National Technical University of Athens

School of Electrical
and Computer Engineering

Division of Computer Science

Personalized, Semantic and Exploratory Data Analysis

PhD Thesis

of

Nikos Bikakis

Diploma in Electronic & Computer Engineering
Tech. Univ. of Crete (2009)

Athens, January 2016

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Personalized, Semantic and Exploratory Data Analysis

PhD Thesis

of

Nikos Bikakis

Diploma in Electronic & Computer Engineering
Tech. Univ. of Crete (2009)

Supervising Committee: T. Sellis
Y. Vassiliou
G. Stamou

Approved by the Examination Committee January 2016.

T. Sellis Prof. NTUA	Y. Vassiliou Prof. NTUA	G. Stamou Asst. Prof. NTUA
K. Kontogiannis Assoc. Prof. NTUA	T. Dalamagas Sr. Rsr. Athena	M. Koubarakis Prof. UOA

A. Deligiannakis
Assoc. Prof. TUC

Athens, January 2016

...

Nikos Bikakis
Electronic & Computer Engineer, PhD, NTUA

© 2016 - All rights reserved

Η παρούσα διατριβή εκπονήθηκε με χρηματοδότηση από τον Ειδικό Λογαριασμό Έρευνας Ε.Μ.Π.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Η έγκριση της διδακτορικής διατριβής από την Ανώτατη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε. Μ. Πολυτεχνείου δεν υποδηλώνει αποδοχή των γνωμών του συγγραφέα (Ν. 5343/1932, άρθρο 202).

Contents

List of Figures

List of Tables

PREFACE

This thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy, in the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Greece. The presented work has been carried out in the Knowledge and Database Systems Laboratory of NTUA and in the Institute for the Management of Information Systems of ATHENA Research Center.

There are several people who helped and supported me during the work on this thesis. First of all, I am grateful to Prof. Timos Sellis for his support, guidance and inspiration, without which this thesis could not have been accomplished. I also need to special thank Dimitris Sacharidis for his guidance, patience, and for always finding the time to discuss and review several parts of the work presented in this thesis. I would like also to thank Prof. Stavros Christodoulakis for his inspiration and his valuable advices and comments, as well as Prof. Yiannis Vasiliou for his support.

I am thankful that I had the pleasure to collaborate and discuss some of the issues presented in this thesis with: George Papastefanatos, Giorgos Giannopoulos, John Liagouris, Theodore Dalamagas, and Chrisa Tsinaraki. Also, I would like to thank Giorgos Giannopoulos, Akis Bikakis and Yiannis Karagiorgos for helping me improve the writing quality of my work.

I appreciate the contribution of (in alphabetically order): Karim Benouaret, Nektarios Gioldasis, Maria Krommyda, Melina Skourla, and Ioannis Stavrakantonakis who worked on issues related to this work. Additionally, I want to thank all the members of the Knowledge and Database Systems Laboratory and the Institute for the Management of Information Systems, for our discussions and for creating a very friendly "working" environment. Finally, I would like to thank my family for their support and patience throughout all these years.

nikos bikakis
Athens, January 2016

This dissertation was funded by the NTUA, Special Account for Research Grants, and the ATHENA Research Center under the research grant "Efficient Management for Big Data: Challenges, Methods and Techniques".

to my family

Abstract

In the *Big Data* era, systems in several application areas face significant efficiency and effectiveness challenges, due to the ever increasing *Volume*, *Variety* and *Velocity* of data. In this context, systems have to handle *vast amounts* of data in *real time* and operate in environments where *different users*, working on *different scenarios*, generate, explore and analyse *different forms* of data. To this direction, this thesis studies the development of *personalization*, *exploration* and *semantic* techniques for facilitating Big Data management and analysis. Specifically, we propose methods for: (a) scalable preference-aware data management and analysis; (b) efficient exploration and visualization over large datasets; and (c) semantic data integration, exploration and retrieval.

In the context of *personalized data analysis*, we study the following problems. First, we study the problem of finding and ranking objects that are preferable by a group of users based on their preferences. We propose an objective and fair interpretation of this problem. Based on this interpretation, we develop efficient index-based algorithms and we introduce an objective ranking scheme satisfying several theoretical properties. In the next problem, we thoroughly study the performance of some of the most well-known external memory skyline algorithms. Particularly, the considered algorithms are redesigned following a formal external memory model. Then, we propose numerous different design choices and we study the resulted algorithms' variations.

Regarding *exploratory data analysis* two problems are considered. In the first one we handle efficient on-the-fly visual exploration over large sets of data. For this problem we propose a multilevel framework that exploits a tree-based structure to hierarchically aggregate objects. Considering different exploration scenarios, we enable efficient exploration via incremental hierarchy construction and prefetching based on user interaction. Further, we provide on-the-fly efficient adaptation of the hierarchies based on user preferences. The second problem considers the exploration and visualization of very large graphs. We propose a new paradigm that allows efficient large graph visual exploration, similar to the exploration paradigm used in maps. Also, we present a disk-based scheme in order to index and store the visualized graph. In this setting, user's interactions are translated to efficient spatial operations. Finally, in order to visualize very large graphs, a partition-based visualization approach is introduced.

With respect to *semantic data analysis*, we focus on three problems. The first problem regards the integration between XML and Semantic Web. We present an interoperability framework that bridges the heterogeneity gap by exploiting a model for the expression of OWL–RDF/S to XML Schema mappings, a method for SPARQL to XQuery translation, and model which transforms XML Schemas into OWL ontologies. The second problem regards the use of semantics in document annotation and retrieval. For this problem we propose a semantic-based annotation model, as well as a learning method for recommending annotations. Finally, we introduce an effective retrieval method that enriches information retrieval techniques with semantics. In the last problem, we study the modelling and the exploration of evolving data, adopting the Linked Data paradigm. As a result, we propose a RDF-based change model and we develop a Linked Data infrastructure that allows exploration and retrieval over evolving data.

Περίληψη

Στην εποχή των *Μεγάλων Δεδομένων* (Big Data), τα συστήματα αντιμετωπίζουν σημαντικές προκλήσεις που σχετίζονται με την αποδοτικότητα και την αποτελεσματικότητα τους. Οι προκλήσεις αυτές απορρέουν κυρίως από τον *Όγκο*, την *Ετερογένεια* και την *Ταχύτητα* που χαρακτηρίζει τα δεδομένα σήμερα. Σε αυτό το πλαίσιο, τα σημερινά συστήματα πρέπει σε *πραγματικό χρόνο* να διαχειρίζονται *μεγάλους όγκους* δεδομένων, καθώς και να λειτουργούν σε περιβάλλοντα όπου *διαφορετικοί χρήστες* οι οποίοι εργάζονται σε *διαφορετικά σενάρια*, δημιουργούν, διερευνούν και αναλύουν *διαφορετικές μορφές* δεδομένων. Προς την κατεύθυνση αυτή, η παρούσα διατριβή μελετά την ανάπτυξη εξατομικευμένων, διερευνητικών και σημασιολογικών τεχνικών για την διαχείριση και ανάλυση Μεγάλων Δεδομένων. Πιο συγκεκριμένα, προτείνονται μέθοδοι για: (α) *κλιμακούμενη διαχείριση και ανάλυση δεδομένων βασισμένη σε προτιμήσεις χρηστών*; (β) *αποδοτική διερεύνηση και οπτικοποίηση μεγάλων συνόλων δεδομένων*; και (γ) *σημασιολογική ολοκλήρωση, διερεύνηση και ανάκτηση δεδομένων*.

Όσον αφορά στο πρώτο μέρος εργασιών, αντικείμενο έρευνας αποτέλεσε η *εξατομικευμένη ανάλυση δεδομένων*, όπου μελετήθηκαν τα ακόλουθα προβλήματα. Αρχικά μελετάται το πρόβλημα της εύρεσης και ταξινόμησης αντικείμενων τα οποία θεωρούνται προτιμητέα από μια ομάδα χρηστών, με βάση τις προτιμήσεις τους. Αποτέλεσμα της μελέτης, είναι η διατύπωση μιας αντικειμενική και δίκαιης ερμηνεία αυτού του προβλήματος. Με βάση αυτή την ερμηνεία, αναπτύχθηκαν αποδοτικοί αλγόριθμοι βασισμένοι σε ευρετήρια και προτάθηκε ένα σχήμα αντικειμενικής ταξινόμησης, το οποίο ικανοποιεί αρκετές θεωρητικές ιδιότητες. Σε επόμενο πρόβλημα, πραγματοποιήθηκε εκτεταμένη μελέτη και σύγκριση τεχνικών αποτίμησης ερωτημάτων κορυφογραμμής δευτερεύουσας μνήμη. Πιο συγκεκριμένα, ένα σύνολο αλγορίθμων κορυφογραμμής μοντελοποιήθηκαν και υλοποιήθηκαν σύμφωνα με το μοντέλο εξωτερικής μνήμης. Επιπλέον, για τους υπό εξέταση αλγόριθμους προτείνεται ένα σύνολο παραλλαγών. Η εκτεταμένη πειραματική μελέτη ανέδειξε νέα συμπεράσματα σχετικά με την σχεδίαση και την απόδοση των αλγορίθμων κορυφογραμμής.

Στο δεύτερο μέρος εργασιών, του οποίου αντικείμενο έρευνας αποτέλεσε η *διερευνητική ανάλυση δεδομένων*, μελετήθηκαν δυο προβλήματα. Πιο συγκεκριμένα, μελετήθηκε το πρόβλημα της αποδοτικής και άμεσης οπτικής διερεύνησης σε μεγάλα σύνολα δεδομένων. Αποτέλεσμα της μελέτης, είναι η ανάπτυξη ενός πλαισίου πολλαπλών επιπέδων βασιζόμενο σε μια δεντρική δομή η οποία πραγματοποιεί την ιεραρχική ομαδοποίηση των δεδομένων. Λαμβάνοντας υπόψη διαφορετικά σενάρια διερεύνησης, το πλαίσιο επιτρέπει την αποδοτική διερεύνηση μέσω της σταδιακής κατασκευής της ιεραρχίας, η οποία βασίζεται στην αλληλεπίδραση του χρήστη. Επιπλέον, περιγράφεται μια μέθοδος η οποία παρέχει αποδοτική και άμεση προσαρμογή των ιεραρχιών με βάση τις προτιμήσεις του χρήστη. Τέλος, παρουσιάζεται μια εκτεταμένη θεωρητική και πειραματική ανάλυση. Στο δεύτερο πρόβλημα μελετάται η διερεύνηση και οπτικοποίηση πολύ μεγάλων γράφων. Από αυτή τη μελέτη προέκυψε μια καινοτόμα μεθοδολογία η οποία επιτρέπει την αποδοτική οπτική διερεύνηση πολύ μεγάλων γράφων. Η μεθοδολογία που προτείνεται είναι παρόμοια με την μεθοδολογία που έχει υιοθετηθεί για την διερεύνηση γεωγραφικών χαρτών. Επιπλέον, παρουσιάζεται μια νέα τεχνική για την ευρετηρίαση και την αποθήκευση γράφων. Σε αυτό το πλαίσιο, οι αλληλεπιδράσεις του χρήστη μεταφράζονται σε αποδοτικούς χωρικούς τελεστές. Τέλος, προκειμένου να είναι εφικτή η οπτικοποίηση πολύ

μεγάλων γράφων, μια προσέγγιση η οποία βασίζεται σε κατάτμηση εισάγεται.

Όσον αφορά στο τρίτο μέρος εργασιών, αντικείμενο έρευνας αποτέλεσε η *σημασιολογική ανάλυση δεδομένων*, όπου μελετήθηκαν τα ακόλουθα προβλήματα. Αρχικά μελετήθηκε το πρόβλημα της ενοποίησης μεταξύ του Σημασιολογικού και του XML περιβάλλοντος. Για το πρόβλημα αυτό, παρουσιάζεται ένα διαλειτουργικό πλαίσιο το οποίο προσφέρει δυνατότητες μετάφρασης ερωτήσεων καθώς και αντιστοίχισης και μετασχηματισμού σχημάτων. Πιο συγκεκριμένα παρουσιάζονται: ένα μοντέλο για την διατύπωση αντιστοιχίσεων μεταξύ OWL-RDF/S και XML Schema, μια μέθοδος για την μετάφραση SPARQL ερωτήσεων σε XQuery, καθώς και ένα μοντέλο για τον μετασχηματισμό XML Schemas σε OWL οντολογίες. Το δεύτερο πρόβλημα αφορά στη χρήση της σημασιολογίας στην επισημείωση και ανάκτηση εγγράφων. Για το πρόβλημα αυτό προτείνεται ένα σημασιολογικό μοντέλο επισημειώσεων, καθώς και μια μέθοδο εκμάθησης για τη σύσταση επισημειώσεων. Τέλος, παρουσιάζεται μια αποτελεσματική μέθοδος ανάκτησης, η οποία εμπλουτίζει τεχνικές ανάκτηση πληροφορίας με σημασιολογία. Στο τελευταίο πρόβλημα, μελετάται η μοντελοποίηση και η εξερεύνηση εξελισσόμενων δεδομένων, υιοθετώντας τεχνικές Διασυνδεμένων Δεδομένων (Linked Data). Αποτέλεσμα αυτής της μελέτης είναι η περιγραφή ενός μοντέλου αλλαγών βασισμένο σε RDF, καθώς και η ανάπτυξη υποδομής Διασυνδεμένων Δεδομένων, η οποία επιτρέπει την διερεύνηση και ανάκτηση εξελισσόμενων δεδομένων.

Chapter 1

Introduction

Volume, *Variety*, and *Velocity* are the three main characteristics that are commonly used to describe *Big Data* era. In this context, several challenges emerge for the today's systems which should be able to effectively handle *massive amounts* of data in *real time*. Furthermore, modern systems have to operate in highly *heterogeneous* environments which are characterized by *different users* (e.g., interests, skills, characteristics) which work on a plethora of *different scenarios* and generate, explore and analyse data in *different forms*.

In the described setting, a large number of tasks related to data management and analysis are remarkably challenging for both users and systems. In what follows, we discuss the adoption of *personalization*, *exploration* and *semantic* techniques towards facilitating a series of tasks in the context of Big Data management and analysis, as well as describing major challenges of the respective settings.

The large volume of data in conjunction with the diverse users and scenarios make it extremely difficult for users to discover useful information according to their interests and needs. In this context, the adoption of *personalization* techniques is a fundamental part in the development of modern systems. Personalization techniques attempt to provide personalized services by exploiting users' profiles, preferences, interests, etc. Personalization systems help users to retrieve, organize and manage information, as well as to customize their overall experience, based on their preferences and the needs emerged by their usage context. Further, they allow service providers to increase their customer numbers, by improving user user experience and satisfaction.

Therefore, the need for methods that provide personalized services to users is ever increasing. Personalization techniques should be applicable to an even broader range of real-life applications. Particularly, such techniques should be able to consider various and complex types of objects, their relations, as well as possible existing schemas that describing them. Further, personalization systems should be able to address more complex tasks, such as scenarios where the systems should offer personalized services to groups of users. In this setting, personalization should take into account all users' information in order conclude to a result. Regarding performance, such methods should be able to efficiently handle very large numbers of both objects and users, where the users' information (e.g., preferences, needs, context) is constantly modified.

Another complementary direction towards improving user experience, consists in enabling users to effectively explore and analyse large and complex sets of data. The purpose of *data exploration* is to facilitate information perception and manipulation, knowledge extraction and inference. Exploratory systems are of great importance in nowadays' era, in which the volume and heterogeneity of available information make it extremely difficult for humans to explore and analyse data. The visualization techniques adopted by the majority of modern exploratory systems, provides users with an intuitive means to explore the content of the data, identify interesting patterns, infer correlations and causalities,

and supports sense-making activities that are not always possible with traditional data analysis techniques.

