“CDG″*
doi_{k+2}	*hasCity*	*“Paris″*
doi_{k+3}	τ	*Airport*
doi_{k+3}	*hasName*	*“John F. Kennedy International Airport″*
doi_{k+3}	*hasCode*	*“JFK″*
doi_{k+3}	*hasCity*	*“New York″*
doi_{k+4}	τ	*Airport*
doi_{k+4}	*hasName*	*“San Paulo – Guarulhos International Airport″*
doi_{k+4}	*hasCode*	*“GRU″*
doi_{k+4}	*hasCity*	*“San Pablo″*
...	...	...
doi_{k+l}	τ	*Airport*
doi_{k+l}	*hasName*	*“Madrid – Barajas Airport″*
doi_{k+l}	*hasCode*	*“MAD″*
doi_{k+l}	*hasCity*	*“Madrid′*
doi_{k+l+1}	τ	*Flight*
doi_{k+l+1}	*hasAirline*	doi_k
doi_{k+l+1}	*hasFlightNumber*	*“AF* 394″
doi_{k+l+1}	*hasDepartureTime*	14/06/2013, 23.20
doi_{k+l+1}	*hasEstimatedArrivalTime*	15/06/2013, 08.05
doi_{k+l+1}	*hasOriginAirport*	doi_{k+2}
doi_{k+l+1}	*hasDestinationAirport*	doi_{k+1}
doi_{k+l+2}	τ	*Flight*
doi_{k+l+2}	*hasAirline*	doi_k
doi_{k+l+2}	*hasFlightNumber*	*“AF* 1400″
doi_{k+l+2}	*hasDepartureTime*	14/06/2013, 20.00
doi_{k+l+2}	*hasEstimatedArrivalTime*	14/06/2013, 22.05
doi_{k+l+2}	*hasOriginAirport*	doi_{k+2}
doi_{k+l+2}	*hasDestinationAirport*	doi_{k+l}
doi_{k+l+3}	τ	*Flight*
doi_{k+l+3}	*hasAirline*	doi_1
doi_{k+l+3}	*hasFlightNumber*	*“AR* 1132″
doi_{k+l+3}	*hasDepartureTime*	17/06/2013, 23.55
doi_{k+l+3}	*hasEstimatedArrivalTime*	18/06/2013, 17.10
doi_{k+l+3}	*hasOriginAirport*	doi_{k+1}
doi_{k+l+3}	*hasDestinationAirport*	doi_{k+l}
...	...	...
doi_{k+l+n}	τ	*Flight*
doi_{k+l+n}	*hasAirline*	doi_0
doi_{k+l+n}	*hasFlightNumber*	*“DL* 1164″
doi_{k+l+n}	*hasDepartureTime*	17/06/2013, 14.55
doi_{k+l+n}	*hasEstimatedArrivalTime*	18/06/2013, 06.40
doi_{k+l+n}	*hasOriginAirport*	doi_{k+3}
doi_{k+l+n}	*hasDestinationAirport*	doi_{k+2}

Fig. 2.15: RDF flights dataset

Second, RDF specification supports a form of incomplete information through blank nodes, enabling us to refer and state facts about unknown literals or URIs using blank nodes in the triples. Whereas in standard relational databases all attribute values must be defined, either with constants or the special value NULL.

Example 8 (Flights continued). *For instance, continuing with the example above, we could state that there's an unknown resource* $\_{:}b_0$ *of type Airline, with abbreviation "TAM" and name "TAM Linhas Aereas" for which there's a flight from San Pablo to Buenos Aires scheduled on the 19th of June, as shown in Figure 2.16.*

$$\begin{array}{l}
\langle \_{:}b_0\ \tau\ Airline\rangle,\\
\langle \_{:}b_0\ hasName\ \text{"}TAM\ Linhas\ Aereas''\rangle,\\
\langle \_{:}b_0\ hasAbbreviation\ \text{"}TAM''\rangle,\\
\langle doi_{k+l+n+1}\ \tau\ Flight\rangle,\\
\langle doi_{k+l+n+1}\ hasAirline\ \_{:}b_0\rangle,\\
\langle doi_{k+l+n+1}\ hasFlightNumber\ \text{"}TAM8018''\rangle,\\
\langle doi_{k+l+n+1}\ hasDepartureTime\ 19/06/2013,\ 14.25\rangle,\\
\langle doi_{k+l+n+1}\ hasEstimatedArrivalTime\ 19/06/2013,\ 17.15\rangle,\\
\langle doi_{k+l+n+1}\ hasOriginAirport\ doi_{k+4}\rangle,\\
\langle doi_{k+l+n+1}\ hasDestinationAirport\ doi_{k+1}\rangle
\end{array}$$

Fig. 2.16: RDF extended flights dataset

On the other hand, classic relational databases cannot represent incomplete information. Therefore, the flight cannot be represent in the database presented in Figure 2.14, *or*, if the database constraints allows it, the flight can be represented using the special value NULL in the airline attribute as its shown in Figure 2.17.

Flights	*FlightId*	*AirlineId*	*Flight Number*	*Orig. AirportId*	*Dest. AirportId*	*Departure Time*	*Est. Arrival Time*
	1	3	"*AF* 394″	2	1	14/06/2013, 23.20	15/06/2013, 08.05
	2	3	"*AF* 1400″	2	5	14/06/2013, 20.00	14/06/2013, 22.05
	3	2	"*AR* 1132″	1	5	17/06/2013, 23.55	18/06/2013, 17.10
	...	...	...	...	...	...	...
	4	1	"*DL* 1164″	3	2	17/06/2013, 14.55	18/06/2013, 06.40
	5	*NULL*	"*TAM* 8018″	4	1	19/06/2013, 14.25	19/06/2013, 17.15

Fig. 2.17: Extended flights relational table

Finally, the RDF data model follows the Open World Assumption, in contrast with typical relational databases that are under the CWA instead. Therefore, in traditional relational databases all the data is explicit, while RDF *implicit triples* are considered to be part of the dataset, even though they are not explicitly present in it. The main source of implicit triples are the (domain) constraints specified in the RDF Schema (optional) [26].

Example 9 (Flights extended). *For instance, if we extend the Example 7 as its shown in Figure 2.18, using RDFS constraints, the fact stated by the (implicit) triple* $\langle doi_i\ \tau\ Airline\rangle$ *(*$i = k+l+n+2$*, for readability) holds in the dataset because the triples* $\langle doi_{i+2}\ hasAirline\ doi_i\rangle$ *and* $\langle hasAirline\ \hookrightarrow_r\ Airline\rangle$ *are present in the dataset. Moreover, the facts* $\langle doi_{i+1}\ \tau\ Airport\rangle$ *and* $\langle \_{:}b_1\ \tau\ Airport\rangle$ *also holds in the dataset because the following triples are present:*

$$\begin{array}{c}
\langle doi_{i+2}\ hasOriginAirport\ doi_{i+1}\rangle,\\
\langle hasOriginAirport\ \hookrightarrow_r\ Airport\rangle,\\
\langle doi_{i+2}\ hasDestinationAirport\ \_{:}b_1\rangle,\\
\langle hasDestinationAirport\ \hookrightarrow_r\ Airport\rangle
\end{array}$$

Data (facts)	**RDF Schema (constraints)**
$\langle doi_{k+l+n+2}$ *hasName* "*American Airlines*"$\rangle$,	$\langle$*Airline* $\prec_{sc}$ *Company*$\rangle$,
$\langle doi_{k+l+n+3}$ *hasName* "*BrusselsAirport*"$\rangle$,	$\langle$*hasAirline* $\hookleftarrow_d$ *Flight*$\rangle$,
$\langle$_:b_1 *hasName* "*London Heathrow Airport*"$\rangle$,	$\langle$*hasAirline* $\hookrightarrow_r$ *Airline*$\rangle$,
$\langle doi_{k+l+n+4}$ τ *Flight*$\rangle$,	$\langle$*hasOriginAirport* $\hookleftarrow_d$ *Flight*$\rangle$,
$\langle doi_{k+l+n+4}$ *hasAirline* $doi_{k+l+n+2}\rangle$,	$\langle$*hasOriginAirport* $\hookrightarrow_r$ *Airport*$\rangle$,
$\langle doi_{k+l+n+4}$ *hasFlightNumber* "*AA6399*"$\rangle$,	$\langle$*hasDestinationAirport* $\hookleftarrow_d$ *Flight*$\rangle$,
$\langle doi_{k+l+n+4}$ *hasDepartureTime* 19/06/2013, 11.50$\rangle$,	$\langle$*hasDestinationAirport* $\hookrightarrow_r$ *Airport*$\rangle$,
$\langle doi_{k+l+n+4}$ *hasEstimatedArrivalTime* 19/06/2013, 12.05$\rangle$,	
$\langle doi_{k+l+n+4}$ *hasOriginAirport* $doi_{k+l+n+3}\rangle$,	
$\langle doi_{k+l+n+4}$ *hasDestinationAirport* _:$b_1\rangle$	

Fig. 2.18: RDF extended flights dataset

Figure 2.19 summarize in a tabular way, as a (large) triplet table, the RDF dataset of the extended Example 9.

Notations Without loss of generality, in the following we will use the conjunctive query notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$ for both `ASK` and `SELECT` queries (for Boolean queries, $\bar{x}$ is empty). We use x, y, and z (possibly with subscripts) to denote variables in queries. We denote by `VarBl`(q) the set of variables and blank nodes occurring in the query q. The set of values (URIs, blank nodes, literals) of a dataset D is denoted `Val`(D).

Query evaluation Given a query q and a dataset D, the *evaluation of* q *against* D is:

$$q(D) = \{\bar{x}_\mu \mid \mu : \texttt{VarBl}(q) \rightarrow \texttt{Val}(D) \text{ is a total assignment s.t. } (t_1, \ldots, t_\alpha)_\mu \subseteq D\}$$

where for a given triple (or triple set) O, we denote by O_μ the result of replacing every occurrence of a variable or blank node $e \in \texttt{VarBl}(q)$ in O, by the value $\mu(e) \in \texttt{Val}(D)$. If q is Boolean, the empty answer set encodes $false$, while the non-empty answer set made of the empty tuple $\emptyset_\mu = \langle\rangle$ encodes *true*.

Notice that (normative) query evaluation *treats the blank nodes in a query as non-distinguished variables.* That is, one could consider equivalently queries with blank nodes or queries with non-distinguished variables.

Query answering It is important to keep in mind the distinction between query *evaluation* and query *answering.* The evaluation of q against D only uses D's explicit triples, thus may lead to an incomplete answer set. The (complete) *answer set* of q against D is obtained by the evaluation of q against the closure of D, denoted by $q(D^\infty)$.

Example 10 (**Query answering**)**.** *Consider again the dataset used in example 9. The following query asks for all the airports:*

$$q(x)\text{:- } x\ \tau\ Airport$$

The evaluation of q against the data in Figure 2.19, $q(D)$*, leads to the incomplete answer data set* $q(D_{data}) = \{doi_{k+1}, doi_{k+2}, doi_{k+3}, doi_{k+4}, \ldots, doi_{k+l}\}$*. The evaluation of the same query q against the data and the schema shown in the same figure (i.e., the evaluation of the query against the closure of the dataset,* $q(D^\infty)$*) leads to the correct answer (including the implicit triples):* $q(D^\infty) = \{doi_{k+1}, doi_{k+2}, doi_{k+3}, doi_{k+4}, \ldots, doi_{k+l}, doi_{k+l+n+3}, \_{:}b_1\}$.

Data (facts)

Subject	**Property**	**Object**
doi_0	τ	*Airline*
doi_0	*hasName*	“*Delta Air Lines*″
doi_0	*hasAbbreviation*	“*DL*″
doi_1	τ	*Airline*
doi_1	*hasName*	“*Aerolineas Argentinas*″
doi_1	*hasAbbreviation*	“*AR*″
...	...	...
doi_k	τ	*Airline*
doi_k	*hasName*	“*Air France*″
doi_k	*hasAbbreviation*	“*AF*″
doi_{k+1}	τ	*Airport*
doi_{k+1}	*hasName*	“*Ministro Pistarini International Airport*″
doi_{k+1}	*hasCode*	“*EZE*″
doi_{k+1}	*hasCity*	“*Buenos Aires*″
doi_{k+2}	τ	*Airport*
doi_{k+2}	*hasName*	“*Charles de Gaulle Airport*″
doi_{k+2}	*hasCode*	“*CDG*″
doi_{k+2}	*hasCity*	“*Paris*″
doi_{k+3}	τ	*Airport*
doi_{k+3}	*hasName*	“*John F. Kennedy International Airport*′
doi_{k+3}	*hasCode*	“*JFK*″
doi_{k+3}	*hasCity*	“*New York*″
doi_{k+4}	τ	*Airport*
doi_{k+4}	*hasName*	“*San Paulo − Guarulhos International Airport*″
doi_{k+4}	*hasCode*	“*GRU*″
doi_{k+4}	*hasCity*	“*San Pablo*″
...	...	...
doi_{k+l}	τ	*Airport*
doi_{k+l}	*hasName*	“*Madrid − Barajas Airport*″
doi_{k+l}	*hasCode*	“*MAD*″
doi_{k+l}	*hasCity*	“*Madrid*″
doi_{k+l+1}	τ	*Flight*
doi_{k+l+1}	*hasAirline*	doi_k
doi_{k+l+1}	*hasFlightNumber*	“*AF* 394″
doi_{k+l+1}	*hasDepartureTime*	14/06/2013, 23.20
doi_{k+l+1}	*hasEstimatedArrivalTime*	15/06/2013, 08.05
doi_{k+l+1}	*hasOriginAirport*	doi_{k+2}
doi_{k+l+1}	*hasDestinationAirport*	doi_{k+1}
doi_{k+l+2}	τ	*Flight*
doi_{k+l+2}	*hasAirline*	doi_k
doi_{k+l+2}	*hasFlightNumber*	“*AF* 1400″
doi_{k+l+2}	*hasDepartureTime*	14/06/2013, 20.00
doi_{k+l+2}	*hasEstimatedArrivalTime*	14/06/2013, 22.05
doi_{k+l+2}	*hasOriginAirport*	doi_{k+2}
doi_{k+l+2}	*hasDestinationAirport*	doi_{k+l}
...	...	...
doi_{k+l+3}	τ	*Flight*
doi_{k+l+3}	*hasAirline*	doi_1
doi_{k+l+3}	*hasFlightNumber*	“*AR* 1132″
doi_{k+l+3}	*hasDepartureTime*	17/06/2013, 23.55
doi_{k+l+3}	*hasEstimatedArrivalTime*	18/06/2013, 17.10
doi_{k+l+3}	*hasOriginAirport*	doi_{k+1}
doi_{k+l+3}	*hasDestinationAirport*	doi_{k+l}
...	...	...
doi_{k+l+n}	τ	*Flight*
doi_{k+l+n}	*hasAirline*	doi_0
doi_{k+l+n}	*hasFlightNumber*	“*DL* 1164″
doi_{k+l+n}	*hasDepartureTime*	17/06/2013, 14.55
doi_{k+l+n}	*hasEstimatedArrivalTime*	18/06/2013, 06.40
doi_{k+l+n}	*hasOriginAirport*	doi_{k+3}
doi_{k+l+n}	*hasDestinationAirport*	doi_{k+2}
$\_{:}b_0$	τ	*Airline*
$\_{:}b_0$	*hasName*	“*TAM Linhas Aereas*″
$\_{:}b_0$	*hasAbbreviation*	“*TAM*″
$doi_{k+l+n+1}$	τ	*Flight*
$doi_{k+l+n+1}$	*hasAirline*	$\_{:}b_0$
$doi_{k+l+n+1}$	*hasFlightNumber*	“*TAM*8018″
$doi_{k+l+n+1}$	*hasDepartureTime*	19/06/2013, 14.25
$doi_{k+l+n+1}$	*hasEstimatedArrivalTime*	19/06/2013, 17.15
$doi_{k+l+n+1}$	*hasOriginAirport*	doi_{k+4}
$doi_{k+l+n+1}$	*hasDestinationAirport*	doi_{k+1}
$doi_{k+l+n+2}$	*hasName*	“*American Airlines*″
$doi_{k+l+n+3}$	*hasName*	“*BrusselsAirport*″
$\_{:}b_1$	*hasName*	“*London Heathrow Airport*″
$doi_{k+l+n+4}$	τ	*Flight*
$doi_{k+l+n+4}$	*hasAirline*	$doi_{k+l+n+2}$
$doi_{k+l+n+4}$	*hasFlightNumber*	“*AA*6399″
$doi_{k+l+n+4}$	*hasDepartureTime*	19/06/2013, 11.50
$doi_{k+l+n+4}$	*hasEstimatedArrivalTime*	19/06/2013, 12.05
$doi_{k+l+n+4}$	*hasOriginAirport*	$doi_{k+l+n+3}$
$doi_{k+l+n+4}$	*hasDestinationAirport*	$\_{:}b_1$

RDF Schema (constraints)

$\langle Airline \prec_{sc} Company \rangle$,
$\langle hasAirline \hookleftarrow_{d} Flight \rangle$,
$\langle hasAirline \hookrightarrow_{r} Airline \rangle$,
$\langle hasOriginAirport \hookleftarrow_{d} Flight \rangle$,
$\langle hasOriginAirport \hookrightarrow_{r} Airport \rangle$,
$\langle hasDestinationAirport \hookleftarrow_{d} Flight \rangle$,
$\langle hasDestinationAirport \hookrightarrow_{r} Airport \rangle$

Fig. 2.19: RDF extended flights dataset

2.6 RDF and RDBMs

A generalization of the traditional relational tables with NULL values, called *V-tables*, allows the use of variables in their tuples. Moreover, the use of the same variable in different tuples of a V-table allows us to express joins on unknown values. Figure 2.20 shows an example of a V-table for the Ligue 1 (French Football League) 2013/2014 season fixture.

Home	*Away*	*Week*
FC Nantes	*SC Bastia*	1
LOSC Lille	*FC Lorient*	1
Olympique Lyonnais	x	1
y	*Paris Saint – Germain*	1
...	...	...
x	*Stade Rennais FC*	2
Paris Saint – Germain	*AC Ajaccio*	2
z	*LOSC Lille*	2
AS Monaco FC	y	2
...	...	...
x	y	4
...	...	...

Fig. 2.20: Example of a V-table.

Previous work [33] introduced a representation system for incomplete information relational databases, based on V-Tables, that supports projection, positive selection, union, join, and remaining of attributes. A key result on V-table querying is that standard relational evaluation (which sees variables in V-tables as constants) computes the exact answer set of any conjunctive query [33]. Further, provides a possible way of answering conjunctive queries against V-tables using standard relational database management systems (or RDBMSs, in short).

RDF datasets turn out to be a special case of incomplete relational databases based on *V-tables.* Therefore, using RDBMS evaluation, we can obtain complete answer sets to BGP queries as follows:

Given a dataset D, we encode it into the V-table $\texttt{Triples}(\texttt{s}, \texttt{p}, \texttt{o})$ storing the triples of D as tuples, in which blank nodes become variables. Then, given a BGP query $q(\bar{x})$:- $\texttt{s}_1\ \texttt{p}_1\ \texttt{o}_1, \ldots, \texttt{s}_n\ \texttt{p}_n\ \texttt{o}_n$, in which blank nodes have been equivalently replaced by fresh non-distinguished variables, the SPARQL evaluation $q(D)$ of q against D is obtained by the relational evaluation of the conjunctive query:

$$q(\bar{x}) \text{:- } \bigwedge_{i=1}^{n} \texttt{Triples}(\texttt{s}_i, \texttt{p}_i, \texttt{o}_i)$$

against the `Triples` table. Indeed, *SPARQL and relational evaluations coincide with the above encoding*, as relational evaluation amounts to finding all the total assignments from the variables of the query to the values (constants and variables) in the `Triples` table, so that the query becomes a subset of that `Triples` table [1].

Trough the two established techniques for handling RDF entailment, namely *saturation* and *reformulation*, we can compute the answer set of q against D in the following ways:

- Saturation: evaluating $q(\bar{x})$:- $\bigwedge_{i=1}^{n}$ $\texttt{Triples}(\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$ against the $\texttt{Triples}$ table containing D^∞ (the closure of D).

- Reformulation: evaluating $q'(\bar{x})$:- $\bigwedge_{i=1}^{m}$ $\texttt{Triples}(\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$ against the $\texttt{Triples}$ table containing D facts, where q' is the reformulation of the query q w.r.t. D schema (constraints).

$$\frac{\{\mathbf{c}_1 \ \prec_{sc}\ \mathbf{c}_2, \mathtt{s}\ \tau\ \mathbf{c}_1\} \subseteq D}{D = D \cup \{\mathtt{s}\ \tau\ \mathbf{c}_2\}} \tag{2.1}$$

$$\frac{\{\mathbf{p}\ \hookleftarrow_d\ \mathbf{c}, \mathtt{s}\ \mathbf{p}\ \mathtt{o}\} \subseteq D}{D = D \cup \{\mathtt{s}\ \tau\ \mathbf{c}\}} \tag{2.2}$$

$$\frac{\{\mathbf{p}\ \hookrightarrow_r\ \mathbf{c}, \mathtt{s}\ \mathbf{p}\ \mathtt{o}\} \subseteq D}{D = D \cup \{\mathtt{o}\ \tau\ \mathbf{c}\}} \tag{2.3}$$

$$\frac{\{\mathbf{p}_1\ \prec_{sp}\ \mathbf{p}_2, \mathtt{s}\ \mathbf{p}_1\ \mathtt{o}\} \subseteq D}{D = D \cup \{\mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\}} \tag{2.4}$$

Fig. 2.21: Saturation rules for a dataset D.

Saturation-based query answering Given a dataset D, the closure of the dataset is computed using the entailment rules shown in Figure2.21; then, the evaluation of every query against the saturation leads to the complete answer set. This method is straightforward and easy to implement. Its disadvantages are that dataset saturation requires computation time and storage space for all the entailed triples; moreover, the saturation must be recomputed upon every update. Incremental algorithms for saturation maintenance had been proposed in previous work [1]. However, the recursive nature of entailment makes this process costly (in time and space) and this method not suitable for datasets with a high rate of updates.
Table 2.1, extracted from the cited work [1], presents the characteristics of well-known datasets and shows that saturation adds between 10% and 41% to the dataset size, while multiset-based saturation (required for the incrementally maintaining the saturation technique presented by the authors) increase the size between 116% and 227%.

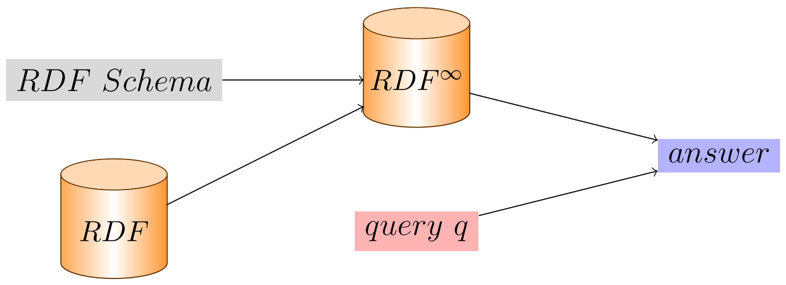


Fig. 2.22: Saturation-based query answering overview.

Reformulation-based query answering Given a query q and a dataset D, reformulate (using the immediate entailment rules) q w.r.t. D schema into another query q', such that

Graph	#Schema	#Instance	#Saturation	Saturation increase (%)	#Multiset	Multiset increase (%)
Barton [34]	101	33, 912, 142	38, 969, 023	14.91	73, 551, 358	116.89
DBpedia [24]	5666	26, 988, 098	29, 861, 597	10.65	66, 029, 147	227.37
DBLP [35]	41	8, 424, 216	11, 882, 409	41.05	18, 699, 232	121.97

Tab. 2.1: Datasets and saturation characteristics [1].

the evaluation of q' against D data, denoted $q'(D_{data})$, is the complete answer set of q against D (i.e., $q(D^{\infty})$). The main advantage of this method is that its robust to update, there is no need to (re)compute the closure of the dataset. In the other hand, the reformulation is made at run time, often resulting in a more complex query than the original one and with a costlier evaluation.

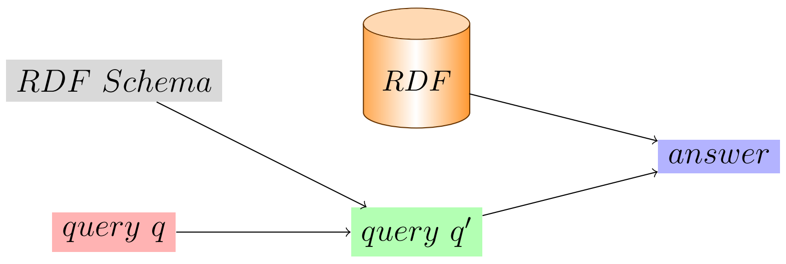


Fig. 2.23: Reformulation-based query answering overview.

In this work, we focus on *reformulation-based query answering* only for *instance-level queries*. The following theorem, introduced in [4], shows that to answer such queries, among the DB fragment's rules shown in Figures 2.6–2.9, it suffices to consider only the entailment rules in Figure 2.8. The proof strategy for Theorem 2.6.1 is based on the original one [4] and adapted to the current work.

Theorem 2.6.1. *[RDF triple entailment] [4] Let D be a dataset, t_1 be a triple of the form* $\mathtt{s}\ \tau\ \mathtt{o}$*, and t_2 be a triple of the form* $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$*. $t_1 \in D^{\propto}$ (respectively, $t_2 \in D^{\propto}$) iff there exists a sequence of application of the rules in Figure 2.8 leading from D to t_1 (respectively t_2), assuming that each entailment step relies on D and all triples previously entailed.*

Proof.

$\Leftarrow$) The proof in this direction is trivial since the rules of Figure 2.8 are among the ones that are used to define $D^{\propto}$.

$\Rightarrow$) Let us call a *derivation* of t any sequence of immediate entailment rules that produces the entailed triple t, starting from D. Let us consider, without loss of generality, a *minimal* derivation (i.e., in which removing a step of rule application does not allow deriving t anymore). A derivation can be minimized by gradually removing steps producing entailed triples that are not further reused in the entailment sequence of t. We show for such a *minimal* derivation of an entailed triple t that any step using a rule that is not in Figure 2.8 can be replaced by a sequence of steps using only rules from Figure 2.8, leading to another derivation of t. Applying exhaustively the above replacement on the minimization of obtained derivations obviously leads to a derivation of t using the rules in Figure 2.8 only.

Consider a minimal derivation of t using the immediate entailment rule from Figure 2.8: $\mathtt{s} \prec_{sc} \mathtt{o}, \mathtt{s}_1\ \tau\ \mathtt{s} \vdash_{\text{RDF}} \mathtt{s}_1\ \tau\ \mathtt{o}$ While the triple $\mathtt{s}_1\ \tau\ \mathtt{s}$ is either in D or produced

by a rule from Figure 2.8 (only the rules in Figure 2.8 produce such a triple), the triple $\mathtt{s}\ \prec_{sc}\ \mathtt{o}$ may result from the triples $\{\mathtt{s}\ \prec_{sc}\ \mathtt{o}_n, \mathtt{o}_n\ \prec_{sc}\ \mathtt{o}_{n-1}, \ldots, \mathtt{o}_1\ \prec_{sc}\ \mathtt{o}\} \subseteq D$ and n applications of the rule $\mathtt{s}\ \prec_{sc}\ \mathtt{o},\ \mathtt{o}\ \prec_{sc}\ \mathtt{o}_1 \vdash_{\text{RDF}} \mathtt{s}\ \prec_{sc}\ \mathtt{o}_1$ from Figure 2.9 (only that rule produces triples of the form $\mathtt{s}\ \prec_{sc}\ \mathtt{o}$). Observe that we do not have to consider the rules from Figures 2.6 and 2.7 in a *minimal* derivation. It is therefore easy to see that the application of $\mathtt{s}\ \prec_{sc}\ \mathtt{o}, \mathtt{s}_1\ \tau\ \mathtt{s} \vdash_{\text{RDF}} \mathtt{s}_1\ \tau\ \mathtt{o}$ in the derivation of t can be replaced by the following sequence:
$\mathtt{s}\ \prec_{sc}\ \mathtt{o}_n, \mathtt{s}_1\ \tau\ \mathtt{s} \vdash_{\text{RDF}} \mathtt{s}_1\ \tau\ \mathtt{o}_n,$
$\mathtt{o}_n\ \prec_{sc}\ \mathtt{o}_{n-1}, \mathtt{s}_1\ \tau\ \mathtt{o}_n \vdash_{\text{RDF}}$
$\mathtt{s}_1\ \tau\ \mathtt{o}_{n-1},$
$\ldots,$
$\mathtt{o}_1\ \prec_{sc}\ \mathtt{o}, \mathtt{s}_1\ \tau\ \mathtt{o}_1 \vdash_{\text{RDF}} \mathtt{s}_1\ \tau\ \mathtt{o}$

The rest of the proof is omitted as it amounts to show, similarly as above, that the claim also holds for the three other immediate entailment rules of Figure 2.8.

□

2.7 Query reformulation

Let me reformulate the question

Given a query q and a dataset D, we want to reformulate q w.r.t. D schema into another query q', such that the evaluation of q' against D data, denoted $q'(D_{data})$, is the complete answer set of q against D (i.e., $q(D^\infty)$). The **Reformulate** algorithm, previously introduced in [26] and extended in [1], exhaustively applies the rules shown in Figure 2.24, each of which defines a transformation of the form $\frac{input}{output}$. The input of a transformation can be of the form ⟨logical condition w.r.t. q⟩ or ⟨logical condition w.r.t. D, logical condition w.r.t. q⟩, while the output is a (new) query q' based on the input query q and the schema of dataset D. In general terms, each rule produces a new query (its output) when the rule's input conditions are satisfied, by some query (either the original query q or a query q' produced by a previous application of a rule) and, optionally, by the schema of dataset D (depending on the form of the input of the rule). The reformulation of the query q w.r.t. the dataset schema is the set of all the queries produced by applying the rules.

2.7.1 Partially instantiated queries

Let $q(\bar{x})$:- $t_1, \ldots, t_n$ be a query and σ be a mapping from a subset of the variables and blank nodes of q, to some values (literals, URIs, or blank nodes).

Given a query q and a mapping σ, a *partially instantiated query* w.r.t. σ, denoted q_σ, is a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_n)_\sigma$ where the mapping σ has been applied both on q's head variables $\bar{x}$ and on q's body triple patterns. In the case of $\sigma = \emptyset$, q_σ coincides with the original (non-instantiated) query q. Observe that, in non-standard fashion, in partially instantiated queries some distinguished (head) variables of q_σ can be bound.

By allowing constants in the head, partially instantiated queries go outside the reach of our definition of BGP queries. Accordingly, a slight extension is required to the notions of BGP query evaluation and answer sets, introduced in Section 2.5 for graphs, as in [1].

Given a partially instantiated query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_n)_\sigma$ whose set of variables and blank nodes is $\mathtt{VarBl}(q_\sigma)$ and a dataset D whose set of values (literals, URIs and blank nodes) is $\mathtt{Val}(D)$, the *evaluation* of q_σ against D is:

$$q_\sigma(D) = \{(\bar{x}_\sigma)_\mu \mid \mu : \mathtt{VarBl}(q_\sigma) \rightarrow \mathtt{Val}(D) \text{ is a total assignment } ((t_1, \ldots, t_n)_\sigma)_\mu \subseteq D\}$$

The complete *answer set* of q_σ against the dataset D is the evaluation of q_σ against $D^\propto$, denoted $q_\sigma(D^\propto)$.