Exploration and visualization systems should be able to effectively handle billion objects datasets in real time. Further, modern systems should address issues related to visual presentation and user comprehension, such as overploting. Visualizing a large number of data objects is a challenging task; modern systems have to "*squeeze a billion records into a million pixels*". Finally, the requirement of scalable exploration must be coupled with the diversity of preferences and requirements posed by different users and tasks. Therefore, the systems should provide the user with the ability to customize the exploration experience based on her preferences and the requirements posed by the examined task.

Beyond the challenges emerged by the variety of users, tasks and contexts, additional challenges are posed by the variety that characterize today's data, systems and technologies. Thus, offering uniform access to heterogeneous data sources, and establishing interoperability between different systems and technologies, are of a great importance. Such challenges have lead to the development of the *Semantic Web* (SW) vision, which can be assumed as a collaborative environment where systems use and share data transparently. Semantics allow information to be described in a formal and explicit manner, where complex relations and concepts can be defined. The use of semantics can significantly improve systems' effectiveness in searching, sharing, and combining information in several application areas.

The SW is an open environment comprised of hundreds of large interlinked, and, often, user contributed datasets. It is founded on semantic technologies and standards for Web information representation and management. Since the SW applications and services have to coexist and interoperate with the existing applications that access legacy systems, it is essential for its infrastructure to enable transparent access to information stored in heterogeneous data sources. Additionally, SW users should not consider different data models, languages and technologies for developing their applications or accessing data sources. Therefore, it is crucial to develop methods that will provide interoperability between different infrastructures, as well as facilitate transparent access over heterogeneous data sources.

1.1 Contributions

This dissertation presents novel methods for managing and analysing data. Our work comprises three complementary directions towards facilitating Big Data management and analysis. Specifically, in the first direction, we propose scalable methods for preference-aware data management and analysis. In the second direction we implement techniques for efficient exploration and visualization over large sets of numeric, temporal and graph data. Finally, we propose methods for semantic-based data integration, access and retrieval. Our contributions include the following.

1. Considering a group of users, each specifying individual preferences over objects' attributes, we study the problem of finding and ranking the objects that are preferable by all users. We introduce and propose an objective and fair interpretation of this problem, based on Pareto-based aggregation. Considering this interpretation, we study three related problems. The first is to find the set of objects that are unanimously considered ideal by the entire group. In the second problem, we relax the requirement for unanimity and only require a percentage of users to agree. Then, in the third problem, we devise an effective ranking scheme based on our Pareto-based aggregation framework. To increase the efficiency when dealing with categorical attributes, we introduce an elegant transformation of categorical attribute

values into numerical values, which exhibits certain nice properties and allows us to use well-known index structures. Based on this transformation, we propose index-based algorithms which employ a space partitioning index to hierarchically group objects. Regarding the ranking problem, we theoretically study the behaviour of our ranking scheme and present a number of theoretical properties satisfied by our approach. Several interesting extensions of the aforementioned problems have also been studied, involving the following issues: multi-values attributes, non-tree hierarchies, subspace indexing, and objective attributes. A thorough experimental evaluation validates the efficiency and the effectiveness of the proposed methods. Particularly, our index-based techniques are an order of magnitude faster than baseline approaches, scaling up to millions of objects and thousands of users. Finally, our ranking scheme outperforms traditional rank aggregation methods in terms of precision and recall. The methods discussed and the results obtained can be found in [72, 71].

2. Skyline queries return the set of non-dominated objects, where an object is dominated, if there exists another with better values on all attributes. Considering the skyline problem, we thoroughly study some of the most well-known skyline algorithms. Although the studied algorithms are specifically designed to operate in external memory, little attention has been given to important implementation and design details, regarding memory and object management. For example, all algorithms assume that the unit of transfer during an I/O operation is the object, whereas in a real system is the block, i.e., a set of objects. Our work addresses such shortcomings by appropriately adapting the algorithms based on a realistic I/O model that better captures performance in a real system. Furthermore, we thoroughly study the core computational challenge in these algorithms, which is the management of in-memory objects. In particular, we introduce various policies for two basic tasks: traversing and evicting in-memory objects. Both tasks can have significant consequences for the number of required I/Os and for the CPU time. For the studied algorithms, we experimentally evaluate real disk-based implementations, rather than simulations, and derive useful conclusions for synthetic and real datasets. Particularly, we demonstrate that, in many cases and contrary to common belief, algorithms that pre-process (typically sort) the database are not faster. Finally, we perform an extensive study of our proposed policies, and reach the conclusion that in some settings (dimensionality and dataset distribution) these policies can reduce the number of dominance checks by more than 50%. The methods discussed and the results obtained have been published in [79].

3. We study the problem of on-the-fly visual exploration over large sets of data. As a result, we propose a framework that offers personalized multilevel exploration and analysis over numeric and temporal data. Our framework is built on top of a lightweight tree-based structure. This structure aggregates input objects into a hierarchical multilevel model. We define two versions of this model, that adopt different data organization approaches, well-suited to exploration and analysis context. When user preferences are not available, a method that considers input data characteristics, as well as environment settings (i.e., screen resolution, visualization parameters) estimates the best-fit parameters for hierarchy construction. On top of our model, we define different exploration scenarios, assuming various user exploration preferences. In order to enable efficient exploration over large datasets, our framework offers incremental hierarchy construction and prefetching based on user interaction. Further, it provides a method which dynamically and efficiently adapts an existing hierarchy to a new, taking into account a set of user preferences. A

thorough theoretical analysis, a performance evaluation and a user study, illustrate the efficiency and the effectiveness of the proposed framework. The framework is realized in a web-based prototype tool, called *synopsViz* that offers multilevel visual exploration and analysis over Linked Data datasets. The methods discussed and the results obtained can be found in [78, 80, 81].

4. We consider the problem of visualizing and exploring very large graphs. For this problem we introduce *graphVizdb*, a novel platform that offers large graph interactive visualization. The proposed platform is based on a new paradigm to interact with the visualized graph in a way that is similar to maps exploration. The platform derives its efficiency from a novel technique for indexing and storing the graph. The core idea is that in an offline preprocessing phase, the graph is drawn, using a layout algorithm. After drawing the graph, the coordinates assigned to its nodes (with respect to a Euclidean plane) are indexed with a spatial data structure, i.e., an R-tree, and stored in a database. In runtime, while the user is navigating over the graph, our system maps user operations into efficient spatial operations (i.e., window queries). Based on the coordinates, specific parts of the graph are retrieved from the database and sent to the user. In order to visualize very large graphs, we propose a partition-based graph visualization approach. In this setting, the input graph is divided into a set of smaller sub-graphs, then each sub-graph is visualized, the resulting graphs are organized and combined into a single graph. The sub-graphs are organized and combined based on a greedy algorithm that attempts to improve (e.g., minimize edges length, avoid overlaps) the resulting graph layout. We evaluate the performance of our methods using several real graph datasets. Our platform can offer efficient visual exploration over very large graphs (e.g., 300M edges/nodes) using commodity hardware. Finally, we develop a web-based prototype which supports four main operations: interactive navigation, multilevel exploration, subgraph management, and keyword search. The methods discussed and the results obtained have been published in [77, 76].

5. We study the problem of interoperability between the XML and Semantic Web worlds. As a result, we propose the *SPARQL2XQuery* framework which bridges the heterogeneity gap and creates an interoperable environment. The framework allows SPARQL queries posed over semantic sources to be automatically translated to XQuery expressions w.r.t. a set of predefined mappings. In more detail, we define a mapping model for the expression of OWL–RDF/S to XML Schema mappings, as well as a method for SPARQL to XQuery translation. Further, our framework supports both manual and automatic mapping specification between ontologies and XML Schemas. In the automatic mapping specification scenario, a transformational model which transforms XML Schemas into OWL ontologies has been defined, supporting the latest versions of the schema standards (i.e., XML Schema 1.1 and OWL 2). A thorough experimental evaluation has been conducted in order to demonstrate the efficiency of the proposed methods. The methods discussed and the results obtained can be found in [85, 82, 83, 353, 74, 75].

6. We consider the problem of semantic information retrieval. For this purpose, we propose the *GoNTogle* framework that supports ontology-based annotation and retrieval, in a fully collaborative environment. The framework provides both manual and automatic annotation mechanisms. Automatic annotation is based on a learning method that exploits user annotation history and textual information to automatically recommend annotations for new documents. Further, we introduce a hybrid retrieval method that provides a flexible combination of textual-based and

semantic-based retrieval in conjunction with advanced semantic-based operations. The proposed methods are implemented in a fully functional tool and their effectiveness is experimentally validated. The methods discussed and the results obtained have been published in [73, 183].

7. We consider the problem of modeling, publishing and exploring evolving life science data, adopting the Linked Data paradigm. We propose an RDF-based change model to capture versioned entities. Based on this model we convert legacy data from biological databases to evolving Linked Data. Our Linked Data infrastructure can assist biologists to explore biological entities and their evolution, and provides a SPARQL endpoint for applications and services to query historical miRNA data and track changes, their causes and effects. Our methods have been published in [142].

Table 1.1: Thesis overview

Part	Problem	Ch./Sec.	Results
I: Personalized	Preferable Objects under Group Preferences	2	[72, 71]
	External Memory Skyline Algorithms	3	[79]
II: Exploratory	Efficient Multilevel Exploration	4.1	[78, 80, 81]
	Scalable Graph Exploration	4.2	[77, 76]
III: Semantic	XML & Semantic Web Interoperability	5	[85, 82, 83, 353, 74, 75]
	Semantic Information Retrieval	6.1	[73, 183]
	Publishing & Exploring Evolving Linked Data	6.2	[142]

1.2 Outline

The thesis is organized into three main parts: (I) *Personalized*, (II) *Exploratory*, and (III) *Semantic* data analysis. We summarize the thesis structure as well as the results in Table 1.1. Further, Figure 1.1 illustrates the relations between the examined problems and the three main research areas.

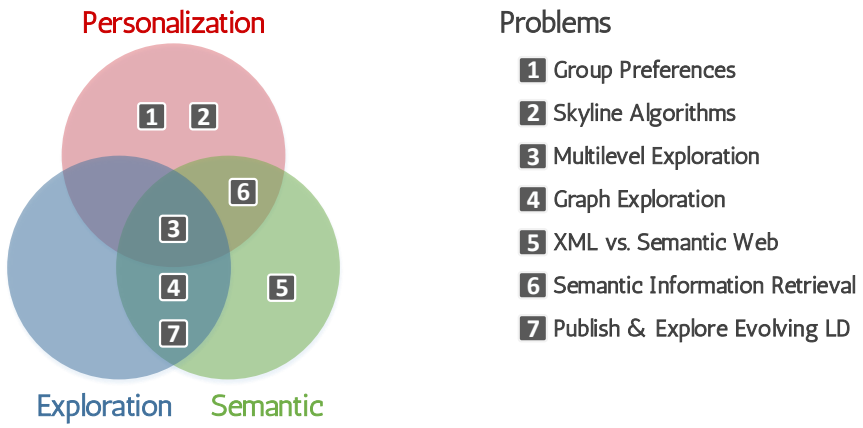


Figure 1.1: Problems vs. Research areas

In more detail, the remainder of this thesis is structured as follows. *Chapter 2* presents our methods for finding and ranking objects considering preferences from a group of users. *Chapter 3* studies external memory skyline algorithms. *Chapter 4* presents our approaches on visual exploration and analysis over large sets of data. *Chapter 5* presents our methods for establishing interoperability between the XML and Semantic Web worlds. *Chapter 6* introduces our approaches for semantic exploration and retrieval. Finally, *Chapter 7* concludes the discussion of this thesis summarizing its contributions, and presents possible extensions and ideas for future work.

Part I

Personalized Data Analysis

Chapter 2

Preferable Objects under Group Preferences

Considering a group of users, each specifying individual preferences over categorical attributes, the problem of determining a set of objects that are objectively preferable by all users is challenging on two levels. First, we need to determine the preferable objects based on the categorical preferences for each user, and second we need to reconcile possible conflicts among users' preferences. A naive solution would first assign degrees of match between each user and each object, by taking into account all categorical attributes, and then for each object combine these matching degrees across users to compute the total score of an object. Such an approach, however, performs two series of aggregation, among categorical attributes and then across users, which completely obscure and blur individual preferences. Our solution, instead of combining individual matching degrees, is to directly operate on categorical attributes, and define an objective Pareto-based aggregation for group preferences. Building on our interpretation, we tackle two distinct but relevant problems: finding the Pareto-optimal objects, and objectively ranking objects with respect to the group preferences. To increase the efficiency when dealing with categorical attributes, we introduce an elegant transformation of categorical attribute values into numerical values, which exhibits certain nice properties and allows us to use well-known index structures to accelerate the solutions to the two problems. In fact, experiments on real and synthetic data show that our index-based techniques are an order of magnitude faster than baseline approaches, scaling up to millions of objects and thousands of users.

2.1 Introduction

Recommender systems have the general goal of proposing objects (e.g., movies, restaurants, hotels) to a user based on her preferences. Several instances of this generic problem have appeared over the past few years in the Information Retrieval and Database communities; e.g., [20, 215, 92, 355]. More recently, there is an increased interest in *group recommender systems*, which propose objects that are well-aligned with the preferences of a set of users [219, 284, 108, 97]. Our work deals with a class of these systems, which we term *Group Categorical Preferences* (GCP), and has the following characteristics. (1) Objects are described by a set of categorical attributes. (2) User preferences are defined on a subset of the attributes. (3) There are multiple users with distinct, possibly conflicting, preferences. The GCP formulation may appear in several scenarios; for instance, colleagues arranging for a dinner at a restaurant, friends selecting a vacation plan for a holiday break.

To illustrate GCP, consider the following example. Assume that three friends in New York are looking for a restaurant to arrange a dinner. Suppose that, the three friends

Table 2.1: New York restaurants

	Attributes				
Restaurant	**Cuisine**	**Attire**	**Place**	**Price**	**Parking**
o_1	Eastern	Business casual	Clinton Hill	\$\$\$	Street
o_2	French	Formal	Time Square	\$\$\$\$	Valet
o_3	Brazilian	Smart Casual	Madison Square	\$\$	No
o_4	Mexican	Street wear	Chinatown	\$	No

Table 2.2: User preferences

	Preferences				
User	**Cuisine**	**Attire**	**Place**	**Price**	**Parking**
u_1	European	Casual	Brooklyn	\$\$\$	Street
u_2	French, Chinese	–	–	–	Valet
u_3	Continental	–	Time Square, Queens	–	–

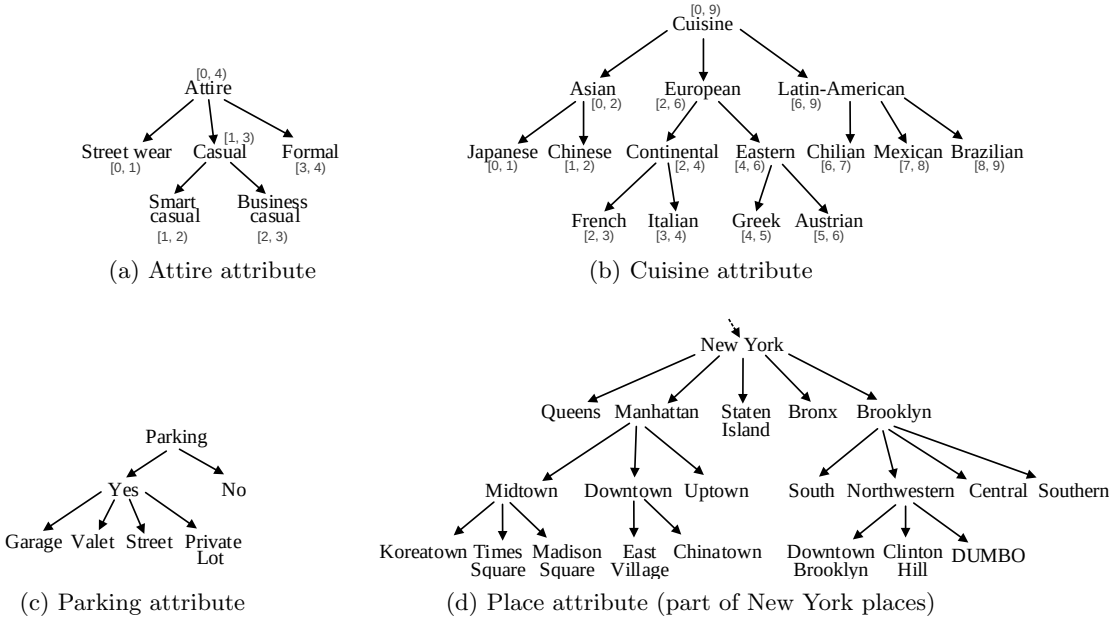
are going to use a Web site (e.g., Yelp[1]) in order to search and filter restaurants based on their preferences. Note that in this setting, as well as in other Web-based recommendation systems, categorical description are prevalent compared to numerical attributes. Assume a list of available New York restaurants, shown in Table 2.1, where each is characterized by five categorical attributes: *Cuisine*, *Attire*, *Place*, *Price* and *Parking*. In addition, Figure 2.1 depicts the hierarchies for these attributes. Attire and Parking are three-level hierarchies, Cuisine and Place are four-level hierarchies, and Price (not shown in Figure 2.1) is a two-levels hierarchy with four leaf nodes ($\$, ..., \$\$\$\$$). Finally, Table 2.2 shows the three friends' preferences. For instance, u_1 prefers European cuisine, likes to wear casual clothes, and prefers a moderately expensive (\$\$\$) restaurant in the Brooklyn area offering also street parking. On the other hand, u_2 likes French and Chinese cuisine, and prefers restaurants offering valet parking, without expressing any preference on attire, price and place.

Observe that if we look at a particular user, it is straightforward to determine his ideal restaurant. For instance, u_1 clearly prefers o_1, while u_2 clearly favors o_2. These conclusions per user can be reached using the following reasoning. Each preference attribute value $u_j.A_k$ is *matched* with the corresponding object attribute value $o_i.A_k$ using a matching function, e.g., the Jaccard coefficient, and a matching degree per preference attribute is derived. Given these degrees, the next step is to "compose" them into an overall matching degree between a user u_j and an object o_i. Note that several techniques are proposed for "composing" matching degrees; e.g., [284, 283, 237, 355, 122]. The simplest option is to compute a linear combination, e.g., the sum, of the individual degrees. Finally, alternative aggregations models (e.g., Least-Misery, Most-pleasure, etc.) could also be considered.

Returning to our example, assume that the matching degrees of user u_1 are: $\langle 1/2, 1/2, 1/6, 1, 1\rangle$ for restaurant o_1, $\langle 1/4, 0, 0, 0, 0\rangle$ for o_2, $\langle 0, 1/2, 0, 0, 0\rangle$ for o_3, and $\langle 0, 0, 0, 0, 0\rangle$ for o_4 (these degrees correspond to Jaccard coefficients computed as explained in Section 3.2.1). Note that for almost any "composition" method employed (except those that only, or strongly, consider the Attire attribute), o_1 is the most favorable restaurant for user u_1. Using similar reasoning, restaurant o_2, is ideal for both users u_2, u_3.

When all users are taken into consideration, as required by the GCP formulation, several questions arise. Which is the best restaurant that satisfies the entire group? And more importantly, *what does it mean to be the best restaurant*? A simple answer to the latter, would be the restaurant that has the highest "composite" degree of match to all

[1] www.yelp.com



(a) Attire attribute

(b) Cuisine attribute

(c) Parking attribute

(d) Place attribute (part of New York places)

Figure 2.1: Attribute hierarchies

users. Using a similar method as before, one can define a collective matching degree that "composes" the overall matching degrees for each user. This interpretation, however, enforces an additional level of "composition", the first being across attributes, and the second across users. These compositions obscure and blur the individual preferences per attribute of each user.

To some extent, the problem at the first "composition" level can be mitigated by requiring each user to manually define an importance weight among his specified attribute preferences. On the other hand, it is not easy, if possible at all, to assign weights to users, so that the assignment is fair. There are two reasons for this. First, *users may specify different sets of preference attributes*, e.g., u_1 specifies all five attributes, while u_2 only Cuisine and Parking. Second, even when considering a particular preference attribute, e.g., Cuisine, *users may specify values at different levels of the hierarchy*, e.g., u_1 specifies a European cuisine, while u_2 French cuisine, which is two levels beneath. Similarly, objects can also have attribute values defined at different levels. Therefore, any "composition" is bound to be *unfair*, as it may favor users with specific preferences and objects with detailed descriptions, and disfavor users with broader preferences and objects with coarser descriptions. This is an inherent difficulty of the GCP problem.

In this work, we introduce the *double Pareto-based aggregation*, which provides an objective and fair interpretation to the GCP formulation without "compositing" across preference attributes and users. Under this concept, the matching between a user and an object forms a *matching vector*. Each coordinate of this vector corresponds to an attribute and takes the value of the corresponding matching degree. The first Pareto-based aggregation is defined over attributes and induces a partial order on these vectors. Intuitively, *for a particular user*, the first partial order objectively establishes that an object is *better*, i.e., more preferable, than another, if it is better on all attributes. Then, the second Pareto-based aggregation, defined across users, induces the second and final partial order on objects. According to this order, an object is better than another, if it is more preferable according to *all users*.

Based on the previous interpretation of the GCP formulation, we seek to solve two distinct problems. The first, which we term the *Group-Maximal Categorical Objects* (GMCO) problem, is finding the set of maximal, or Pareto-optimal, objects according to the final

partial order. Note that since this order is only partial, i.e., two objects may not be comparable, there may exist multiple objects that are maximal; recall, that an object is maximal if there exists no other object that succeeds it in the order considered. In essence, it is the fact that this order is partial that guarantees objectiveness. The GMCO problem has been tackled in our previous work [71].

The second problem, which we term the *Group-Ranking Categorical Objects* (GRCO) problem, consists of determining an objective ranking of objects. Recall that the double Pareto-based aggregation, which is principal in guaranteeing objectiveness, induces only a partial order on the objects. On the other hand, ranking implies a total order among objects. Therefore, it is impossible to rank objects without introducing additional ordering relationships among objects, which however would sacrifice objectiveness. We address this contradiction, by introducing an objective *weak order* on objects. Such an order allows objects to share the same tier, i.e., ranked at the same position, but defines a total order among tiers, so that among two tiers, it is always clear which is better.

The GMCO problem has at its core the problem of finding maximal elements according to some partial order. Therefore, it is possible to adapt an existing algorithm to solve the core problem, as we discuss in Section 2.3.2. While there exists a plethora of main-memory algorithms, e.g., [251, 63], and more recently of external memory algorithms (termed skyline query processing methods), e.g., [98, 123, 300], they all suffer from two performance limitations. First, they need to compute the matching degrees and form the matching vectors for all objects, before actually executing the algorithm. Second, it makes little sense to apply index-based methods, which are known to be the most efficient, e.g., the state-of-the-art method of [300]. The reason is that the entries of the index depend on the specific instance, and need to be rebuilt from scratch when the user preferences change, even though the description of objects persists.

To address these limitations, we introduce a novel index-based approach for solving GMCO, which also applies to GRCO. The key idea is to index the set of objects that, unlike the set of matching vectors, remains constant across instances, and defer expensive computation of matching degrees. To achieve this, we apply a simple transformation of the categorical attribute values to intervals, so that each object translates to a rectangle in the Euclidean space. Then, we can employ a space partitioning index, e.g., an R*-Tree, to hierarchically group the objects. We emphasize that this transformation and index construction is a one-time process, whose cost is amortized across instances, since the index requires no maintenance, as long as the collection of objects persists. Based on the transformation and the hierarchical grouping, it is possible to efficiently compute upper bounds for the matching degrees for *groups* of objects. Therefore, for GMCO, we introduce an algorithm that uses these bounds to guide the search towards objects that are more likely to belong to the answer set, avoid computing unnecessary matching degrees.

For the GRCO problem, i.e., finding a (weak) order among objects, there has been a plethora of works on the related topic of combining/fusing multiple ranked lists, e.g., [174, 157, 42, 292, 170, 284]. However, such methods are not suitable for our GCP formulation. Instead, we take a different approach. We first relax the unanimity in the second Pareto-based aggregation, and require only a percentage $p\%$ of users to agree, resulting in the p-GMCO problem. This introduces a *pre-order* instead of a partial order, i.e., the induced relation lacks antisymmetry (an object may at the same time be before and after another). Then, building on this notion, we define tiers based on p values, and rank objects according to the tier they belong, which results in an objective weak order. To support the effectiveness of our ranking scheme, we analyze its behaviour in the context of rank aggregation and show that it posseses several desirable theoretical properties.

Table 2.3: Notation

Symbol	Description
$\mathcal{A}$, d	Set of attributes, number of attributes ($\lvert\mathcal{A}\rvert$)
A_k, $\lvert A_k\rvert$	Attribute, number of distinct values in A_k
$\mathcal{H}(A_k)$, $\lvert\mathcal{H}(A_k)\rvert$	Hierarchy of A_k, number of hierarchy nodes
$\mathcal{O}$, o_i	Set of objects, an object
$\mathcal{U}$, u_j	Set of users, a user
$o_i.A_k$, $u_j.A_k$	Value of attribute A_k in object o_i, user u_j
$o_i.I_k$, $u_j.I_k$	Interval representation of the value of A_k in o_i, u_j
m_i^j	Matching vector of object o_i to user u_j
$m_i^j.A_k$	Matching degree of o_i to user u_j on attribute A_k
$o_a \succ o_b$	Object o_a is collectively preferred over o_b
$\mathcal{T}$	The R*-Tree that indexes the set of objects
N_i, e_i	R*-Tree node, the entry for N_i in its parent node
$e_i.ptr$, $e_i.mbr$	The pointer to node N_i, the MBR of N_i
M_i^j	Maximum matching vector of entry e_i to user u_j
$M_i^j.A_k$	Maximum matching degree of e_i to user u_j on A_k

Contributions. The main contributions of this work are summarized as follows.

1. We introduce and propose an objective and fair interpretation of group categorical preference (GCP) recommender systems, based on double Pareto-based aggregation.
2. We introduce three problems in GCP systems, finding the group-maximal objects (GMCO), finding relaxed group-maximal objects (p-GMCO), and objectively ranking objects (GRCO).
3. We present a method for transforming the hierarchical domain of a categorical attribute into a numerical domain.
4. We propose index-based algorithms for all problems, which employ a space partitioning index to hierarchically group objects.
5. We theoretically study the behaviour of our ranking scheme and present a number of theoretical properties satisfied by our approach.
6. We present several extensions involving the following issues: multi-values attributes, non-tree hierarchies, subspace indexing, and objective attributes.
7. We conduct a thorough experimental evaluation using both real and synthetic data.

2.2 Group Categorical Preferences

Table 6.1 shows the most important symbols and their definition. Consider a set of d categorical *attributes* $\mathcal{A} = \{A_1, \ldots, A_d\}$. The domain of each attribute A_k is a *hierarchy* $\mathcal{H}(A_k)$. A hierarchy $\mathcal{H}(A_k)$ defines a tree, where a leaf corresponds to a lowest-level value, and an internal node corresponds to a category, i.e., a set, comprising all values within the subtree rooted at this node. The root of a hierarchy represents the category covering all lowest-level values. We use the symbol $|A_k|$ (resp. $|\mathcal{H}(A_k)|$) to denote the number of leaf (resp. all hierarchy) nodes. With reference to Figure 2.1, consider the "Cuisine" attribute. The node "Eastern" is a category and is essentially a shorthand for the set {"Greek", "Austrian"}, since it contains the two leaves, "Greek" and "Austrian".

Assume a set of *objects* $\mathcal{O}$. An object $o_i \in \mathcal{O}$ is defined over *all* attributes, and the value of attribute $o_i.A_k$ is one of the nodes of the hierarchy $\mathcal{H}(A_k)$. For instance, in Table 2.1, the value of the "Cuisine" attribute of object o_1, is the node "Eastern" in the hierarchy of Figure 2.1.

Table 2.4: Matching vectors

	User		
Restaurant	u_1	u_2	u_3
o_1	$\langle 1/2, 1/2, 1/6, 1, 1\rangle$	$\langle 0, 1, 1, 1, 0\rangle$	$\langle 0, 1, 0, 1, 1\rangle$
o_2	$\langle 1/4, 0, 0, 0, 0\rangle$	$\langle 1, 1, 1, 1, 1\rangle$	$\langle 1/2, 1, 1, 1, 1\rangle$
o_3	$\langle 0, 1/2, 0, 0, 0\rangle$	$\langle 0, 1, 1, 1, 0\rangle$	$\langle 0, 1, 0, 1, 1\rangle$
o_4	$\langle 0, 0, 0, 0, 0\rangle$	$\langle 0, 1, 1, 1, 0\rangle$	$\langle 0, 1, 0, 1, 1\rangle$

Further, assume a set of *users* $\mathcal{U}$. A user $u_i \in \mathcal{U}$ is defined over a *subset* of the attributes, and for each *specified* attribute $u_i.A_j$, its value in one of the hierarchy $\mathcal{H}(A_j)$ nodes. For all unspecified attributes, we say that user u_i is *indifferent* to them. Note that, an object (resp. a user) may has (resp. specify) multiple values for each attribute (see Section 2.6.1).

Given an object o_i, a user u_j, and a specified attribute A_k, the *matching degree* of o_i to u_j with respect to A_k, denoted as $m_i^j.A_k$, is specified by a *matching function* $\mathbf{M}$: $dom(A_k) \times dom(A_k) \rightarrow [0, 1]$. The matching function defines the *relation* between the user's preferences and the objects attribute values. For an indifferent attribute A_k of a user u_j, we define $m_i^j.A_k = 1$.

Note that, different matching functions can be defined per attribute and user; for ease of presentation, we assume a single matching function. Moreover, note that this function can be *any user defined function* operating on the cardinalities of intersections and unions of hierarchy attributes. For example, it can be the Jaccard coefficient, i.e., $m_i^j.A_k = \frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k \cup u_j.A_k|}$. The numerator counts the number of leaves in the intersection, while the denominator counts the number of leaves in the union, of the categories $o_i.A_k$ and $u_j.A_k$. Other popular choices are the Overlap coefficient: $\frac{|o_i.A_k \cap u_j.A_k|}{\min(|o_i.A_k|, |u_j.A_k|)}$, and the Dice coefficient: $2\frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k| + |u_j.A_k|}$.

In our running example, we assume the Jaccard coefficient. Hence, the matching degree of restaurant o_1 to user u_1 w.r.t. "Attire" is $\frac{|\text{"Business casual"} \cap \text{"Casual"}|}{|\text{"Business casual"} \cup \text{"Casual"}|} = \frac{|\{\text{"Business casual"}\}|}{|\{\text{"Business casual"}, \text{"Smart casual"}\}|} = \frac{1}{2}$, where we substituted "Casual" with the set {"Business casual", "Smart casual"}.

Given an object o_i and a user u_j, the *matching vector* of o_i to u_j, denoted as m_i^j, is a d-dimensional point in $[0, 1]^d$, where its k-th coordinate is the matching degree with respect to attribute A_k. Furthermore, we define the norm of the matching vector to be $\|m_i^j\| = \sum_{A_k \in \mathcal{A}} m_i^j.A_k$. In our example, the matching vector of restaurant o_1 to user u_1 is $\langle 1/2, 1/2, 1/6, 1, 1\rangle$. All matching vectors of this example are shown in Table 2.4.