2.7.2 Reformulation rules

$$\frac{\langle \mathtt{s}\ x\ \mathtt{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \tau\}}} \tag{2.5}$$

$$\frac{\langle \mathtt{s}\ \mathbf{p}\ \mathtt{o} \in D, \mathtt{s}_1\ x\ \mathtt{o}_1 \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{p}\}}} \tag{2.6}$$

$$\frac{\langle \mathtt{p}_1 \ \prec_{sp}\ \mathbf{p} \in D, \mathtt{s}\ x\ \mathtt{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{p}\}}} \tag{2.7}$$

$$\frac{\langle \mathbf{p} \ \prec_{sp}\ \mathtt{p}_1 \in D, \mathtt{s}\ x\ \mathtt{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{p}\}}} \tag{2.8}$$

$$\frac{\langle \mathtt{s}_1\ \tau\ \mathbf{c} \in D, \mathtt{s}\ \tau\ x \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{c}\}}} \tag{2.9}$$

$$\frac{\langle \mathtt{c}_1 \ \prec_{sc}\ \mathbf{c} \in D, \mathtt{s}\ \tau\ x \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{c}\}}} \tag{2.10}$$

$$\frac{\langle \mathbf{c} \ \prec_{sc}\ \mathtt{c}_1 \in D, \mathtt{s}\ \tau\ x \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{c}\}}} \tag{2.11}$$

$$\frac{\langle \mathtt{s}_1 \ \hookleftarrow_d\ \mathbf{c} \in D, \mathtt{s}\ \tau\ x \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{c}\}}} \tag{2.12}$$

$$\frac{\langle \mathtt{s}_1 \ \hookrightarrow_r\ \mathbf{c} \in D, \mathtt{s}\ \tau\ x \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{x \rightarrow \mathbf{c}\}}} \tag{2.13}$$

$$\frac{\langle \mathbf{c}_1 \ \prec_{sc}\ \mathbf{c} \in D, \mathtt{s}\ \tau\ \mathbf{c} \in q_\sigma \rangle}{q_{\sigma[\mathtt{s}\ \tau\ \mathbf{c} / \mathtt{s}\ \tau\ \mathbf{c}_1]}} \tag{2.14}$$

$$\frac{\langle \mathbf{p} \ \hookleftarrow_d\ \mathbf{c} \in D, \mathtt{s}\ \tau\ \mathbf{c} \in q_\sigma \rangle}{q_{\sigma[\mathtt{s}\ \tau\ \mathbf{c} / \mathtt{s}\ \mathbf{p}\ x]}} \tag{2.15}$$

$$\frac{\langle \mathbf{p} \ \hookrightarrow_r\ \mathbf{c} \in D, \mathtt{s}\ \tau\ \mathbf{c} \in q_\sigma \rangle}{q_{\sigma[\mathtt{s}\ \tau\ \mathbf{c} / x\ \mathbf{p}\ \mathtt{s}]}} \tag{2.16}$$

$$\frac{\langle \mathbf{p}_1 \ \prec_{sp}\ \mathbf{p} \in D, \mathtt{s}\ \mathbf{p}\ \mathtt{o} \in q_\sigma \rangle}{q_{\sigma[\mathtt{s}\ \mathbf{p}\ \mathtt{o} / \mathtt{s}\ \mathbf{p}_1\ \mathtt{o}]}} \tag{2.17}$$

Fig. 2.24: Reformulation rules for a partially instantiated query q_σ w.r.t. a dataset D.

The set of rules in Figure 2.24 can be divided in two groups. The first group, rules (2.5)–(2.13), reformulate queries by binding one of their variables, either to the RDF built-in property τ or to a class or property defined in the dataset. For instance, the rule (2.5) states that if q_σ contains a triple of the form $\mathtt{s}\ x\ \mathtt{o}$, i.e., having a variable in the property position beyond the values in the subject and object positions, then a new query, $q_{\sigma \cup \nu}$, binding the variable x to the RDF built-in property τ is created. Observe that if x was a distinguished variable in the original query, q_σ, a head variable in the new query, $q_{\sigma\cup\nu}$, will be bound after the rule application.

Consider rule (2.6) regarding some query q_σ. If the dataset schema contains a triple in which property $\mathbf{p}$ is defined, and a triple of the form $\mathtt{s}\ x\ \mathtt{o}$ appears in q_σ, then by this rule a new query $q_{\sigma\cup\nu}$ where x is bound to the property $\mathbf{p}$ is created.

The RDFS axiomatics triples $\langle rdfs{:}subPropertyOf\ rdfs{:}domain\ rdf{:}Property\rangle$ and $\langle rdfs{:}subPropertyOf\ rdfs{:}range\ rdf{:}Property\rangle$ states that both the subject and the object of $\prec_{sp}$ statements are properties. Therefore, they can be used to instantiate the variables in the property position of a triple appearing in a query. Given a query q_σ, the rule (2.7) instantiate query variables appearing in the property position of a triple contained in the query, to the value appearing in the object position of a $\prec_{sp}$ statement that appears in the dataset schema. Similarly, if q_σ contains a triple of the form $\mathtt{s}\ x\ \mathtt{o}$ and a $\prec_{sp}$ statement appears in D schema, the rule (2.8) creates a new query binding the variable x to the value appearing in the subject position of the $\prec_{sp}$ statement.

Rules (2.9)–(2.13) instantiate the variable x in a query triple of the form $\mathtt{s}\ \tau\ x$. The RDF meta-model specifies that the values of the τ property are classes. Therefore, the rules bind x to D values of which it can be inferred that they are classes, i.e., those appearing in specific positions of schema-level triples. For instance, the rule (2.9) says: if a triple of the from $\mathtt{s}\ \tau\ x$, i.e., having any kind of subject, but having the RDF built-in property τ in the property position and a variable in the object position, appears in q_σ, and the dataset schema contains a triple of the form $\mathtt{s}_1\ \tau\ \mathbf{c}$, then create a new query that binds the variable x with the class $\mathbf{c}$. Similarly, the rules (2.10) and (2.11) instantiate query variables appearing in the subject position to values appearing in a $\prec_{sc}$ statement content in D schema. In this case, the RDFS axiomatics triples $\langle rdfs{:}subClassOf\ rdfs{:}domain\ rdfs{:}Class\rangle$ and $\langle rdfs{:}subClassOf\ rdfs{:}range\ rdfs{:}Class\rangle$, which states that both the subject and the object of $\prec_{sc}$ statements are classes, allows the binding of query variables appearing in the subject position of triples contained in the query. Finally, rules (2.12) and (2.13) binds variables appearing in the object position of a triple of the form $\mathtt{s}\ \tau\ x$ contained in the query q_σ, to the class appearing in the object position of a $\hookleftarrow_d$ or $\hookrightarrow_r$ statement appearing in the schema of dataset D.

The second group of rules (2.14)–(2.17) alter the query by replacing (denoted *old triple / new triple*) one of its triples by another, using schema triples. Rule (2.14) exploits $\prec_{sc}$ statements contained in D schema: if the query q_σ has a triple of the form $\mathtt{s}\ \tau\ \mathbf{c}$, i.e., seeks for instances of class $\mathbf{c}$, and $\mathbf{c}_1$ is a subclass of $\mathbf{c}$, then instances of $\mathbf{c}_1$ should also be returned. Similarly, rules (2.15) and (2.16) uses $\hookleftarrow_d$ and $\hookrightarrow_r$ statements in the schema, of the from $\mathbf{p}\ \hookleftarrow_d\ \mathbf{c}$ and $\mathbf{p}\ \hookrightarrow_r\ \mathbf{c}$ respectively to replace the triple $\mathbf{p}\ \hookleftarrow_d\ \mathbf{c}$, appearing in q_σ, with the new triples $\mathtt{s}\ \mathbf{p}\ x$, in the case of rule (2.15), and $x\ \mathbf{p}\ \mathtt{s}$, in the case of rule (2.16). Finally, rule (2.17) exploits $\prec_{sp}$ statements appearing in the schema analogously to rule (2.14) exploits $\prec_{sc}$ statements. If a triple of the form $\mathtt{s}\ \mathbf{p}\ \mathtt{o}$ is contained in the query q_σ and $\mathbf{p}_1$ is a sub property of $\mathbf{p}$, then the triples with property $\mathbf{p}$ should also be returned.

Example 11 (**Reformulation rules**). *Consider the DBLP dataset [35] and the query $q(x, y, z)$:- $x\ y\ z$ asking for the triples of the dataset (including the entailed ones). We show how some of the rules in Figure 2.24 can be used to reformulate q w.r.t. DBLP schema.*

(*i*) *Using q as input for rule (2.5) produces the query:*
$q_{\{y\to\tau\}}$, i.e., $q(x, \tau, z)$:- $x\ \tau\ z$.

(*ii*) *Using* $q_{\{y \to \tau\}}$ *as input for rule (2.12) can lead to:*
$q_{\{y \to \tau, z \to dblp:Document\}}$*, i.e.,* $q(x, \tau, dblp{:}Document)$*:-* $x\ \tau\ dblp{:}Document$*.*

(*iii*) *Finally, using* $q_{\{y \to \tau, z \to dblp:Document\}}$ *as input for rule (2.14) can lead to:*
$q(x, \tau, dblp{:}Document)$*:-* $x\ \tau\ dblp{:}Book$*.*

2.7.3 Reformulation-based query answering

Reformulation-based query answering, as explained in Section 2.6, reformulates a given query q w.r.t. the constraints defined in the RDF schema of the given dataset D into a query q' such that the evaluation of q' against D data (or facts), denoted $q'(D_{data})$, is the complete answer set of q against D (i.e., $q(D^{\infty})$). It is required that q' (the reformulated query) is equivalent to (or contained in) q (the original query) w.r.t. the constraints defined in the dataset D; otherwise the evaluation of q' might produce erroneous answers.

The query reformulation technique introduced in Sections 2.7.1 and 2.7.2, using the previously introduced definitions of evaluation and of answer set of a query w.r.t. a dataset, does not satisfy the requirement above. Moreover, the reason for which the requirement is not meet is due to blank nodes. The problem, as pointed out in [4], is that a blank node in a BGP query is a subject or object (might be both when appears in more than one RDF triple of the query) that is not identified by a URI and is not a literal, i.e., *a non-distinguished variable*; while in the context of the **Reformulate** algorithm a blank node is distinguished from any other blank node, i.e., when the algorithm brings a particular blank node by replacing a triple or binding a variable, it reference that singular blank node in the dataset.

Example 12 (**Erroneous answer**). *For example, consider now the dataset* D *in Figure 2.25 and the query* $q(x, y)$*:-* $x\ \tau\ y$*, whose reformulation (w.r.t.* D*) is presented in Figure 2.26.*

$$D = \begin{array}{l} \{doi_0\ \tau\ \_{:}b_0,\ doi_0\ hasTitle\ \text{“}Around\ the\ World\ in\ Eighty\ Days\text{''}, \\ doi_1\ hasWritten\ doi_0,\ doi_1\ hasName\ \text{“}Jules\ Verne\text{''},\ doi_1\ \tau\ Person, \\ hasName\ \hookleftarrow_d\ Person,\ hasName\ \hookrightarrow_r\ \text{rdfs:Literal},\ hasTitle\ \hookleftarrow_d\ Writing, \\ hasTitle\ \hookrightarrow_r\ \text{rdfs:Literal},\ Writer\ \prec_{sc}\ Person,\ hasWritten\ \hookleftarrow_d\ Writer, \\ hasWritten\ \hookrightarrow_r\ Writing,\ \_{:}b_0\ \prec_{sc}\ Writing\} \end{array}$$

Fig. 2.25: RDF dataset D.

$$\begin{array}{lcl} \textbf{Reformulate}^0(q, D) & = & \{q(x, y)\text{:- } x\ \tau\ y\} \\ \textbf{Reformulate}^1(q, D) & = & \textbf{Reformulate}^0(q, D) \cup \\ \multicolumn{3}{l}{\{q(x, Person)\text{:- } x\ \tau\ Person,\ q(x, Writer)\text{:- } x\ \tau\ Writer,} \\ \multicolumn{3}{l}{\ q(x, Writing)\text{:- } x\ \tau\ Writing, q(x, \_{:}b_0)\text{:- } x\ \tau\ \_{:}b_0\}} \\ \textbf{Reformulate}^2(q, D) & = & \textbf{Reformulate}^1(q, D) \cup \\ \multicolumn{3}{l}{\{q(x, Person)\text{:- } x\ \tau\ Writer,\ q(x, Writer)\text{:- } x\ hasWritten\ v,} \\ \multicolumn{3}{l}{\ q(x, Writing)\text{:- } x\ hasTitle\ v,\ q(x, Writing)\text{:- } v\ hasWritten\ x,} \\ \multicolumn{3}{l}{\ q(x, Writing)\text{:- } x\ \tau\ \_{:}b_0,\ q(x, Person)\text{:- } x\ hasName\ v\}} \\ \textbf{Reformulate}^3(q, D) & = & \textbf{Reformulate}^2(q, D) \end{array}$$

Fig. 2.26: Reformulation of the query $q(x, y)$:- $x\ \tau\ y$ w.r.t. the dataset D.

Observe that the evaluation of the query $q(x, Writing)$*:-* $x\ \tau\ \_{:}b_0 \in \textbf{Reformulate}(q, D)$ *against the dataset* D*, with the assignment* $\mu = \{x \to doi_1, \_{:}b_0 \to Person\}$*, produce the* *erroneous answer $\langle doi_1\ \tau\ Writing\rangle$. This is because using $q(x, Writing)$:- $x\ \tau\ Writing \in$* **Reformulate**$^1(q, D)$ *as input for rule (2.14), the* **Reformulate** *algorithm produces the query $q(x, Writing)$:- $x\ \tau\ \_{:}b_0$ with the sole purpose of finding writings, in particular of the subclass $\_{:}b_0$, values for the variable x.*

In order to solve the issue detailed and exemplified above, the authors [4] introduced:

- Alternate notions of both, *evaluation* and *answer set of a partially instantiated query*, against a database.
- A property showing the relation between standard and non-standard definitions of query evaluation and answer set of queries.
- A novel technique for reformulation-based query answering technique.

Non-standard evaluation and answer set of a query against a database Unlike the standard definitions introduced in Sections 2.5 and 2.7.1, the definitions introduced here take into account blank nodes.

Given a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ and a dataset D, the standard evaluation is based on assignments of the variables and blank nodes of q to values in D into database values, whereas the alternate definition binds only the query variables (leaving unchanged the blank nodes as the URIs and literals).

Let $\texttt{Val}(D)$ be the set of values (URIs, blank nodes, literals) of the dataset D, and let $\texttt{Var}(q_\sigma)$ be the set of variables (no blank nodes) of q:

- The *non-standard evaluation* of q_σ against D is defined as:

$$\tilde{q_\sigma}(D) = \{(\bar{x}_\sigma)_\mu \mid \mu : \texttt{Var}(q_\sigma) \to \texttt{Val}(D) \text{ is a total assignment s.t. } ((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq D\}.$$

- The *non-standard answer set* of q_σ w.r.t. D is obtained by the non-standard evaluation of q_σ against $D^\propto$, denoted $\tilde{q_\sigma}(D^\propto)$.

Property 1, as stated before, presents the relation between standard and non-standard definitions of query evaluation and answer set of queries.

Property 1. *Let D be a database and q_σ a (partially instantiated) query against D.*

1. *$\tilde{q}_\sigma(D) \subseteq q_\sigma(D)$ and $\tilde{q}_\sigma(D^\propto) \subseteq q_\sigma(D^\propto)$ hold.*
2. *If q_σ does not contain blank nodes then $\tilde{q}_\sigma(D) = q_\sigma(D)$ and $\tilde{q}_\sigma(D^\propto) = q_\sigma(D^\propto)$.*

The property above follows directly from the fact that the assignments μ involved in non-standard evaluations are defined only on the variables of q, which are a subset of the ones allowed in standard evaluations ($\texttt{VarBl}(q)$).

Using the above notation of non-standard evaluation, Theorem 2.7.1 presents a novel technique for reformulation-based query answering; the theorem and it proof can be found in [4].

Theorem 2.7.1. *Given a BGP query q without blank nodes and a dataset D whose schema is $\mathcal{S}$, the following holds:*

$$q(D^\propto) = \bigcup_{q'_{\sigma'} \in \textbf{Reformulate}(q, \mathcal{S})} \tilde{q}'_{\sigma'}(D).$$

As stated in Section 2.5, query evaluation treats the blank nodes in a query as non-distinguished variables. That is, one could consider equivalently queries with blank nodes or queries with non-distinguished variables. Therefore, the assumption is made by the authors [4] *without loss of generality.*

Observe that query reformulation w.r.t. $\mathcal{S}$ and non-standard evaluation of partially instantiated queries allow computing the exact answer set, *without* saturating the dataset.

Example 13 (**Continued**). *Now, consider again the query $q(x, y)$:- $x\ \tau\ y$ and the dataset D presented in Figure 2.25, both used in the Example 12. The correct and complete answer set of q against the dataset D is:*

$$\bigcup_{q_\sigma \in \mathbf{Reformulate}(q,\mathcal{S})} \tilde{q}_\sigma(D) = \{\langle \mathrm{doi}_0, \_{:}b_0\rangle,\ \langle \mathrm{doi}_1, \mathrm{Person}\rangle,\ \langle \mathrm{doi}_1, \mathrm{Writer}\rangle,\ \langle \mathrm{doi}_0, \mathrm{Writing}\rangle\}$$

with $\mathcal{S}$ the schema of D.

In this thesis, *non-standard query evaluation* is applied when *reformulation* is used, because we allow blank nodes in the database.

2.7.4 Reformulation algorithm

Given a BGP query q and a dataset D whose schema is $\mathcal{S}$, the output of $\mathbf{Reformulate}(q, \mathcal{S})$ is defined [1] as the fix-point $\mathbf{Reformulate}^{\infty}(q, D)$, where:

$$\begin{array}{ll}
\mathbf{Reformulate}^0(q,\mathcal{S}) & = \{q\} \\
\mathbf{Reformulate}^{k+1}(q,\mathcal{S}) & = \mathbf{Reformulate}^k(q,\mathcal{S}) \cup \\
 & \quad \{q''_{\sigma''} \mid \exists r \in [\textit{rules } 2.5, \ldots, 2.17] \text{ s.t. applying } r \text{ on } \mathcal{S} \\
 & \quad\ \text{and some query } q'_{\sigma'} \in \mathbf{Reformulate}^k(q,\mathcal{S}) \\
 & \quad\ \text{yields the query } q''_{\sigma''}\}
\end{array}$$

Observe that the fix point exist and is unique because **Reformulate** is monotone.

The reformulation algorithm uses the set of rules in Figure 2.24 in a backward-chaining fashion [36]. The evaluate and saturate functions used in Algorithm 1 provide, respectively, the standard query evaluation for plain RDF, and the saturation of a dataset w.r.t. an RDF Schema (Figures 2.8 and 2.9).

More precisely, Algorithm 1 uses the original query q and the rules in Figure 2.24 to generate new queries by a backward application of the rules on the atoms of the query. Then, it applies the same procedure on the newly obtained queries and repeats this process until no new queries can be constructed. Finally, the algorithm returns the union of the generated queries (including the original one).
The inner loop of the algorithm (lines 5–17) consist of six *if* statements covering the thirteen rules in Figure 2.24. The conditions of these statements represent the inputs of the rules, whereas the consequents correspond to their outputs. The comment at the end of each *if* statement indicates the subset of rules being applied. In each loop iteration, when a query atom matches the condition of an *if* statement, the respective rule is triggered, replacing the atom with the one that appears in the output of the rule. Thus, when the condition of an *if* statement is satisfied and its body is executed, the rules related to it are applied; an atom of the query is reformulated into another and therefore a new conjunctive query is created. Note that some statements are related with multiple rules, then creating multiple conjunctive queries when the condition of the statement is fulfilled.

Algorithm 1: Reformulate$(q, \mathcal{S})$

```
Input  : an RDF schema $\mathcal{S}$ and a conjunctive query $q$ over $\mathcal{S}$
Output: a union of conjunctive queries $ucq$ such that for any dataset $D$:
        evaluate$(q, \text{saturate}(D, \mathcal{S}))$ = evaluate$(ucq, D)$
1  $ucq \leftarrow \{q\}$, $ucq' \leftarrow \emptyset$
2  while $ucq \neq ucq'$ do
3      $ucq' \leftarrow ucq$
4      foreach conjunctive query $q' \in ucq'$ do
5          foreach atom $g$ in $q'$ do
6              if $g = \langle \mathtt{s}, \tau, c_2 \rangle$ and $c_1 \prec_{\text{sc}} c_2 \in \mathcal{S}$ then        //rule (2.14)
7                  $ucq \leftarrow ucq \cup \{q'_{[g/\langle \mathtt{s}, \tau, c_1 \rangle]}\}$
8              if $g = \langle \mathtt{s}, p_2, \mathtt{o} \rangle$ and $p_1 \prec_{\text{sp}} p_2 \in \mathcal{S}$ then        //rule (2.17)
9                  $ucq \leftarrow ucq \cup \{q'_{[g/\langle \mathtt{s}, p_1, \mathtt{o} \rangle]}\}$
10             if $g = \langle \mathtt{s}, \tau, c \rangle$ and $p \hookleftarrow_{\text{d}} c \in \mathcal{S}$ then        //rule (2.15)
11                 $ucq \leftarrow ucq \cup \{q'_{[g/\exists x\ \langle \mathtt{s}, p, x \rangle]}\}$
12             if $g = \langle \mathtt{o}, \tau, c \rangle$ and $p \hookrightarrow_{\text{r}} c \in \mathcal{S}$ then        //rule (2.16)
13                 $ucq \leftarrow ucq \cup \{q'_{[g/\exists x\ \langle x, p, \mathtt{o} \rangle]}\}$
14             if $g = \langle \mathtt{s}, \tau, x \rangle$ and $c_1, c_2 \ldots, c_n$ are all the classes in $\mathcal{S}$ then
               //rules (2.9)–(2.13)
15                 $ucq \leftarrow ucq \cup \bigcup_{i=1}^{n} \{(q'_{[g/\langle \mathtt{s}, \tau, c_i \rangle]})_{\sigma=[x \rightarrow c_i]}\}$
16             if $g = \langle \mathtt{s}, x, \mathtt{o} \rangle$ and $p_1, p_2 \ldots, p_m$ are all the properties in $\mathcal{S}$ then
               //rules (2.5)–(2.8)
17                 $ucq \leftarrow ucq \cup \bigcup_{i=1}^{m} \{(q'_{[g/\langle \mathtt{s}, p_i, \mathtt{o} \rangle]})_{\sigma=[x \rightarrow p_i]}\} \cup \{(q'_{[g/\langle \mathtt{s}, \tau, \mathtt{o} \rangle]})_{\sigma=[x \rightarrow \tau]}\}$
18 return $ucq$
```

The rules (2.5)–(2.13) requires binding a variable x to a constant (literal or URI) c_i, p_i, or τ; a mapping σ is used then to bind all the occurrences of the variable x in the query. Observe that some distinguished (head) variables of q_σ might be bound as part of this process, resulting in a partially instantiated query.

Theorem 2.7.2 shows that the reformulation algorithm terminates and provides an upper bound for the size of its output. This theorem also exhibits that query reformulation is polynomial in the size of the schema and exponential in the size of the query. The theorem and its corresponding proof was introduce in [37] and then extended in [1] to support a *more expressive* RDF fragment (the DB fragment). Observe that the complexity shown below corresponds to the theorem extension in [1], as we are working with the DB fragment; proof can be found in [1]. Moreover, Proposition 2.7.3, Theorem 2.7.4 and their proofs (which are adapted to the set of rules used in the algorithm) where introduced in previous work [37].

Theorem 2.7.2 (Termination of **Reformulate**$(q, \mathcal{S})$)**.** *[1, 37]*
Given a BGP query q over an RDF schema $\mathcal{S}$, the size (number of queries) of the output of $Reformulate(q, \mathcal{S})$ is in $O((6 * \#\mathcal{S}^2)^n)$, with $\#\mathcal{S}$ and n the sizes (number of triples) of $\mathcal{S}$ and q respectively.

Semantic relationship	RDF Statement	FOL notation
Class assertion	$\texttt{s}\ \tau\ c$	$c(\texttt{s})$
Property assertion	$\texttt{s p o}$	$p(\texttt{s}, \texttt{o}))$

Tab. 2.2: Semantic relationships expressible in RDF statements.

Semantic relationship	RDFS statement	FOL notation
Class inclusion	$c_1 \prec_{sc} c_2$	$\forall x(c_1(x) \Rightarrow c_2(x))$
Property inclusion	$p_1 \prec_{sp} p_2$	$\forall x \forall y(p_1(x,y) \Rightarrow\ p_2(x,y))$
Domain typing of a property	$p \hookleftarrow_d c$	$\forall x \forall y(p(x,y) \Rightarrow c(x))$
Range typing of a property	$p \hookrightarrow_r c$	$\forall x \forall y(p(x,y) \Rightarrow c(y))$

Tab. 2.3: Semantic relationships expressible in an RDF schema.

Proposition 2.7.3 (Complexity). *[37]*
The query reformulation is polynomial in the size of the schema and exponential in the size of the query. The query answering against a dataset D is in LogSpace *in the size of D and exponential in the size of the query.*

Proof. The complexity of the query reformulation follows readily from the proof of Theorem 2.7.2. The complexity of query answering comes from the fact that the language can be reduced to FOL, for which the given results have been proved [38]. □

Theorem 2.7.4 (Correctness of Algorithm 1). *[37]*
Let ucq be the output of **Reformulate**$(q, \mathcal{S})$*, for a given query q over an RDF schema $\mathcal{S}$. For any dataset D associated to $\mathcal{S}$:*

$$\text{evaluate}(q, \text{saturate}(D, \mathcal{S})) = \text{evaluate}(ucq, D)$$

Proof. To prove the correctness of the algorithm, it suffices to show that its *sound* and *complete.*

Soundness In order to proof that **Reformulate** algorithm is *sound* we need to show that for any t:

$$t \in \text{evaluate}(ucq, D) \Rightarrow t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S})) \tag{2.18}$$

Therefore we have two cases:

- $t \notin \text{evaluate}(ucq, D)$: The proof in this case is trivial as per utterance $t \notin \text{evaluate}(ucq, D)$, therefore the antecedent of the conditional 2.18 is $false$ and accordingly the conditional itself is *true*.

- $t \in \text{evaluate}(ucq, D)$: Given that $t \in \text{evaluate}(ucq, D)$, then there's a query $q_i \in ucq$ such as t is an answer to q_i. Moreover, as q_i belongs to ucq, and being ucq the output of our **Reformulate** algorithm we know, by construction, that q_i is subsumed by q w.r.t. $\mathcal{S}$, i.e., $\text{evaluate}(q_i, \text{saturate}(D, \mathcal{S})) \subseteq \text{evaluate}(q, \text{saturate}(D, \mathcal{S}))$. Thus $t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S}))$.

Completeness To prove that Algorithm 1 is complete we have to show that for any t:

$$t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S})) \Rightarrow t \in \text{evaluate}(ucq, D) \tag{2.19}$$

As before, we have two cases. The trivial one in which the falsehood of the antecedent makes the hole conditional *true*, and a second one (and more interesting) in which the antecedent of the conditional, $t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S}))$, is *true*. Let n be the size (number of atoms) of the query q. Since $t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S}))$ is *true*, t results from the projection upon n triples $t_1, \ldots, t_n$. Therefore we have to proof that any $t_i (1 \leq i \leq n)$ is also exhibited by a reformulation of the i-th atom of q. We will prove it by induction on the number of applications of the saturation rules, described in Table 2.3.

Before we proceed with the proof by induction, as stated above, observe that the set of rules, shown in Figure 2.24, covers all possible cases of query atoms. As described in Section 2.5, the atoms in a query are triple patterns of the form $\langle \texttt{s p o} \rangle$, composed by a *subject*, a *property*, and a *object* respectively. Subjects and properties can be URIs, or variables, whereas objects can also be literals.

- When a triple pattern, that belongs to the query, is of the form $\texttt{s}\ x\ \texttt{o}$, i.e., having a variable in the property position, rules (2.5)–(2.8) are triggered.
- If the atom is of the form $\texttt{s p o}$, i.e., having a URI specified in the property position, rules (2.17) or (2.9)–(2.13) can apply, depending on whether the value of the $\texttt{p}$ is the built-in property τ or not, respectively.
- When the query contains a triple pattern of the form $\texttt{s}\ \tau\ \mathbf{c}$, i.e., having the built-in property τ in the property position and a constant (literal or URI) in the object position, the rules (2.14), (2.15) and (2.16) can be applied.

Thus, all possible cases of triple patterns belonging to a query are being treated.

We proceed now to prove the proposition 2.19 by induction on the number of applications of the saturation rules. Let α be the number of applications of the saturation rules (shown in Table 2.3) applied in a forward-chaining fashion [2], needed for the triple t_i to be added in saturate$(D, \mathcal{S})$, i.e., $t_i \in \text{saturate}^{\alpha}(D, \mathcal{S}) \land \forall_{\beta=0}^{\alpha-1} t_i \notin \text{saturate}^{\alpha}(D, \mathcal{S})$.

First, we will show that the free variables of the i-th atom of q that are bounded by t_i, are equally bounded by the evaluation of a reformulation of the i-th atom of q.

Basis We are going to show now that the statement 2.19 holds for $\alpha = 0$.

- If $t_i \notin D$ (i.e., $t_i \notin \text{saturate}^0(D, \mathcal{S})$), then the antecedent of the conditional if $false$ and therefore the statement as a hole is *true*.
- If $t_i \in D$ (i.e., t_i is an explicit triple), therefore t_i is also a triple for the evaluation of the non-reformulated i-th atom of q.

Inductive step Assuming that the conditional 2.19 holds for $\alpha < k$, we will show it also holds for $\alpha = k$.

Let assume t_i is finally added to saturate$(D, \mathcal{S})$ after the application of the first closure rule on a triple $t_{\alpha-1}$. Then, $t_{\alpha-1}$ and t_i are triples of the form $\langle \texttt{s}_1\ \tau\ \mathbf{c_1} \rangle$ and $\langle \texttt{s}_1\ \tau\ \mathbf{c_2} \rangle$, respectively, where $\texttt{s}_1$ is a URI, whereas $\mathbf{c_1}$ and $\mathbf{c_2}$ are URIs or literals. Using the induction hypothesis $t \in \text{evaluate}(q, \text{saturate}(D, \mathcal{S}))$, and thus t_i matches the i-th atom of q. Therefore t_i is of the form: $\langle \texttt{s}\ \tau\ \mathbf{c_2} \rangle$, $\langle \texttt{s}\ x\ \mathbf{c_2} \rangle$, $\langle \texttt{s}\ \tau\ x \rangle$ or $\langle \texttt{s}\ x\ y \rangle$.

In the first case, we perform a reformulation of the i-th atom using the rule (2.14) and we obtain the atom $\langle \mathtt{s}\ \tau\ \mathbf{c_1} \rangle$, which indeed returns $\mathtt{s}_1$ in the result, as if the triple $t_i \in D$ (i.e., t_i is stored in the dataset D). In the second case, we reformulate the query atom with rules (2.5)–(2.8) and obtain (among others) the atom $\langle \mathtt{s}\ \tau\ \mathbf{c_2} \rangle$, which, after one more reformulation using rule (2.14), results in the atom $\langle \mathtt{s}\ \tau\ \mathbf{c_1} \rangle$ that was treated by the first case. In the third case, we apply the rules (2.9)–(2.13) and obtain an atom of the form $\langle \mathtt{s}\ \tau\ \mathbf{c_1} \rangle$. Therefore we also return $\mathtt{s}_1$ in the result. Finally, in the last case we apply rules (2.5)–(2.8), and on the new atom $\langle \mathtt{s}\ \tau\ y \rangle$ we apply rules (2.9)–(2.13), and then fall into the previous case.