2.3 The Group-Maximal Categorical Objects (GMCO) Problem

Section 2.3.1 introduces the GMCO problem, and Section 2.3.2 describes a straightforward baseline approach. Then, Section 2.3.3 explains a method to convert categorical values into intervals, and Section 2.3.4 introduces our proposed index-based solution.

2.3.1 Problem Definition

We first consider a particular user u_j and examine the matching vectors. The *first Pareto-based aggregation* across the attributes of the matching vectors, induces the following partial and strict partial "preferred" orders on objects. An object o_a is *preferred* over o_b, for user u_j, denoted as $o_a \succeq^j o_b$ iff for every specified attribute A_k of the user it holds that $m_a^j.A_k \geq m_b^j.A_k$. Moreover, object o_a is *strictly preferred* over o_b, for user u_j, denoted as

$o_a \succ^j o_b$ iff o_a is preferred over o_b and additionally there exists a specified attribute A_k such that $m_a^j.A_k > m_b^j.A_k$. Returning to our example, consider user u_1 and its matching vector $\langle 0, 1/2, 0, 0, 0\rangle$ for o_3, and $\langle 0, 0, 0, 0, 0\rangle$ for o_4. Observe that o_3 is strictly preferred over o_4.

We now consider all users in $\mathcal{U}$. The *second Pareto-based aggregation* across users, induces the following strict partial "collectively preferred" order on objects. An object o_a is *collectively preferred* over o_b, if o_a is preferred over o_b for all users, and there exists a user u_j for which o_a is strictly preferred over o_b. From Table 2.4, it is easy to see that restaurant o_1 is collectively preferred over o_3, because o_1 is preferred by all three users, and strictly preferred by user u_1.

Given the two Pareto-based aggregations, we define the *collectively maximal objects* in $\mathcal{O}$ with respect to users $\mathcal{U}$, as the set of objects for which there exists no other object that is collectively preferred over them. In our running example, o_1 and o_2 objects are both collectively preferred over o_3 and o_4. There exists no object which is collectively preferred over o_1 and o_2, and thus are the collectively maximal objects. We next formally define the GMCO problem.

Problem 1. [GMCO] Given a set of objects $\mathcal{O}$ and a set of users $\mathcal{U}$ defined over a set of categorical attributes $\mathcal{A}$, the *Group-Maximal Categorical Objects* (GMCO) problem is to find the collectively maximal objects of $\mathcal{O}$ with respect to $\mathcal{U}$.

2.3.2 A Baseline Algorithm (BSL)

The GMCO problem can be transformed to a maximal elements problem, or a skyline query, where the input elements are the matching vectors. Note, however, that the GMCO problem is different than computing the conventional skyline, i.e., over the object's attribute values.

The *Baseline* (BSL) method, whose pseudocode is depicted in Algorithm 1, takes advantage of this observation. The basic idea of BSL is for each object o_i (*loop in line 1*) and for all users (*loop in line 2*), to compute the matching vectors m_i^j (*line 3*). Subsequently, BSL constructs a $|\mathcal{U}|$-dimensional tuple r_i (*line 4*), so that its j-th entry is a composite value equal to the matching vector m_i^j of object o_i to user u_j. When all users are examined, tuple r_i is inserted in the set $\mathcal{R}$ (*line 5*).

The next step is to find the maximal elements, i.e., compute the skyline over the records in $\mathcal{R}$. It is easy to prove that tuple r_i is in the skyline of $\mathcal{R}$ iff object o_i is a collectively maximally preferred object of $\mathcal{O}$ w.r.t. $\mathcal{U}$. Notice, however, that due to the two Pareto-based aggregations, each attribute of a record $r_i \in \mathcal{R}$ is also a record that corresponds to a matching vector, and thus is partially ordered according to the preferred orders defined in Section 3.2.1. Therefore, in order to compute the skyline of $\mathcal{R}$, we need to apply a skyline algorithm (*line 6*), such as [98, 300, 188].

Algorithm 1. BSL

Input: objects $\mathcal{O}$, users $\mathcal{U}$
Output: CM the collectively maximal
Variables: $\mathcal{R}$ set of intermediate records

```
1 foreach o_i ∈ O do
2     foreach u_j ∈ U do
3         compute m_i^j
4         r_i[j] ← m_i^j
5     insert r_i into R
6 CM ← SkylineAlgo (R)
```

Computational Complexity. The computational cost of BSL is the sum of two parts. The first is computing the matching degrees, which takes $O(|\mathcal{O}| \cdot |\mathcal{U}|)$ time. The second is computing the skyline, which requires $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ comparisons, assuming a quadratic time skyline algorithms is used. Therefore, BSL takes $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ time.

2.3.3 Hierarchy Transformation

This section presents a simple method to transform the hierarchical domain of a categorical attribute into a numerical domain. The rationale is that numerical domains can be ordered, and thus tuples can be stored in multidimensional index structures. The index-based algorithm of Section 2.3.4 takes advantage of this transformation.

Consider an attribute A and its hierarchy $\mathcal{H}(A)$, which forms a tree. We assume that any internal node has at least two children; if a node has only one child, then this node and its child are treated as a single node. Furthermore, we assume that there exists an ordering, e.g., the lexicographic, among the children of any node that totally orders all leaf nodes.

The hierarchy transformation assigns an interval to each node, similar to labeling schemes such as [25]. The i-th leaf of the hierarchy (according to the ordering) is assigned the interval $[i-1, i)$. Then, each internal node is assigned the smallest interval that covers the intervals of its children. Figure 2.1 depicts the assigned intervals for all nodes in the two car hierarchies.

Following this transformation, the value on the A_k attribute of an object o_i becomes an interval $o_i.I_k = [o_i.I_k^-, o_i.I_k^+)$. The same holds for a user u_j. Therefore, the transformation translates the hierarchy $H(A_k)$ into the numerical domain $[0, |A_k|]$.

An important property of the transformation is that it becomes easy to compute matching degrees for metrics that are functions on the cardinalities of intersections or unions of hierarchy attributes. This is due to the following properties, which use the following notation: for a closed-open interval $I = [\alpha, \beta)$, define $\|I\| = \beta - \alpha$.

Proposition 1. For objects/users x, y, and an attribute A_k, let $x.I_k$, $y.I_k$ denote the intervals associated with the value of x, y on A_k. Then the following hold:

(1) $|x.A_k| = \|x.I_k\|$

(2) $|x.A_k \cap y.A_k| = \|x.I_k \cap y.I_k\|$

(3) $|x.A_k \cup y.A_k| = \|x.I_k\| + \|y.I_k\| - \|x.I_k \cap y.I_k\|$

Proof. For a leaf value $x.A_k$, it holds that $|x.A_k| = 1$. By construction of the transformation, $\|x.I_k\| = 1$. For a non-leaf value $x.A_k$, $|x.A_k|$ is equal to the number of leaves under $x.A_k$. Again, by construction of the transformation, $\|x.I_k\|$ is equal to the smallest interval that covers the intervals of the leaves under $x.A_k$, and hence equal to $|x.A_k|$. Therefore for any hierarchy value, it holds that $x.A_k = \|x.I_k\|$.

Then, the last two properties trivially follow. The third holds since $|x.A_k \cup y.A_k| = |x.A_k| + |y.A_k| - |x.A_k \cap y.A_k|$. ∎

2.3.4 An Index-based Algorithm (IND)

This section introduces the *Index-based* GMCO (IND) algorithm. The key ideas of IND are: (1) apply the hierarchy transformation, previously described, and index the resulting intervals, and (2) define upper bounds for the matching degrees of a group of objects, so as to guide the search and quickly prune unpromising objects.

We assume that the set of objects $\mathcal{O}$ and the set of users $\mathcal{U}$ are transformed so that each attribute A_k value is an interval I_k. Therefore, each object (and user) defines a

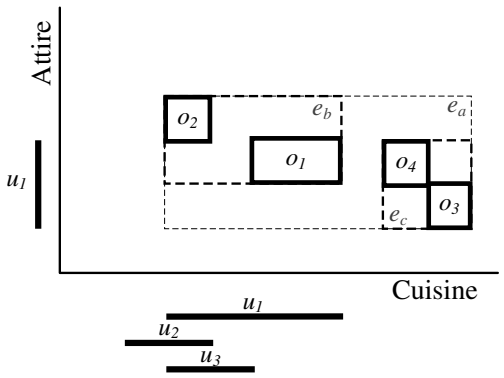


Figure 2.2: Transformed objects and users (considering Cuisine & Attire attributes)

(hyper-)rectangle on the d-dimensional cartesian product of the numerical domains, i.e., $[0, |A_1|) \times \cdots \times [0, |A_d|)$.

Figure 2.2 depicts the transformation of the objects and users shown in Tables 2.1 & 2.2, considering only the attributes Cuisine and Attire. For instance, object o_1 is represented as the rectangle $[4, 6) \times [2, 3)$ in the "Cuisine" × "Attire" plane. Similarly, user u_1 is represented as two intervals, $[2, 6)$, $[1, 3)$, on the transformed "Cuisine", "Attire" axes, respectively.

The IND algorithm indexes the set of objects in this d-dimensional space. In particular, IND employs an R*-Tree $\mathcal{T}$ [58], which is well suited to index rectangles. Each $\mathcal{T}$ node corresponds to a disk page, and contains a number of entries. Each entry e_i comprises (1) a pointer $e_i.ptr$, and (2) a Minimum Bounding Rectangle (MBR) $e_i.mbr$. A leaf entry e_i corresponds to an object o_i, its pointer $o_i.ptr$ is *null*, and $e_i.mbr$ is the rectangle defined by the intervals of o_i. A non-leaf entry e_i corresponds to a child node N_i, its pointer $e_i.ptr$ contains the address of N_i, and $e_i.mbr$ is the MBR of (i.e., the tightest rectangle that encloses) the MBRs of the entries in N_i.

Due to its enclosing property, the MBR of an entry e_i encloses all objects that are stored at the leaf nodes within the $\mathcal{T}$ subtree rooted at node N_i. It is often helpful to associate an entry e_i with all the objects it encloses, and thus treat e_i as a group of objects.

Consider a $\mathcal{T}$ entry e_i and a user $u_j \in \mathcal{U}$. Given only the information within entry e_i, i.e., its MBR, and not the contents, i.e., its enclosing objects, at the subtree rooted at N_i, it is impossible to compute the matching vectors for the objects within this subtree. However, it is possible to derive an *upper bound* for the matching degrees of any of these objects.

We define the *maximum matching degree* $M_i^j.A_k$ of entry e_i on user u_j w.r.t. specified attribute A_k as the highest attainable matching degree of any object that may reside within $e_i.mbr$. To do this we first need a way to compute lower and upper bounds on unions and intersections of a user interval with an MBR.

Proposition 2. Fix an attribute A_k. Consider an object/user x, and let I_x, denote the interval associated with its value on A_k. Also, consider another object/user y whose interval I_y on A_k is contained within a range R_y. Given an interval I, $\delta(I)$ returns 0 if I is empty, and 1 otherwise. Then the following hold:

(1) $1 \leq |y.A_k| \leq \|R_y\|$

(2) $\delta(I_x \cap R_y) \leq |x.A_k \cap y.A_k| \leq \|I_x \cap R_y\|$

(3) $\|I_x\| + 1 - \delta(I_x \cap R_y) \leq |x.A_k \cup y.A_k| \leq \|I_x\| + \|R_y\| - \delta(I_x \cap R_y)$

Proof. Note that for the object/user y with interval I_y on A_k, it holds that $I_y \subseteq R_y$.

(1) For the left inequality of the first property, observe that value $y.A_k$ is a node

that contains at least one leaf, hence $1 \leq |y.A_k|$. Furthermore, for the right inequality, $|y.A_k| = \|I_y\| \leq \|R_y\|$.

(2) For the left inequality of the second property, observe that the value $x.A_k \cap y.A_k$ contains either at least one leaf when the intersection is not empty, and no leaf otherwise. The right inequality follows from the fact that $I_x \cap I_y \subseteq I_x \cap R_y$.

(3) For the left inequality of the third property, assume first that $I_x \cap I_y = \varnothing$; hence $\delta(I_x \cap R_y) = 0$. In this case, it holds that $\|I_x \cup I_y\| = \|I_x\| + \|I_y\|$. By the first property, we obtain $1 \leq \|I_y\|$. Combining the three relations, we obtain the left inequality. Now, assume that $I_x \cap I_y \neq \varnothing$; hence $\delta(I_x \cap R_y) = 1$. In this case, it also holds that $\|I_x\| \leq \|I_x \cup I_y\|$, and the left inequality follows.

For the right inequality of the third property, observe that $\|I_x \cup I_y\| = \|I_x\| + \|I_y\| - \|I_x \cap I_y\|$. By the first property, we obtain $\|I_y\| \leq \|R_y\|$, and $-\|I_x \cap I_y\| \leq -\delta(I_x \cap R_y)$, by the second. The right inequality follows from combining these three relations. ■

Then, defining the maximum matching degree reduces to appropriately selecting the lower/upper bounds for the specific matching function used. For example, consider the case of the Jaccard coefficient, $\frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k \cup u_j.A_k|}$. Assume e_i is a non-leaf entry, and let $e_i.R_k$ denote the range of the MBR on the A_k attribute. We also assume that $u_j.I_k$ and $e_i.R_k$ overlap. Then, we define $M_i^j.A_k = \frac{\|e_i.R_k \cap u_j.I_k\|}{\|u_j.I_k\|}$, where we have used the upper bound for the intersection in the enumerator and the lower bound for the union in the denominator, according to Proposition 2. For an indifferent to the user attribute A_k, we define $M_i^j.A_k = 1$. Now, assume that e_i is a leaf entry, that corresponds to object o_i. Then the maximum matching degree $M_i^j.A_k$ is equal to the matching degree $m_i^j.A_k$ of o_i to u_j w.r.t. A_k.

Computing maximum matching degrees for other metrics is straightforward. In any case, the next lemma shows that an appropriately defined maximum matching degree is an upper bound to the matching degrees of all objects enclosed in entry e_i.

Proposition 3. The maximum matching degree $M_i^j.A_k$ of entry e_i on user u_j w.r.t. specified attribute A_k is an upper bound to the highest matching degree in the group that e_i defines.

Proof. The maximum matching degree is an upper bound from Proposition 2. ■

In analogy to the matching vector, the *maximum matching vector* M_i^j of entry e_i on user u_j is defined as a d-dimensional vector whose k-th coordinate is the maximum matching degree $M_i^j.A_k$. Moreover, the norm of the maximum matching vector is $\|M_i^j\| = \sum_{A_k \in \mathcal{A}} M_i^j.A_k$.

Next, consider a $\mathcal{T}$ entry e_i and the entire set of users $\mathcal{U}$. We define the *score* of an entry e_i as $score(e_i) = \sum_{u_j \in \mathcal{U}} \|M_i^j\|$. This score quantifies how well the enclosed objects of e_i match against all users' preferences. Clearly, the higher the score, the more likely that e_i contains objects that are good matches to users.

Algorithm Description. Algorithm 2 presents the pseudocode for IND. The algorithm maintains two data structures: a heap H which stores $\mathcal{T}$ entries sorted by their score, and a list CM of collectively maximal objects discovered so far. Initially the list CM is empty (*line 1*), and the root node of the R*-Tree is read (*line 2*). The score of each root entry is computed and all entries are inserted in H (*line 3*). Then, the following process (*loop in line 4*) is repeated as long as H has entries.

The H entry with the highest score, say e_x, is popped (*line 5*). If e_x is a non-leaf entry (*line 6*), it is *expanded*, which means that the node N_x identified by $e_x.ptr$ is read (*line*

Algorithm 2. IND

Input: R*-Tree $\mathcal{T}$, users $\mathcal{U}$
Output: CM the collectively maximal
Variables: H a heap with $\mathcal{T}$ entries sorted by $score()$

```
1  CM ← ∅
2  read T root node
3  insert in H the root entries
4  while H is not empty do
5      e_x ← pop H
6      if e_x is non-leaf then
7          N_x ← read node e_x.ptr
8          foreach e_i ∈ N_x do
9              pruned ← false
10             foreach u_j ∈ U do
11                 compute M_i^j
12             foreach o_a ∈ CM do
13                 if ∀A_j : m_a^j ≥ M_i^j ∧ ∃A_k : m_a^k ≻ M_i^k then
14                     pruned ← true
15                     break
16             if not pruned then
17                 insert e_i in H
18     else
19         o_x ← e_x
20         result ← true
21         foreach o_a ∈ CM do
22             if o_a ≻ o_x then
23                 result ← false
24                 break
25         if result then
26             insert o_x in CM
```

7). For each child entry e_i of N_x (*line 8*), its maximum matching degree M_i^j with respect to every user $u_j \in \mathcal{U}$ is computed (*lines 10–11*). Then, the list CM is scanned (*loop in line 12*). If there exists an object o_a in CM such that (1) for each user u_j, the matching vector m_a^j of o_a is better than M_i^j, and (2) there exists a user u_k so that the matching vector m_a^k of o_a is strictly better than M_i^k, then entry e_i is discarded (*lines 13–15*). It is straightforward to see (from Proposition 3) that if this condition holds, e_i cannot contain any object that is in the collectively maximal objects, which guarantees IND' correctness. When the condition described does not hold (*line 16*), the score of e_i is computed and e_i is inserted in H (*line 17*).

Now, consider the case that e_x is a leaf entry (*line 18*), corresponding to object o_x (*line 19*). The list CM is scanned (*loop in line 21*). If there exists an object that is collectively preferred over o_x (*line 22*), it is discarded. Otherwise (*line 25–26*), o_x is inserted in CM.

The algorithm terminates when H is empty (*loop in line 4*), at which time the list CM contains the collectively maximal objects.

Computational Analysis. IND performs object to object comparisons as well as object to non-leaf entries. Since there are at most $|\mathcal{O}|$ non-leaf entries, IND performs $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ comparisons in the worst case. Further it computes matching degrees on the fly at a cost of $O(|\mathcal{O}| \cdot |\mathcal{U}|)$. Overall, IND takes $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ time, the same as BSL. However, in practice IND is more than an order of magnitude faster than BSL (see Section 2.7).

Example. We demonstrate IND, using our running example, as depicted in Figure 2.2. The four objects are indexed by an R*-Tree, whose nodes are drawn as dashed rectangles. Objects o_1, o_2 are grouped in entry e_b, while o_3, o_4 in entry e_c. Entries e_b and e_c are the entries of the root e_a. Initially, the heap contains the two root entries, $H = \{e_b, e_c\}$. Entry e_b has the highest score (the norm of its maximum matching vector is the largest), and is thus popped. The two child entries o_1 and o_2 are obtained. Since the list CM is empty, no child entry is pruned and both are inserted in the heap, which becomes $H = \{o_1, o_2, e_c\}$. In the next iteration, o_2 has the highest score and is popped. Since this is a leaf entry, i.e., an object, and CM is empty, o_2 is inserted in the result list, $CM = \{o_2\}$. Subsequently, o_1 is popped and since o_2 is not collectively preferred over it, o_1 is also placed in the result list, $CM = \{o_2, o_1\}$. In the final iteration, entry e_c is popped, but the objects in CM are collectively preferred over both e_c child. Algorithm IND concludes, finding the collectively maximal $CM = \{o_2, o_1\}$.

2.4 The p-Group-Maximal Categorical Objects (p-GMCO) Problem

Section 2.4.1 introduces the p-GMCO problem, and Section 2.4.2 presents an adaptation of the BSL method, while Section 2.4.3 introduces an index-based approach.

2.4.1 Problem Definition

As the number of users increases, it becomes more likely that the users express very different and conflicting preferences. Hence, it becomes difficult to find a pair of objects such that the users unanimously agree that one is worst than the other. Ultimately, the number of maximally preferred objects increases. This means that the answer to an GMCO problem with a large set of users becomes less meaningful.

The root cause of this problem is that we require unanimity in deciding whether an object is collectively preferred by the set of users. The following definition relaxes this requirement. An object o_a is *p-collectively preferred* over o_b, denoted as $o_a \succ_p o_b$, iff there exist a subset $\mathcal{U}_p \subseteq \mathcal{U}$ of at least $\lceil \frac{p}{100} \cdot |\mathcal{U}| \rceil$ users such that for each user $u_i \in \mathcal{U}_p$ o_a is preferred over o_b, and there exists a user $u_j \in \mathcal{U}_p$ for which o_a is strictly preferred over o_b. In other words, we require only p% of the users votes to decide whether an object is universally preferred. Similarly, the *p-collectively maximal objects* of $\mathcal{O}$ with respect to users $\mathcal{U}$, is defined as the set of objects in $\mathcal{O}$ for which there exists no other object that is p-collectively preferred over them. The above definitions give rise to the p-GMCO problem.

Problem 2. [**p-GMCO**] Given a set of objects $\mathcal{O}$ and a set of users $\mathcal{U}$ defined over a set of categorical attributes $\mathcal{A}$, the *p-Group-Maximal Categorical Objects* (p-GMCO) problem is to find the p-collectively maximal objects of $\mathcal{O}$ with respect to $\mathcal{U}$.

Following the definitions, we can make a number of important observations, similar to those in the k-dominance notion [112]. First, if an object is collectively preferred over some other object, it is also p-collectively preferred over that same object for any p. As a result, an object that is p-collectively maximal is also collectively maximal for any p. In other words, the answer to the p-GMCO problem is a *subset* of the answer to the corresponding GMCO.

Second, consider an object o that is not p-collectively maximal. Note that it is possible that no p-collectively maximal object is p-collectively preferred over o. As a result checking if o is a result by considering only the p-collectively maximal objects may lead to false positives. Fortunately, it holds that there must exist a collectively maximal object that

is p-collectively preferred over o. So it suffices to check o against the collectively maximal objects only (and not just the subset that is p-collectively maximal).

Example. Consider the example in Tables 2.1 & 2.2. If we consider $p = 100$, we require all users to agree if an object is collectively preferred. So, the 100-collectively maximal objects are the same as the collectively maximal objects (i.e., o_1, o_2). Let's assume that $p = 60$; i.e., $\lceil \frac{60}{100} \cdot 3 \rceil = 2$ users. In this case, only the restaurant o_2 is 60-collectively maximal, since, o_2 is 60-collectively preferred over o_1, if we consider the set of users u_2 and u_3. Finally, if $p = 30$, we consider only one user in order to decide if an object is collectively preferred. In this case, the 30-collectively maximal is an empty set, since o_2 is 30-collectively preferred over o_1, if we consider either user u_2 or u_3, and also o_1 is 30-collectively preferred over o_2, if we consider user u_1.

2.4.2 A Baseline Algorithm (p-BSL)

Based on the above observations, we describe a baseline algorithm for the p-GMCO problem, based on BSL. Algorithm 3 shows the changes with respect to the BSL algorithm; all omitted lines are identical to those in Algorithm 1. The p-BSL algorithm first computes the collectively maximal objects applying BSL (*lines 1–6*). Then, each collectively maximal object, is compared with all other collectively maximal objects (*lines 7–14*). Particularly, each object o_i is checked whether there exists another object in CM that is p-collectively preferred over o_i (*lines 10–12*). If there is no such object, object o_i is inserted in p-CM (*line 14*). When the algorithm terminates, the set p-CM contains the p-collectively maximal objects.

Algorithm 3. p-BSL

```
   Input: objects 𝒪, users 𝒰
   Output: p-CM the p-collectively maximal
   Variables: CM the collectively maximal
   ⋮
 7 foreach o_i ∈ CM do
 8     inpCM ← true
 9     foreach o_j ∈ CM\o_i do
10         if o_j ≻_p o_i then
11             inpCM ← false
12             break;
13     if inpCM then
14         insert o_i to p-CM
```

Computational Analysis. Initially, the algorithm is computing the collectively maximal set using the BSL algorithm (*lines 1–6*), which requires $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$. Then, finds the p-collectively maximal objects (*lines 7–14*), performing in the worst case $O(|\mathcal{O}|^2)$ comparisons. Since, in worst case we have that $|CM| = |\mathcal{O}|$. Therefore, the computational cost of Algorithm 3 is $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$.

2.4.3 An Index-based Algorithm (p-IND)

We also propose an extension of IND for the p-GMCO problem, termed p-IND. Algorithm 4 shows the changes with respect to the IND algorithm; all omitted lines are identical to those in Algorithm 2.

In addition to the set CM, p-IND maintains the set p-CM of p-collectively maximal objects discovered so far (*line 1*). It holds that p-$CM \subseteq CM$; therefore, an object may appear in both sets. When a leaf entry o_x is popped (*line 19*), it is compared against

Algorithm 4. p-IND

```
Input: R*-Tree 𝒯, users 𝒰
Output: p-CM the p-collectively maximal
Variables: H a heap with 𝒯 entries sorted by score(), CM the collectively maximal object
1  CM ← ∅; p-CM ← ∅
⋮
4  while H is not empty do
   ⋮
18    else
19       o_x ← e_x
20       inCM ← true; inpCM ← true
21       foreach o_a ∈ CM do
22          if o_a ≻ o_x then
23             inCM ← false
24             break
25          if inpCM then
26             if o_a ≻_p o_x then
27                inpCM ← false
28          if o_a ∈ p-CM then
29             if o_x ≻_p o_a then
30                remove o_a from p-CM
31       if inCM then
32          insert o_x to CM
33          if inpCM then
34             insert o_x to p-CM
```

each object o_a in CM (*lines 21–30*) in three checks. First, the algorithm checks if o_a is collectively preferred over o_x (*lines 22–24*). In that case, object o_x is not in the CM and thus not in the p-CM. Second, it checks if o_a is p-collectively preferred over o_x (*lines 25–27*). In that case, object o_x is not in the p-CM, but is in the CM. Third, the algorithm checks if the object o_x is p-collectively preferred over o_a (*lines 28–30*). In that case, object o_a is removed from the p-collectively maximal objects (*line 30*), but remains in CM.

After the three checks, if o_x is collectively maximal (*line 31*) it is inserted in CM (*line 32*). Further, if o_x is p-collectively maximal (*line 33*) it is also inserted in p-CM (*line 34*). When the p-IND algorithm terminates, the set p-CM contains the answer to the p-GMCO problem.

Computational Analysis. p-IND performs at most 3 times more object to object comparisons than IND. Hence its running time complexity remains $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$.

2.5 The Group-Ranking Categorical Objects (GRCO) Problem

Section 2.5.1 introduces the GRCO problem, and Section 2.5.2 describes an algorithm for GRCO. Then, Section 2.5.3 discusses some theoretical properties of our proposed ranking scheme.

2.5.1 Problem Definition

As discussed in Section 2.1, it is possible to define a ranking among objects by "composing" the degrees of match for all users. However, any "compositing" ranking function is unfair, as there is no objective way to aggregate individual degrees of match. In contrast, we propose an objective ranking method based on the concept of p-collectively preference.

The obtained ranking is a weak order, meaning that it is possible for objects to share the same rank (ranking with ties). We define the *rank* of an object o to be the smallest integer τ, where $1 \leq \tau \leq |\mathcal{U}|$, such that o is p-collectively maximal for any $p \geq \frac{\tau}{|\mathcal{U}|} \cdot 100$. The non-collectively maximal objects are assigned the lowest possible rank $|\mathcal{U}| + 1$. Intuitively, rank τ for an object o means that any group $\mathcal{U}' \subseteq \mathcal{U}$ of at least τ users (i.e., $|\mathcal{U}'| \geq \tau$) would consider o to be preferable, i.e., o would be collectively maximal for these $\mathcal{U}'$ users. At the highest rank 1, an object o is preferred by each user individually, meaning that o appears in all possible p-collectively maximal object sets.

Problem 3. **[GRCO]** Given a set of objects $\mathcal{O}$ and a set of users $\mathcal{U}$ defined over a set of categorical attributes $\mathcal{A}$, the *Group-Ranking Categorical Objects* (GRCO) problem is to find the rank of all collectively maximal objects of $\mathcal{O}$ with respect to $\mathcal{U}$.

Example. Consider the restaurants and the users presented in Tables 2.1 & 2.2. In our example, the collectively maximals are the restaurants o_1 and o_2. As described in the previous example (Section 2.4), the restaurant o_2 is collectively maximal for any group of two users. Hence, the rank for the restaurant o_2 is equal to two. In addition, o_1 requires all the three users in order to be considered as collectively maximal; so its rank is equal to three. Therefore, the restaurant o_2 is ranked higher than o_1.

2.5.2 A Ranking Algorithm (RANK-CM)

The RANK-CM algorithm (Algorithm 5), computes the rank for all collectively maximal objects. The algorithm takes as input, the collectively maximal objects CM, as well as the number of users $|\mathcal{U}|$. Initially, in each object is assigned the highest rank; i.e., $rank(o_i) \leftarrow 1$ (*line 2*). Then, each object is compared against all other objects in CM (*loop in line 3*). Throughout the objects comparisons, we increase τ (*lines 5–11*) from the current rank (i.e., rank(o_x)) (*line 4*) up to $|\mathcal{U}|$. If o_i is not p-collectively maximal (*line 7*), for $p = \frac{\tau}{|\mathcal{U}|} \cdot 100$ (*line 6*), then o_x cannot be in the p-CM and can only have rank at most $\tau + 1$ (*line 8*). Finally, each object is inserted in the rCM based on its rank (*line 12*).

Algorithm 5. RANK-CM

Input: CM the collectively maximal objects, $|\mathcal{U}|$ the number of users
Output: rCM the ranked collectively maximal objects

```
1  foreach o_i ∈ CM do
2      rank(o_i) ← 1
3      foreach o_j ∈ CM\o_i do
4          τ ← rank(o_i)
5          while τ ≤ |U| − 1 do
6              p ← (τ / |U|) · 100
7              if o_j ≻_p o_i then
8                  rank(o_i) = τ + 1
9              else
10                 break;
11             τ ← τ + 1
12     insert o_i in rCM at rank(o_i)
```

Computational Analysis. The algorithm compares each collective maximal object with all other collective maximal objects. Between two objects the algorithm performs at most $|\mathcal{U}| - 1$ comparisons. Since, in worst case we have that $|CM| = |\mathcal{O}|$, the computational cost of Algorithm 5 is $O(|\mathcal{O}|^2 \cdot |\mathcal{U}|)$.

2.5.3 Ranking Properties

In this section, we discuss some theoretical properties in the context of the rank aggregation problem. These properties have been widely used in voting theory as evaluation criteria for the fairness of a voting system [366, 41, 319]. We show that the proposed ranking scheme satisfies several of these properties.

Property 1. [**Majority**] If an object is strictly preferable over all other objects by the *majority* of the users, then this object is ranked above all other objects.

PROOF. Assume that k_a users strictly prefer o_a over all other objects, where $k_a > \frac{|\mathcal{U}|}{2}$. We will prove that the rank r_a of the object o_a is lower than the rank of any other object.

Since, k_a users strictly prefer o_a over all other objects, any group of at least $|\mathcal{U}|-k_a+1$ users, will consider o_a as collectively maximal. This holds since, any group of at least $|\mathcal{U}|-k_a+1$ users, contains at least one user which strictly prefers o_a over all other objects. Note that, $|\mathcal{U}|-k_a+1$ may not be the smallest group size. That is, it may hold that, for any group of less than $|\mathcal{U}|-k_a+1$ users, o_a is collectively maximal.