Thus, for all four cases that can appear after applying the first saturation rule, we have proved by induction that the conditional 2.19 holds. The rest of the proof is omitted as it amounts to show, similarly as above, that the claim also holds for the three other saturation rules. $\square$

Example 14 (Query reformulation). *Consider the DBLP dataset [35] and the query $q(x, y)$:- $x\ \tau\ y$ asking for all resources and the classes to which they belong. Figure 2.27 shows the reformulation of q w.r.t. the dataset schema, using the* **Reformulate** *algorithm.*

$$\begin{aligned}
&\textbf{Reformulate}^0(q,D) = \{q(x,y)\text{:- } x\ \tau\ y\}\\
&\textbf{Reformulate}^1(q,D) = \textbf{Reformulate}^0(q,D) \cup\\
&\ q(x, xs{:}int)\text{:- } x\ \tau\ xs{:}int,\ q(x, dblp{:}Www)\text{:- } x\ \tau\ dblp{:}Www,\\
&\ q(x, dblp{:}Collection)\text{:- } x\ \tau\ dblp{:}Collection,\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Document,\\
&\ q(x, xs{:}string)\text{:- } x\ \tau\ xs{:}string,\ q(x, dblp{:}Series)\text{:- } x\ \tau\ dblp{:}Series,\\
&\ q(x, dblp{:}Book)\text{:- } x\ \tau\ dblp{:}Book,\ q(x, dblp{:}Mastersthesis)\text{:- } x\ \tau\ dblp{:}Mastersthesis,\\
&\ q(x, dblp{:}Publisher)\text{:- } x\ \tau\ dblp{:}Publisher,\ q(x, dblp{:}Phdthesis)\text{:- } x\ \tau\ dblp{:}Phdthesis,\\
&\ q(x, dblp{:}Inproceedings)\text{:- } x\ \tau\ dblp{:}Inproceedings,\\
&\ q(x, dblp{:}Proceedings)\text{:- } x\ \tau\ dblp{:}Proceedings,\ q(x, rdfs{:}Thing)\text{:- } x\ \tau\ rdfs{:}Thing,\\
&\ q(x, dblp{:}Article)\text{:- } x\ \tau\ dblp{:}Article,\ q(x, dblp{:}Citation)\text{:- } x\ \tau\ dblp{:}Citation\}\\
&\textbf{Reformulate}^2(q,D) = \textbf{Reformulate}^1(q,D) \cup\\
&\ q(x, xs{:}int)\text{:- } v\ dblp{:}volume\ x,\ q(x, xs{:}int)\text{:- } v\ dblp{:}year\ x,\\
&\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Proceedings,\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Article,\\
&\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Www,\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Collection,\\
&\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Series,\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Book,\\
&\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Phdthesis,\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Inproceedings,\\
&\ q(x, dblp{:}Document)\text{:- } x\ \tau\ dblp{:}Mastersthesis,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}crossref\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}objectField\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}datatypeField\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}cite\ v,\ q(x, dblp{:}Document)\text{:- } v\ dblp{:}crossref\ x,\\
&\ q(x, dblp{:}Document)\text{:- } v\ dblp{:}cite\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}datatypeField\ x,\\
&\ q(x, rdfs{:}Thing)\text{:- } v\ dblp{:}ee\ x\}\\
&\textbf{Reformulate}^3(q,D) = \textbf{Reformulate}^2(q,D) \cup\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}url\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}ee\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}isbn\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}year\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}month\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}number\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}series\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}editor\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}address\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}volume\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}title\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}journal\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}chapter\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}school\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}cdrom\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}booktitle\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}author\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}publisher\ v,\\
&\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}note\ v,\ q(x, dblp{:}Document)\text{:- } x\ dblp{:}pages\ v,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}isbn\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}year\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}month\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}number\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}series\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}editor\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}address\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}volume\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}title\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}journal\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}chapter\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}school\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}cdrom\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}booktitle\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}author\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}publisher\ x,\\
&\ q(x, xs{:}string)\text{:- } v\ dblp{:}note\ x,\ q(x, xs{:}string)\text{:- } v\ dblp{:}pages\ x\}\\
&\textbf{Reformulate}^4(q,D) = \textbf{Reformulate}^3(q,D)
\end{aligned}$$

Fig. 2.27: Sample reformulation of a query q w.r.t. the DBLP schema [35].

3. DEVELOPMENT

3.1 Problem statement

As large join queries are an inherent characteristic of searching RDF data [13], reformulated queries may be syntactically very large unions of conjunctive queries [1]. This makes query evaluation very inefficient.
The size of the union of conjunctive queries, for a given query q, depends on the number of its reformulations $|q|_r$. By minimizing the number of reformulations of the queries we hand to the database engine for evaluation, we aim to increase the efficiency of the overall evaluation process.

Query evaluation method Given a query q, *a method for evaluating it* consists of:

1. finding a set of *subqueries* (or *query clusters*), each of which will be reformulated by existing algorithms, into a union of conjunctive queries (or UCQ, in short). On performance reasons, we consider that each subquery should be Cartesian-product free, that is, the query graph of each subquery (cluster) must be connected;

2. combining the results of the cluster UCQs thus obtained, through join operations, in order to obtain query results.

Clearly, there are many evaluation methods for a given query q:

- there are many ways in which a query could be decomposed into clusters;

- the SQL language provides many ways of writing "recombination" queries (that put together the results of the clusters, in order to build the global query results);

- finally, the RDBMS in charge of evaluating all the SQL statements issued in order to answer the original query, may have different alternatives of executing them.

Given that we aim at building an efficient processor of reformulated queries by relying on the functionality of an RDBMS (as opposed to altering these functionalities), in this work we explore the freedom degrees mentioned in the points 1. and 2. above and are not concerned with the internals of the RDBMS optimization and query evaluation. An advantage of this perspective is that our techniques can be applied on top of any RDBMS.

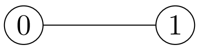
Goal The goal of this work is to investigate algorithms for choosing an efficient evaluation method for a given query q, exploiting: (*i*) clustering alternatives, and (*ii*) SQL formulation alternatives.

Execution model Given an evaluation method, *executing the query according to this method* means:

1. Let $c_1, c_2, \ldots, c_k$ be the clusters selected out of q by the given evaluation method.
2. Let c_i^r be the reformulation of c_i, $1 \leq i \leq k$. (Each c_i is reformulated using the **Reformulate** algorithm presented in Section 2.7.4).
3. An SQL *main query*, in which $c_1^r, c_2^r, \ldots, c_k^r$ are *subqueries* and the joins across them are the conditions, is sent to the RDBMS.

Example 15 (Evaluation alternatives for reformulated queries). *Consider the following query against the real-life DBLP dataset [35]:*

$$\begin{aligned} q(x) \quad &\text{:- } x\ \tau\ dblp{:}Document, && (0)\\ & x\ dblp{:}objectField\ http{:}//www.example.org/dblp/ && (1)\end{aligned}$$



Triple patterns	*#Reformulations*
$\{0\}$	36
$\{1\}$	4
$\{0,1\}$	144

Tab. 3.1: Query q and reformulations characteristics.

The given query q is composed by two atoms, namely (0) *and* (1)*, having 36 and 4 reformulations respectively. Hence, the query has 144 reformulations. Figures 3.1 and 3.2 presents the reformulations of the atoms of q, while Figure 3.3 shows the reformulations of the query.*

The SQL translation of atom (1)*, found in Listing 3.2, shows each of the four reformulations (of q_1) mapped into a conjunctive query, and SQL UNION operators combining the result sets.*

Listing 3.1 presents the SQL translation of atom (0) *while Listing 3.2 presents the SQL translation of atom* (1)*. Finally, Figure 3.3 present q as a "main query" joining the SQL translations of its atoms. Observe that the* main query *joins the SQL translations of the reformulations of the atoms of q, the* subqueries *with names q_0 and q_1 that corresponds to the atoms* (0) *and* (1) *respectively. The join condition between the subqueries is given by the variable in common between the atoms they represent.*

For what concerns the SQL translation of the reformulated q as a whole, it is a union of 144 *conjunctive queries, shown in Listing 6.1 which is delegated in Section 6 of the Appendix for readability. To illustrate the present discussion, Listing 3.4 depicts the first 5 of these 144 conjunctive queries. Both the full Listing 6.1 and its reduced version 3.4 illustrate the fact that when translated into a single SQL statement, the SQL query is quite large (syntactically) and contains many repeated sub-expressions, which in practice led to a significant increase in query evaluation time [1].*

{$q_0(x)$:- x τ dblp:Document, $q_0(x)$:- x τ dblp:Proceedings, $q_0(x)$:- x τ dblp:Article,
$q_0(x)$:- x τ dblp:Www, $q_0(x)$:- x τ dblp:Collection, $q_0(x)$:- x τ dblp:Series,
$q_0(x)$:- x τ dblp:Book, $q_0(x)$:- x τ dblp:Phdthesis, $q_0(x)$:- x τ dblp:Inproceedings,
$q_0(x)$:- x τ dblp:Mastersthesis, $q_0(x)$:- x dblp:crossref v, $q_0(x)$:- x dblp:objectField v,
$q_0(x)$:- x dblp:datatypeField v, $q_0(x)$:- x dblp:cite v, $q_0(x)$:- v dblp:crossref x,
$q_0(x)$:- v dblp:cite x, $q_0(x)$:- x dblp:url v, $q_0(x)$:- x dblp:ee v,
$q_0(x)$:- x dblp:isbn v, $q_0(x)$:- x dblp:year v, $q_0(x)$:- x dblp:month v,
$q_0(x)$:- x dblp:number v, $q_0(x)$:- x dblp:series v, $q_0(x)$:- x dblp:editor v,
$q_0(x)$:- x dblp:address v, $q_0(x)$:- x dblp:volume v, $q_0(x)$:- x dblp:title v,
$q_0(x)$:- x dblp:journal v, $q_0(x)$:- x dblp:chapter v, $q_0(x)$:- x dblp:school v,
$q_0(x)$:- x dblp:cdrom v, $q_0(x)$:- x dblp:booktitle v, $q_0(x)$:- x dblp:author v,
$q_0(x)$:- x dblp:publisher v, $q_0(x)$:- x dblp:note v, $q_0(x)$:- x dblp:pages v}

Fig. 3.1: Reformulations of the atom (0) w.r.t. DBLP [35].

{$q_1(x)$:- x dblp:objectField http://www.example.org/dblp/,
$q_1(x)$:- x dblp:url http://www.example.org/dblp/,
$q_1(x)$:- x dblp:ee http://www.example.org/dblp/,
$q_1(x)$:- x dblp:cite http://www.example.org/dblp/}

Fig. 3.2: Reformulations of the atom (1) w.r.t. DBLP [35].

{$q(x)$:- x τ dblp:Document, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Proceedings, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Article, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Www, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Collection, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Series, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Book, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Phdthesis, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Inproceedings, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Mastersthesis, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:crossref v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:objectField v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:datatypeField v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:cite v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- v dblp:crossref x, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- v dblp:cite x, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Document, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Document, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Document, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Proceedings, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Proceedings, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Proceedings, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Article, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Article, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Article, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Www, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Www, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Www, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Collection, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Collection, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Collection, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Series, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Series, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Series, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Book, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Book, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Book, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Phdthesis, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Phdthesis, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Phdthesis, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Inproceedings, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Inproceedings, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Inproceedings, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Mastersthesis, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Mastersthesis, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x τ dblp:Mastersthesis, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:crossref v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:crossref v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:crossref v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:url v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:ee v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:objectField v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:objectField v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:objectField v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:isbn v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:year v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:month v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:number v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:series v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:editor v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:address v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:volume v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:title v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:journal v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:chapter v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:school v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:cdrom v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:booktitle v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:author v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:publisher v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:note v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:pages v, x dblp:objectField http://www.example.org/dblp/,
$q(x)$:- x dblp:datatypeField v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:datatypeField v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:datatypeField v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:cite v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:cite v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:cite v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- v dblp:crossref x, x dblp:url http://www.example.org/dblp/,
$q(x)$:- v dblp:crossref x, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- v dblp:crossref x, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- v dblp:cite x, x dblp:url http://www.example.org/dblp/,
$q(x)$:- v dblp:cite x, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- v dblp:cite x, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:url v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:url v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:url v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:ee v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:ee v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:ee v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:isbn v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:isbn v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:isbn v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:year v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:year v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:year v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:month v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:month v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:month v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:number v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:number v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:number v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:series v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:series v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:series v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:editor v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:editor v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:editor v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:address v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:address v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:address v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:volume v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:volume v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:volume v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:title v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:title v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:title v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:journal v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:journal v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:journal v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:chapter v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:chapter v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:chapter v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:school v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:school v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:school v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:cdrom v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:cdrom v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:cdrom v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:booktitle v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:booktitle v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:booktitle v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:author v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:author v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:author v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:publisher v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:publisher v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:publisher v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:note v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:note v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:note v, x dblp:cite http://www.example.org/dblp/,
$q(x)$:- x dblp:pages v, x dblp:url http://www.example.org/dblp/,
$q(x)$:- x dblp:pages v, x dblp:ee http://www.example.org/dblp/,
$q(x)$:- x dblp:pages v, x dblp:cite http://www.example.org/dblp/}

Fig. 3.3: Reformulations of the query q w.r.t. DBLP [35].

Listing 3.1: SQL translation of the reformulation of atom (0) w.r.t. DBLP [35].

```
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                               AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
UNION
SELECT DISTINCT p0.o AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
UNION
SELECT DISTINCT p0.o AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
```

Listing 3.2: SQL translation of the reformulation of atom (1) w.r.t. DBLP [35].

```
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                   AND p0.o='http://www.example.org/dblp/'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                   AND p0.o='http://www.example.org/dblp/'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                   AND p0.o='http://www.example.org/dblp/'
UNION
SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                   AND p0.o='http://www.example.org/dblp/'
```

Listing 3.3: SQL translation with subqueries of the reformulation of query q w.r.t. DBLP [35].

```
SELECT DISTINCT temp_0.x AS x FROM
(
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                   AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
        UNION
        SELECT DISTINCT p0.o AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
        UNION
        SELECT DISTINCT p0.o AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
        UNION
        SELECT DISTINCT p0.s AS x FROM triples AS p0 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
) AS temp_0,
(
        SELECT DISTINCT p1.s AS x FROM triples AS p1 WHERE p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                   AND p1.o='http://www.example.org/dblp/'
        UNION
        SELECT DISTINCT p1.s AS x FROM triples AS p1 WHERE p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                   AND p1.o='http://www.example.org/dblp/'
        UNION
        SELECT DISTINCT p1.s AS x FROM triples AS p1 WHERE p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                   AND p1.o='http://www.example.org/dblp/'
        UNION
        SELECT DISTINCT p1.s AS x FROM triples AS p1 WHERE p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                   AND p1.o='http://www.example.org/dblp/'
) AS temp_1
WHERE temp_0.x = temp_1.x
```

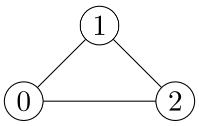
```
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION ...
```

Fig. 3.4: Partial SQL translation of the reformulation of query q w.r.t. DBLP [35] (the full translation can be found in the Appendix).

The clusterization of the original query, q, and the consequent execution as a *main query* joining the SQL translations of the reformulations of the clusters (of atoms) of q, allows us to keep the number of reformulations contained while relying on the selectivity of the atoms to reduce intermediate results.

Example 16 (Clusterization and atoms selectivity). *For example, consider the DBLP dataset [35] and the following query:*

$$\begin{array}{lll} q(x,v) & \text{:-}\ x\ dblp{:}datatypeField\ v, & (0) \\ & \quad x\ purl{:}publisher\ \text{“}Springer\text{''}, & (1) \\ & \quad x\ \tau\ dblp{:}Document & (2) \end{array}$$



Triple pattern	#Reformulations	#Results
0	19	2,629,667
1	1	4,367
2	36	711,174

Tab. 3.2: Query q and reformulations characteristics.

Observe that atom (1) *has a single reformulation and* 4.367 *results (a few given that the DBLP dataset contains* 8,424,216 *tuples). By clustering atom* (1) *with atom* (0) *and executing the corresponding subquery (q_1, illustrated below), the number of reformulations holds (i.e., the number of reformulations of q_1 is the same as that of atom* (0) *alone) while the number of tuples resulting by the evaluation of q_1 (*30,450*) is significantly less.*

Similarly, the clusterization of atom (1) *with atom* (2) *and execution of the corresponding query (q_2, illustrated below) results in* 4,367 *tuples, a significant reduction given that the atom* (2) *alone matches* 711,174 *tuples.*

$$q_1(x, v) \quad \text{:-}\ x\ \mathit{dblp{:}datatypeField}\ v, \quad (0)$$
$$x\ \mathit{purl{:}publisher}\ \text{“}\mathit{Springer}\text{”} \quad (1)$$

$$q_2(x) \quad x\ \mathit{purl{:}publisher}\ \text{“}\mathit{Springer}\text{”}, \quad (1)$$
$$x\ \tau\ \mathit{dblp{:}Document} \quad (2)$$

Moreover, this technique results in a lower number of reformulations and therefore a (syntactically) shorter SQL query (with less complexity), decreasing the required evaluation time. Further, there is a reduction in the number of times a common sub-expression is evaluated, as shown in Example 15.

3.2 Solution overview

We address the described problem of optimizing reformulated RDF queries in a two-step manner:

i) Introduce algorithms to find a clusterization that accelerates the execution for a given BGP query.

ii) Introduce techniques for improving the execution of clustered queries.

First, we introduce in Section 3.4 a novel and practical cost model for BGP queries taking into account both the number of results and the number of reformulations. Second, by using this cost model, we devise efficient heuristic algorithms for BGP queries nodes clusterization (with and without clusters overlapping) found in Section 3.5. Moreover, we use two different clusterization techniques namely *partition* and *fragmentation*.

Given a BGP query q, the partition of q is a division of its atoms into *non-overlapping* and *non-empty* parts that cover all the atoms of q. The fragmentation of the same query q is a division of its atoms into *non-empty* fragments that may have overlapping. Since the efficiency of BGP query evaluation is directly related to both the number of results and the number of reformulations, there are cases in which it becames more efficient to have common atoms (with high selectivity and low reformulations) between the fragments.

Finally, in Section 3.6 we introduce a naïve algorithm and optimization opportunities for clustered query execution (execute each subquery and the join(s) combining the subqueries).

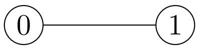
Extensive experiments with large RDF data performed on different RDF stores, found in Chapter 4, confirm the efficiency and effectiveness of our approach over the baseline techniques, presented in previous work [1].

3.3 Eliminate duplicate queries

Given a conjunctive query q over a RDF Schema $\mathcal{S}$ corresponding to a dataset D, different sequences of immediate entailment rules may result in equivalent output queries (queries with the same atoms in different order, variable renaming, etc). Observe that the output of Algorithm 1 is a union of conjunctive queries and hence, having two equivalent queries in the union won't add results (when evaluating the reformulated query) but it will add syntactic complexity and increase evaluation time. Therefore, to increase the efficiency of the query evaluation it is desirable that the output of the **Reformulate** algorithm does not have equivalent queries.

Example 17 (Equivalent queries during reformulation). *Consider the following query w.r.t. the DBLP dataset [35]:*

$$\begin{aligned} q(x) \quad &\text{:-}\ x\ dblp{:}datatypeField\ y, && (0) \\ & x\ \tau\ dblp{:}Document && (1) \end{aligned}$$



Tab. 3.3: Example query q.

The query q is composed of two atoms, namely (0) and (1), each of which have 19 and 36 reformulations respectively. Figures 3.5 and 3.6 presents the reformulations of the atoms of q. The expected number of reformulations of the query q is then 684, being this the multiplication of the number of reformulations of its atoms. However, the size of the output of the Algorithm 1 applied to the query q and the RDF Schema $\mathcal{S}$, corresponding to the DBLP dataset, is 513 (a union of 513 conjunctive queries).

$\{q_0(x)$:- x *dblp:datatypeField* y, $q_0(x)$:- x *dblp:Series* y,
$q_0(x)$:- x *dblp:booktitle* y, $q_0(x)$:- x *dblp:Publisher* y,
$q_0(x)$:- x *dblp:number* y, $q_0(x)$:- x *dblp:year* y,
$q_0(x)$:- x *dblp:note* y, $q_0(x)$:- x *dblp:volume* y,
$q_0(x)$:- x *dblp:title* y, $q_0(x)$:- x *dblp:chapter* y,
$q_0(x)$:- x *dblp:address* y, $q_0(x)$:- x *dblp:pages* y,
$q_0(x)$:- x *dblp:author* y, $q_0(x)$:- x *dblp:school* y,
$q_0(x)$:- x *dblp:cdrom* y, $q_0(x)$:- x *dblp:month* y,
$q_0(x)$:- x *dblp:journal* y, $q_0(x)$:- x *dblp:isbn* y,
$q_0(x)$:- x *dblp:editor* $y\}$

Fig. 3.5: Reformulations of the atom (0) w.r.t. the DBLP dataset [35].

$\{q_1(x)$:- x τ *dblp:Document*, $q_1(x)$:- x τ *dblp:Proceedings*, $q_1(x)$:- x τ *dblp:Article*,
$q_1(x)$:- x τ *dblp:Www*, $q_1(x)$:- x τ *dblp:Collection*, $q_1(x)$:- x τ *dblp:Series*,
$q_1(x)$:- x τ *dblp:Book*, $q_1(x)$:- x τ *dblp:Phdthesis*, $q_1(x)$:- x τ *dblp:Inproceedings*,
$q_1(x)$:- x τ *dblp:Mastersthesis*, $q_1(x)$:- x *dblp:crossref* v, $q_1(x)$:- x *dblp:objectField* v,
$q_1(x)$:- x *dblp:datatypeField* v, $q_1(x)$:- x *dblp:cite* v, $q_1(x)$:- v *dblp:crossref* x,
$q_1(x)$:- v *dblp:cite* x, $q_1(x)$:- x *dblp:url* v, $q_1(x)$:- x *dblp:ee* v,
$q_1(x)$:- x *dblp:isbn* v, $q_1(x)$:- x *dblp:year* v, $q_1(x)$:- x *dblp:month* v,
$q_1(x)$:- x *dblp:number* v, $q_1(x)$:- x *dblp:series* v, $q_1(x)$:- x *dblp:editor* v,
$q_1(x)$:- x *dblp:address* v, $q_1(x)$:- x *dblp:volume* v, $q_1(x)$:- x *dblp:title* v,
$q_1(x)$:- x *dblp:journal* v, $q_1(x)$:- x *dblp:chapter* v, $q_1(x)$:- x *dblp:school* v,
$q_1(x)$:- x *dblp:cdrom* v, $q_1(x)$:- x *dblp:booktitle* v, $q_1(x)$:- x *dblp:author* v,
$q_1(x)$:- x *dblp:publisher* v, $q_1(x)$:- x *dblp:note* v, $q_1(x)$:- x *dblp:pages* $v\}$

Fig. 3.6: Reformulations of the atom (1) w.r.t. the DBLP dataset [35].

The difference is due to the fact that atoms (0) and (1) of q have some reformulations in common (variable renames), and therefore some of the resulting conjunctive queries are equivalent. Here we present a short list including some, but not all, of the pairs of equivalent conjunctive queries resultant form applying the reformulation rules to q w.r.t. $\mathcal{S}$:

- *$q(x)$:- x dblp:address y, x dblp:booktitle y and*
  $q(x)$:- x dblp:booktitle y, x dblp:address y.

- $q(x)$:- x *dblp*:*address* y, x *dblp*:*cdrom* y *and*
  $q(x)$:- x *dblp*:*cdrom* y, x *dblp*:*address* y.

- $q(x)$:- x *dblp*:*author* y, x *dblp*:*cdrom* y *and*
  $q(x)$:- x *dblp*:*cdrom* y, x *dblp*:*author* y.

- $q(x)$:- x *dblp*:*chapter* y, x *dblp*:*isbn* y *and*
  $q(x)$:- x *dblp*:*isbn* y, x *dblp*:*chapter* y.

- $q(x)$:- x *dblp*:*datatypeField* y, x *dblp*:*editor* y *and*
  $q(x)$:- x *dblp*:*editor* y, x *dblp*:*datatypeField* y.

In general, conjunctive queries equivalence is an NP-complete problem [27]. However, in the context of the **Reformulate** algorithm we can take advantage of the facts that:

(*i*) All the conjunctive queries (produced by the algorithm trough the application of the rules) have the same number of atoms and head terms.

(*ii*) The atoms can be safely ordered using a common scheme.

(*iii*) Variables have the same names at each given position.

Therefore, assuming these facts, we are able to avoid equivalent queries in the output of our **Reformulate** algorithm. To do so, we implement a mechanism based on signatures:

- Triple pattern signature w.r.t. the conjunctive query it belongs to.
- Conjunctive query signature, which relies on the signatures of its atoms and the fact that the atoms can be ordered in a canonical way.

The signature of a triple pattern in the conjunctive query is computed by a 11-tuple consisting of:

- Aggregate hash over the literals of the triple pattern.
- Aggregate hash over the URIs of the triple pattern.
- 9 numbers accounting in how many s-s, s-p, s-o, p-s, p-p, p-o, o-s, o-p, o-o joins does each variable of this triple pattern participate.

The signature of a conjunctive query is computed as the hash code of the list of head constants (URIs and literals), the list of head variables, the list of signature of it triples (in order) and the variables in the query.

Moreover, the *SignedConjunctiveQueriesSet* class uses the *ConjunctiveQuerySignature* to answer if a given conjunctive query q is contained in the set or not, and to decide if a conjunctive query should be added to the set when the *add* method is called (or another query $q2$, with the same signature, is already present); the *addAll* method have the same behavior with each of the conjunctive queries contained in the given collection.

Algorithm 2: CQSignature(q)

```
Input  : a conjunctive query $q$
Output: an int that represents the given query signature
1  $triplePatternNames \leftarrow [], triplePatternSignatures \leftarrow []$
2  foreach $p \in q.getTriplePatterns()$ do
3      $triplePatternNames.add(p.name)$
4      $triplePatternSignatures.put(p.name,$ TPSignature$(p, q))$
5  $headConstants \leftarrow []$
6  foreach $term \in q.getHead()$ do
7      if $term.isConstant()$ then
8          $headConstants.add(term)$
9  $variablesClusters = VariablesClusters(q.getTriplePatterns())$
10 $headVariables \leftarrow []$
11 foreach $term \in q.getHead()$ do
12     if $term.isVariable()$ then
13         $headVariables.add(variablesClusters.getClusterIndex(term))$
14 Canonize($headConstants, headVariables, triplePatternNames,$
   $triplePatternSignatures, variablesClusters$)
15 return
   HashCode($headConstants, headVariables, triplePatternSignatures, variablesClusters$)
```

Algorithm 2 begins by initializing two variables used to store the query triple patterns names and their signatures, as empty lists (line 1), and proceeds to populate the lists, respectively, with the name of the atoms of the query (maintaining the original order) and the signatures of the atoms. The signature of a triple pattern w.r.t. the query to which it belongs is a hash code built from:

- The constants (URIs and literals) appearing in the triple pattern.
- The joins among terms of the triple pattern.
- The joins between terms of the given triple pattern and other atoms of the query.

Second, the head constants (URIs and literals) are stored in a list (lines 5–8). Then, in line 9, the variables appearing in the atoms of the query are clustered. We call cluster here a *class of equivalence* for the variables, which allow us to compare the variables in a query by their role. Variables are now seen not as a name nor a symbol but as the positions they have in the triple patterns of the query and the joins they generate, within a single or different atoms. Lines 10–13 stores in another variable a list with the variables appearing in the head of the query. Observe that two queries that differ only in the names of the variables will result in the same list of head variables, as the list includes not the variables but their clusters.

Using the variables previously created, the query is canonized by the Canonize algorithm and the hash code of the canonized query is finally returned. The Canonize algorithm re-orders the atoms of the query by their signature, which involves updating the reference to the atoms (positions in the query) and their terms in the variables stored in memory.

Example 18 (BGP query signature). *For example, consider the queries q and q' shown below:*

$$
\begin{array}{lll|lll}
q(x) & \text{:- } x\ \tau\ dblp{:}Book, & (0) & q'(y) & \text{:- } y\ purl{:}publisher\ \text{“}Springer\text{''}, & (0) \\
 & x\ purl{:}publisher\ \text{“}Springer\text{''} & (1) & & y\ \tau\ dblp{:}Book & (1)
\end{array}
$$

Observe that the head and atoms of q and q' are equal (up to variables rename and query atoms order).

(a) Using query q as input for Algorithm 2, first the following variables are set:

- $triplePatternNames \leftarrow [atom_0, atom_1]$;
- $triplePatternSignatures \leftarrow \{atom_0 : [379432498, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, $atom_1 : [-356503219, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]\}$;
- $headConstants \leftarrow []$;
- $variablesClusters \leftarrow [0, 3]$;
- $headVariables \leftarrow [0]$.

Then, the canonize algorithm is invoked with these variables, and proceeds as follows:

(a) Re-order the atoms of the query by their signature. As atom (1) signature is less that the signature of atom (0) (according to the compareTo method), atom (0) and atom (1) exchange positions in the query.

$$
\begin{array}{lll}
q(x) & \text{:- } x\ purl{:}publisher\ \text{“}Springer\text{''} & (1) \\
 & x\ \tau\ dblp{:}Book, & (0)
\end{array}
$$

(b) The references to the atoms and their terms (positions) are updated in the variables. In this case, the only variable that change is $triplePatternNames$: $triplePatternNames \leftarrow [atom_1, atom_0]$.

Finally, 1400082051 *is the returned signature for q.*

(b) Using query q as input for Algorithm 2, first the following variables are set:

- $triplePatternNames \leftarrow [atom_0, atom_1]$;
- $triplePatternSignatures \leftarrow \{atom_0 : [379432498, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, $atom_1 : [-356503219, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]\}$;
- $headConstants \leftarrow []$;
- $variablesClusters \leftarrow [0, 3]$;
- $headVariables \leftarrow [0]$.

*Observe that the signatures of the triple patterns are equal (as expected). Therefore when the canonize algorithm is invoked (with the variables above as arguments), no re-order of the atoms is needed and thus no update of the variables either. Moreover, the signature of q' is equal to the signature of q (*1400082051*) as the variables have the same values.*

The conjunctive query technique, as it is, in some particular cases can lead to false negatives. As shown in Example 19, there are some cases in which two equivalent queries have different signatures. Observe that false negatives do increase the size of the union of conjunctive queries but do not lead to wrong answers, given that the queries are equivalent. Moreover, the cases on which false negatives were detected are very particular.

Example 19 (False negative). *Consider the query shown below:*

$$\begin{array}{rll} q(w1, w2, w3, w4) & \text{:- } x\ y\ w1 & (0) \\ & z\ y\ w2, & (1) \\ & x\ t\ w3, & (2) \\ & z\ t\ w4, & (3) \end{array}$$

Here all the triples have the signature [1408, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]. Observe that ordering the triple patterns of the query in another way:

$$\begin{array}{rll} q(w1, w2, w3, w4) & \text{:- } x\ y\ w1 & (0) \\ & x\ t\ w3, & (2) \\ & z\ y\ w2, & (1) \\ & z\ t\ w4, & (3) \end{array}$$

leads to a different signature because the variables clusters are different, beyond that the queries are equivalent.

3.4 Cost model

The design of our cost model is motivated by the significant impact of the number of reformulations on the performance of execution of reformulated BGP queries. Given a RDF data set, a *triple pattern* t_i has an estimated number of tuples $|t_i|_t$, for the data set, that matches it, and an exact number of reformulations $|t_i|_r$. Moreover, given a BGP query $q(\bar{x})$:- $t_1, \ldots, t_n$, we denote by $|q|_t$ the estimated number of tuples matching it, and by $|q|_r$ its number of reformulations.

For the purposes of the cost model we assume that:

- The join cost is linear in the size of the inputs. Moreover, the cost a n-way join, $t_1 \bowtie t_2 \bowtie \cdots \bowtie t_n$, is linear in the size of the inputs ($\sum_{i=1}^{n} |t_i|_t$).
- The cost of materialization is $c_1 * |t|_t$, with c_1 some constant and t a triple pattern.
- RDBMS will select for pipelining (not materialize) the subquery with largest result.

Given a triple pattern t_i, we estimate its cost w.r.t. a RDF dataset as:

$$Cost(t_i) = c_0 + \alpha|t_i|_t + \beta|t_i|_r$$

where c_0 is the fixed overhead of connecting to SQL, and α and β are ponder factors for the number of results and number of reformulations respectively.

Given a BGP query $q(\bar{x})$:- $t_1, \ldots, t_n$, the cost of q w.r.t. a RDF dataset is estimated as:

$$Cost(q) = c_0 + \alpha| \bowtie_{k=1}^{n} t_i|_t + \beta| \bowtie_{k=1}^{n} t_i|_r + \sum_{k=1}^{n} |t_i|_t$$

Given an RDF graph G and a clusterization G' of G, the cost of evaluating G, through the clusterization G', can be calculated as the sum of the costs of evaluating each part (subquery), $g \in G'$ plus the sum of the costs of joining the results of the parts, plus the cost of materialize $n-1$ subqueries:

$$Cost(G') = \sum_{g \in G'} Cost(g) + \sum_{g \in G'} |g|_t + \sum_{g \in G' \setminus g_m} MaterializeCost(g)$$

where g_m is the subquery which is picked for pipelining. We make the assumption that this sub-query is the one having the largest estimated number of results.

3.4.1 Cardinality estimation

Notations For a triple atom t:

- $dist(X,t)$ is the distinct number of values of the variable X appearing in t;
- $min(X,t)$ is the minimum value of X occurring in t;
- $max(X,t)$ is the maximum value of X occurring in t;
- $|t|$ is the estimated number of tuples in t.

Assumptions For a triple atom t and variables X, Y of t, we make a set of assumptions:

Uniform distribution in the interval We assume that the values of X are uniformly distributed across the interval. Formally, this is:

$$P(X < x_o) = (x_0 - min(X,t))/(max(X,t) - min(X,t))$$

where $P(\alpha)$ is the probability that an event α happens. As customary, the probability of the events we consider is defined as the number of cases where the event happens, divided by the total number of possible cases, $0 \leq P(\alpha) \leq 1$.

Uniform distribution across distinct values We assume that the values of X are uniformly distributed across the $dist(X,t)$ distinct values, i.e.:

$$\forall x_1, x_2 \text{ such that } x_1 \in \pi_X(t), x_2 \in \pi_X(t),\ P(X = x_1) = P(X = x_2)$$

In other words, the probability that X takes a given value does not depend on that value (of course, as long as the value does appear in $t.X$).

As a consequence, and given that there are $dist(X,t)$ distinct values, for any value $x_1 \in \pi_X(t)$, we have $P(X = x_1) = 1/dist(X,t)$. Equivalently, for any value $x_1 \in \pi_X(t)$, there are $|t|/dist(X,t)$ tuples satisfying $X = x_1$.

Uniform distribution of distinct values across the interval For two values $x_1, x_2 \in \pi_X(t)$ that are consecutive (i.e., such that there does not exist an $x_3 \in \pi_X(t)$ between x_1 and x_2), $x_2 - x_1 = (max(X,t) - min(X,t))/dist(X,t)$.

This assumption is a consequence of the two previous ones. It is proved as follows. Let $\epsilon > 0$ be a small constant. Then,

$P((x_1 - \epsilon) \leq X \leq (x_2 - \epsilon)) = (x_2 - \epsilon - min(X,t) - x_1 + \epsilon + min(X,t))/(max(X,t) - min(X,t)) =$

$(x_2 - x_1)/(max(X,t) - min(X,t))$.

Thus, the probability of encountering an X value between $x_1 - \epsilon$ and $x_2 - \epsilon$ is $(x_2 - x_1)/(max(X,t) - min(X,t))$. But in that interval, the only possible value of X is x_1, and we know that the probability $P(X = x_1) = 1/dist(X,t)$.

This leads to:

$$(x_2 - x_1)/(max(X,t) - min(X,t)) = 1/dist(X,t)$$

or, equivalently:

$$(x_2 - x_1) = (max(X,t) - min(X,t))/dist(X,t)$$

Independent distributions Let X, Y be two variables in t. Then, the distributions of X and Y are independent, i.e.:

$$P(X = x_0 | Y = y_0) = P(X = x_0)$$

where $P(X = x_0 | Y = y_0)$ is, as usual, the probability that $X = x_0$ once we know that $Y = y_0$. The assumption states that knowing the value of Y does not inform us on the value of X.

Moreover, we also make an assumption when joining, say, expression e_1 with e_2 on $e_1.X = e_2.Y$.

Facing values Assume that in a given interval $[min, max]$, $e_1.X$ is determined to take n_1 distinct values, and $e_2.Y$ is determined to take n_2 distinct values, with $n_1 < n_2$. Then, we consider that the distinct values of $e_1.X$ are all among the distinct values of $e_2.Y$.

This assumption is optimistic as it assumes that relations have facing (corresponding) values in the columns on which they are joined.

Estimating the parameters characterizing a join result Let t_1, t_2 be two triples which join on $t_1.X = t_2.Y$. We estimate the number of tuples in the expression $e = t_1 \bowtie_{X=Y} t_2$, as well as the other interesting parameters of e, as follows:

1. Compute $[min_{\bowtie}, max_{\bowtie}]$ which is the intersection of the intervals

$$[min(X, t_1), max(X, t_1)] \text{ and } [min(Y, t_2), max(Y, t_2)]$$

2. Compute the following ratio:

$$k_1 = (max_{\bowtie} - min_{\bowtie})/(max(X, t_1) - min(X, t_1))$$

It is the ratio between the number of t_1 tuples which have matches in t_2, and the number of all t_1 tuples. Due to our assumptions, it is also the ratio between the number of $t_1.X$ distinct values which have matches in t_2, and the total number of distinct values in $t_1.X$.

3. Similarly, compute the ratio:

$$k_2 = (max_{\bowtie} - min_{\bowtie})/(max(X, t_2) - min(X, t_2))$$

It is the ratio between the number of t_2 tuples which have matches in t_1, and the number of all t_2 tuples. It is also the ratio between the number of $t_2.Y$ distinct values which have matches in t_1, and the total number of $t_2.Y$ distinct values.

4. How many distinct values there will be in the result? By the “facing value” hypothesis (which assumes that the values coincide “as much as possible”), we get:

$$k = min((dist(X, t_1) * k_1), (dist(Y, t_2) * k_2))$$

5. How many distinct tuples there will be in the result? For each of the k distinct values of the join attribute, which correspond to matches between t_1 and t_2:

   - t_1 had $|t_1|/dist(X, t_1)$ tuples
   - t_2 had $|t_2|/dist(Y, t_2)$ tuples

   Therefore, there will be

$$k * |t_1| * |t_2|/(dist(X, t_1) * dist(Y, t_2))$$

   tuples.

In the large The previous section provided formulas for capturing the number of tuples and distinct values in the result of one join over two query atoms. This generalizes as follows:

- It may occur that the cardinality of joins computed at a given point can be smaller than the number of distinct values on the column candidate for the next join. In such a case, we simply replace the number of distinct values on the column by the previous join's cardinality.
  For example, let 3 atoms be t_1, t_2 and t_3, variable X belongs to t_1 and t_2 and Y belongs to t_2 and t_3, then $dist(Y, t_2) := min(|t_1 \bowtie t_2|, dist(Y, t_2))$
- To extend to n atoms connected by $n-1$ joins, apply this formula $n-1$ times (of course, each time on the result of the previous application).
- To extend to n atoms connected by more than $n-1$ joins:
  - pick $n-1$ joins that connect the n atoms
  - do as above (n atoms with $n-1$ predicates connecting them)
  - then, for each remaining predicate of the form $Z = U$:
    * multiply the cardinality of the expression by $P(Z = U)$, which can be derived from the numbers of distinct values of Z, respectively, U in that expression.
    * update the number of distinct values in Z (respectively U), i.e., the one with more distinct values will take the others number of distinct values.

3.4.2 Reformulations estimation

Theorem 2.7.2 shows that given a BGP query q and a dataset D whose RDF schema is $\mathcal{S}$, the size (number of queries) of the output of $Reformulate(q, \mathcal{S})$ is in $O((6 * \#\mathcal{S}^2)^n)$, with $\#\mathcal{S}$ and n the sizes (number of triples) of $\mathcal{S}$ and q respectively. However, in practice, the size of a reformulated query is much smaller than the theoretical upper bound [1]. Therefore, the only way to get the exact number of reformulations of a given query q w.r.t. a RDF schema $\mathcal{S}$ is to calculate $Reformulate(q, \mathcal{S})$ and retrieve the size of the output.

Given a BGP query $q(\bar{x})$:- $t_1, \ldots, t_n$ and a RDF schema, there's a relationship between the number of reformulations of the query, $|q|_r$, and the number of reformulations of the atoms of q: $|q|_r \leq \prod_{i=1}^{n} |t_i|_r$, i.e., the number of reformulations of the query has an upper bound given by the multiplication of the number of reformulations of its atoms. Moreover, the multiplication of the number of reformulations of the atoms of $q(\bar{x})$ is also a good estimator of $|q|_r$ in general.

Our $Reformulate$ algorithm avoids equivalent queries in the output. Therefore, in some cases $|q|_r$ might be smaller than $\prod_{i=1}^{n} |t_i|_r$; however since this is not the case in the majority of the queries and, to avoid the execution of a fix point algorithm several times for multiple queries containing a subset of a shared set of triples during the clusterization algorithms presented in Section 3.5, we estimate $|q|_r$ as $\prod_{i=1}^{n} |t_i|_r$.

3.5 BGP query clusterization

Divide and conquer.

Query reformulation is an established technique for handling RDF entailments and had been subject of recent works [1, 37]. However, increases in the size of the reformulated query usually lead to an increase in the execution time (of the reformulated query). Therefore, this technique is not suitable for cases in which the size of the reformulated query is very large [1]. Our approach aims to increase the efficiency of the query evaluation by holding the number of reformulations bounded (when possible) w.r.t. a given threshold (over the number of reformulations). The parametrization of the threshold makes it easy to adapt, as the threshold may vary within different RDBMS engines.

Recall from Section 3.2, to do this we use BGP query clusterization. Moreover, we use two different clusterization techniques namely *partition* and *fragmentation*. This section presents our approach to find a clusterization given a dataset, a BGP query and a threshold over the number of reformulations, using as basis our cost model.

We describe two different clusterization techniques namely *partition*, introduced in Section 3.5.1, and *fragmentation*, described in Section 3.5.2. Given a BGP query, q, the partition of q is a division of its atoms into *non-overlapping* and *non-empty* parts that cover all the atoms of q; while the fragmentation of the same query q is a division of its atoms into *non-empty* fragments that cover all the atoms of q and may overlap.

Section 3.5.3 presents a *naïve algorithm* for finding a partition of a given query, such that the number of reformulations of the parts are bounded by a given threshold. Furthermore, Section 3.5.4 introduces an heuristic algorithm to find a partition for the query, such that the number of reformulations of the parts are bounded, while Section 3.5.5 presents variant, allowing overlapping query fragments.

3.5.1 BGP query partition

A partition of a given set S is a division of S into *non-overlapping* and *non-empty* subsets of S which, together, covers all of S. More formally, a partition for a given set S is a collection of disjoint subsets of S whose union is S. The number of partitions of an n-set is called a *Bell number*, B_n [39]., whose formula is:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} * B_k, \text{ starting with } B_0 = 1.$$

Generating set partitioning is a well-known and studied problem that can be found in articles like [40] and in multiple algorithms books such as [41, 42] to name a few.

BGP query partition resembles *set partition* since it can be seen as the partition of the set of triple patterns (i.e., the BGP). A key difference between set partition and query partition is that the first allows any subset of the given set to be a part, while in query partition all the atoms in a part must be joined (directly or indirectly). Recall that two atoms $t_i, t_j (i \neq j)$ of a given query are joined when there is at least one variable in common among them. Moreover, the number of partitions of a BGP query having n-atoms is bounded by the number of partitions of an n-set, but in practice the size is smaller since BGP queries are not a complete graph usually.

As the size of the reformulated BGP query has a strong impact on the execution time, our first approach then is to find a partition for the given query.

Definition 3.5.1. *[BGP query partition] A BGP query partition C is a partition $C = \{c_1, c_2, \ldots, c_k\}$, given a dataset D, a BGP query q and a threshold in the number of reformulations t, such that:*

- *C is a partition for the atoms of q.*
- *For all $c_i \in C$, $|c_i|_r$ is bounded ($\leq$) by t* or *the cluster contains only one triple pattern t_j with more than one reformulation (where $|t_j|_r > t$), and zero or more atoms with a single reformulation. In other words, the cluster size is bounded by the given threshold* or *its composed by one atom with more than t reformulations and zero or more atoms with a single reformulation.*
- *All the atoms in a cluster are joined (directly or indirectly).*

Theorem 3.5.1 shows that for a given query, the evaluation of the reformulated query w.r.t. a dataset have the same answer set as the join(s) between the evaluation of the reformulated atoms of the query w.r.t. the dataset. Moreover, Theorem 3.5.2 extends this to partitions, and states that the evaluation of the reformulation of a given query w.r.t. a dataset is equivalent to the join(s) between the evaluation of the clusters (contained in the given partition) w.r.t. the dataset.

Theorem 3.5.1. *Given a BGP query $q(\bar{x})$:- $t_1, t_2, \ldots, t_n$ and a dataset D whose RDF schema is $\mathcal{S}$, the following holds:*

$$\begin{aligned}\text{evaluate}(\mathbf{Reformulate}(q,\mathcal{S}),D) = \pi_{\bar{x}}[\text{evaluate}(\mathbf{Reformulate}(t_1,\mathcal{S}),D) \bowtie \\ \text{evaluate}(\mathbf{Reformulate}(t_2,\mathcal{S}),D) \bowtie \ldots \bowtie \text{evaluate}(\mathbf{Reformulate}(t_n,\mathcal{S}),D)]\end{aligned} \tag{3.1}$$

where the head of (the query inferred form) an atom t_i are those variables contained either in the head of q or in another atom t_j ($i \neq j$).

Proof. The theorem directly follows from [1], where it is shown that for any query Q against a database D whose RDF schema is $\mathcal{S}$: evaluate($\mathbf{Reformulate}(Q,\mathcal{S}),D$) = evaluate($Q, \mathbf{Saturate}(D)$) holds. Indeed,

$$\begin{aligned}\text{evaluate}(\mathbf{Reformulate}(q,\mathcal{S}),D) = \text{evaluate}(q,\mathbf{Saturate}(D)) = \\ \pi_{\bar{x}}[\text{evaluate}(t_1,\mathbf{Saturate}(D)) \bowtie \text{evaluate}(t_2,\mathbf{Saturate}(D)) \bowtie \ldots \\ \bowtie \text{evaluate}(t_n,\mathbf{Saturate}(D))] = \pi_{\bar{x}}[\text{evaluate}(\mathbf{Reformulate}(t_1,\mathcal{S}),D) \bowtie \\ \text{evaluate}(\mathbf{Reformulate}(t_2,\mathcal{S}),D) \bowtie \ldots \bowtie \text{evaluate}(\mathbf{Reformulate}(t_n,\mathcal{S}),D)] \text{ holds.}\end{aligned}$$

□

Theorem 3.5.2. *Given a BGP query $q(\bar{x})$, a dataset D whose RDF schema is $\mathcal{S}$ and a partition $C = \{c_1, c_2, \ldots, c_k\}$, the following holds:*

$$\begin{aligned}\text{evaluate}(\mathbf{Reformulate}(q,\mathcal{S}),D) = \pi_{\bar{x}}[\text{evaluate}(\mathbf{Reformulate}(c_1,\mathcal{S}),D) \bowtie \\ \text{evaluate}(\mathbf{Reformulate}(c_2,\mathcal{S}),D) \bowtie \ldots \bowtie \text{evaluate}(\mathbf{Reformulate}(c_n,\mathcal{S}),D)]\end{aligned} \tag{3.2}$$

Proof.

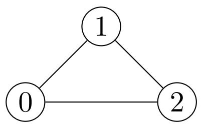
$$\begin{aligned}\text{evaluate}(\textbf{Reformulate}(q,\mathcal{S}),D) = \text{evaluate}(q,\textbf{Saturate}(D)) =\\ \pi_{\bar{x}}[\text{evaluate}(c_1,\textbf{Saturate}(D)) \bowtie \text{evaluate}(c_2,\textbf{Saturate}(D)) \bowtie \ldots\\ \bowtie \text{evaluate}(c_n,\textbf{Saturate}(D))] = \pi_{\bar{x}}[\text{evaluate}(\textbf{Reformulate}(c_1,\mathcal{S}),D) \bowtie\\ \text{evaluate}(\textbf{Reformulate}(c_2,\mathcal{S}),D) \bowtie \ldots \bowtie \text{evaluate}(\textbf{Reformulate}(c_n,\mathcal{S}),D)]\end{aligned}$$

□

Note that Theorems 3.5.1 and 3.5.2 consider queries without blank nodes. This assumption is made *without loss of generality* since blank nodes act as (thus can be equivalently replaced by) non-distinguished variables in BGP queries.

Example 20 (**Query partition**). *For example, consider the DBLP dataset [35], the query shown in Figure 3.7 and the threshold $t = 20$.*

$$\begin{array}{lll} q(x,v) & \text{:- } x\ dblp{:}datatypeField\ v, & (0)\\ & x\ purl{:}publisher\ \text{“}Springer\text{''}, & (1)\\ & x\ \tau\ dblp{:}Document & (2)\end{array}$$



Triple pattern	#Reformulations	#Results
0	19	2,629,667
1	1	4,367
2	36	711,174

Fig. 3.7: Query q and it characteristics.

Partition	Clusters	#Reformulations
(i)	$\{0,1,2\}$	684
(ii)	$\{0,1\},\{2\}$	55
(iii)	$\{0\},\{1,2\}$	55
(iv)	$\{0,2\},\{1\}$	684
(v)	$\{0\},\{1\},\{2\}$	55

Fig. 3.8: Query partitions and their #reformulations.

Figure 3.8 shows all the possible partitions (regardless the number of reformulations) for the reformulations of the query q w.r.t. to D schema and the number of reformulations for each one. Observe that the partitions (i) and (iv) are not valid since the parts $\{0,1,2\}$ *and* $\{0,2\}$ *both have* 684 *reformulations, which is greater than the given threshold. Also note that although all other three (valid) partitions have the same number of reformulations, the partition (v) is less efficient since the single reformulation atom* (1) *is alone in a cluster, which will lead to a subquery (adding extra cost) and does not take advantage of the selectivity that atom* (1) *may add when its joined with other atoms.*

The "best" partition will be selected by the use of our cost model, based on the intermediate number of results of the subqueries, and the factors α and β.

(a) Using partition (ii):

(i) Cluster {0, 1} produces the subquery:

$$\begin{array}{lll} q_0(x,v) & \text{:- } x\ dblp{:}datatypeField\ v, & (0)\\ & x\ purl{:}publisher\ \text{“}Springer\text{''} & (1)\end{array}$$

matching 30,450 *tuples.*

(ii) Cluster {2} produces the subquery:

$$q_1(x) \qquad x\ \tau\ \textit{dblp:Document} \quad (2)$$

matching 711, 174 *tuples.*

(b) Using partition (*iii*)*:*

(i) Cluster {*0*} *produces the subquery:*

$$q_0(x, v) \quad \text{:-}\ x\ \textit{dblp:datatypeField}\ v \quad (0)$$

matching 2, 629, 667 *tuples.*

(ii) Cluster {*1, 2*} *produces the subquery:*

$$\begin{array}{lll} q_1(y) & \text{:-}\ x\ \textit{purl:publisher}\ \text{“Springer”}, & (1) \\ & \quad x\ \tau\ \textit{dblp:Document} & (2) \end{array}$$

matching 4, 367 *tuples.*

3.5.2 BGP query fragmentation

The relational database approach to execute a query that joins relations, is to select the best order in which these relations should be joined in order to minimize the time consumed. Under the cost model assumptions in Section 3.4, given a query $Q = R \bowtie S \bowtie T$, executing the join of the 3 relations in some order will take less time than $(R \bowtie S) \bowtie (S \bowtie T)$, or the join of any other combination of pairs of relations equivalent to Q.
Reformulated RDF queries introduce a new element to take into account, the number of reformulations (given a query and a dataset). The execution time, and therefore the execution plan, does not depend mostly on the number of results of each atom but also, and usually with more incidence, on the number of reformulations of the atoms. It is thus desirable then, for atoms with a single (or a few) reformulations and high selectivity to be included in several clusters, since their selectivity will reduce the number of tuples resulting from the subquery without incrementing (or not drastically) the number of reformulations of the subquery.

Definition 3.5.2. *[BGP query fragmentation] A BGP query fragmentation C is a partition $C = \{c_1, c_2, \ldots, c_k\}$, given a dataset D, a BGP query q and a threshold in the number of reformulations t, such that:*

- *There are no empty clusters in C, i.e., $\forall c_i \in C(size(c_i) > 0)$.*
- *The clusters in C are a subset of q atoms such that, together, covers all the atoms of q. More formally: $\bigcup_{c_i \in C} c_i \equiv \texttt{Body}(q)$, where $\texttt{Body}(q)$ is the set of triples patterns of the query q.*
- *For all $c_i \in C$, $|c_i|_r$ is bounded ($\leq$) by t* or *the cluster contains only one triple pattern t_j with more than one reformulation (where $|t_j|_r > t$), and zero or more atoms with a single reformulation. In other words, the cluster size is bounded by the given threshold* or *its composed by one atom with more than t reformulations and zero or more atoms with a single reformulation.*
- *All the atoms in a cluster are joined (directly or indirectly).*

Theorem 3.3 states that the evaluation of the reformulation of a given query w.r.t. a dataset is equivalent to the join(s) between the evaluation of the clusters (in the given fragmentation) w.r.t. the dataset. The proof is omitted as it is similar to the proof presented for Theorem 3.5.2.

Theorem 3.5.3. *Given a BGP query $q(\bar{x})$, a dataset D whose RDF schema is $\mathcal{S}$ and a fragmentation $C = \{c_1, c_2, \ldots, c_k\}$, the following holds:*

$$\begin{aligned}\text{evaluate}(\mathbf{Reformulate}(q,\mathcal{S}),D) = \pi_{\bar{x}}[\text{evaluate}(\mathbf{Reformulate}(c_1,\mathcal{S}),D) \bowtie \\ \text{evaluate}(\mathbf{Reformulate}(c_2,\mathcal{S}),D) \bowtie \ldots \bowtie \text{evaluate}(\mathbf{Reformulate}(c_n,\mathcal{S}),D)]\end{aligned} \tag{3.3}$$

Example 21 (**Query fragmentation**)**.** *Consider the query and the threshold given in the Example 20. In that example we use the selectivity of the atoms to decide if atom* (1) *should be clustered with atom* (0) *while atom* (2) *remains alone in a cluster, or the other way around. As fragmentation allows overlapping fragments, atom* (1) *can be clustered with atom* (0) *in one fragment and also with atom* (2) *in another fragment, as illustrated in Figure 3.9.*

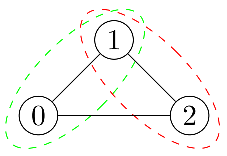


Fig. 3.9: Fragmentation example.

The number of reformulations of this fragmentation is still 55*. The difference with Example 20 is in the intermediate number of results of the subqueries (corresponding to the clusters). Using the fragments* $\{0,1\}$ *and* $\{1,2\}$ *results in subqueries matching* 30,450 *and* 4,367 *tuples respectively. Indeed, significantly less than the intermediate number of results produced by the subqueries (corresponding to the partitions) in Example 20.*

The reduction in the intermediate number of results leads to two main advantages that ultimately increase the efficiency of the query:

- *Reduction in the number of tuples being materialized, as all the subqueries but one are materialized.*
- *Reduction in the size of the intermediate result sets being joined (the subqueries).*

3.5.3 Naive threshold algorithm

Our naïve algorithm first generates all the partitions for the set of triple patterns contained in the given query, which is analog to generate all the partitions for the set $\{1, \ldots, n\}$ (where n is the size of the query) since each atom can be identified by an integer (its position in the query). Then, those partitions that do not meet BGP query partition requirements are removed (i.e., first those including non joined atoms, and then those whose number of reformulations exceeds the given threshold and have more than one atom with multiple reformulations). Finally, the cost of the remaining (valid) partitions is calculated using our cost model (previously described in Section 3.4) and the one with minimum cost is returned as the output of the algorithm.

Observe that looping only once trough the generated set of partitions is enough. By keeping in memory the best partition (up to the current iteration) and its cost (the minimum cost up to the current iteration), we iterate over the generated partitions and for each of them:

1. Check if the partition meets the BGP partition requirements. If not, then just continue with the next partition.
2. Calculate the cost of the partition using the given factors.
3. If the cost of the partition is less than the minimum cost known up to the current iteration (or the first partition that reach the requirements), then update the best partition and minimum cost variables.

Algorithm 3, presented below, uses the signature described in [41] to generate all the partitions for the set $\{1, \ldots, n\}$.

Algorithm 3: Naive Threshold Reduction Factor Algorithm (**NTRFA**)

```
Input  : BGP query $q$, dataset $D$, # reformulations threshold $t$, # tuples factor $\alpha$,
         # reformulations factor $\beta$
Output: A partition $partition$ for the BGP query $q$ such that the number of
         reformulations of the parts w.r.t. $D$ schema is bounded by $t$ (when is
         possible)
1  $bestPartition \leftarrow \emptyset$; $minCost \leftarrow NULL$; $\mathcal{S} \leftarrow D.getSchema()$
2  $partitions \leftarrow SetPartitions(q.size())$
3  foreach $partition$ in $partitions$ do
4      if $HasDisjoinedAtom(partition, q)$ then
5          continue
6      if $Reformulations(partition, q, \mathcal{S}) > t$ then
7          continue
8      $cost \leftarrow CalculateCostUsingFactors(q, partition, D, \alpha, \beta)$
9      if $minCost == NULL$ or $minCost > cost$ then
10         $minCost \leftarrow cost$; $bestPartition \leftarrow partition$
11 return $bestPartition$
```

First, the variables that will keep the best partition and its cost (the minimum one) are initialized together with the RDF schema of the given dataset (line 1). Line 2 generates all the partitions for the set $\{1, \ldots, n\}$, where n is the number of atoms of the given query. Lines 3–10 walk trough the partitions filtering those that do not reach the BGP query partition requirements (*if* statements in lines 4 and 6); for those partitions that meet all the requirements the cost is calculated (line 8) w.r.t. the given dataset. Finally, line 9 checks if the cost is less than the minimum (cost) up to the current iteration, *or* it is the first partition that fulfill the BGP query partition requirements; if so, the best partition and minimum cost variables are updated, respectively, with the partition in the current iteration and its cost.

Analysis of the algorithm The worst case is when the BGP of the given query is a complete graph (i.e., there is a join between any pair of atoms of the query) and the number of reformulations of all the possible parts of atoms are bounded by the given threshold. If that is the case, then the cost is calculated for every possible partition.

Given a BGP query of size n, there are B_n possible partitions of the query atoms. Therefore, the cost algorithm (which involves the **Reformulate** algorithm and the cardinality estimator, respectively, to estimate the number of reformulations and number of results of the parts) is invoked B_n times. However, note that for the set of n-atoms of the query there are $\sum_{k=1}^{n} \binom{n}{k}$ different parts that can appear in the partitions. Hence, by calculating and keeping in memory the number of reformulations and the number of results of all the possible parts, we can calculate the cost of all the possible partitions without invoking the **Reformulate** algorithm nor the cardinality estimator twice for the same part. Therefore, the **Reformulate** algorithm and the cardinality estimator are invoked, in the worst case, $\sum_{k=1}^{n} \binom{n}{k}$ times each.

Theorem 2.7.2 states that the complexity of the **Reformulate** algorithm is in $O((6 * \#\mathcal{S}^2)^n)$, with $\#\mathcal{S}$ and n the sizes (number of triples) of $\mathcal{S}$ and q respectively; while the complexity of the cardinality estimator is in $O(n)$ as the algorithm is linear w.r.t. the size of the query. This assumes that statistics needed w.r.t. the given dataset are available in memory.

Hence, the complexity of Algorithm 3 is $O((\sum_{k=1}^{n} \binom{n}{k}) * ((6 * \#\mathcal{S}^2)^n + n))$. However, if we decide to approximate the number of reformulations of a query by the number of reformulations of its atoms, we can reduce the complexity, to be $O(n * (6 * \#\mathcal{S}^2)^n + (\sum_{k=1}^{n} \binom{n}{k}) * (2 * n))$.

3.5.4 Greedy threshold algorithm

A first idea was to ensure that no selected cluster has more than t reformulations, for a given fixed threshold t. Based on this idea, Algorithm 4 greedily selects clusters of atoms by "growing" subqueries from single atoms, until the cluster obtained by adding atoms has more reformulations than t. Moreover, Algorithm 4 considers *non-overlapping clusters* only (i.e., a partition), that is: any two different clusters will cover disjoint sets of atoms.

Algorithm 4: Greedy Threshold Reduction Factor Algorithm (**GTRFA**)

```
Input  : BGP query $q$, dataset $D$, # reformulations threshold $t$, # tuples factor $\alpha$,
         # reformulations factor $\beta$
Output: A partition $partition$ for the BGP query $q$ such that the number of
         reformulations of the parts w.r.t. $D$ schema is bounded by $t$ (when is
         possible)
1  $partition \leftarrow \emptyset$; $\mathcal{S} \leftarrow D.getSchema()$
2  $sortedNodes \leftarrow SortNodesByCostUsingFactors(q.nodes(), D, \alpha, \beta)$
3  while $sortedNodes \neq \emptyset$ do
4      $part \leftarrow \{sortedNodes.head()\}$
5      $sortedNodes \leftarrow sortedNodes.tail()$
6      $node = FilterByNeighborAndThreshold(part, sortedNodes, q, \mathcal{S}, t)$
7      while $node \neq NULL$ do
8          $sortedNodes \leftarrow sortedNodes \setminus \{node\}$
9          $part \leftarrow part \cup \{node\}$
10         $node = FilterByNeighborAndThreshold(part, sortedNodes, q, \mathcal{S}, t)$
11     $partition \leftarrow partition \cup \{part\}$
12 return $partition$
```

First, the variable that keeps the (growing) partition is initialized as an empty set while $\mathcal{S}$ is initialized with the RDF schema of the given dataset (line 1). Then, in line 2, query atoms are sorted by their cost (w.r.t. the given dataset) using the given factors. The SortNodesByCostUsingFactors algorithm calculates the cost of each atom w.r.t. the dataset also relying on the α and β parameters of our cost model (introduced in Section 3.4). Then, SortNodesByCostUsingFactors sorts the atoms incrementally by their cost and return the ordered nodes.
Algorithm 4 has two nested loops. The main loop, between lines 3 and 11, incrementally grows the BGP query partition by adding a new part in each iteration. The inner loop (lines 7–10) grows a (currently being constructed) part from a single atom. Moreover, lines 4 and 5 take the atom having the lower cost, which does not appear in a previously created part, initializes a new part with it and removes it (the atom) from the sorted nodes list. Lines 6–10 seek for atoms (one by one) with minimum cost such that:

- The atom is not included in a previously created or the current part (being constructed).
- The atom has a join with at least one atom of the current part.
- The number of reformulations of the current *part* to which is added the set $\{node\}$ is lower than or equal to the given threshold: $|part \cup \{node\}|_r \leq t$.

When no atom can be found which meets these requirements (the part cannot grow more), the current part is added to the partition (line 11). This procedure continues until the list of sorted nodes is empty.

Analysis of the algorithm Sorting the nodes by their cost using the given factors requires the execution of the **Reformulate** algorithm and the cardinality estimator, and sorting the nodes by cost after. Therefore, the SortNodesByCostUsingFactors algorithm, invoked on line 2, is in $O(n*((6*\#\mathcal{S}^2)^n+n+log(n)))$, with $\#\mathcal{S}$ and n the sizes (number of triples) of $\mathcal{S}$ and q respectively.

Observe that the outer loop will be executed, at most, n times since at least one atom is removed from the list of sorted nodes at each iteration.

The algorithm FilterByNeighborAndThreshold is in $O((n-k)*(m+(6*\#\mathcal{S}^2)^{m+1}))$, with k and m being, respectively, the number of nodes already in the partition and the size of the part being constructed (*part*). Observe that in the worst case all the remaining $n-k$ nodes will be analyzed and for each of them:

- The existence of at least one atom in the given *part* to which the node is joined will be checked;
- The number of reformulations of $|part \cup \{node\}|_r \leq t$ will be calculated.

To check if a node and a part are connected, we are using a naïve algorithm that iterates the atoms in the part and returns *true* when it finds one that is joined with the given node; $false$ is there's no such atom. Therefore, the complexity could be slightly improved by the use of a better method, but it would still be dominated by the fact that the **Reformulate** algorithm is invoked $n-k$ times in the worst case.

Note that there is a strong relation between the times the outer and the inner loop are executed. The worst case of the inner loop is when all the nodes in the sorted nodes list can be arranged in the same part; this is the best case for the outer join, since the list of sorted nodes will be empty by the end of the first iteration. Hence, Algorithm 4 is in $O(n*((6*\#\mathcal{S}^2)^n+n)+\sum_{i=1}^{n-1}(n-i)*(i+(6*\#\mathcal{S}^2)^{i+1}))$. Once again, if we decide to approximate the number of reformulations of a query by the number of reformulations of its atoms, we can reduce the complexity, to be $O(n*(6*\#\mathcal{S}^2)^n+n^3)$.