Recall the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group). Therefore, in any case we have that, the rank r_a of o_a is at most $|\mathcal{U}|-k_a+1$, i.e., $r_a \leq |\mathcal{U}|-k_a+1$ (1).

On the other hand, let an object $o_i \in \mathcal{O}\backslash o_a$. Then, o_i is not collectively maximal, for any group with $|\mathcal{U}|-k_a+1$ users. This holds since, we have that $k_a > \frac{|\mathcal{U}|}{2}$. So, there is a group of $|\mathcal{U}|-k_a+1$ users, for which, each user strictly preferred o_a over o_i. As a result, in order for o_i to be considered as collectively maximal for any group of a specific size, we have to consider groups with more than $|\mathcal{U}|-k_a+1$ users. From the above, it is apparent that, in any case, the rank r_i for an object o_i is greater than $|\mathcal{U}|-k_a+1$, i.e., $r_i > |\mathcal{U}|-k_a+1$ (2).

Therefore, from (1) and (2), in any case the rank of the object o_a will be lower than the rank of any other object. This concludes the proof of the property. ■

Property 2. [**Independence of Irrelevant Alternatives**] The rank of each object is not affected if non-collectively maximal objects are inserted or removed.

PROOF. According to the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group).

As a result, the rank of an object is specified from the minimum group size, for which, for any group of that size, the object is collectively maximal. Therefore, it is apparent that, the rank of each object is not affected by the non-collectively maximal objects. To note that, the non-collectively maximal objects are ranked with the lowest possible rank, i.e., $|\mathcal{U}|+1$. ■

Property 3. [**Independence of Clones Alternatives**] The rank of each object is not affected if non-collectively maximal objects similar to an existing object are inserted.

PROOF. Similarly to the Property 2. Based on the ranking scheme definition, the non-collectively maximal objects do not affect the ranking. ■

Property 4. [**Users Equality**] The result will remain the same if two users switch their preferences. This property is also know as *Anonymity*.

PROOF. According to the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group).

As a result, the rank of an object is specified from the minimum group size, for which, for any group of that size, the object is collectively maximal. Hence, if two users switch preferences, it is apparent that, the minimum group of any users, for which an object is collectively maximal, remains the same, for all objects. Therefore, the rank of all objects remains the same. ■

Let an object $o_i \in \mathcal{O}$ and a user $u_j \in \mathcal{U}$. Also, let m_i^j be the matching vector between u_j and o_i. We say that the user u_j *increases his interest* over o_i, if $\exists A_k : m_i^j.A_k < \grave{m}_i^j.A_k$, where $\grave{m}_i^j$ is the matching degree resulted by the interest change.

Property 5. [**Monotonicity**] If an object o_a is ranked above an object o_b, and a user increases his interest over o_a, then o_a maintains its position above o_b.

Proof. Let r_a and r_b be the rank of objects o_a and o_b, respectively. Since, o_a is ranked above the object o_b, we have that $r_a < r_b$.

According to the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group). So, we have that for any group of at least r_a and r_b members, o_a and o_b will be collectively maximal.

Assume a user $u_j \in \mathcal{U}$ increases his interest over the object o_a. Further, assume that r'_a and r'_b are the new ranks of the objects o_a and o_b, resulting from the interest change. We show that in any case $r'_a \leq r_a$ and $r'_b \geq r_b$.

First let us study what holds for the new rank of the object o_a. After the interest change, r'_a is the smallest group size that o_a is collectively maximal for any group of that size. We suppose for the sake of contradiction that $r'_a > r_a$. Hence, after the interest change, we should consider larger group sizes in order to ensure that o_a will be collectively maximal for any group of that size. This means that, after the interest change, there is a group of r_a users for which o_a is not collectively maximal. Hence, since o_a is not collectively maximal, there must exist an object $o_i \in \mathcal{O} \backslash o_a$ that is collectively preferred over o_a. To sum up, considering r_a users, we have that: before the interest change, there is no object that is collectively preferred over o_a; and, after the interest change, there is an object that is collectively preferred over o_a. This cannot hold, since the matching degrees between all other users and objects remain the same, while some matching degrees between o_a and u_j have increased (due to interest change). So, for any group of r_a users, there cannot exist an object o_i which is collectively preferred over o_a. Hence, we proved by contradiction that in any case $r'_a \leq r_a$.

Now, let us study what holds for the new rank of the object o_b. After the interest change, r'_b is the smallest group size, that, for any group of that size, o_b is collectively maximal. For the sake of contradiction, we assume that $r'_b < r_b$. Hence, after the interest change, we should consider smaller group sizes, in order to ensure that o_b will be collectively maximal for any group of that size. This means that, before the interest change, there is a group of r'_b users, for which o_b is not collectively maximal. Hence, since o_b is not collectively maximal, there must be an object $o_i \in \mathcal{O} \backslash o_b$ that is collectively preferred over o_b. To sum up, considering r'_b users, we have that: before the interest change, there is an object that is collectively preferred over o_b; and, after the interest change, there is no object that is collectively preferred over o_b. It is apparent that this also cannot hold. So, we proved by contradiction, that in any case $r'_b \geq r_b$.

We show that, $r'_a \leq r_a$ and $r'_b \geq r_b$. Since, $r_a < r_b$, in any case the object o_a will be ranked above o_b. This concludes the proof. ■

For some user u, the following property ensures that the result when u participates is the same or better (w.r.t. u's preferences) compared to that when u does not participate.

Property 6. [**Participation**] *Version 1:* If the object o_a is ranked above the object o_b, then after adding one or more users, which strictly prefer o_a over all other objects, object o_a maintains its position above o_b.

Version 2: Assume an object o_a that is ranked above the object o_b, and that there is at least one user $u \in \mathcal{U}$ which has not stated any preferences; then if u expresses that strictly prefers o_a over all other objects, object o_a maintains its position above o_b.

PROOF. *Version 1:* Let r_a and r_b be the ranks of the objects o_a and o_b, respectively. Since, o_a is ranked above the object o_b, we have that $r_a < r_b$.

According to the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group). Hence, we have that for any group of at least r_a and r_b members, o_a and o_b will be collectively maximal, respectively.

We assume a new user u_n, where $u_n \cap \mathcal{U} = \varnothing$. The new user u_n strictly prefers o_a over all other objects $\mathcal{O} \backslash o_a$. For the sake of simplicity, we consider a singe new user; the proof for more users is similar. The new user set $\mathcal{U}_n$ is generated by adding the new user u_n to the user set $\mathcal{U}$, i.e., $\mathcal{U}_n = \mathcal{U} \cup u_n$.

Let r'_a and r'_b be the ranks for the objects o_a and o_b, respectively, for the new user set $\mathcal{U}_n$. We show that, in any case, rank r'_a is lower than r'_b.

First let us study what holds for the new rank of the object o_a. We show that for any group of r_a members from the new user set $\mathcal{U}_n$, o_a will be collectively maximal. We assume a set S of r_a members from $\mathcal{U}_n$; i.e., $S \subseteq \mathcal{U}_n$ and $|S| = r_a$. Then, based on the users contained in S, we have two cases: (a) All users from S initially belong to $\mathcal{U}$; i.e., $S \subseteq \mathcal{U}_n$. In this case o_a is collectively maximal based on the initial hypothesis. (b) The new user u_n is included to S; i.e., $u_n \in S$. Also in this case o_a is collectively maximal, since for the user u_n, o_a is strictly preferred over all other objects.

Hence, in any case for any group of r_a members from $\mathcal{U}_n$, o_a will be collectively maximal. Also, depending on $\mathcal{U}$, the minimum size of any group of $\mathcal{U}_n$ for which o_a is collectively maximal, may be smaller than r_a; i.e., $r'_a \leq r_a$. Therefore, we have that in any case $r'_a \leq r_a$ (1).

Now, let's determine the new rank for the object o_b. It easy to verify that, if we consider groups of less than r_b users from $\mathcal{U}_n$, then o_b cannot be collectively maximal for any group of that size. Therefore, we have to select groups with equal to or greater than r_b users from $\mathcal{U}_n$, in order for any group of that size to consider o_b as collectively maximal. Hence, we have that in any case $r'_b \geq r_b$ (2).

Since, $r_a < r_b$, for (1) and (2) we have that $r'_a < r'_b$. This concludes the proof of Version 1.

Version 2: The second version can be proved in similar way, since it can be "transformed" into the first version.

Assume we have a user $u_j \in \mathcal{U}$ that has not expressed any preferences. Note that the following also holds if we have more than one users that have not expressed any preferences.

In this case, it is apparent that the ranking process "ignores" the user u_j. In other words: let r_a and r_b be the rank for the objects o_a and o_b, respectively, when we consider the set of users $\mathcal{U}$. In addition, let r'_a and r'_b be the ranks if we consider the users $\mathcal{U} \backslash u_j$. Based on our ranking scheme, it is apparent that, if $r_a > r_b$, then $r'_a > r'_b$.

In this version of the property, we assume that a user u_j has not initially expressed any preferences. Afterwards, u_j states that he strictly prefers o_a over any other object. This scenario is equivalent to the following.

Since, as described above, the rankings are not effected if we remove u_j; we initially consider the users $\mathcal{U} \backslash u_j$. Afterwards, a user that strictly prefers o_a over all other objects

is inserted in the users set $\mathcal{U}\backslash u_j$. This is the same as the first version of our property.

Note that, in order for the second version to be considered in our implementation, we have to modify the initialization of matching vector for the indifferent attributes. Particularly, the matching vector for indifferent attributes should be setting to 0, instead of 1. ■

The following property ensures a low possibility of objects being ranked in the same position.

Property 7. [Resolvability] *Version 1:* If two objects are ranked in the same position, adding a new user can cause an object to be ranked above the other.

Version 2: Assume that two objects are ranked in the same position, and that there is at least one user u which has not stated any preferences; if u expresses preferences, then this can cause an object to be ranked above the other.

Proof. *Version 1:* Assume that we have the objects o_a and o_b. Let r_a and r_b be the rank of objects o_a and o_b, respectively. Initially, the objects are ranked in the same position, so we have that $r_a = r_b$. In order to prove this property, we consider the following example.

Assume that we have an object set $\mathcal{O}$ and four users $\mathcal{U}$ (i.e., $|\mathcal{U}| = 4$). For each of the first two users (i.e., u_1 and u_2) the object o_a is strictly preferred over all other objects in $\mathcal{O}$.

On the other hand, for each of the users u_3 and u_4, the object o_b is strictly preferred over the all other objects in $\mathcal{O}$.

So, for the object o_a, we have that, for any group of three members, o_a will be collectively maximal. This holds, since at least one of the three members is one of the first two users (u_1 or u_2), for which o_a is strictly preferred over all other objects. In addition, it is apparent that three is the smallest size for which, for any group of that size, o_a will be collectively maximal.

According to the definition of the ranking scheme, if the rank of an object o is τ, then τ is the smallest integer that, for any group of at least τ users, o will be collectively maximal (for this group).

As a result, for the rank of o_a we have that $r_a = 3$. Using similar reasoning, for the rank of o_b we have that $r_b = 3$. Hence, we have that in our example both objects o_a and o_b have the same rank, i.e., $r_a = r_b = 3$.

Now lets assume that we add a new user u_5, for which the object o_a is strictly preferred over the all other objects in $\mathcal{O}$. So, for the following, we consider the new users set $\mathcal{U}'$ that includes the new user u_5, i.e., $\mathcal{U}' = \mathcal{U} \cup u_5$. We show that, considering the new users set $\mathcal{U}'$, the new rank r'_b of object o_b will be greater than the initial rank r_b; and for o_a its new rank r'_a will be the same as the initial r_a rank. Hence, in any case, if we also consider a new user u_5, the objects o_a and o_b will have different ranks.

Considering the new users $\mathcal{U}'$, there is a group with three users for which o_b is not collectively maximal. For example, if we select the users u_1, u_2 and u_5, then o_b is not collectively maximal. Hence, in order for o_b to be collectively maximal, we have to select a larger group (at least four users) from $\mathcal{U}'$. So, four users is the smallest group, for which for any group of that size, o_b will be collectively maximal. As a result, $r'_b = 4$. Hence, the new rank of the object o_b is greater than the initial rank.

Regarding the object o_a, for any group of three users from $\mathcal{U}'$, o_a will be collectively maximal. This hold since, for the three out of the five users (i.e., u_1, u_2, u_5), the object o_a is strictly preferred over all other objects. In addition, three users is the smallest group, for which for any group of that size, o_a will be collectively maximal. Therefore, the new rank of o_a is $r'_a = 3$.

So, the new ranks after the addition of user u_5 will be $r'_a = 3$ and $r'_b = 4$, i.e., the objects o_a and o_b will have different ranks. This concludes the proof of Version 1.

Version 2: The second version can be proved in similar way, since it can be "transformed" to the first version as in the proof of Property 6. ■

Property 8. [Users' Preferences Neutrality] Users with different number of preferences, or different preference granularity, are equally important.

PROOF. It is apparent from the ranking scheme definition that this property holds. ■

Property 9. [Objects' Description Neutrality] Objects with different description (i.e., attributes values) granularity are equal important.

PROOF. It is apparent from the ranking scheme definition that this property holds. ■

2.6 Extensions

Section 2.6.1 discusses the case of multi-valued attributes and Section 2.6.2 the case of non-tree hierarchies. Section 2.6.3 presents an extension of IND (and thus of p-IND) for the case when only a subset of the attributes is indexed. Section 2.6.4 discusses semantics of objective attributes.

2.6.1 Multi-valued Attributes

There exist cases where objects have, or users specify, multiple values for an attribute. Intuitively, we want the matching degree of an object to a user w.r.t. a multi-valued attribute to be determined by the *best possible match* among their values. Note that, following a similar approach, different semantics can be adopted for the matching degree of multi-valued attributes. For example, the matching degree of multi-valued attributes may be defined as the average or the minimum match among their values.

Consider an attribute A_k, an object o and a user u, and also let $\{o.A_k[i]\}$, $\{u.A_k[j]\}$ denote the set of values for the attribute A_k for object o, user u, respectively. We define the matching degree of o to u w.r.t. A_k to be the largest among matching degrees computed over pairs of $\{o.A_k[i]\}$, $\{u.A_k[j]\}$ values. For instance, in case of Jaccard coefficient we have, $m.A_k = \max_{i,j} \frac{|o.A_k[i] \cap u.A_k[j]|}{|o.A_k[i] \cup u.A_k[j]|}$.

In order to extend IND to handle multi-valued attributes, we make the following changes. We can relate an object o_x to multiple virtual objects $\{o_x[i]\}$, corresponding to different values in the multi-valued attributes. Each of these virtual objects correspond to different rectangles in the transformed space. For object o_x, the R*-Tree $\mathcal{T}$ contains a leaf entry e_x whose MBR is the MBR enclosing all rectangles of the virtual objects $\{o_x[i]\}$. The leaf entry e_x also keeps information on how to re-construct all virtual objects. During execution of IND, when leaf entry e_x is de-heaped, all rectangles corresponding to virtual objects $\{o_x[i]\}$ are re-constructed. Then, object o_x is collectively maximal, if there exists no other object which is collectively preferred over all virtual objects. If this is the case, then all virtual objects are inserted in the list CM, and are used to prune other entries. Upon termination, the virtual objects $\{o_x[i]\}$ are replaced by object o_x.

2.6.2 Non-Tree Hierarchies

We consider the general case where an attribute hierarchy forms a directed acyclic graph (dag), instead of a tree. The distinctive property of such a hierarchy is that a category is allowed to have multiple parents. For example, consider an Attire attribute hierarchy slightly different than that of Figure 2.1, which also has an new attire category "Sport

casual". In this case, the "Sport casual" category will have two parents, "Street wear" and "Casual".