3.5.5 Greedy threshold algorithm with overlapping fragments

Preliminary tests seem to show that in some cases efficient evaluation methods may involve overlapping clusters, i.e., such that some query atoms appear in more than one cluster. This happens especially when an atom is selective (returns few triples) and its reformulation is not very large either. Accordingly, we have devised a three-phase algorithm, which is a variant on Algorithm 4, extended to also allow overlapping query fragments.
The phases of Algorithm 5 can be summarized as:

i) Identify "bad nodes" (nodes with many reformulations).

ii) Mitigate "bad nodes" selectivity by creating a cluster for each of them and joining with nodes with high selectivity and few reformulations.

iii) Group the remaining nodes (that are not included in any cluster yet) through the fragmentation approach described in Section

Algorithm 5: Greedy Overlapping Fragments Threshold Reduction Factor Algorithm (**GOFTRFA**)

```
Input  : BGP query $q$, dataset $D$, # reformulations threshold $t$, # tuples factor $\alpha$,
         # reformulations factor $\beta$
Output: A fragmentation $fragmentation$ for the BGP query $q$
1  $fragmentation \leftarrow \emptyset$; $\mathcal{S} \leftarrow D.getSchema()$
2  $badNodes \leftarrow FilterByReformulationsThreshold(q.nodes(), D, t)$
3  $nodes \leftarrow q.nodes() \setminus badNodes$
4  $sortedNodes \leftarrow SortByCostUsingFactors(nodes, D, \alpha, \beta)$
5  foreach $Node\ node \in badNodes$ do
6      $fragmentation \leftarrow fragmentation \cup Extend(node, D, t, sortedNodes, \alpha, \beta)$
7  $remainingNodes \leftarrow nodes \setminus Flatten(fragmentation)$
8  while $remainingNodes \neq \emptyset$ do
9      $node \leftarrow remainingNodes.head()$
       $fragment \leftarrow Extend(node, D, t, sortedNodes, \alpha, \beta)$
       $remainingNodes \leftarrow remainingNodes \setminus fragment$
       $fragmentation \leftarrow fragmentation \cup fragment$
10 return $fragmentation$
```

Within Algorithm 5, first the variable that keeps the (growing) fragmentation is initialized as an empty set and $\mathcal{S}$ is initialized with the RDF schema of the given dataset (line 1).

Second, the "bad nodes" (nodes with more reformulations than the given threshold) are identified and separated from the rest in line 2.

Then, the other nodes (those with less reformulations than the given threshold) are sorted by their cost (w.r.t. the given dataset) using the given factors in line 4. The SortNodesByCostUsingFactors algorithm calculates the cost of each atom w.r.t. the dataset given using the factors also given as arguments by the use of our cost model (introduced in Section 3.4). Then, it sorts the atoms incrementally by their cost and return the ordered nodes. Unlike Algorithm 4, the number of reformulations are not calculated by the SortByCostUsingFactors algorithm, since they are already present in memory (from the "bad nodes" identification).

Once the nodes with less reformulations than the threshold are sorted, the algorithm iterates over the "bad nodes" creating a fragment for each of them and adding it to the fragmentation. The algorithm Extend is used to grow a fragmentation starting from an atom. This is performed by looping through the list of ordered nodes received as argument and for each of them, if the node joins with at least one atom of the fragment, then the cost of the fragment to which we add the atom is compared to the cost of the fragment itself: $CalculateCostUsingFactors(fragment \cup \{atom\}, D, \alpha, \beta \leq CalculateCostUsingFactors(fragment, D, \alpha, \beta))$. If the cost is less or equal to the cost of the fragment, then the atom is added to the fragment and the procedure continues with the next atom in the sorted list of nodes. Otherwise the loop finalizes and the fragmentation is returned as it is.
Finally, those atoms that are not present in any fragment of the fragmentation are identified and one by one, analogously as was done with the "bad nodes", a fragment is created

and added to the fragmentation. Observe that unlike the list of "bad nodes", in some cases one fragment may include more than one of the "remaining nodes".

Observe that unlike Algorithm 4, the number of reformulations of a cluster (having more than one atom with multiple reformulations) may be over the given threshold since the threshold t is used to identify "bad nodes".

Analysis of the algorithm The algorithm FilterByReformulationsThreshold invokes the **Reformulate** algorithm for each atom in the query and therefore is in $O(n * (6 * \#\mathcal{S}^2)^n)$, with $\#\mathcal{S}$ and n the sizes (number of triples) of $\mathcal{S}$ and q respectively.

The algorithm SortNodesByCostUsingFactors is similar to the one used in Algorithm 4. However, as the number of reformulations for each atom of the query had been calculated already in line 2, by keeping these in memory the order of SortNodesByCostUsingFactors is in $O(n * (n + log(n)))$.

Algorithm 5 is composed by two loops, each of which have the same body. The number of iterations of one is directly related with the number of iterations of the other. In the worst case, the body of the loops will be executed n times in total, whether it is executed inside the first or the second loop. The body of the loops is composed by the algorithm Extend which is in $O(\sum_{i=1}^{m-1}(m - i) * (r + (6 * \#\mathcal{S}^2)^{r+i}))$, with m and r the sizes of the sorted list of nodes and the cluster respectively. Therefore, the loops (together) are in $O(\sum_{i=1}^{n-1}(n - i) * (i + (6 * \#\mathcal{S}^2)^i))$.

Hence, Algorithm 5 is in:

$$O(n * ((6 * \#\mathcal{S}^2)^n + n) + \textstyle\sum_{i=1}^{n-1}(n - i) * (i + (6 * \#\mathcal{S}^2)^i)).$$

If we decide to approximate the number of reformulations of a query by the number of reformulations of its atoms, we can reduce the complexity, to be $O(n * (6 * \#\mathcal{S}^2)^n + n^3)$.

GOFTRFA-TS Early experiments showed that Algorithm 5 often lead to (redundant) single atom fragments. Experimental evaluation of the algorithm revealed that when no "bad nodes" are detected (or "bad nodes" that are joined with the low cost atoms to be precise), low cost atoms end up being clustered alone (as the cost of the single atom fragment is usually less than clustering the atom together with any other node) and the atom is also included in other clusters (because the low cost atom reduces the cost of the cluster). This, as shown in Chapter 4, leads to performance deterioration. Therefore, an slightly modification of the proposed algorithm was created to overcome this such cases. Once the fragmentation is computed, a loop trough the fragments removes redundant single atom fragments contained in the fragmentation.

Observe that the modification in the algorithm has no impact in the complexity.

GOFTRFA-ETS Experiments showed that the modification proposed above may also lead to sub-optimal fragmentations as it often lead to (redundant) fragments (a generalization of the problem described above). To overcome such cases we propose a modification of the proposed algorithm that extend the one proposed before to redundant fragments (instead of single atom redundant fragments). Once the fragmentation is computed, a loop trough the fragments removes redundant fragments contained in the fragmentation. To do so, first the fragments are sorted (increasingly) by size (number of atoms in the fragment); then a loop trough the sorted fragments remove redundant ones (if all the atoms in the fragment are contained in another fragment of the fragmentation, then the fragment is removed from the fragmentation).

Again, observe that the modification in the algorithm has no impact in the complexity.

3.6 Executing a clustered BGP query

Given a BGP query q, a dataset D and threshold t over the number of reformulations, we already have techniques and algorithms to reformulate q (Section 2.7) and find a clusterization bounded by t (Section 3.5). The next step is to execute the clustered BGP query against the RDBMS.

This section introduce techniques for clustered query execution, that is: execute each subquery and then the join(s) combining the subqueries. Given a BGP query q, and a clusterization C for q, for non-overlapping clusters there will be a join between two clusters, c_i and c_j $(i \neq j)$, for each edge e in the original graph such that one of the nodes connected by e is in c_i and the other in c_j. Moreover, multiple joins between a pair of fragments will simplified as one join with multiple, conjuncted, conditions. In addition, overlapping clusters will also have a join between two clusters, c_i and c_j $(i \neq j)$, for each node n such that $n \in c_i \cap c_j$. Therefore, there may be many ways in which to execute the join(s) combining the subqueries.
Section 3.6.1 introduce a naïve strategy that can be used to execute a clustered query, while Section 3.6.2 presents some opportunities for optimizations.

3.6.1 Naive strategy

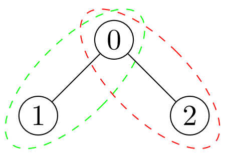
Given a BGP query q, and a clusterization that may have overlapping clusters $C = \{c_1, c_2, \ldots, c_n\}$, a first approach to execute the join(s) combining the subqueries corresponding to the clusters $\{q_1, q_2, \ldots, q_n\}$. is to add a join condition in the "the main query" for each variable that appears in the head of two (or more) subqueries.
The head of the a subquery q_i, will be the union of the inferred heads for the triple patterns in the body of q_i. Moreover, given a triple pattern $t_j = \mathtt{s\ p\ o}$, that belongs to the body of q_i, the inferred head corresponding to t_j will be composed of:

- The variables of t_j that are in the head of q_i.
- The variables of t_j that are present in another triple pattern, t_k, that belongs to (the body) q_i, thus implementing a join between t_j and t_k.

Example 22 (**Executing clustered query**). *Consider the following query:*

$$\begin{array}{lll} q(w) & \text{:- } x\ \mathtt{p}_1\ y, & (0) \\ & y\ \mathtt{p}_2\ w & (1) \\ & x\ \mathtt{p}_3\ z & (2) \end{array}$$

and the clusterization $C = \{\{0,1\},\{0,2\}\}$, illustrated below.



We show now how to execute the join(s) combining the subqueries.

(i) *The cluster containing the atoms* (0) *and* (1) *produces the subquery:*

$$\begin{array}{lll} q_0(x,y,w) & \text{:-}\ x\ \mathtt{p}_1\ y, & (0) \\ & \quad y\ \mathtt{p}_2\ w & (1) \end{array}$$

Observe that all the three variables (x, y, w) appearing in q_0 must be in the head since variables x and y appears in triple patterns of q_1, while w is in the head of the original query q.

(ii) *The cluster containing the atoms* (0) *and* (2) *produces the subquery:*

$$\begin{array}{lll} q_1(x,y) & \text{:-}\ x\ \mathtt{p}_1\ y, & (0) \\ & \quad x\ \mathtt{p}_3\ z & (2) \end{array}$$

Note that in this case only the variables x and y are used in the head of the (sub)query as they appear in triple patterns of q_0; variable z is a non-distinguished variable that does not appear in any other triple pattern (of other subqueries) nor in the head of the original query, and therefore it is not contained in the head of q_1.

(iii) *Finally, using q_1 and q_2 as subqueries leads to a join using x and y as join variables.*

$[(x\ \mathtt{p}_1\ y), (y\ \mathtt{p}_2\ w)] \bowtie_{x,y} [(x\ \mathtt{p}_1\ y), (x\ \mathtt{p}_3\ z)]$

3.6.2 Optimizations

Observe that in some cases, the naïve strategy can be sub-optimal.

In the previous example, the joins on x and y, regarding the original query q, are enforced by q_1 and q_0 respectively. Moreover, there is no need to use both x and y to enforce the join (in the "the main query") between the subqueries. Therefore, the use of a minimal number of variables when executing the join(s) combining the subqueries (in the "the main query") may lead to subqueries with results of smaller size (entailing the materialization of smaller intermediate result sets and join(s) of smaller relations) and a faster "main query".
The selection of variables that will be used to enforce the join(s) in the "the main query" may be driven then by their selectivity (in those subqueries in which the variable appears), shown in Example 23. Furthermore, an optimal selection of (head) variables for a subquery may also lead to a reduced number of reformulations, illustrated Example 24.

Queries characterization Observe that the optimization mentioned above is not possible for any BGP query. Given a BGP query q and a clusterization $C = \{c_1, c_2, \ldots, c_n\}$ for it, such that:

- Exists at least one cluster in C featuring all the atoms of q in which a variable x appears;
- All the clusters in C are connected without using the variable x in the join condition of all those clusters in which x appears.

The clustered query execution can be optimized as discussed above. Note that if fragmentation is employed, some of the atoms may also belong to other clusters.

Definition 3.6.1 (Connected clusters)**.** *Two clusters c_i and c_k are said to be connected when there is (at least) a join between a triple pattern contained in c_i and a triple pattern contained in c_k.*

Theorem 3.6.1. *Given a BGP query $q(\bar{x})$:- $t_1, t_2, \ldots, t_n$ and a clusterization $C = \{c_1, c_2, \ldots, c_m\}$ for it, such that:*

- *There is at least one cluster $c_i \in C$ containing any triple pattern t appearing in q such that the variable x is contained in t.*
- *The clusters in C are connected by other variables than x.*

The following holds:

$$\begin{aligned}\text{evaluate}(\textbf{Reformulate}(c_1, \mathcal{S}), D) \bowtie \ldots \bowtie \text{evaluate}(\textbf{Reformulate}(c_m, \mathcal{S}), D) =\\ \text{evaluate}(\textbf{Reformulate}(c_1, \mathcal{S}), D) \bowtie_{(\texttt{HeadVar}(c_1)\cap\texttt{HeadVar}(c_2))\setminus\{x\}} \cdots \\ \bowtie_{(\texttt{HeadVar}(c_{m-1})\cap\texttt{HeadVar}(c_m))\setminus\{x\}} \text{evaluate}(\textbf{Reformulate}(c_m, \mathcal{S}), D)\end{aligned} \tag{3.4}$$

where `HeadVar` *are the variables contained in the head of the subquery (corresponding to the cluster).*

Proof. The claim directly follows from the fact that, for any c_k, $i \neq k$:
evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie$ evaluate(**Reformulate**$(c_k, \mathcal{S}), D)$
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie$ evaluate(**Reformulate**$(c'_k \cup c''_k, \mathcal{S}), D)$
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie$ evaluate(**Reformulate**$(c'_k, \mathcal{S}), D)$ $\bowtie$
evaluate(**Reformulate**$(c''_k, \mathcal{S}), D)$ by theorems 3.5.2 and 3.5.3.
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie_{\setminus\{x\}}$ evaluate(**Reformulate**$(c'_k, \mathcal{S}), D)$ $\bowtie_{\setminus\{x\}}$
evaluate(**Reformulate**$(c''_k, \mathcal{S}), D)$ since the first atom is contained in the second one, and since the third atom does not contain x.
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie_{\setminus\{x\}}$ (evaluate(**Reformulate**$(c'_k, \mathcal{S}), D)$ $\bowtie$
evaluate(**Reformulate**$(c''_k, \mathcal{S}), D)$) since the third atom does not contain x.
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie_{\setminus\{x\}}$ evaluate(**Reformulate**$(c'_k \cup c''_k, \mathcal{S}), D)$
= evaluate(**Reformulate**$(c_i, \mathcal{S}), D)$ $\bowtie_{\setminus\{x\}}$ evaluate(**Reformulate**$(c_k, \mathcal{S}), D)$. □

Example 23 (**Continued**)**.** *Consider the query and the clusterization given in Example 22. The following executions of the clustered query produce the same results:*

(a) Using x to enforce the join condition between the subqueries:

(i) Cluster $\{0, 1\}$ produces the subquery:

$$\begin{array}{lrl} q_0(x, w) & \text{:-}\ x\ \mathtt{p}_1\ y, & (0)\\ & y\ \mathtt{p}_2\ w & (1)\end{array}$$

(ii) Cluster $\{0, 2\}$ produces the subquery:

$$\begin{array}{lrl} q_1(x) & \text{:-}\ x\ \mathtt{p}_1\ y, & (0)\\ & x\ \mathtt{p}_3\ z & (2)\end{array}$$

(iii) Finally, using x to join the subqueries q_1 and q_2, leads to "the main query":

$[(x\ \mathtt{p}_1\ y), (y\ \mathtt{p}_2\ w)] \bowtie_x [(x\ \mathtt{p}_1\ y), (x\ \mathtt{p}_3\ z)]$

(b) Using y to enforce the join condition between the subqueries:

(i) Cluster $\{0,1\}$ produces the subquery:

$$\begin{aligned} q_0(y,w) \quad &\text{:- } x \ \mathtt{p}_1 \ y, \quad (0) \\ & y \ \mathtt{p}_2 \ w \quad (1) \end{aligned}$$

(ii) Cluster $\{0,2\}$ produces the subquery:

$$\begin{aligned} q_1(y) \quad &\text{:- } x \ \mathtt{p}_1 \ y, \quad (0) \\ & x \ \mathtt{p}_3 \ z \quad (2) \end{aligned}$$

(iii) Finally, using y to join the subqueries q_1 and q_2, leads to "the main query":

$$[(x \ \mathtt{p}_1 \ y), (y \ \mathtt{p}_2 \ w)] \bowtie_y [(x \ \mathtt{p}_1 \ y), (x \ \mathtt{p}_3 \ z)]$$

As pointed in Section 3.1, the impact of one triple pattern in another atom of a given subquery (or query) does not depend only on the selectivity estimation (of the atoms), which is less accurate with increasing BGP graph size and heterogeneity [7], but also in the number of reformulations both (atoms) have in common and the variables being selected (head). Observe that the *query clusterization* and *clustered query execution* processes are related. Continuing with the example above, the decision of which variable (x or y) is used to enforce the join among the subqueries will also make that variable no longer necessary in the head of both subqueries (q_0 and q_1). Hence, the decision (related to clustered query execution) will have an impact on the process of query clusterization, since the removal of a variable from the head of a query might result in changes to both the number of results and the number of reformulations of the subquery.

Example 24 (Continued). *Consider the clustered query executions shown in Example 23, and the following query:*

$$\begin{aligned} q(w) \quad &\text{:- } x \ \mathit{dblp{:}datatypeField} \ y, \quad (0) \\ & y \ \mathit{dblp{:}datatypeField} \ w \quad (1) \\ & x \ \tau \ z \quad (2) \end{aligned}$$

Triple pattern	#Reformulations	#Results
0	19	2,629,667
1	19	2,629,667
2	62	1,248,889

with 22,382 *reformulations.*

(a) Using x and *y to enforce the join condition between the subqueries:*

(i) Cluster $\{0,1\}$ produces the subquery:

$$\begin{aligned} q_0(x,y,w) \quad &\text{:- } x \ \mathit{dblp{:}datatypeField} \ y, \quad (0) \\ & y \ \mathit{dblp{:}datatypeField} \ w \quad (1) \end{aligned}$$

with 361 *reformulations.*

(ii) Cluster $\{0,2\}$ produces the subquery:

$$\begin{aligned} q_1(x, y) \quad &\text{:- } x \; dblp{:}datatypeField \; y, \quad (0) \\ &x \; \tau \; z \quad (2) \end{aligned}$$

with 1, 178 *reformulations.*

(b) Using x to enforce the join condition between the subqueries:

(i) Cluster $\{0, 1\}$ *produces the subquery:*

$$\begin{aligned} q_0(x, w) \quad &\text{:- } x \; dblp{:}datatypeField \; y, \quad (0) \\ &y \; dblp{:}datatypeField \; w \quad (1) \end{aligned}$$

with 361 *reformulations.*

(ii) Cluster $\{0, 2\}$ *produces the subquery:*

$$\begin{aligned} q_1(x) \quad &\text{:- } x \; dblp{:}datatypeField \; y, \quad (0) \\ &x \; \tau \; z \quad (2) \end{aligned}$$

with 1, 007 *reformulations.*

(c) Using y to enforce the join condition between the subqueries:

(i) Cluster $\{0, 1\}$ *produces the subquery:*

$$\begin{aligned} q_0(y, w) \quad &\text{:- } x \; dblp{:}datatypeField \; y, \quad (0) \\ &y \; dblp{:}datatypeField \; w \quad (1) \end{aligned}$$

with 361 *reformulations.*

(ii) Cluster $\{0, 2\}$ *produces the subquery:*

$$\begin{aligned} q_1(y) \quad &\text{:- } x \; dblp{:}datatypeField \; y, \quad (0) \\ &x \; \tau \; z \quad (2) \end{aligned}$$

with 1, 178 *reformulations.*

Observe that using x and y to enforce the join(s) in the "the main query" produces subqueries with 361 *and* 1, 178 *reformulations.*

Using only x show a decrease in the number of reformulations of q_1 (the subquery for the cluster $\{0, 2\}$*), while using y have no differences in the number of reformulations of the subqueries.*

Clearly, improving the clustered query execution capability to consider the number of reformulations of the clusters (besides the number of results) when selecting the (head) variables may further speed up the evaluation of subqueries, and therefore "the main query" (the reformulated query).

3.7 Clustered queries and RDBMs

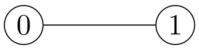
Consider the execution model introduced in Section 3.1. Given a dataset D, a BGP query q and a clusterization C for it, there are several syntactic ways of writing in SQL the *main query* joining the subqueries (which corresponds to the reformulations of the clusters in C). We considered two different techniques to write the *main query*:

(i) Defining each subquery using Common Table Expressions (or CTEs, in short) and use them in the *main query*. CTEs can be thought of as defining temporary tables that exist just for the duration of processing one query [43].

(ii) Defining each subquery in the FROM clause of the *main query*.

Example 25 (Subqueries execution). *Consider now the query shown in Figure 3.10 and the clusterization $C = \{\{0\}, \{1\}\}$. We show the SQL translation of q (using the given clusterization) defining the subqueries with CTEs in Listing 3.4, and in the FROM clause in Listing 3.5.*

$$\begin{aligned} q(x) \quad &\text{:-}\ x\ \tau\ dblp{:}Article, & (0) \\ & x\ dblp{:}objectField\ http{:}//www.example.org/dblp/ & (1) \end{aligned}$$



Triple patterns	*#Reformulations*
$\{0\}$	1
$\{1\}$	4

Fig. 3.10: Query q and reformulations characteristics.

Listing 3.4: SQL translation defining the subqueries with CTEs of q w.r.t. the clusterization C.

```
WITH
        temp_0 AS (
                SELECT DISTINCT s FROM triples AS p1 WHERE p1.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                        AND p1.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'),
        temp_1 AS (
                SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                        AND p2.o='http://www.example.org/dblp/'
                UNION
                SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                        AND p2.o='http://www.example.org/dblp/'
                UNION
                SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                        AND p2.o='http://www.example.org/dblp/'
                UNION
                SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                        AND p2.o='http://www.example.org/dblp/')
SELECT DISTINCT temp_0.s AS att_1 FROM temp_0, temp_1 WHERE temp_1.s=temp_0.s
```

Listing 3.5: SQL translation defining the subqueries in the FROM clause of q w.r.t. the clusterization C.

```
SELECT DISTINCT temp_0.s AS x FROM
        (SELECT DISTINCT s FROM triples AS p1 WHERE p1.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                AND p1.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article') AS temp_0,
        (SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                AND p2.o='http://www.example.org/dblp/'
         UNION
         SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                AND p2.o='http://www.example.org/dblp/'
         UNION
         SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                AND p2.o='http://www.example.org/dblp/'
         UNION
         SELECT DISTINCT s FROM triples AS p2 WHERE p2.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                AND p2.o='http://www.example.org/dblp/') AS temp_1
WHERE temp_1.s=temp_0.s
```

Preliminary tests seem to show that in most cases defining the subqueries in the FROM clause of the *main query* is more efficient than using CTEs.

4. EXPERIMENTAL EVALUATION

This chapter presents a variety of experiments done during our research that lead to, and support our approach. Several well-known RDF dataset were used during this experimentation stage, including DBpedia [24] and DBLP [35]. Section 4.2 presents some insides regarding the preparation of the datasets in order to be used in the experiments.

Schema-level triples are kept in memory, while instance-level triples are stored in a Triple(s, p, o) table, indexed by all permutations of the (s, p, o) columns, leading a total of 6 indexes. Our indexing choice is inspired by [6] and [13] to give the RDBMS efficient query evaluation opportunities.

Previous works [4, 6, 13] have used dictionary encoding when storing a RDF dataset, to avoid storing repeated times and joining string-encoded RDF attributes. As the experiments shows the use of this technique tends to increase the efficiency of query evaluation against RDBMS and decrease the size of both, the tables and indexes used to store the instance-level triples. Section 4.3 presents experiments regarding the impact of dictionary-encoding storage in both the sizes of the tables and indexes, and query execution time.

Finally, Section 4.4 shows the results of executing *clustered BGP queries* and compare them to *BGP query reformulation* execution, using it as baseline.

4.1 Settings

This section introduce general specifications regarding the equipment and common settings used during the measurements; the particulars of each experiment, will be introduced when necessary.

All our algorithms are fully implemented in Java^{TM}7 [44] and deployed them on top of PostgreSQL [14], version 8.5 (using standard default settings) as the database back-end for its reputation as a (free) efficient platform that has been used in several related works [4, 6, 26]. All measured times are averaged over 5 executions, since no major variations were detected between the different executions.

As in [4, 6, 13, 26], for efficiency we stored the data in a dictionary-encoded `Triples(s, p, o)` table, using a unique integer for each distinct value (URIs and literals) in the `s`, `p` and `o` positions of the dataset. The encoded `Triples` table is indexed by all the possible combinations of the three columns (i.e., a total of six indexes). Moreover, the encoding dictionary was stored as a separate table indexed both by the integer dictionary code and by the encoded value (URI or literal).

As part of the set up for each experiment, the VACUUM ANALYZE command is executed before the experiment starts, to reclaim storage occupied by dead tuples. Moreover, before measuring each query we warm up the database cache by executing the query once. Queries whose evaluation requires more than 3 hours were interrupted and duly pointed out in the experiments.

Hardware The PostgreSQL [14] server ran on a 8-core Intel Xeon (E5506) 2.13 GHz machine with 16GB RAM, running Mandriva Linux release 2010.0 (Official).

4.2 Datasets preparation

The datasets used during the experiments are available in multiple formats, including RDF. However, most of them are stored as N-Triples [45] and sometimes include annotations. Therefore, data transformation is needed in order to import the datasets into the triples table. We converted and polish the datasets into a format that can be imported into the RDBMS engine through a program implemented in Bash [46], that make use of the Raptor RDF Syntax Library [47], the AWK utility and regular expressions. Once the dump is transformed, we import the resultant (Tab Separated Values) file using the PostgreSQL *COPY* command (copy data between a file and a table). Moreover, an utility implemented in Java [44] creates the dictionary table and encode the triples table.

4.3 Dictionary-encoded Triples table

We now present experiments on the impact of using a *dictionary-encoded* `Triples` *table* on both, the size of the database (Section 4.3.1), and the execution time of the reformulated BGP queries (Section 4.3.2). Early results, and previous works [4, 6, 13, 26], motivated the use of a dictionary-encoded `Triples` table, as stated above, to store the data and ran the rest of the experiments.

4.3.1 Dictionary encoding and DB size

Database size reduction is one of the dictionary-encoding key advantages. Figure 4.1 presents the sizes of the tables, respectively, indexes corresponding to the DBLP [35] dataset (this dataset consists of 8, 424, 216 triples). We incrementally loaded the DBLP [35] dataset into the RDBMS engine (an eighth, a quarter, half and the complete dataset) and measure the size of the database at each stage. Moreover, we stored the dataset in two different ways:

- Plain: the triples of the dataset are stored as tuples in a table `Triples(s, p, o)`, using triple column indexes (i.e., a total of six indexes with all the possible combinations of the three columns).
- Encoded: a unique integer is assigned to each different value in the dataset, and these pairs are stored in a table `Dictionary`(*key*, *value*), using one index for the keys and another for the values. Each RDF triple is encoded using the *dictionary* into a triple of integer values, that are stores as tuples in a table `Triples(s, p, o)`, using triple column indexes (i.e., a total of six indexes with all the possible combinations of the three columns).

Figure 4.2 illustrates the sizes of the tables and indexes comprising the database for the DBLP [35] and DBpedia [24] (ontology info box) datasets.

Figures 4.1 and 4.2 also illustrate the important space occupancy of RDF indexes, confirming the initial reports made in [6].

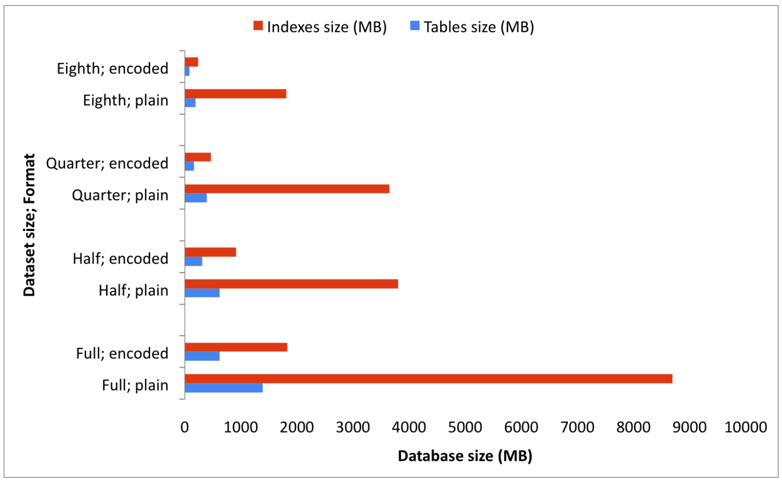


Fig. 4.1: Database size for the DBLP dataset.

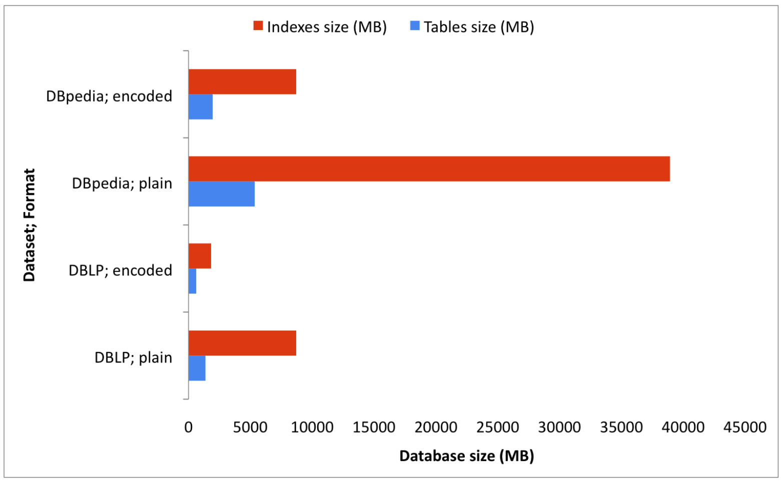


Fig. 4.2: Database size for the DBLP and DBpedia (info box) datasets.

4.3.2 Dictionary encoding and query execution time

Dictionary encoding is used not only to diminish the space occupancy but also to decrease the evaluation time of BGP queries. Figures 4.3 and 4.4 illustrate two queries, their characteristics, and the evaluation time of the reformulated query against both dictionary-encoded and non-encoded `Triples` tables.

Figure 4.3 presents a single atom query with many reformulations, while Figure 4.4 introduces a complex query with six atoms and a dense BGP.

Finally, Chart 4.5 shows that in both cases the execution of the query against the dictionary-encoded `Triples` table is (considerably) more efficient than the execution of the queries against the non-encoded `Triples` table.

$q_1(x, y)$:- $x\ \tau\ y$ (0)

Triple pattern	#Reformulations	#Tuples
0	73	1,977,010

Fig. 4.3: Query q_1 and its characteristics.

$$
\begin{array}{lll}
q_{101}(u, z) & \text{:- } x\ dblp{:}editor\ y, & (0)\\
& y\ foaf{:}name\ z, & (1)\\
& x\ purl{:}title\ u, & (2)\\
& x\ dblp{:}datatypeField\ v, & (3)\\
& x\ purl{:}publisher\ \text{“}Springer\text{”}, & (4)\\
& x\ \tau\ dblp{:}Document & (5)
\end{array}
$$

Triple pattern	#Reformulations	#Tuples
0	1	14,645
1	1	446,964
2	1	716,647
3	19	2,629,667
4	1	4,367
5	36	711,174

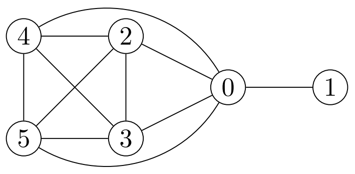


Fig. 4.4: Query q_{101} and its characteristics.