In the following, we extend the hierarchy transformation to handle dags. The extension follows the basic idea of labeling schemes for dags, as presented in [25]. First, we obtain a spanning tree from the dag by performing a depth-first traversal. Then, we assign intervals to nodes for the obtained tree hierarchy as in Section 2.3.3. Next for each edge, i.e., child to parent relationship, not included in the spanning tree, we propagate the intervals associated with a child to its parent, merging adjacent intervals whenever possible. In the end, each node might be associated with more than one interval.

The IND algorithm can be adapted for multi-interval hierarchy nodes similar to how it can handle multi-valued attributes (Section 2.6.1). That is, an object may be related to multiple virtual objects grouped together in a leaf entry of the R*-Tree.

The following properties extend Proposition 1 for the general case of non-tree hierarchies.

Proposition 4. For objects/users x, y, and an attribute A_k, let $\{x.I_k\}$, $\{y.I_k\}$ denote the set of intervals associated with the value of x, y on A_k. Then it holds that:

(1) $|x.A_k| = \sum_{I_x \in \{x.I_k\}} \|I_x\|$

(2) $|x.A_k \cap y.A_k| = \sum_{\substack{I_x \in \{x.I_k\} \\ I_y \in \{y.I_k\}}} \|I_x \cap I_y\|$

(3) $|x.A_k \cup y.A_k| = \sum_{I_x \in \{x.I_k\}} \|I_x\| + \sum_{I_y \in \{y.I_k\}} \|I_y\| - \sum_{\substack{I_x \in \{x.I_k\} \\ I_y \in \{y.I_k\}}} \|I_x \cap I_y\|$

Proof. Regarding the first property, observe that $|x.A_k| = \left\| \bigcup_{I_x \in \{x.I_k\}} I_x \right\| = \sum_{I_x \in \{x.I_k\}} \|I_x\|$, since the intervals I_x are disjoint.

Also, $|x.A_k \cap y.A_k| = \left\| \left(\bigcup_{I_x \in \{x.I_k\}} I_x \right) \cap \left(\bigcup_{I_y \in \{y.I_k\}} I_y \right) \right\| = \left\| \bigcup_{I_x \in \{x.I_k\}, I_y \in \{y.I_k\}} I_x \cap I_y \right\|$ $= \sum_{I_x \in \{x.I_k\}, I_y \in \{y.I_k\}} \|I_x \cap I_y\|$, since the intervals $I_x \cap I_y$ are disjoint.

Finally, the third property holds since $|x.A_k \cup y.A_k| = |x.A_k| + |y.A_k| - |x.A_k \cap y.A_k|$. ■

2.6.3 Subspace Indexing

This section deals with the case that the index on the set of objects is built on a subset of the object attributes. Recall that R*-Tree indices are efficient for small dimensionalities, e.g., when the number of attributes is less than 10. Therefore, to improve performance, it makes sense to build an index only on a small subspace containing the attributes most frequently occuring in users' preferences. In the following, we present the changes to the IND algorithm necessary to handle this case.

First, a leaf R*-Tree entry e_i contains a pointer to the disk page storing the non-indexed attributes of the object o_i corresponding to this entry. Second, given a non-leaf R*-Tree entry e_i, we define its maximum matching degree on user u_j to be $M_i^j.A_k = \frac{\|e_i.mbr.I_k \cap u_j.I_k\|}{\|u_j.I_k\|}$ with respect to an indexed attribute A_k (as in regular IND), and $M_i^j.A_{k'} = 1$ with respect to a non-indexed attribute $A_{k'}$. Third, for a leaf entry e_i corresponding to object o_i, its maximum matching degree is equal to the matching degree of o_i to u_j w.r.t. A_k, as in regular IND. Note that in this case an additional I/O operation is required to retrieve the non-indexed attributes.

It is easy to see that the maximum matching degree $M_i^j.A_k$ of entry e_i on user u_j w.r.t. specified attribute A_k is an upper bound to the highest matching degree among all objects in the group that e_i defines. However, note that it is not a *tight* upper bound as in the case of the regular IND (Proposition 3).

The remaining definitions, i.e., the maximum matching vector and the score of an entry, as well as the pseudocode are identical to their counterparts in the regular IND algorithm.

2.6.4 Objective Attributes

In this section we describe *objective attributes*. As objective attributes we refer to the attributes that the order of their values is the same for all users. Hence, in contrast to the attributes considered before, the users are not expressing any preferences over the objective attributes. Particularly, in objective attributes, their preference relation is derived from the attributes' semantics and it is the same for all users. For instance, in our running example in addition to the restaurants' attributes in which different users may have different preferences (i.e., subjective attributes); we can assume an objective attribute "Rating", representing the restaurant's score. Attribute Rating is totally ordered, and higher rated restaurants are more preferable from all users.

Based on the attributes' semantics, we categorized attributes into two groups: (1) *objective attributes*, and (2) *subjective attributes*. Let $\mathcal{A}_o \in \mathcal{A}$ and $\mathcal{A}_s \in \mathcal{A}$ denote the objective and subjective attributes respectively, where $\mathcal{A}_o \cup \mathcal{A}_s = \mathcal{A}$ and $\mathcal{A}_o \cap \mathcal{A}_s = \varnothing$.

Let two objects o_a and o_b, having an objective attribute $A_k \in \mathcal{A}_o$. As $o_a.A_k$ we denote the value of the attribute A_k for the object o_a. Here, without loss of generality, we assume that objective attributes are single-value numeric attributes, and the object o_a is *better* than another object o_b on the objective attribute A_k, iff $o_a.A_k > o_b.A_k$.

Considering objects with both objective and subjective attributes, the preferred and strictly preferred relations presented in Section 3.2.1, are defined as follows.

An object o_a is *preferred* over o_b, for user u_j, denoted as $o_a \succeq^j o_b$ iff (1) for every specified subjective attribute $A_h \in \mathcal{A}_s$ it holds that $m_a^j.A_h \geq m_b^j.A_h$, and (2) for each objective attribute $A_k \in \mathcal{A}_o$ hold that $o_a.A_k \geq o_b.A_k$. Moreover, object o_a is *strictly preferred* over o_b, for user u_j, denoted as $o_a \succ^j o_b$ iff (1) o_a is preferred over o_b, (2) there exists a specified subjective attribute $A_h \in \mathcal{A}_s$ such that $m_a^j.A_h > m_b^j.A_h$, and (3) there exists an objective attribute $A_k \in \mathcal{A}_o$ such that $o_a.A_k > o_b.A_k$.

2.7 Experimental Analysis

Section 2.7.1 describes the datasets used for the evaluation. Sections 2.7.2 and 2.7.3 study the efficiency of the GMCO and p-GMCO algorithms, respectively. Finally, Section 2.7.4 investigates the effectiveness of the ranking in the GRCO problem.

2.7.1 Datasets & User preferences

We use five datasets in our experimental evaluation, one synthetic and four real. The first is Synthetic, where objects and users are synthetically generated. All attributes have the same hierarchy, a binary tree of height $\log|A|$, and thus all attributes have the same number of leaf hierarchy nodes $|A|$. To obtain the set of objects, we fix a level, ℓ_o (where $\ell_o = 1$ corresponds to the leaves), in all attribute hierarchies. Then, we randomly select nodes from this level to obtain the objects' attribute value. The number of objects is denoted as $|\mathcal{O}|$, while the number of attributes for each object is denoted as d. Similarly, to obtain the set of users, we fix a level, ℓ_u, in all hierarchies. The group size (i.e., number of users) is denoted as $|\mathcal{U}|$.

Table 2.5: Real datasets basic characteristics

Dataset	Number of Objects	Attributes (Hierarchy height)
RestaurantsF	85,691	*Cuisine* (6), *Attire* (3), *Parking* (3)
ACM	281,476	*Category* (4)
Cars	30,967	*Engine* (3), *Body* (4), *Transmission* (3)
RestaurantsR	130	*Cuisine* (5), *Smoke* (3), *Dress* (3), *Ambiance* (3)

The second dataset is RestaurantsF, which contains 85,681 US restaurant retrieved from Factual[2]. We consider *three* categorical attributes, *Cuisine*, *Attire* and *Parking*. The hierarchies of these attributes are presented in Figure 2.1 (the figure only depicts a subset of the hierarchy for Cuisine). Particularly, for the attributes Cuisine, Attire and Parking, we have 6, 3, 3 levels and 126, 5, 5 leaf hierarchy nodes, respectively.

The third dataset is ACM, which contains 281,476 research publications from the ACM, obtained from datahub[3]. The *Category* attribute is categorical and is used by the ACM in order to classify research publications. The hierarchy for this attributed is defined by the ACM Computing Classification System[4], and is organized in 4 levels and has 325 leaf nodes.

The fourth dataset is Cars, containing a set of 30,967 car descriptions retrieved from the Web[5]. We consider *three* attributes, *Engine*, *Body* and *Transmission*, having 3, 4, 3 levels, and 11, 23, 5 leaf hierarchy nodes, respectively. We note that this is not the same dataset used in [71].

The fifth dataset is RestaurantsR, obtained from a recommender system prototype[6]. This dataset contains a set of 130 restaurants descriptions and a set of 138 users along with their preferences. For our purposes, we consider *four* categorical attributes, *Cuisine*, *Smoke*, *Dress*, and *Ambiance*, having 5, 3, 3, 3 levels, and 83, 3, 3, 3 leaf hierarchy nodes, respectively. This dataset is used in the effectiveness analysis of the GRCO problem, while the other datasets are used in the efficiency evaluation of the GMCO algorithms.

For the efficiency evaluation, the user preferences for real datasets are obtained following two different approaches. In the first approach, denoted as *Real preferences*, we attempt to simulate real user preferences. Particularly, for the RestaurantF dataset, we use as user preferences the restaurants' descriptions from the highest rated New York restaurant list[7]. For the Car dataset, the user preferences are obtained from the top rated cars[8]. Finally, for the ACM dataset, the user preferences are obtained by considering ACM categories from the papers published within a research group[9]. In the second approach, denoted as *Synthetic preferences*, the user preferences are obtained using a method similar to this followed in Synthetic dataset. Particularly, the user preferences are specified by randomly selecting hierarchy nodes from the second hierarchy level (i.e., $\ell_u = 2$). Table 2.5 summarizes the basic characteristics of the employed real datasets.

2.7.2 Efficiency of the GMCO algorithms

For the GMCO problem, we implement IND (Section 2.3) and three flavors of the BSL algorithm (Section 2.3.2), denoted BSL-BNL, BSL-SFS, and BSL-BBS, which use the skyline algorithms BNL [98], SFS [123], BBS [300], respectively.

[2] www.factual.com
[3] datahub.io/dataset/rkb-explorer-acm
[4] www.acm.org/about/class/ccs98-html
[5] www.epa.gov
[6] archive.ics.uci.edu/ml/datasets/Restaurant+&+consum er+data
[7] www.yelp.com
[8] www.edmunds.com/car-reviews/top-rated.html
[9] www.dblab.ntua.gr/pubs

Table 2.6: Parameters (Synthetic)

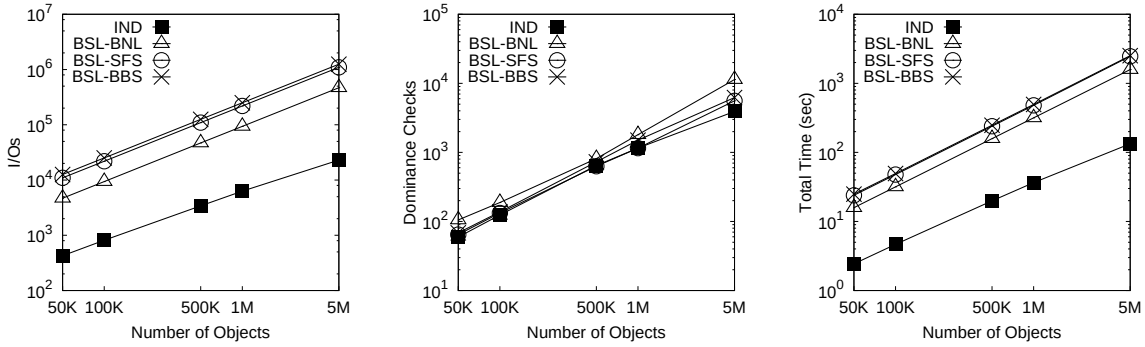
Description	Symbol	Values
Number of objects	$\|\mathcal{O}\|$	50K, 100K, **500K**, 1M, 5M
Number of attribute	d	2, 3, **4**, 5, 6
Group size	$\|\mathcal{U}\|$	2, 4, **8**, 16, 32
Hierarchy height	$\log\|A\|$	4, 6, **8**, 10, 12
Hierarchy level for objects	ℓ_o	**1**, 2, 3, 4, 5
Hierarchy level for users	ℓ_u	**2**, 3, 4, 5, 6

To gauge the efficiency of all algorithms, we measure: (1) the number of disk I/O operations, denoted as I/Os; (2) the number of dominance checks, denoted as Dom. Checks; and (3) the total execution time, denoted as Total Time, and measured in secs. In all cases, the reported time values are the averages of 3 executions. All algorithms were written in C++, compiled with gcc, and the experiments were performed on a 2GHz CPU.

2.7.2.1 Results on Synthetic Dataset

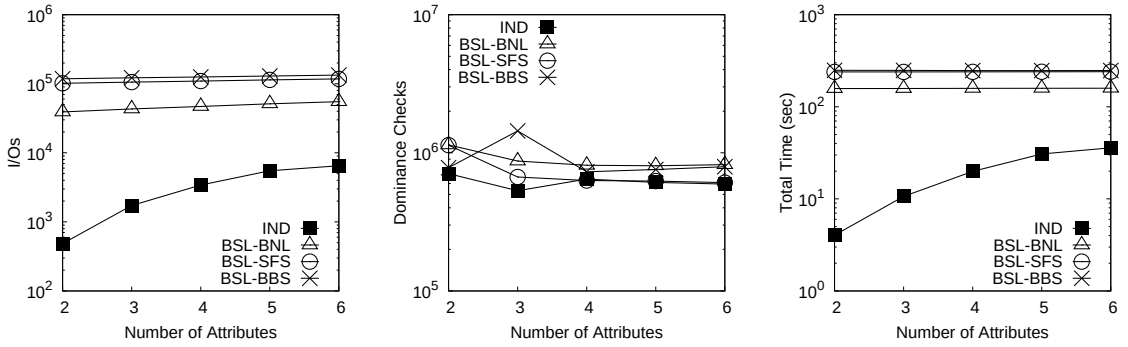
In this section we study the efficiency of the GMCO algorithms using the Synthetic dataset described in Section 2.7.1.

Parameters. Table 2.6 lists the parameters that we vary and the range of values examined for Synthetic. To segregate the effect of each parameter, we perform six experiments, and in each we vary a single parameter, while we set the remaining ones to their default (bold) values.