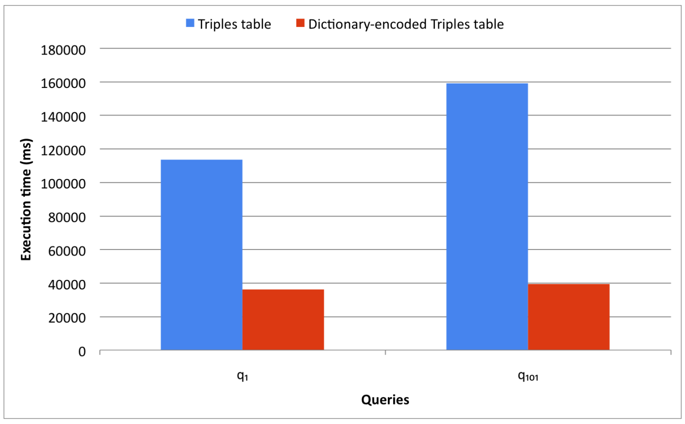


Fig. 4.5: Execution time of queries q_2 and q_{101} against plain and dictionary-encoded `Triples` table.

4.4 Clustered queries and RDBMs

This section studies query evaluation performance for the two techniques introduced in Section 3.7, used to translate a clustered BGP query into an SQL *main query* joining the subqueries (which corresponds to the reformulations of the clusters) in order to be executed in a RDBMS engine. The execution time of the reformulated queries is also included in the experiment as a baseline.

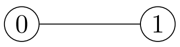
We hand-picked a set of queries for the DBLP dataset with different characteristics. The queries and their characteristics are detailed in Figures 4.6- 4.17.

Finally, Figure 4.18 summarize the query execution time (in milliseconds) for all the queries.

Two-atom queries First, we introduce three variations of a two-atoms BGP query using the built-in property τ and the DBLP [35] class *dblp:Document*. The queries are listed in Figures 4.6–4.8. The difference between q_2 and q_{102} lies in the fact that the subject variable of q_2 is x whereas in q_{102} it is y. Query q_{104} can be seen as a generalization of q_2 since it states that the type of x is an unknown z as opposed to *dblp:Document* in q_2. Figures 4.6–4.8 show that the number of reformulations and the number of matching tuples for the atoms of the queries vary between these three queries.

The query evaluation times at the bottom of Figures 4.6–4.8 demonstrate that clustered BGP query execution are more than 6 times faster than BGP query reformulation execution, due to the (wide) decrease in the number of reformulations of the executed query (achieved by clustering the atoms separately).

$$\begin{array}{lll} q_2(x,y) & :\!-\ x\ \tau\ \mathit{dblp{:}Document}, & (0) \\ & \quad x\ \mathit{dblp{:}datatypeField}\ y & (1) \end{array}$$



Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	19	2, 629, 667

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	684	2, 629, 667	242, 018		
{0}, {1}	55	2, 629, 667		42, 800	35, 890

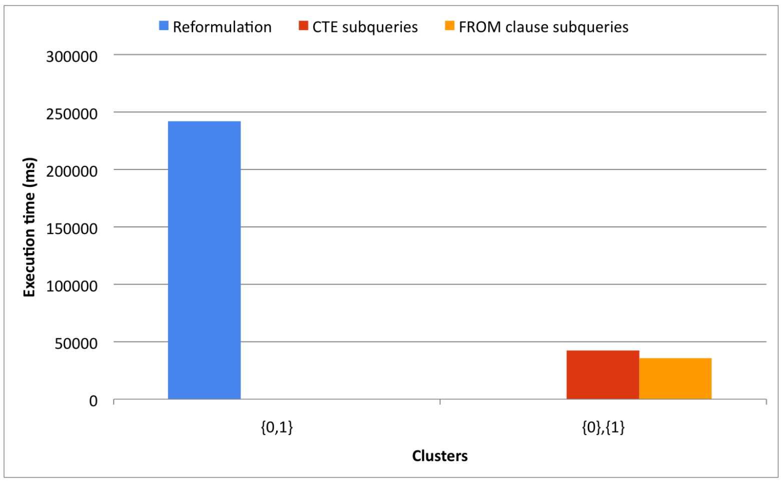
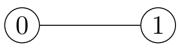


Fig. 4.6: Query q_2: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

$q_{102}(x, y)$:- $y\ \tau\ \mathit{dblp{:}Document},$ (0)
$x\ \mathit{dblp{:}datatypeField}\ y$ (1)



Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	19	2, 629, 667

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	684	4, 898	227, 712		
{0}, {1}	55	4, 898		36, 600	37, 680

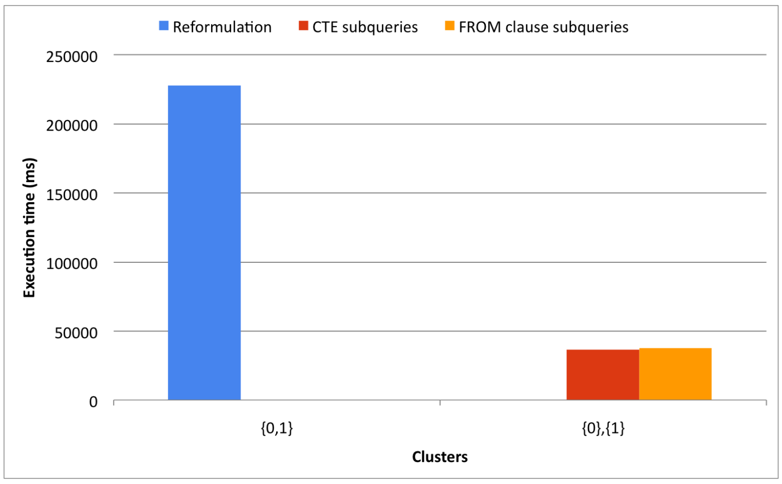
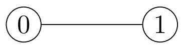


Fig. 4.7: Query q_{102}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

$q_{104}(x, y, z)$:- $x\ \tau\ z,$ (0)
$x\ \mathit{dblp{:}datatypeField}\ y$ (1)



Triple pattern	#Reformulations	#Tuples
0	73	1, 977, 010
1	19	2, 629, 667

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	1, 387	5, 259, 462	595, 376		
{0}, {1}	92	5, 259, 462		72, 082	66, 061

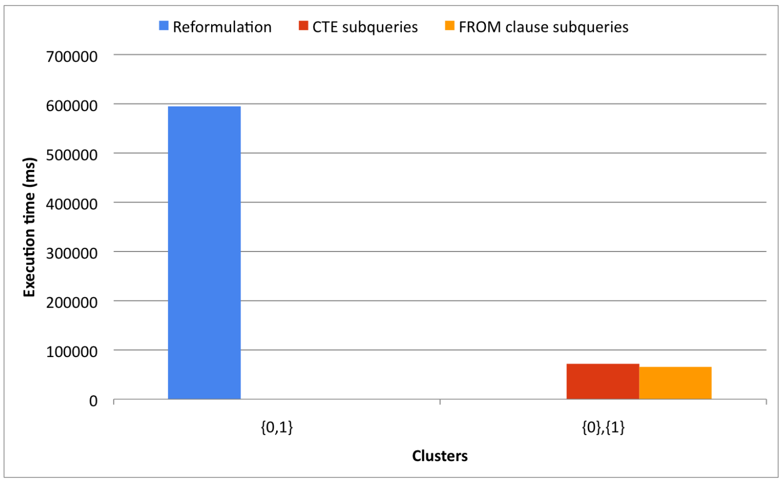


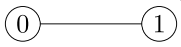
Fig. 4.8: Query q_{104}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

Figures 4.9–4.11 present three new variations of a two-atom query. Thus, q_3 and q_{107} differ in the property value of their second triple, while q_{108} is a generalization of q_3 in that q_{108} does not constrain the property value in the second triple. The experiments show that clustering atoms with few reformulations and high selectivity (e.g., the first atom in q_3 and q_{108}), with many-reformulations, not very selective atoms, is more efficient than evaluating selective atoms separately.

Moreover, as shown in the evaluation performance chart of q_{107} (Figure 4.10), while it is generally profitable to add a highly selective atom to a cluster, the benefit of doing so diminishes as the number of reformulations of that atom grows. In such cases, it is better to leave the highly selective, many-reformulations atom alone in its cluster (while it may still be clustered with others in the future, if its high selectivity gets to offset the risk due to the many reformulations). In the case of query q_{107}, leaving this atom alone leads to important reductions (by a factor of more than 12) in the number of reformulations of the query fragment sent to the RDBMS for evaluation.

$$
\begin{array}{llr}
q_3(x) & \text{:- } x\ \tau\ \mathit{dblp{:}Document}, & (0)\\
 & x\ \mathit{dblp{:}objectField}\ \mathit{http://www.example.org/dblp/} & (1)
\end{array}
$$

Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	4	8, 212



Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	144	8, 212	911		
{0}, {1}	40	8, 212		18, 562	17, 691

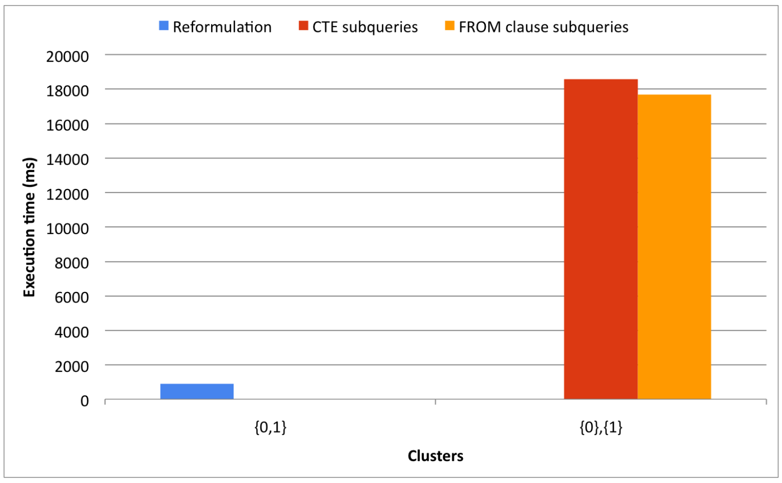


Fig. 4.9: Query q_3: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

$$q_{107}(x) \quad \text{:- } x \ \tau \ \textit{dblp:Document}, \quad (0)$$
$$x \ \textit{dblp:datatypeField} \ \textit{http://www.example.org/dblp/} \quad (1)$$

$$\begin{tikzcd} \boxed{0} \arrow[r, no head] & \boxed{1} \end{tikzcd}$$

Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	19	0

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	684	0	20, 162		
{0}, {1}	55	0		17, 565	16, 169

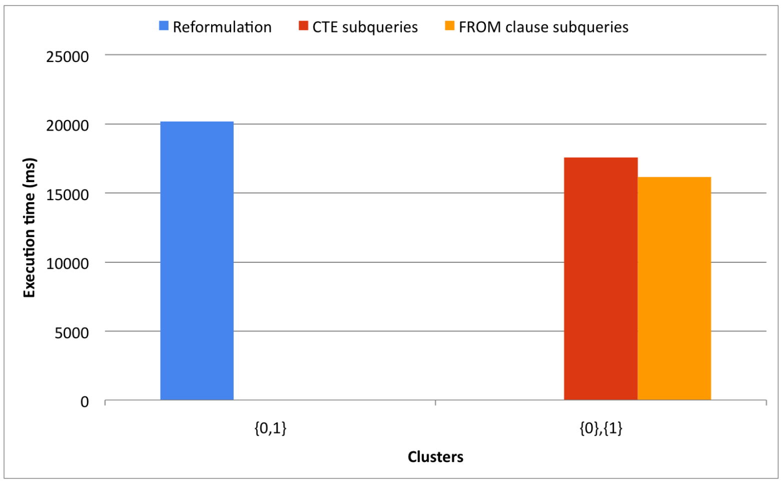


Fig. 4.10: Query q_{107}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

$$q_{108}(x, y) \quad \text{:- } x \ \tau \ \textit{dblp:Document}, \quad (0)$$
$$x \ \textit{dblp:objectField} \ y \quad (1)$$

$$\begin{tikzcd} \boxed{0} \arrow[r, no head] & \boxed{1} \end{tikzcd}$$

Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	4	8, 212

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1}	144	8, 212	7, 554		
{0}, {1}	40	8, 212		19, 040	16, 506

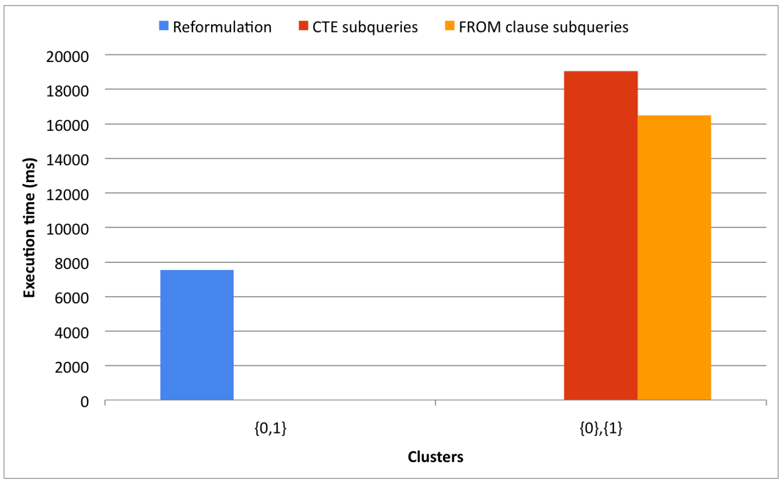
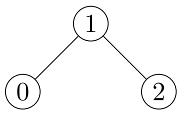


Fig. 4.11: Query q_{108}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

Three-atom query Consider now the three-atom query illustrated in Figure 4.12. Our experiment shows that a cluster including both atoms (0) and (1) should be avoided, as the number of reformulations is exceedingly high.

The evaluation performance chart also shows that in this case, using one cluster for each atom is less efficient than reformulating the whole query. This is because the one-atom clustering fails to take advantage of the high selectivity of atom (2) or the fact that it has a single reformulation. Finally, by exploiting the high selectivity and single reformulation of atom (2) using it in both clusters, BGP query fragmentation execution is more than 16 times faster than the BGP query reformulation execution (or any of the other two strategies).

$$\begin{array}{lll} q_4(x,y) & \text{:- } x\ \mathit{dblp{:}datatypeField}\ y, & (0) \\ & x\ \mathit{purl{:}publisher}\ \text{“}\mathit{Springer}\text{”}, & (1) \\ & x\ \tau\ \mathit{dblp{:}Document} & (2) \end{array}$$



Triple pattern	#Reformulations	#Tuples
0	19	2, 629, 667
1	1	4, 367
2	36	711, 174

Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1, 2}	684	30, 450	16, 999		
{0}, {1}, {2}	56	30, 450		36, 870	32, 974
{0, 1}, {2}	55	30, 450		18, 888	18, 182
{0}, {1, 2}	55	30, 450		15, 996	14, 960
{0, 2}, {1}	685	30, 450		237, 218	262, 914
{0, 1}, {1, 2}	55	30, 450		1, 025	1, 050
{1}, {0, 1}, {1, 2}	56	30, 450		1, 383	29, 622

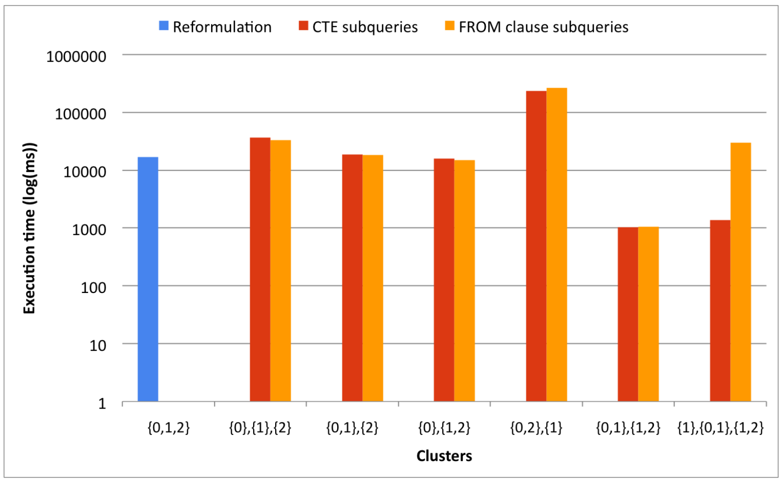
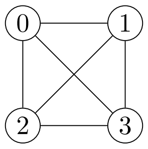


Fig. 4.12: Query q_4: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

Four-atom query Figure 4.13 presents a query composed by a highly selective atoms with 19 reformulations, and three single-reformulation atoms with a high number of matching tuples. As expected, the reformulation of the BGP query (i.e., clustering all the atoms together) is (much) more efficient than other clusterizations since atom (0) gives a high selectivity while the other atoms do not increase the number of reformulations of the evaluated query.

$$
\begin{array}{llr}
q_7(y,u,t) & \text{:- } x\ \mathit{dblp{:}objectField}\ \mathit{http://www.example.org/dblp/}, & (0)\\
 & x\ \mathit{purl{:}title}\ y & (1)\\
 & x\ \mathit{purl{:}creator}\ u & (2)\\
 & x\ \mathit{purl{:}date}\ t & (3)
\end{array}
$$

Triple pattern	#Reformulations	#Tuples
0	4	8,212
1	1	716,647
2	1	1,688,043
3	1	716,434



Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1, 2, 3}	4	19,576	324		
{0}, {1}, {2}, {3}	7	19,576		14,527	12,008
{0}, {1, 2, 3}	5	19,576		21,229	20,318
{0, 1}, {0, 2}, {0, 3}	12	19,576		462	416

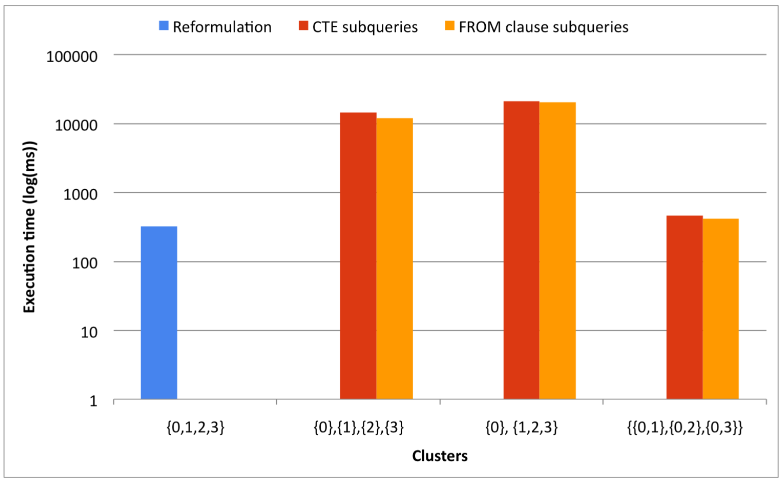


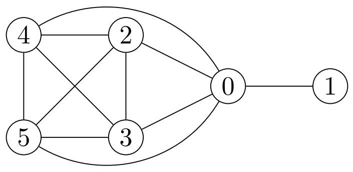
Fig. 4.13: Query q_7: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions (bottom).

Six-atom query, many-reformulation atoms Query q_{101}, shown in Figure 4.14, is composed by six atoms and have a dense BGP. In this case, clusterizations that *do not* put atoms (3) and (5) in the same cluster are much more efficient; avoiding this combination keeps the number of query reformulations under control. To ease interpretation of the results, Figure 4.14 also provides the number of reformulations and of tuples matching each cluster, in the clusterizations studied in our experiments.

As shown in the evaluation performance chart, several BGP partition and BGP fragmentation choices lead to faster execution (up to 314 times) than the execution of the query reformulated as a whole. Observe that the high selectivity of the single reformulated atom (4) makes a big differences when it appears in a cluster.

$$\begin{array}{lll} q_{101}(u,z) & \text{:- } x\ dblp\text{:}editor\ y, & (0) \\ & y\ foaf\text{:}name\ z, & (1) \\ & x\ purl\text{:}title\ u, & (2) \\ & x\ dblp\text{:}datatypeField\ v, & (3) \\ & x\ purl\text{:}publisher\ \text{“}Springer\text{”}, & (4) \\ & x\ \tau\ dblp\text{:}Document & (5) \end{array}$$

Triple pattern	#Reformulations	#Tuples
0	1	14,645
1	1	446,964
2	1	716,647
3	19	2,629,667
4	1	4,367
5	36	711,174



Clusterization	Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries	FROM clause subqueries
(i)	$\{0,1,2,3,4,5\}$	513	9,562	593,804		
(ii)	$\{0,1,2,3\},\{4,5\}$	55	9,562		4,138	4,055
(iii)	$\{2,3,4\},\{0,1,5\}$	55	9,562		7,441	5,794
(iv)	$\{0,1,2,3,4\},\{0,1,2,5\}$	55	9,562		13,987	13,774
(v)	$\{0,2,3,4\},\{0,1,2,5\}$	55	9,562		8,606	8,662
(vi)	$\{0,1,2,3,4\},\{0,5\}$	55	9,562		9,783	8,100
(vii)	$\{0,1,2,3,4\},\{5\}$	55	9,562		22,462	22,618
$(viii)$	$\{0,1,2,3\},\{2,4,5\}$	55	9,562		5,577	13,545
(ix)	$\{0,1,2,3,4\},\{4,5\}$	55	9,562		5,173	5,110
(x)	$\{0,1,2,4\},\{3,4\},\{4,5\}$	56	9,562		1,891	1,552
(xi)	$\{3,4\},\{0,1,2,5\}$	55	9,562		12,378	6,522
(xii)	$\{2,3,4\},\{0,1,2,5\}$	55	9,562		7,010	12,055
$(xiii)$	$\{2,3,4\},\{0,1,4,5\}$	55	9,562		4,207	7,296
(xiv)	$\{0,1,2,4\},\{3\},\{5\}$	56	9,562		31,689	32,508
(xv)	$\{0,1,2,4,5\},\{3\}$	55	9,562		19,808	21,904
(xvi)	$\{0,1\},\{0,4\},\{2,4\},\{3,4\},\{4,5\}$	58	9,562		1,596	18,149

Cluster	#Reformulations	#Tuples
$\{0,1,2,3\}$	19	14,694
$\{4,5\}$	36	4,367
$\{2,3,4\}$	19	4,370
$\{0,1,5\}$	36	14,674
$\{0,1,2,5\}$	36	14,694
$\{0,2,3,4\}$	19	9,539
$\{0,1,2,3,4\}$	19	9,562
$\{0,5\}$	36	14,645
$\{2,4,5\}$	36	4,370
$\{0,1,2,4\}$	1	9,562
$\{3,4\}$	19	4,367
$\{0,1,4,5\}$	36	9,559
$\{0,1,2,4,5\}$	36	9,562
$\{0,1\}$	1	14,674
$\{0,4\}$	1	9,536
$\{2,4\}$	1	4,370

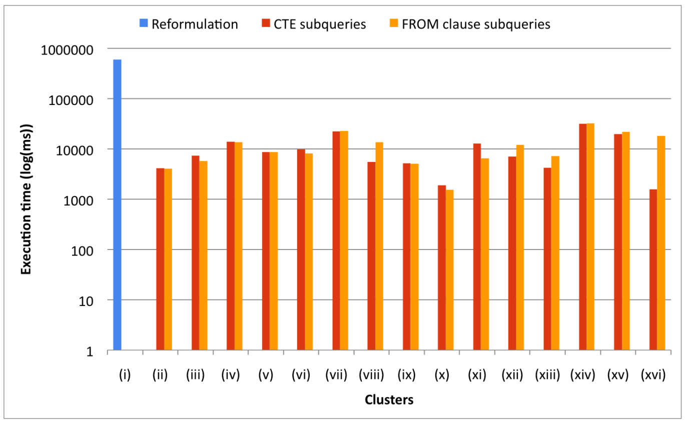
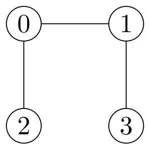


Fig. 4.14: Query q_{101}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions and their clusters (bottom).

Four-atom chain query Figure 4.15 presents a query having two single-reformulation atoms with high selectivity, each of which is connected to an atom with a high number of reformulations (73); the latter two atoms are connected among them. Experiments show the execution of a clusterization such that each of the atoms with 73 reformulations is clustered separately (with a corresponding highly selective, single reformulation atom) is more than 227 times faster than reformulating the whole BGP query.

$$\begin{aligned} q_{110}(x, y, z) \ \ \text{:-}\ & x\ \tau\ z, && (0)\\ & y\ \tau\ z, && (1)\\ & x\ purl\text{:}publisher\ \text{“}Springer\text{”}, && (2)\\ & y\ purl\text{:}publisher\ \text{“}Morgan\ Kaufmann\text{”} && (3) \end{aligned}$$

Triple pattern	#Reformulations	#Tuples
0	73	1,977,010
1	73	1,977,010
2	1	4,367
3	1	39



Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries exec. time (ms)	FROM clause subqueries exec. time (ms)
{0, 1, 2, 3}	1,721	203,462	438,498		
{0}, {1}, {2}, {3}	148	203,462		71,977	69,376
{0, 2}, {1, 3}	146	203,462		1,924	1,528

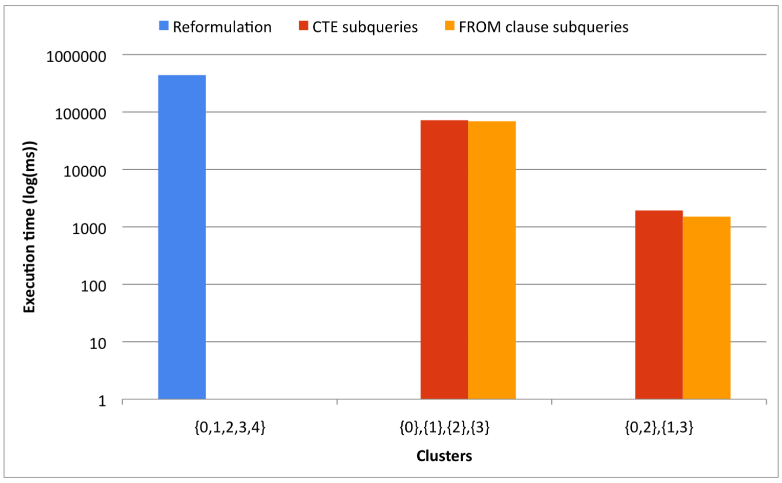


Fig. 4.15: Query q_{110}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions and their clusters (bottom).

Once again, keeping a tab on the number of reformulations while using atom selectivity to reduce the number of intermediate subquery (cluster) results, improves BGP query execution efficiency.

When BGP reformulation fails During our experiments we encountered some BGP queries whose evaluation, due to the high number of reformulations, is simply unfeasible. In some of the cases the number of reformulations is so high that the **Reformulate** algorithm takes hours or even cannot conclude due to an "out of memory" exception. Beyond the fact that the "maximum Java heap size" can be set, this problem is intrinsic to BGP query reformulation. For any given "maximum Java heap size", there is always a BGP query such that the number of reformulations is high enough to require more memory.

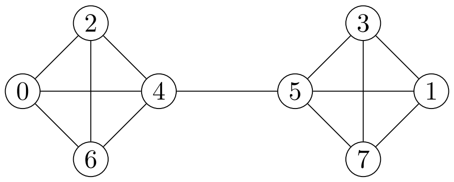
We also experienced cases in which the SQL translation of the reformulated query is syntactically so large that an `org.postgresql.util.PSQLException` is raised" "The SQL statement could not be executed: ERROR: stack depth limit exceeded". Once again, beyond the given hint (`Hint: Increase the configuration parameter ''max_stack_depth'', currently 2048kB, after ensuring the platform's stack depth limit is adequate.`), this error is intrinsic to the very large syntactically size of the reformulated queries. For any given max_stack_depth, there is a BGP query such that the SQL translation of the query reformulation exceed the limit.

Finally, there are some cases in which the reformulations, beyond being tens of thousands, can be calculated and the SQL translation executed. However, the query requires many hours of evaluation making its execution unfeasible.

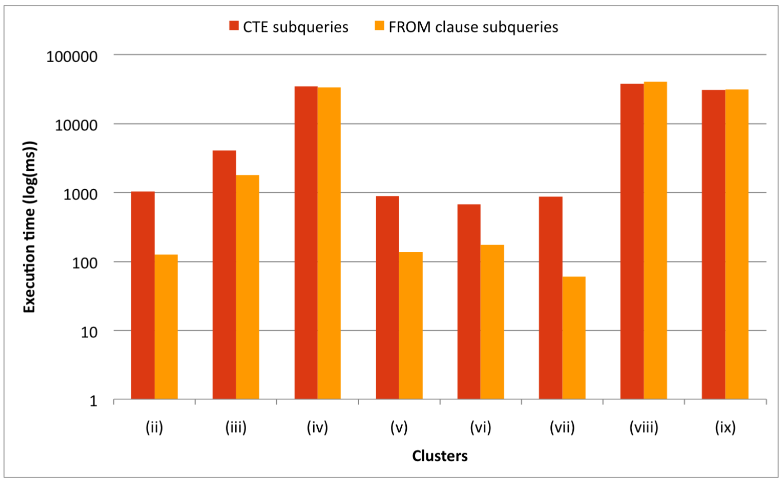
Here we present two such queries. As illustrated in the evaluation performance charts, the execution of the clustered queries is not only feasible but also done in a couple of seconds.

$$
\begin{array}{llr}
q_{111}(z, v, u, w, t) & \text{:- } x\ \tau\ dblp\text{:}Document, & (0) \\
 & y\ \tau\ dblp\text{:}Document, & (1) \\
 & x\ dblp\text{:}objectField\ u, & (2) \\
 & y\ dblp\text{:}objectField\ t, & (3) \\
 & x\ purl\text{:}creator\ z, & (4) \\
 & y\ purl\text{:}creator\ z & (5) \\
 & x\ purl\text{:}publisher\ \text{“}Springer\text{''}, & (6) \\
 & y\ purl\text{:}publisher\ \text{“}Morgan\ Kaufmann\text{''} & (7)
\end{array}
$$

Triple pattern	#Reformulations	#Tuples
0	36	711, 174
1	36	711, 174
2	4	8, 212
3	4	8, 212
4	1	1, 688, 043
5	1	1, 688, 043
6	1	4, 367
7	1	39



Clusterization	Clusters	#Reformu-lations	#Tuples	Reformula-tion exec. time (ms)	CTE subqueries	FROM clause subqueries
(i)	$\{0, 1, 2, 3, 4, 5, 6, 7\}$	20, 736	$*$	$*$		
(ii)	$\{0, 4, 6\}, \{2, 4, 6\}, \{1, 5, 7\}, \{3, 5, 7\}$	80	0		1, 034	127
(iii)	$\{0, 2, 4, 6\}, \{1, 3, 5, 7\}$	288	0		4, 096	1, 789
(iv)	$\{3, 4, 5, 6, 7\}, \{0\}, \{1\}, \{2\}$	80	0		34, 814	33, 509
(v)	$\{0, 6\}, \{1, 7\}, \{2, 6\}, \{3, 7\}, \{4, 6\}, \{5, 7\}$	82	0		893	137
(vi)	$\{0, 6\}, \{1, 7\}, \{2, 6\}, \{3, 7\}, \{2, 4, 6\}, \{5, 7\}$	85	0		674	174
(vii)	$\{0, 6\}, \{1, 7\}, \{3, 7\}, \{2, 4, 6\}, \{5, 7\}$	81	0		856	60
$(viii)$	$\{2, 3, 4, 5, 6, 7\}, \{0\}, \{1\}$	81	0		37, 650	40, 352
(ix)	$\{1, 2, 3, 4, 5, 6, 7\}, \{0\}$	612	0		30, 790	31, 257

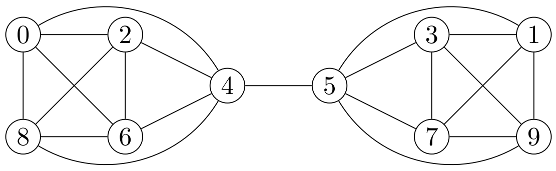


*: The query execution was interrupted after running for three hours.

Fig. 4.16: Query q_{111}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions and their clusters (bottom).

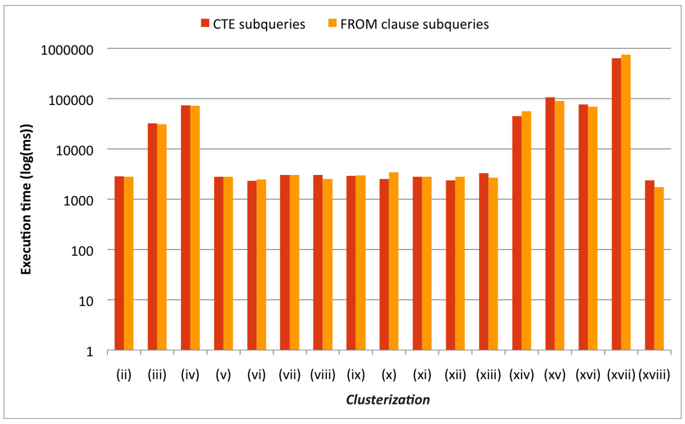
$$
\begin{array}{lll}
q_{106}(z, v, u, w, t) & \text{:-}\ x\ \tau\ v, & (0)\\
& y\ \tau\ w, & (1)\\
& x\ dblp{:}datatypeField\ u, & (2)\\
& y\ dblp{:}datatypeField\ t, & (3)\\
& x\ purl{:}creator\ z, & (4)\\
& y\ purl{:}creator\ z & (5)\\
& x\ purl{:}title\ r, & (6)\\
& y\ purl{:}title\ s, & (7)\\
& x\ purl{:}publisher\ \text{“}Springer\text{''}, & (8)\\
& y\ purl{:}publisher\ \text{“}Morgan\ Kaufmann\text{''} & (9)
\end{array}
$$

Triple pattern	#Reformulations	#Tuples
0	73	1,977,010
1	73	1,977,010
2	19	2,629,667
3	19	2,629,667
4	1	1,688,043
5	1	1,688,043
6	1	716,647
7	1	716,647
8	1	4,367
9	1	39



Clusterization	Clusters	#Reformulations	#Tuples	Reformulation exec. time (ms)	CTE subqueries	FROM clause subqueries
(*i*)	{0,1,2,3,4,5,6,7,8,9}	1,923,769	*	*		
(*ii*)	{0,4,6,8},{2,4,6,8},{1,5,7,9},{3,5,7,9}	184	80		2,878	2,811
(*iii*)	{1,5,7,9},{3},{0,4,6,8},{2}	184	80		32,529	31,112
(*iv*)	{1},{3,5,7,9},{0},{2,4,6,8}	184	80		73,388	71,886
(*v*)	{1,5,7,9},{3,5,7,9},{0,4,6,8},{2,8}	184	80		2,825	2,814
(*vi*)	{1,5,7,9},{3,5,7,9},{0,8},{2,4,6,8}	184	80		2,346	2,468
(*vii*)	{1,5,7,9},{3,9},{0,4,6,8},{2,4,6,8}	184	80		3,020	3,029
(*viii*)	{1,9},{3,5,7,9},{0,4,6,8},{2,4,6,8}	184	80		3,026	2,532
(*ix*)	{1,5,9},{3,7,9},{0,4,6,8},{2,4,6,8}	184	80		2,956	3,017
(*x*)	{1,7,9},{3,5,9},{0,4,6,8},{2,4,6,8}	184	80		2,554	3,426
(*xi*)	{1,5,7,9},{3,5,7,9},{0,4,8},{2,6,8}	184	80		2,815	2,824
(*xii*)	{1,5,7,9},{3,5,7,9},{0,6,8},{2,4,8}	184	80		2,381	2,828
(*xiii*)	{1,7,9},{3,5,9},{0,6,8},{2,4,8}	184	80		3,280	2,717
(*xiv*)	{0,2,4,6,8},{1,3,5,7,9}	2,774	80		44,715	56,464
(*xv*)	{4,5,6,7,8,9},{0},{1},{2},{3}	185	80		106,369	89,579
(*xvi*)	{0,4,5,6,7,8,9},{1},{2},{3}	184	80		760,22	69,182
(*xvii*)	{0,2,4,5,6,7,8,9},{1,3}		80		636,152	750,097
(*xviii*)	{0,8},{1,9},{2,8},{3,9},{6,8},{7,9},{4,8},{5,9}	189	80		2,404	1,736

Cluster	#Reformulations	#Tuples
{0,4,6,8}	73	1,368
{2,4,6,8}	19	2,530
{1,5,7,9}	73	110
{3,5,7,9}	19	110
{2,8}	19	30,450
{0,8}	73	8,734
{3,9}	19	92
{1,9}	73	78
{1,5,9}	73	110
{3,7,9}	19	92
{1,7,9}	73	78
{3,5,9}	19	110
{0,4,8}	73	1,364
{2,6,8}	19	30,472
{0,6,8}	73	8,740
{2,4,8}	19	2,514
{0,2,4,6,8}	1,387	5,056
{1,3,5,7,9}	1,387	220



*: Given the very high number of reformulations of q_{106} the reformulate algorithm could not finished the query reformulation, an java.lang.OutOfMemoryError Exception raised. Moreover, during our experiments we also experienced that queries with too many reformulations may also generate an org.postgresql.util.PSQLException Exception: “The SQL statement could not be executed: ERROR: stack depth limit exceeded”. Beyond the given hint (Hint: Increase the configuration parameter “max_stack_depth” (currently 2048kB), after ensuring the platform’s stack depth limit is adequate.), the error is intrinsic to the very large syntactically size of the reformulated queries.

Fig. 4.17: Query q_{106}: query syntax, query graph and statistics (top), possible decompositions (center), and evaluation performance for these decompositions and their clusters (bottom).

Figure 4.18 summarizes the evaluation performance of all the queries in Figures 4.6-4.17. Observe that the reformulation execution time is missing for queries q_{111} and q_{106} since the reformulation of the queries could not be evaluated.

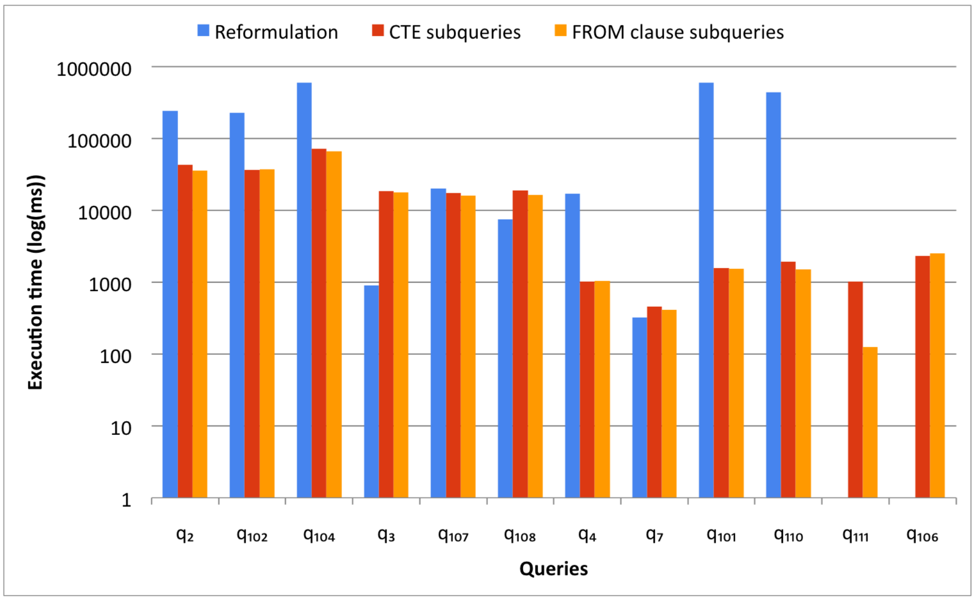


Fig. 4.18: Queries evaluation performance summary (using the known clusterization having provided the best performance).

4.5 Experimenting with our proposed algorithms

In this section, we describe a set of experiments we performed with the algorithms previously introduced in the thesis, namely: the greedy threshold reduction factor algorithm (GTRFA, Section 3.5.4), the greedy overlapping fragments threshold reduction Factor Algorithm (GOFTRFA, Section 3.5.5) and its optimized variant GOFTRFA-TS outlined in the same section. For what concerns the naive threshold reduction factor algorithm (NTRFA, Section 3.5.3), as explained when introducing the algorithm, we do not expect it to be competitive with the other ones and therefore its implementation has been postponed and thus no experiments with it are described here.

For our experiments, we considered the same queries as have been presented in the previous experiments. We are interested in two main metrics:

- the efficiency of each algorithm, that is, its running time;
- the effectiveness, that is, how well the algorithm performs in recommending clusterizations that indeed lead to a fast evaluation of the reformulated query.

Further, we have examined the impact of the values taken by the parameters characterizing the algorithms, which we recall below:

The # tuples factor α is a coefficient used to ponder the number of results returned by the evaluation of a cluster, in the cost estimation assigned to a given clustering;

The # reformulations factor β similarly is a coefficient used to ponder the number of reformulations of a cluster, in the cost estimation assigned to a given clustering;

The reformulations threshold t is the number of reformulations not to be exceeded by a given cluster (when possible).

Throughout our experiments, we assigned the following values to these parameters:

- $\alpha = 10$;
- $\beta \in \{1, 100.000\}$;
- $t \in \{10, 100, 1.000, 1.500\}$

The reason why α was constant is that we were interested in observing the *relative* impact of α and β in the clusterization cost factor, and to this effect, it sufficed to vary β. Further, the values for t were inspired from the behavior of interesting queries on real-life data set we experimented with.

4.5.1 Algorithm GTRFA

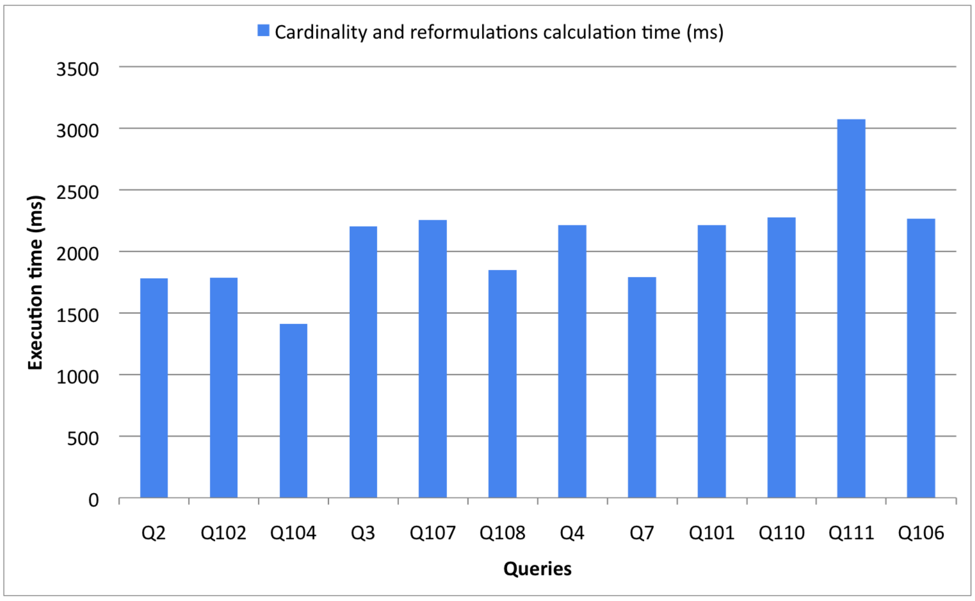


Fig. 4.19: GTRFA running time for $t = 10, \beta = 1, \alpha = 10$.

Figure 4.19 depicts the running time of Algorithm GTRFA for the the parameter configuration $t = 10, \beta = 1, \alpha = 10$. The figure depicts the time taken by the algorithm in order to: obtain the statistics necessary for estimating the number of results of various clusters, reformulate clusters to compute the number of reformulations of each, and the other steps of the algorithm itself as depicted in Algorithm 4. We found that the latter was negligible, and less than 10 ms in all cases. (This remark holds for the remainder of the experiments with our algorithms.)

Figure 4.19 shows that overall, the *algorithm running time* is acceptable and never more than three seconds, for small queries such as q_2 (two atoms) up to relatively large queries such as q_{106} (ten atoms). Observe that the statistics are currently gathered through SQL queries against a statistics database; this could be significantly sped up by relying on more efficient data statistics structures, for instance in memory.

Further, we examined the clusterizations recommended by the algorithm for all the queries considered, to see how efficient their evaluation was.

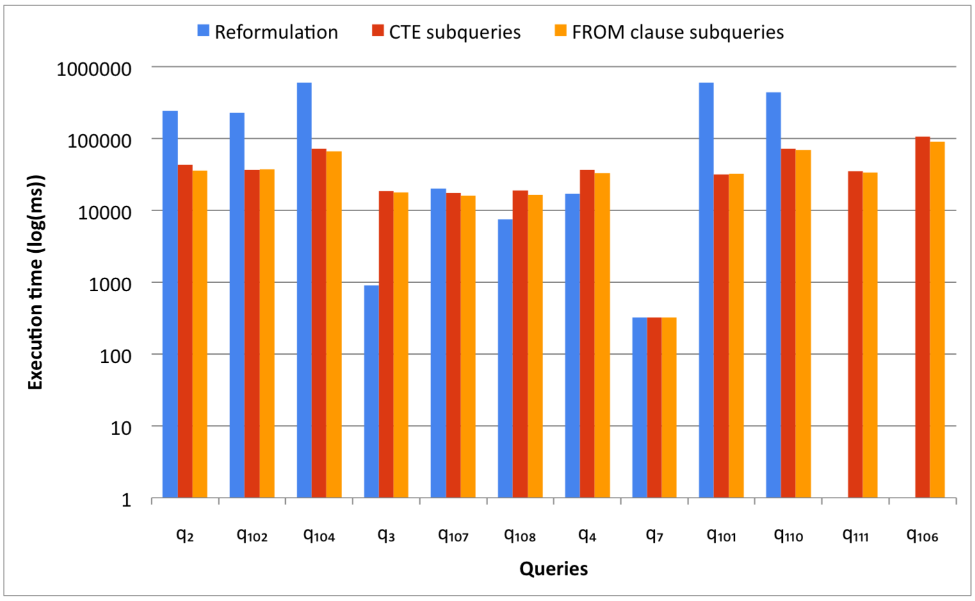


Fig. 4.20: GTRFA queries evaluation performance summary for $t = 10, \beta = \{1, 100.000\}, \alpha = 10$.

In most cases, we found that the recommended clustering was much more efficient than plain reformulation.

In a minority of cases, however, we found that reformulation may turn out to perform better than the recommended choice. This is related to the value of the t threshold: in some cases, if one query atom has (slightly) more than t reformulations, the algorithm will never cluster it together with another atom having more than one reformulation (in order for the possible cluster thus formed, not to go beyond the t threshold); however, of the other atom has a small number of reformulations, the clusterization thus avoided could have been an efficient one. This is exactly the case of query q_3, whose detailed analysis is provided in Figure 4.9.

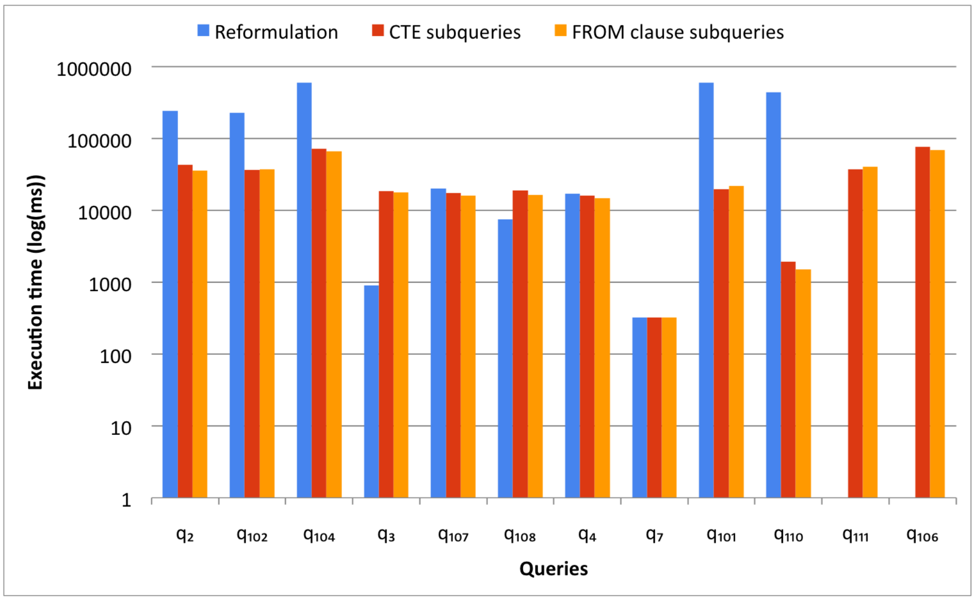


Fig. 4.21: GTRFA queries evaluation performance summary for $t = 100, \beta = \{1, 100.000\}, \alpha = 10$.

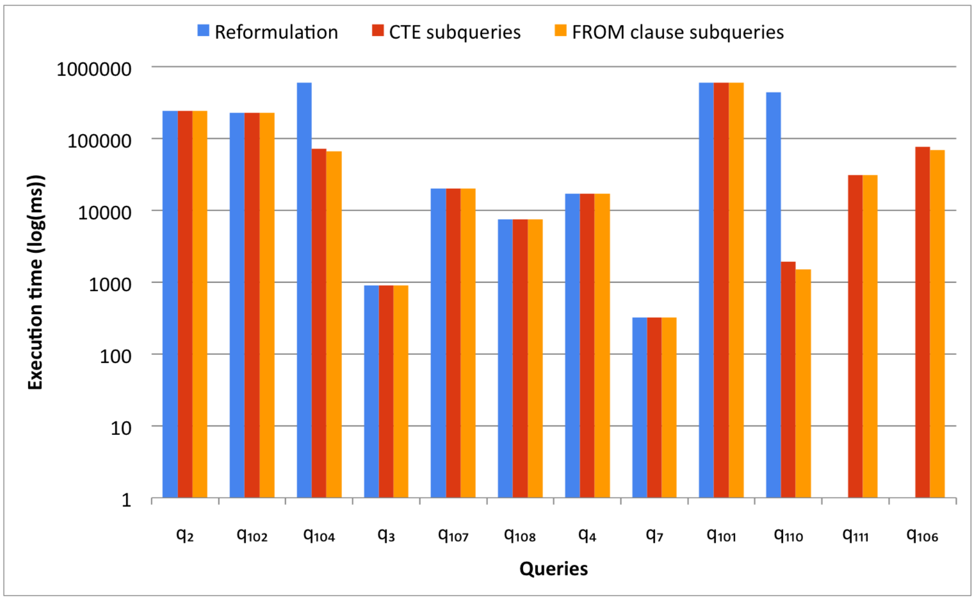


Fig. 4.22: GTRFA queries evaluation performance summary for $t = 1.000, \beta = \{1, 100.000\}, \alpha = 10$.

4.5.2 Algorithm GOFTRFA

Figures 4.23- 4.28 illustrate the performance of the clusterizations recommended by the algorithm for some of the queries considered (q_2-q_4). Unlike Algorithm 4 (mostly guided by the threshold), experiments shown (as expected) that the values of α and β have a bigger incidence in the recommended clusterization.

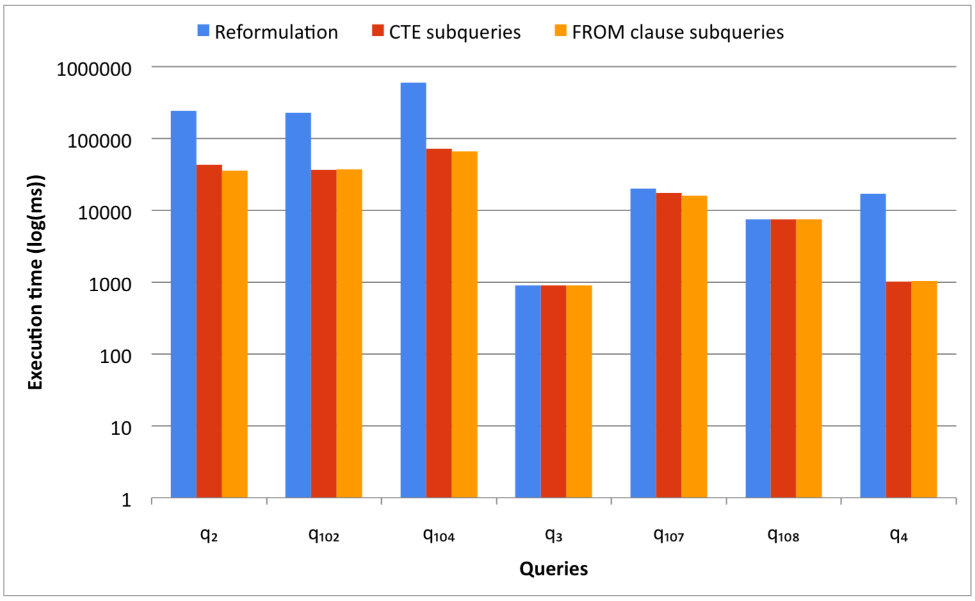


Fig. 4.23: GOFTRFA queries evaluation performance summary for $t = 10, \beta = 1, \alpha = 10$.

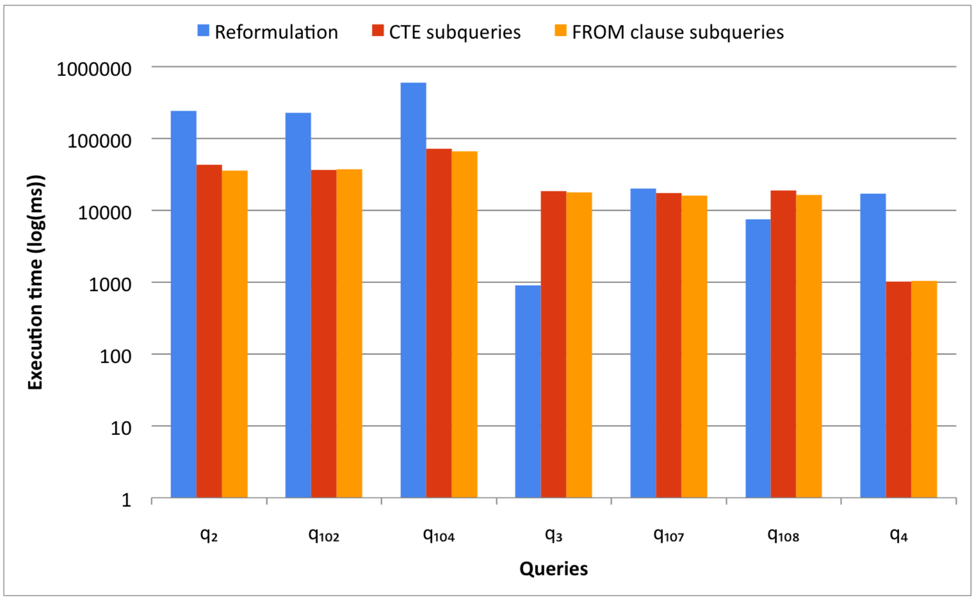


Fig. 4.24: GOFTRFA queries evaluation performance summary for $t = 10, \beta = 100.000, \alpha = 10$.

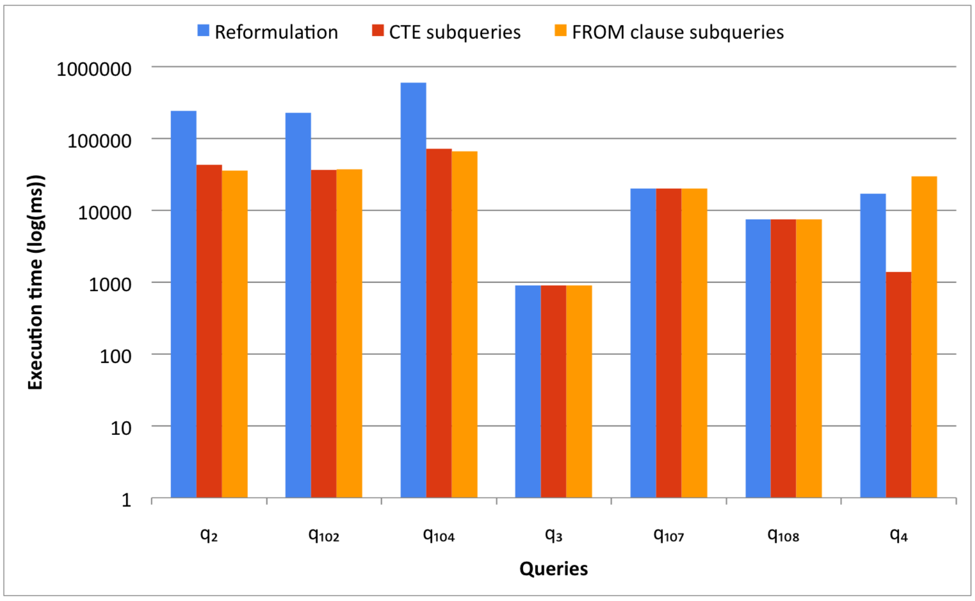


Fig. 4.25: GOFTRFA queries evaluation performance summary for $t = 100, \beta = 1, \alpha = 10$.

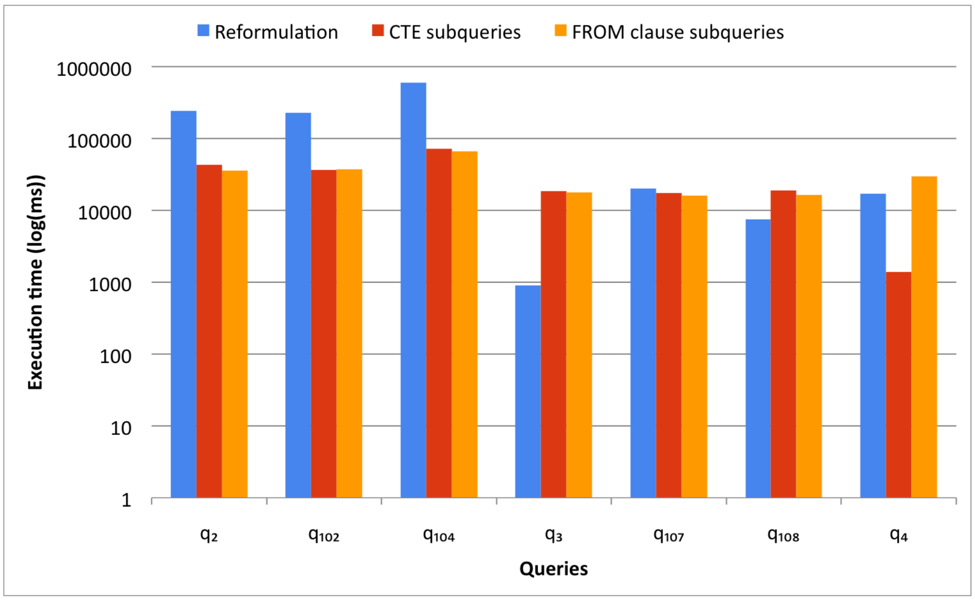


Fig. 4.26: GOFTRFA queries evaluation performance summary for $t = 100, \beta = 100.000, \alpha = 10$.

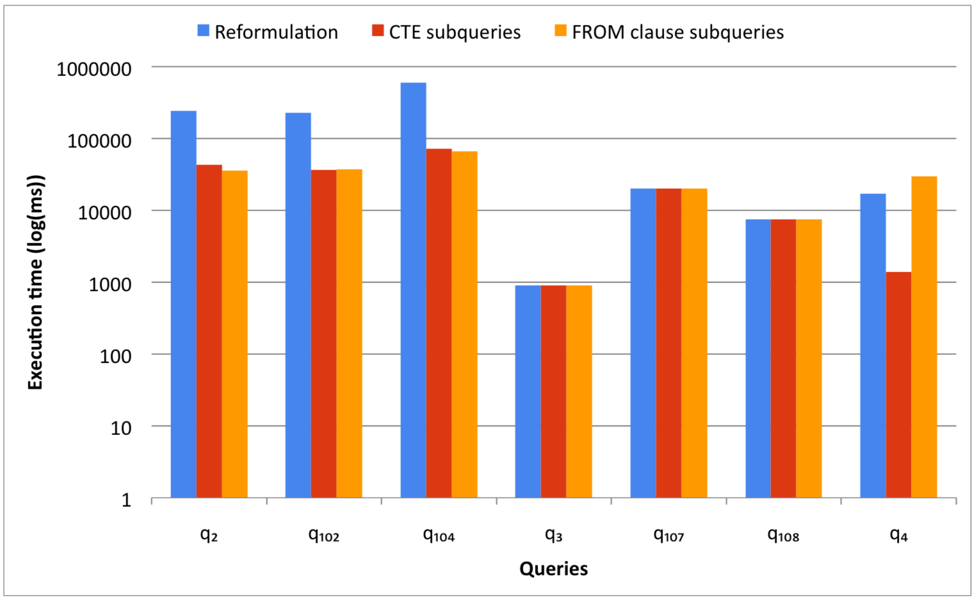


Fig. 4.27: GOFTRFA queries evaluation performance summary for $t = 1.000, \beta = 1, \alpha = 10$.

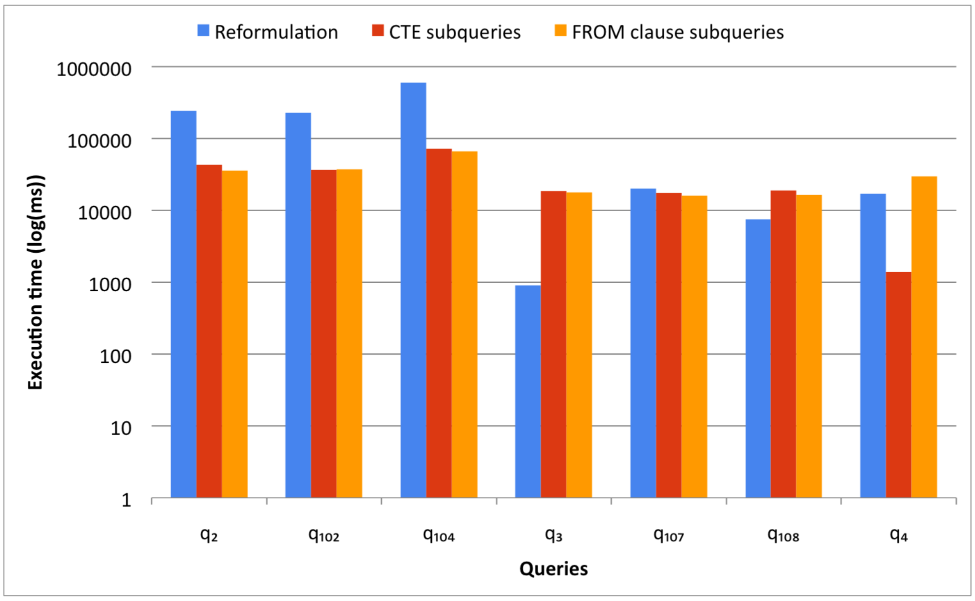


Fig. 4.28: GOFTRFA queries evaluation performance summary for $t = 1.000, \beta = 100.000, \alpha = 10$.

In most cases, we found that the recommended clusterization was much more efficient than plain reformulation, and typically either *the best clustering* we could manually produce (among those shown previously in this experimental study), or *better* than those.

Observe that in many cases Algorithm 5 recommends a single cluster containing all the atoms (plain reformulation) when this is more performant (according to the cost function and the input arguments).

4.5.3 Algorithm GOFTRFA-TS

As pointed in Section 3.5.5, early experiments showed that Algorithm 5 may lead to sub-optimal fragmentations as it often lead to (redundant) fragments (q_4 and q_{111}) and modifications were proposed.

Figures 4.29- 4.30 illustrate the performance of the clusterizations recommended by the algorithm for some of the queries considered (q_4-q_{106}). Two-atoms queries q_2-q_{108} are not considered because the recommended clusterizations are the same given by Algorithm 5.

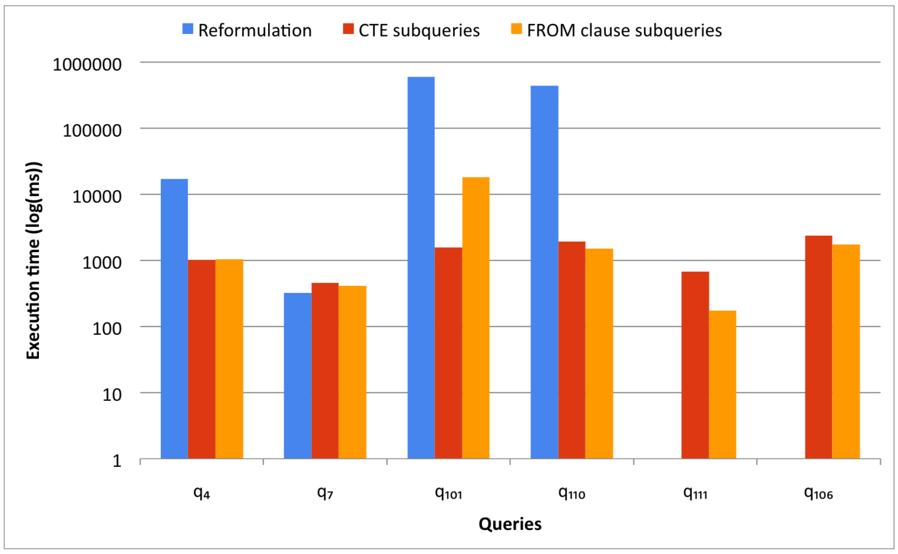


Fig. 4.29: GOFTRFA-TS queries evaluation performance summary for $t = \{10, 100, 1.000\}, \beta = 1, \alpha = 10$.

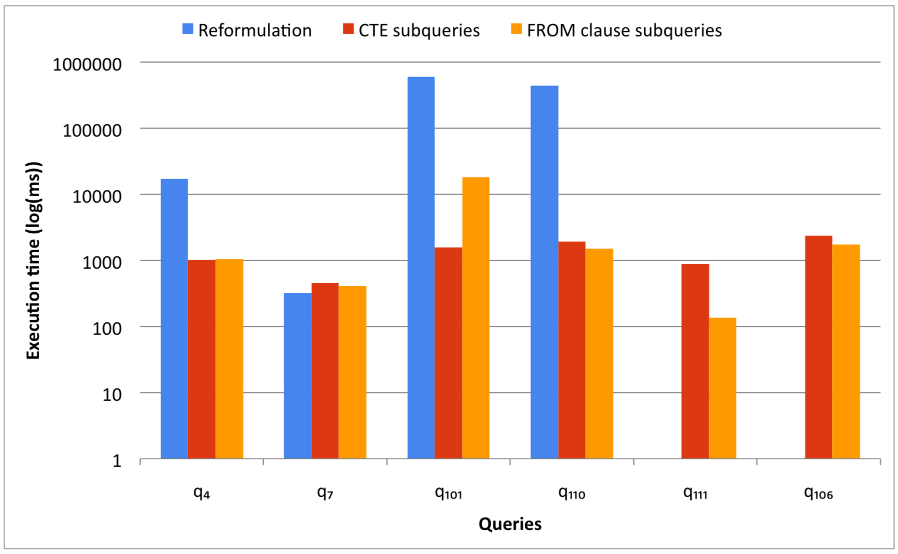


Fig. 4.30: GOFTRFA-TS queries evaluation performance summary for $t = \{10, 100, 1.000\}, \beta = 100.000, \alpha = 10$.

4.5.4 Summary

Finally, Figure 4.31 summarizes the performance of the clusterizations recommended by the algorithms for all the queries in Figures 4.6- 4.17. Observe that the reformulation execution time is missing for queries q_{111} and q_{106} since the reformulation of the queries could not be evaluated.

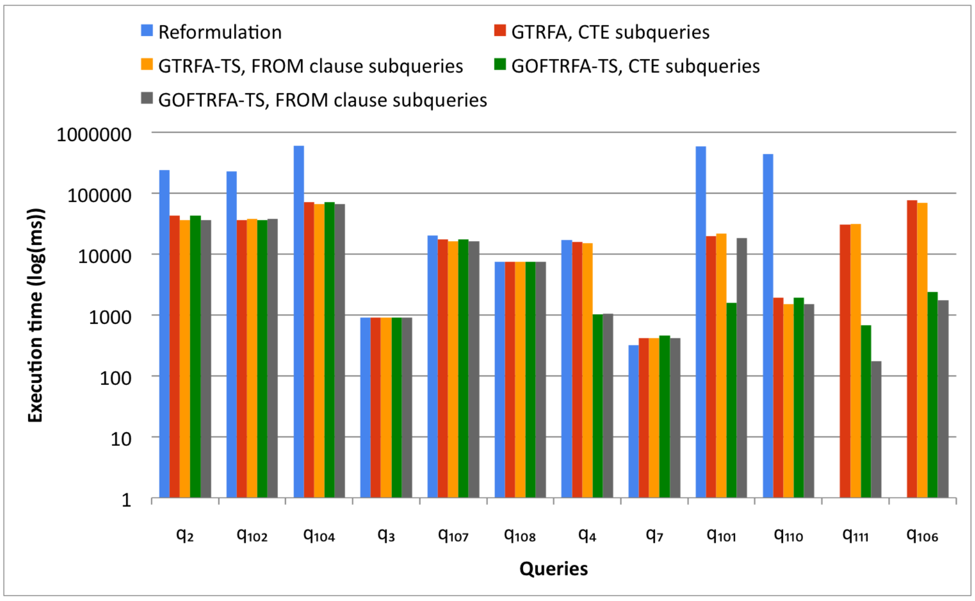


Fig. 4.31: Queries evaluation performance summary (using the known clusterization having provided the best performance).

Figure 4.31 shows that in most cases the recommended clusterization was much more efficient than plain reformulation, and typically either *the best clustering* we could manually produce (among those shown previously in this experimental study), or *better* than those.

Overall, Algorithm 5 (GOFTRFA) and its variations usually proved to lead to equal or more performant clusterizations than Algorithm 4 (GTRFA).

4.6 Experiment conclusion

Given a dataset and a BGP query, our experiments in Section 4.4 have shown that in some cases the use of a main query joining the SQL translation of the reformulated atoms is more efficient than using CTEs (Common Table Expressions) and joining those in the main query and vice versa in others. Opposed to what one would suppose, the two different syntactic ways of writing the query lead in some cases to big performance differences (this is the case of q_4, q_{101} and q_{111}, with some clusterizations).

The relational database approach to execute a query that joins relations, is to select the best order in which these relations should be joined in order to minimize the time consumed. Given a query $Q = R \bowtie S \bowtie T$, executing the join of the 3 relations in some order will take less time than $(R \bowtie S) \bowtie (S \bowtie T)$, or the join of any other combination of pairs of relations equivalent to Q. This observation leads us to considering non-overlapping clustering of queries. Reformulated BGP queries introduce a new element to take into account, i.e., the number of reformulations. The execution time, and therefore the execution plan, does not depend mainly on the number of results of each atom but also (and oftentimes more strongly) on the number of reformulations of the atoms. It is desirable, then, to include in several clusters atoms with one (or a few) reformulations and high selectivity, since it will reduce the intermediate number of results produced by the subquery without incrementing (or not drastically) the number of reformulations.

Our experiments in Section 4.4, comparing the two clusterization methods introduced in Section 3.2, shows that *fragmentation* increases efficiency w.r.t. *partition* when the query contains atoms with few reformulations and high selectivity (which are included in multiple fragments).

Section 4.5 shows that in most cases the clusterization recommended by our algorithms was much more efficient than plain reformulation, and typically either *the best clustering* we could manually produce (among those shown previously in this experimental study), or *better* than those. Experiments also shown that the arguments given (α, β and the threshold) have an important incidence in the recommended clusterization and therefore must be carefully selected.

Experimentation with real-life datasets shows that the number of tuples matching a triple pattern is in the thousands (and even tens and hundreds of thousands) while the number of reformulations of the same triple pattern usually are well below the hundred. Therefore, for the sake of increase the weight of the number of reformulations when calculating the cost, the factor β have to be much greater than the factor α.

5. CONCLUSION

This thesis is placed at the junction of two main classes of systems and algorithms.

Databases and in particular relational database management systems (RDBMSs) have been designed and thoroughly optimized in order to make them provide good and reliable performance when evaluating queries. The cornerstone of such efficient systems is the ability to optimize or rewrite queries into equivalent expressions which return the same results as the original query, regardless of the database they are evaluated upon. To make such equivalence reasoning possible (and efficient), strong simplifications have been typically consented to the expressive power of the query language, and of the constraints known to hold on the database. Thus, it can be (broadly) stated that databases traded quite an amount of expressive power in exchange for efficiency.

Knowledge bases have, in contrast, focused on (very) expressive languages for describing data and the constraints which hold on it. While numerous works have explored the tractability frontier in terms of features of the constraint language, work still remains to be done in order to implement scalable systems that would run efficiently on data sets sizes typically handled by databases. Thus, knowledge management has (broadly speaking) favored expressive power at the expense of efficiency and performance robustness.

A particular class of applications requiring knowledge management tools and techniques originates in the Semantic Web: heterogeneous Web data is represented in RDF, and endowed with constraints expressed in one of the many languages available. In our work, we have considered RDF Schema constraints. To help address the query evaluation needs brought by RDF interogation languages such as SPARQL, many works authored especially since 2007 have proposed taking advantage of databases and in particular RDBMSs, which have been heavily optimized for joins. While SPARQL queries pose unique challenges (they tend to be syntactically very large self-join queries over a single table etc.), properly tuned and engineered relational databases have been able to cope reasonably well with the issues raised by query evaluation, *ignoring schema and semantic constraints*. This can be seen as fitting in an RDBMS, the part of the Semantic Web data management problem that could easily fit – and disregarding the rest.

Once semantic constraints are taken into account, two main methods can be used to reflect their impact on query answering. First, one could attempt to derive all consequences of the constraints, or, equivalently in our context, to compute all the entailed or derived triples, and store them in the database next to the explicit data. This approach is also termed saturation. Once this is done, query answering is reduced to query evaluation and the facilities provided by an RDBMS (possibly one which has been best tuned to the needs of RDF queries) are sufficient. Alternatively, one can leave the database unchanged and reformulate queries asked against the data, so that the reformulated query, when evaluated (through standard RDBMS techniques) against the database, return the same answer as if the original query was evaluated on the saturated database. (We stress that query reformulation does not actually require to saturate the database.)

The trade-offs between the two are easy to see. Saturation increases the size of the database, and may require complex computations to *maintain* when the schema and/or the data change. In contrast, reformulation sometimes produces syntactically large queries, which may raise performance issues to database systems, even to efficient ones. This is because the complexity of query optimization is determined by the syntactic query complexity, and RDF queries are unusually large from an RDBMS perspective, where typical applications feature few relations, each with many attributes; this contrasts with the RDF setting of "narrow" relations of around three attributes.

The starting point of this thesis is the observation driven from the recent research [1] that reformulated query evaluation with an RDBMS remains relatively expensive even after intelligent indexes have been properly implemented in the database. Our goal has thus been to investigate which *alternatives* exist with respect to the method of evaluating such reformulated queries, and we have identified two main dimensions among which choices can be made:

1. First, we have considered different ways of *clustering a query* into sub-queries, such that each sub-query is reformulated and evaluated individually, before joining the results of all sub-queries. We have explored both query clusterizations which amount to partitions of the set of query atoms, and fragmentations with overlapping sets of atoms. Interestingly, we have observed that clusterizations with and without overlap can lead to performance gains of up to three orders of magnitude!

2. Second, we have exploited alternatives provided by modern SQL for expressing the relatively complex queries which result from reformulation. While in principle it could have been hoped that performance would not be impacted by the syntax used to specify the query, we have noted that in practice, the syntax does make a difference, not as strong as the query clusterization strategy, but a significant one nonetheless.

Thus, a first contribution of this thesis is to highlight the existence of this space of alternatives and to experimentally validate their interest through extensive experiments on real-life Semantic Web data sets. We also demonstrate that while partition-based strategies would seem intuitively more efficient (since they do not evaluate any query atom more than once), in practice, overlapping query fragments may lead to better performance, since a highly selective, few-reformulations atom should be joined (evaluated together) with several large-result atoms in order to diminish the number of results without increasing significantly the number of reformulations of the resulting sub-query.

The second contribution of the thesis consists of heuristic algorithms for recommending query clusterization strategies that are likely to provide the most important performance gains to the process of evaluating queries through reformulation. These algorithms exploit statistic information about the database and the schema, such as the number of triples that are likely to match a given sub-query, or the number of reformulations that it may have, in order to identify intelligent clusterization strategies. Our experiments have shown that our algorithms succeed in automatically identifying high-performance clusterization strategies, thus making good profit of the clusterization search space we identified. Moreover, fine tuning of the algorithms parameters (α, β and t) results in more efficient clusterizations.

Our experiments also shown that Algorithm 5 generally produce more efficient clusterizations than Algorithm 4.

Our approach and algorithms were conceived and deployed as standalone Java-based modules working on top of an off-the-shelf RDBMS, in particular the popular and efficient open source PostgreSQL system. Thus, we view the results of our thesis as a contribution towards bridging the gap between expressive models for Web data management and the powerful RDBMS tools which do not currently support the semantic needs of such applications.

6. RELATED WORK

To the best of our knowledge, the problem we are considering has only been addressed so far in [48], where the authors only consider the choice of evaluating all unions before any join. As we shown in the sequel, this is just one point in the search space that we consider. In contrast, our heuristic algorithms may make more complex choices, where some but not all the unions entailed by reformulations are pushed under joins, depending on the number of reformulations of each term as well as the number of triples matching the term.

As pointed out in Section 4.5.1, the statistics are currently gathered through SQL queries against a statistics database. The implementation of more efficient (in memory) data statistics structures may significantly sped up the query clusterization algorithms.

In this thesis we have tried various α, β and t (threshold reformulations). Moreover, aiming to strengthen the contribution, it is interesting to study how (and how much) can we automatically tune these parameters so that we get the best clusterization.

Linear programming had been successfully used solving optimization problems subject to constraints. The proposal of a model for the clusterization problem, and use of optimizers and LP solvers such as Gurobi [49] is subject to future research.

The harnessing of the opportunities presented in Section 3.6.2 for optimizing the execution of a clustered BGP query and study of the implications with respect to the clusterization techniques is also an interesting subject that might improve, even more, the efficacy of the BGP query evaluation.

BIBLIOGRAPHY

[1] François Goasdoué, Ioana Manolescu, and Alexandra Roatiş. Efficient query answering against dynamic rdf databases. In *EDBT*, 2013.

[2] The World Wide Web Consortium (W3C). Resource description framework. `http://www.w3.org/RDF`.

[3] The World Wide Web Consortium (W3C). SPARQL protocol and RDF query language. `http://www.w3.org/TR/rdf-sparql-query`.

[4] François Goasdoué, Ioana Manolescu, and Alexandra Roatiş. Bgp query answering against dynamic rdf databases (extended version, Inria report no. 8018). `http://hal.inria.fr/hal-00719641/`, 2012.

[5] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422. VLDB Endowment, 2007.

[6] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: Sextuple indexing for Semantic Web data management. *PVLDB*, 2008.

[7] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *SIGMOD*, 2009.

[8] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *WWW*, 2008.

[9] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable multi-query optimization for sparql. In *IEEE 28th International Conference on Data Engineering*, pages 666–677, 2012.

[10] 3store. `http://www.aktors.org/technologies/3store`.

[11] Apache jena^{TM}: Java framework for building semantic web applications. `http://jena.apache.org`.

[12] Sesame. `http://www.openrdf.org`.

[13] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of rdf data. *VLDBJ*, 19(1):91–113, 2010.

[14] Postgresql. `http://www.postgresql.org`.

[15] Tim Berners-Lee and Mark Fischetti. *Weaving the Web*. HarperCollins Publishers, 1999.

[16] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American Magazine*, pages 28–37, 2001.

[17] Tim Berners-Lee, Nigel Shadbolt, and Wendy Hall. The semantic web revisited. *IEEE Computer*, pages 96–101, 2006.

[18] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management*. Cambridge University Press, 2011.

[19] The World Wide Web Consortium (W3C). Rdf schema. `http://www.w3.org/TR/rdf-schema`.

[20] The World Wide Web Consortium (W3C). Owl web ontology language overview. `http://www.w3.org/TR/owl-features`.

[21] Hyeong Sik Kim, Padmashree Ravindra, and Kemafor Anyanwu. Scan-sharing for optimizing rdf graph pattern matching on mapreduce. In *IEEE Fifth International Conference on Cloud Computing*, 2012.

[22] Uniprot rdf. `http://dev.isb-sib.ch/projects/uniprot-rdf`.

[23] Semantic web challenge. billion triples track. `http://challenge.semanticweb.org`, 2008.

[24] Dbpedia. `http://wiki.dbpedia.org/About`.

[25] Metaweb technologies: Freebase data dumps. `https://developers.google.com/freebase/index`.

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

[27] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[28] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. Foundations of RDF databases. In *Reasoning Web*, pages 158–204, 2009.

[29] Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis. RDFS reasoning and query answering on DHTs. In *ISWC*, 2008.

[30] The World Wide Web Consortium (W3C). Resource description framework. `http://www.w3.org/DesignIssues/RDFnot.html`.

[31] The World Wide Web Consortium (W3C). Rdf semantics. `http://www.w3.org/TR/rdf-mt/`.

[32] François Picalausa, Yongming Luo, George H.L. Fletcher, Jan Hidders, and Stijn Vansummeren. A structural approach to indexing triples. In *The Semantic Web: Research and Applications*, pages 406–421, 2012.

[33] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[34] Barton library data. `http://simile.mit.edu/rdf-test-data/barton`.

[35] Dblp. `http://kdl.cs.umass.edu/data/dblp/dblp-info.html`.

[36] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[37] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases (rapport de recherche, Inria report no. 7738). `http://hal.archives-ouvertes.fr/docs/00/62/39/42/PDF/RR-7738.pdf`, 2011.

[38] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.

[39] Set partitions. `http://www.theory.csc.uvic.ca/~cos/inf/setp/SetPartitions.html`.

[40] M.C Er. A fast algorithm for generating set partitions. *The Computer Journal*, 31(3):283–284, 1988.

[41] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms.* Academic Press, 1975.

[42] Steven S Skiena. *The Algorithm Design Manual.* Springer, 2nd edition edition, 2008.

[43] Common table expressions. `http://www.postgresql.org/docs/9.2/static/queries-with.html`.

[44] Java^{TM}. `http://docs.oracle.com/javase/7/docs/`.

[45] The World Wide Web Consortium (W3C). N-triples. `http://www.w3.org/2001/sw/RDFCore/ntriples/`.

[46] Gnu bash. `http://www.gnu.org/software/bash/`.

[47] Raptor rdf syntax library. `http://librdf.org/raptor/`.

[48] Michaël Thomazo. Compact rewriting for existential rules. *IJCAI*, 2013.

[49] The gurobi optimizer. `http://www.gurobi.com`.

APPENDIX A

Funding

This work has been partially funded by Institut National de Recherche en Informatique et en Automatique.

APPENDIX B

RDF and RDFS vocabulary

RDF vocabulary [31]	**RDFS vocabulary [31]**
rdf:type	*rdfs:domain*
rdf:Property	*rdfs:range*
rdf:XMLLiteral	*rdfs:Resource*
rdf:nil	*rdfs:Literal*
rdf:List	*rdfs:Datatype*
rdf:Statement	*rdfs:Class*
rdf:subject	*rdfs:subClassOf*
rdf:predicate	*rdfs:subPropertyOf*
rdf:object	*rdfs:member*
rdf:first	*rdfs:Container*
rdf:rest	*rdfs:ContainerMembershipProperty*
rdf:Seq	*rdfs:comment*
rdf:Bag	*rdfs:seeAlso*
rdf:Alt	*rdfs:isDefinedBy*
rdf:_1	*rdfs:label*
rdf:_2	
...	
rdf:value	

RDF and RDFS axiomatic triples

RDF axiomatic triples [31]

⟨*τ τ rdf:Property*⟩
⟨*rdf:subject τ rdf:Property*⟩
⟨*rdf:property τ rdf:Property*⟩
⟨*rdf:object τ rdf:Property*⟩
⟨*rdf:first τ rdf:Property*⟩
⟨*rdf:rest τ rdf:Property*⟩
⟨*rdf:value τ rdf:Property*⟩
⟨*rdf:_1 τ rdf:Property*⟩
⟨*rdf:_2 τ rdf:Property*⟩
...
⟨*rdf:nil τ rdf:List*⟩

RDFS axiomatic triples [31]

⟨*rdf:type rdfs:domain rdfs:Resource*⟩
⟨*rdfs:domain rdfs:domain rdf:Property*⟩
⟨*rdfs:range rdfs:domain rdf:Property*⟩
⟨*rdfs:subPropertyOf rdfs:domain rdf:Property*⟩
⟨*rdfs:subClassOf rdfs:domain rdfs:Class*⟩
⟨*rdf:subject rdfs:domain rdf:Statement*⟩
⟨*rdf:predicate rdfs:domain rdf:Statement*⟩
⟨*rdf:object rdfs:domain rdf:Statement*⟩
⟨*rdfs:member rdfs:domain rdfs:Resource*⟩
⟨*rdf:first rdfs:domain rdf:List*⟩
⟨*rdf:rest rdfs:domain rdf:List*⟩
⟨*rdfs:seeAlso rdfs:domain rdfs:Resource*⟩
⟨*rdfs:isDefinedBy rdfs:domain rdfs:Resource*⟩
⟨*rdfs:comment rdfs:domain rdfs:Resource*⟩
⟨*rdfs:label rdfs:domain rdfs:Resource*⟩
⟨*rdf:value rdfs:domain rdfs:Resource*⟩
⟨*rdf:Alt rdfs:subClassOf rdfs:Container*⟩
⟨*rdf:Bag rdfs:subClassOf rdfs:Container*⟩
⟨*rdf:Seq rdfs:subClassOf rdfs:Container*⟩
⟨*rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property*⟩

⟨*rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso*⟩

⟨*rdf:XMLLiteral rdf:type rdfs:Datatype*⟩
⟨*rdf:XMLLiteral rdfs:subClassOf rdfs:Literal*⟩
⟨*rdfs:Datatype rdfs:subClassOf rdfs:Class*⟩

⟨*rdf:type rdfs:range rdfs:Class*⟩
⟨*rdfs:domain rdfs:range rdfs:Class*⟩
⟨*rdfs:range rdfs:range rdfs:Class*⟩
⟨*rdfs:subPropertyOf rdfs:range rdf:Property*⟩
⟨*rdfs:subClassOf rdfs:range rdfs:Class*⟩
⟨*rdf:subject rdfs:range rdfs:Resource*⟩
⟨*rdf:predicate rdfs:range rdfs:Resource*⟩
⟨*rdf:object rdfs:range rdfs:Resource*⟩
⟨*rdfs:range rdfs:member rdfs:rangerdfs:Resource*⟩
⟨*rdf:first rdfs:range rdfs:Resource*⟩
⟨*rdf:rest rdfs:range rdf:List*⟩
⟨*rdfs:seeAlso rdfs:range rdfs:Resource*⟩
⟨*rdfs:isDefinedBy rdfs:range rdfs:Resource*⟩
⟨*rdfs:comment rdfs:range rdfs:Literal*⟩
⟨*rdfs:label rdfs:range rdfs:Literal*⟩
⟨*rdf:value rdfs:range rdfs:Resource*⟩

⟨*rdf*:$_1$ *rdf:type rdfs:ContainerMembershipProperty*⟩
⟨*rdf*:$_1$ *rdfs:domain rdfs:Resource*⟩
⟨*rdf*:$_1$ *rdfs:range rdfs:Resource*⟩
⟨*rdf*:$_2$ *rdf:type rdfs:ContainerMembershipProperty*⟩
⟨*rdf*:$_2$ *rdfs:domain rdfs:Resource*⟩
⟨*rdf*:$_2$ *rdfs:range rdfs:Resource*⟩
...

Entailment rules

Simple entailment rules

Rule name [31]	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
se_1	$\mathtt{s\ p\ o}$	$\mathtt{s\ p}\ \_{:}b$, *where* $\_{:}b$ *identifies a blank node allocated to* $\mathtt{s}$ *by rule* se_1 *or* se_2
se_2	$\mathtt{s\ p\ o}$	$\_{:}b\ \mathtt{p\ o}$, *where* $\_{:}b$ *identifies a blank node allocated to* $\mathtt{s}$ *by rule* se_1 *or* se_2

Literal entailment rules

Rule name [31]	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
lg	$\mathtt{s\ p\ o}$	$\mathtt{s\ p}\ \_{:}b$, *where* $\_{:}b$ *identifies a blank node allocated to the literal* $\mathtt{s}$ *by this rule*
gl	$\mathtt{s\ p}\ \_{:}b$	$\mathtt{s\ p\ o}$, *where* $\_{:}b$ *identifies a blank node allocated to* $\mathtt{o}$ *by rule* lg

RDF entailment rules

Rule name [31]	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
rdf_1	$\mathtt{s\ p\ o}$	$\mathtt{p}\ \tau\ rdf{:}Property$
rdf_2	$\mathtt{s\ p\ o}, \mathtt{o}\ \tau\ rdf{:}XMLLiteral$	$\_{:}b\ \tau\ rdf{:}XMLLiteral$, *where* $\_{:}b$ *identifies a blank node allocated to* $\mathtt{o}$ *by rule* lg

RDFS entailment rules

Rule name [31]	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
$rdfs_1$	$\mathtt{s\ p\ o}, \mathtt{o\ p}\ rdf{:}Literal$	$\_{:}b\ \tau\ rdf{:}Literal$, *where* $\_{:}b$ *identifies a blank node allocated to* $\mathtt{o}$ *by rule* lg
$rdfs_2$	$\mathtt{p}\ \hookleftarrow_d\ \mathtt{s}, \mathtt{s}_1\ \mathtt{p}\ \mathtt{o}_1$	$\mathtt{s}_1\ \tau\ \mathtt{s}$
$rdfs_3$	$\mathtt{p}\ \hookrightarrow_r\ \mathtt{s}, \mathtt{s}_1\ \mathtt{p}\ \mathtt{o}_1$	$\mathtt{o}_1\ \tau\ \mathtt{s}$
$rdfs_{4a}$	$\mathtt{s\ p\ o}$	$\mathtt{s}\ \tau\ rdfs{:}Resource$
$rdfs_{4b}$	$\mathtt{s\ p\ o}$	$\mathtt{o}\ \tau\ rdfs{:}Resource$
$rdfs_5$	$\mathtt{p}\ \prec_{sp}\ \mathtt{p}_1, \mathtt{p}_1\ \prec_{sp}\ \mathtt{p}_2$	$\mathtt{p}\ \prec_{sp}\ \mathtt{p}_2$
$rdfs_6$	$\mathtt{s}\ \tau\ rdf{:}Property$	$\mathtt{s}\ \prec_{sp}\ \mathtt{s}$
$rdfs_7$	$\mathtt{p}_1\ \prec_{sp}\ \mathtt{p}_2, \mathtt{s}\ \mathtt{p}_1\ \mathtt{o}$	$\mathtt{s}\ \mathtt{p}_2\ \mathtt{o}$
$rdfs_8$	$\mathtt{s}\ \tau\ rdfs{:}Class$	$\mathtt{s}\ \prec_{sc}\ rdfs{:}Resource$
$rdfs_9$	$\mathtt{s}_1\ \prec_{sc}\ \mathtt{s}_2, \mathtt{s}\ \tau\ \mathtt{s}_1$	$\mathtt{s}\ \tau\ \mathtt{s}_2$
$rdfs_{10}$	$\mathtt{s}\ \tau\ rdfs{:}Class$	$\mathtt{s}\ \prec_{sc}\ \mathtt{s}$
$rdfs_{11}$	$\mathtt{s}\ \prec_{sc}\ \mathtt{s}_1, \mathtt{s}_1\ \prec_{sc}\ \mathtt{s}_2$	$\mathtt{s}\ \prec_{sc}\ \mathtt{s}_2$
$rdfs_{12}$	$\mathtt{s}\ \tau\ rdfs{:}ContainerMembershipProperty$	$\mathtt{s}\ \prec_{sp}\ rdfs{:}member$
$rdfs_{13}$	$\mathtt{s}\ \tau\ rdfs{:}Datatype$	$\mathtt{s}\ \prec_{sc}\ rdfs{:}Literal$

Extensional RDFS entailment rules

Rule name [31]	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
ext_1	$\mathtt{p}\ \hookleftarrow_d\ \mathtt{s}_1, \mathtt{s}_1\ \prec_{sc}\ \mathtt{s}$	$\mathtt{p}\ \hookleftarrow_d\ \mathtt{s}$
ext_2	$\mathtt{p}\ \hookrightarrow_r\ \mathtt{s}_1, \mathtt{s}_1\ \prec_{sc}\ \mathtt{s}$	$\mathtt{p}\ \hookrightarrow_r\ \mathtt{s}$
ext_3	$\mathtt{p}\ \prec_{sp}\ \mathtt{p}_1, \mathtt{p}_1\ \hookleftarrow_d\ \mathtt{s}$	$\mathtt{p}\ \hookleftarrow_d\ \mathtt{s}$
ext_4	$\mathtt{p}\ \prec_{sp}\ \mathtt{p}_1, \mathtt{p}_1\ \hookrightarrow_r\ \mathtt{s}$	$\mathtt{p}\ \hookrightarrow_r\ \mathtt{s}$
ext_5	$\tau\ \prec_{sp}\ \mathtt{p}, \mathtt{p}\ \hookleftarrow_d\ \mathtt{p}_1$	$rdfs{:}Resource\ \prec_{sc}\ \mathtt{p}_1$
ext_6	$\prec_{sc} \prec_{sp}\ \mathtt{p}, \mathtt{p}\ \hookleftarrow_d\ \mathtt{p}_1$	$rdfs{:}Class\ \prec_{sc}\ \mathtt{p}_1$
ext_7	$\prec_{sp} \prec_{sp}\ \mathtt{p}, \mathtt{p}\ \hookleftarrow_d\ \mathtt{p}_1$	$rdfs{:}Property\ \prec_{sc}\ \mathtt{p}_1$
ext_8	$\prec_{sc} \prec_{sp}\ \mathtt{p}_1, \mathtt{p}\ \hookrightarrow_r\ \mathtt{p}_1$	$rdfs{:}Class\ \prec_{sc}\ \mathtt{p}_1$
ext_9	$\prec_{sp} \prec_{sp}\ \mathtt{p}_1, \mathtt{p}\ \hookrightarrow_r\ \mathtt{p}_1$	$rdfs{:}Property\ \prec_{sc}\ \mathtt{p}_1$

APPENDIX C

SQL translation of reformulated queries

```
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.o=p1.s
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.o=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Document'
                                                                AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                AND p1.o='http://www.example.org/dblp/'
                                                                AND p0.s=p1.s
```

```sql
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Inproceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Phdthesis'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Collection'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Www'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Proceedings'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                  AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
```

```
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Book'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
                                                                 AND p0.o='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Mastersthesis'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
```

```
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#datatypeField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#objectField'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.o=p1.s
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.o=p1.s
```

```
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.o=p1.s
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.o
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.o
UNION
SELECT DISTINCT p0.o AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#crossref'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.o=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#isbn'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#author'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#series'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#volume'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#school'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#publisher'
                                                                 AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                 AND p1.o='http://www.example.org/dblp/'
                                                                 AND p0.s=p1.s
```

```
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#booktitle'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#note'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#title'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#chapter'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#year'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#month'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#journal'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0,triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                               AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                               AND p1.o='http://www.example.org/dblp/'
                                                               AND p1.s=p0.s
```

```
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#editor'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#pages'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#number'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#address'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cdrom'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#cite'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p0.s=p1.s
UNION
SELECT DISTINCT p0.s AS att_1 FROM triples AS p0, triples AS p1 WHERE p0.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#url'
                                                                  AND p1.p='http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#ee'
                                                                  AND p1.o='http://www.example.org/dblp/'
                                                                  AND p1.s=p0.s
```

Fig. 6.1: SQL translation of the reformulation of query q w.r.t. DBLP [35].