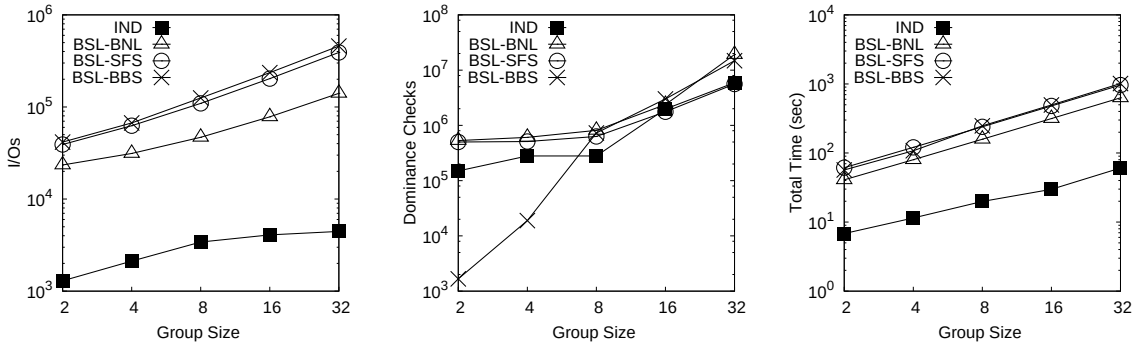
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.3: GMCO algorithms, Synthetic: varying $|\mathcal{O}|$



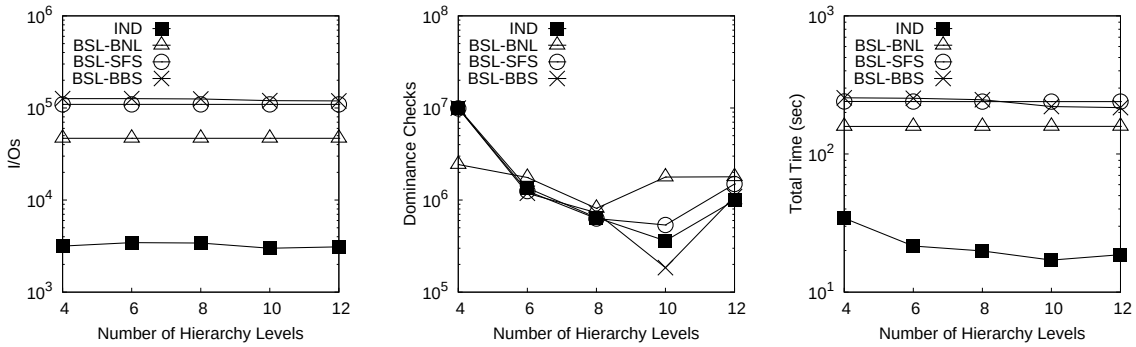
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.4: GMCO algorithms, Synthetic: varying d



(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.5: GMCO algorithms, Synthetic: varying $|\mathcal{U}|$



(a) I/O Operations (b) Dom. Checks (c) Total Time

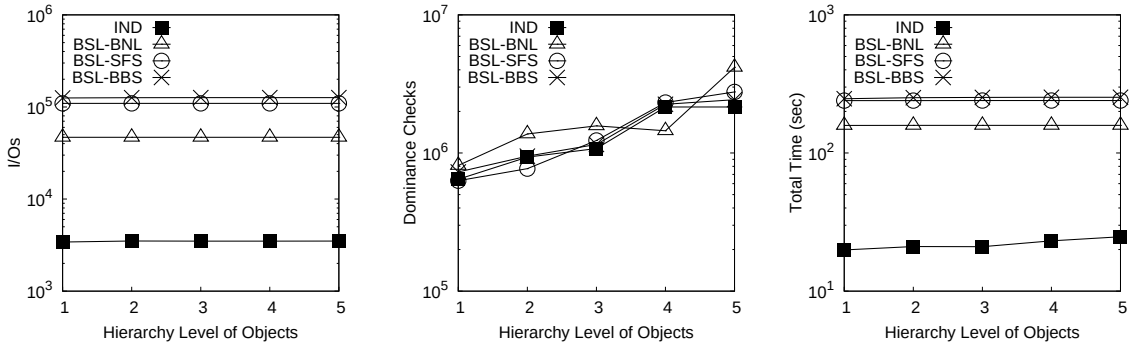
Figure 2.6: GMCO algorithms, Synthetic: varying $\log|A|$

Varying the number of objects. In the first experiment, we study performance with respect to the objects' set cardinality $|\mathcal{O}|$. Particularly, we vary the number of objects from 50K up to 5M and measure the number of I/Os, the number of dominance checks, and the total processing time, in Figures 2.3a, 2.3b and 2.3c, respectively.

When the number of objects increases, the performance of all methods deteriorates. The number of I/Os performed by IND is much less than the BSL variants, the reason being BSL needs to construct a file containing matching degrees. Moreover, the SFS and BBS variants have to preprocess this file, i.e., sort it and build the R-Tree, respectively. Hence, BSL-BNL requires the fewest I/Os among the BSL variants.

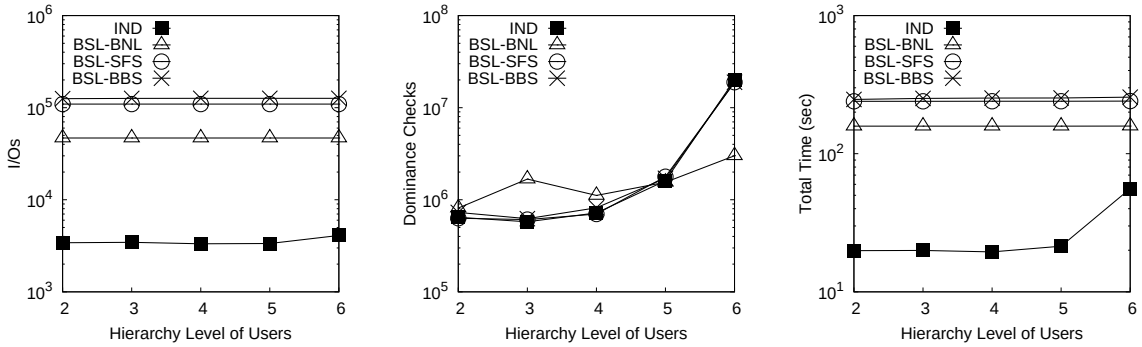
All methods require roughly the same number of dominance checks as seen in Figure 2.3b. IND performs fewer checks, while BSL-BNL the most. Compared to the other BSL variants, BSL-BNL performs more checks because, unlike the others, computes the skyline over an unsorted file. IND performs as well as BSL-SFS and BSL-BBS, which have the easiest task. Overall, Figure 2.3c shows that IND is more than an order of magnitude faster than the BSL variants.

Varying the number of attributes. Figure 2.4 investigates the effect as we increase the number of attributes d from 2 up to 6. The I/O cost, shown in Figure 2.4a of the BSL variants does not depend on $|\mathcal{O}|$ and thus remains roughly constant as d increases. On the other hand, the I/O cost of IND increases slightly with d. The reason is that d determines the dimensionality of the R-Tree that IND uses. Further, notice that the number of dominance checks depicted in Figure 2.4b is largely the same across methods. Figure 2.4c shows that the total time of IND increases with d, but it is still significantly smaller (more than 4 times) than the BSL methods even for $d = 6$.



(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.7: GMCO algorithms, Synthetic: varying ℓ_o



(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.8: GMCO algorithms, Synthetic: varying ℓ_u

Varying the group size. In the next experiment, we vary the users' set cardinality $|\mathcal{U}|$ from 2 up to 32; results are depicted in Figure 2.5. The performance of all methods deteriorates with $|\mathcal{U}|$. The I/O cost for IND is more than an order of magnitude smaller than the BSL variants, and the gap increases with $|\mathcal{U}|$, as Figure 2.5a shows. As before, BSL-BNL requires the fewest I/Os among the BSL variants.

Regarding the number of dominance checks, shown in Figure 2.5b, IND performs the fewest, except for 2 and 4 users. In these settings, the BBS variant performs the fewest checks, as it is able to quickly identify the skyline and prune large part of the space. Note that $|\mathcal{U}|$ determines the dimensionality of the space that BSL-BBS indexes. As expected, for more than 4 dimensions the performance of BBS starts to take a hit. Overall, Figure 2.3c shows that IND is more than an order of magnitude faster than all the BSL variants, among which BSS-BNL is the fastest.

Varying the hierarchy height. In this experiment, we vary the hierarchy height $\log|A|$ from 4 up to 12 levels. Figure 2.6 illustrates the results. All methods are largely unaffected by this parameter. Note that the number of dominance checks varies with $\log|A|$, and IND performs roughly as many checks as the BSL variants which operated on a sorted file, i.e., BSL-SFS and BSL-BBS. Overall, IND is more than an order of magnitude faster than all BSL variants.

Varying the objects level. Figure 2.7 depicts the results of varying the level ℓ_o from which we draw the objects' values. The performance of all methods is not significantly affected by ℓ_o. Note though that the number of dominance checks increases as we select values from higher levels.

Varying the users level. Figure 2.8 depicts the results of varying the level ℓ_u from

which we draw the users' preference values. As with the case of varying ℓ_o, the number of dominance checks increases with ℓ_u, while the performance of all methods remains unaffected. The total time of IND takes its highest value of $\ell_u = 6$, as the number of required dominance checks increases sharply for this setting. Nonetheless, IND is around 3 times faster than BSL-BNL.

2.7.2.2 Results on Real Datasets

In this section we study the efficiency of the GMCO algorithms using the three real datasets described in Section 2.7.1. For each dataset, we examine both real and synthetic preferences, obtained as described in Section 2.7.1. Also, we vary the group size $|\mathcal{U}|$ from 2 up to 32 users.

Figures 2.9 & 2.10 present the result for RestaurantsF dataset, for real and synthetic preferences, respectively. Similarly, Figures 2.11 & 2.12 present the result for ACM dataset, and Figures 2.13 & 2.14 for Cars dataset. As we can observe, the performance of the examined methods is almost similar for all datasets, real and synthetic. Also, similar performance is observed in real and synthetic user preferences. In most cases, IND outperforms the BSL methods by at least an order of magnitude in terms of I/Os and total time. Additionally, IND performs less dominance checks than the BSL methods in almost all cases.

Regarding BSL methods, BSL-BNL outperforms the others in terms of I/Os and total time; while BSL-SFS and BNL-BBS have the almost the same performance. Regarding the number of dominance checks, for less than 16 users BSL-BNL performs more dominance checks than other BSL methods; while for 32 users, in many cases (Figures 2.9b, 2.10b, 2.11b) BSL-BNL performs the fewest dominance checks from BSL methods. Finally, for less than 8 users, BNL-BBS perform fewer dominance checks than other BSL methods.

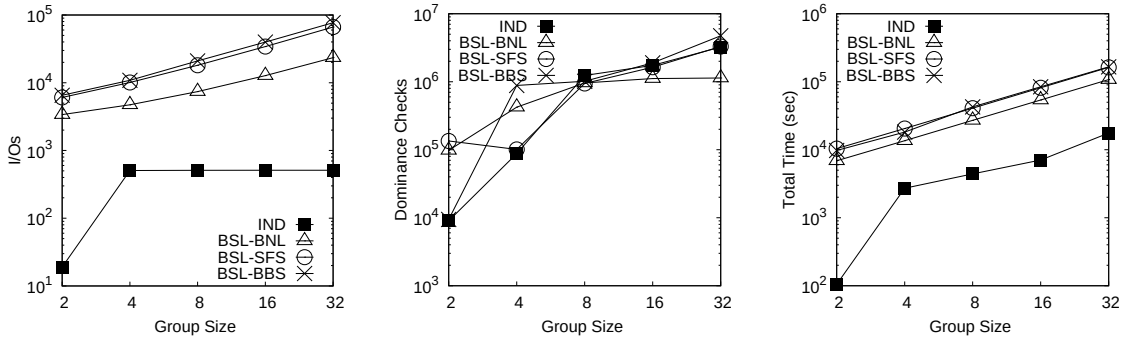
2.7.3 Efficiency of the p-GMCO Algorithms

In this section, we investigate the performance of the p-GMCO algorithms (Section 2.4). For the p-GMCO problem, we implement the respective extensions of all algorithms (IND and BSL variants), distinguished by a p prefix. As before, we measure the number of I/O operations, dominance checks and the total time. In the following experiments, we use the three real datasets and vary the number of users from 2 up to 1024, while $p = 30\%$. Also, we also vary the parameter p from 10% up to 50%. However, the performance of all methods (in terms of I/Os and total time) remains unaffected by p; hence, the relevant figures are omitted.

Figures 2.15 & 2.16 present the result for RestaurantsF dataset, for real and synthetic preferences, respectively. Similarly, Figures 2.17 & 2.18 corresponds to the ACM dataset, and Figures 2.19 & 2.20 to Cars.

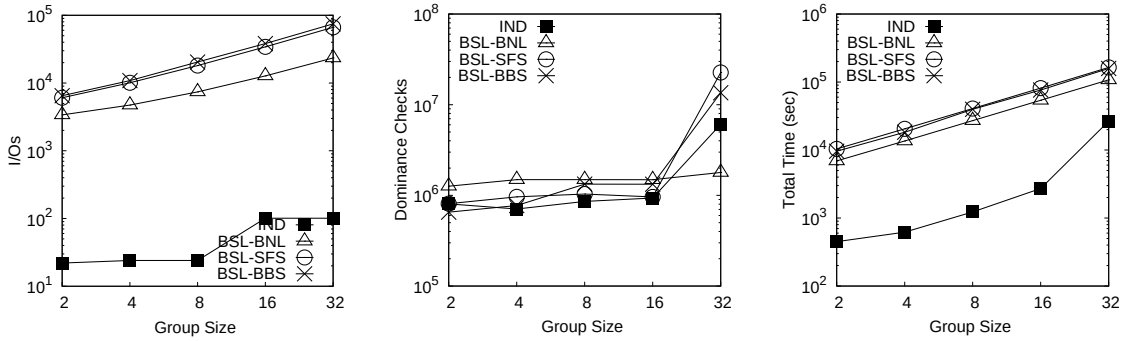
As we can observe, IND outperforms the BSL methods in almost all cases. Particularly, the number of I/O operations performed by IND is several order of magnitude lower than the BSL variants. In addition, in almost all cases, IND performs fewer dominance checks than the BSL methods. The number of I/Os performed by IND remains stable for more than 16 users; while for BSL methods, the I/O operations are constantly increased up to 256 users. Regarding dominance check, the number of dominance checks increases with $|\mathcal{U}|$ following an almost similar trend for all methods.

Finally, regarding BSL methods, BSL-BNL outperforms the other BSL methods in terms of I/Os and total time; while BSL-SFS and BNL-BBS have almost the same performance. As far as dominance checks, in some cases (Figures 2.15b & 2.17b) BSL-BNL outperforms all BSL methods, while in other cases (Figures 2.16b, 2.18b, 2.19b, 2.20b), BSL-BNL performs more dominance checks than the other BSL methods.



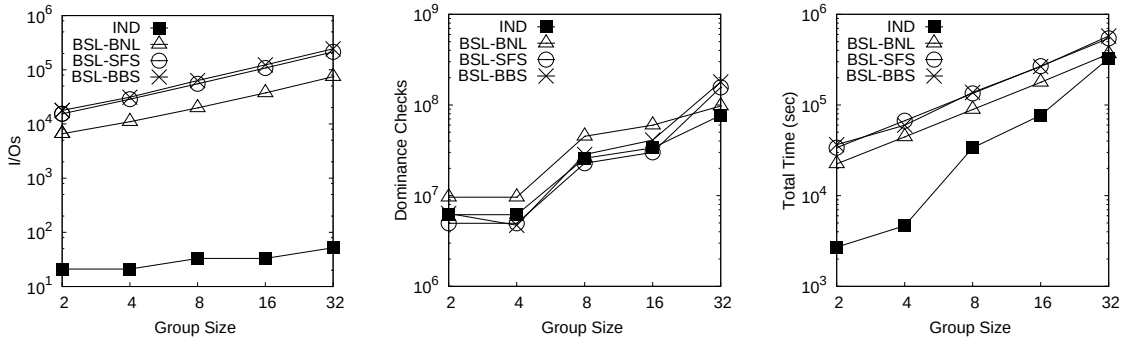
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.9: GMCO algorithms, RestaurantsF (Real preferences): varying $|\mathcal{U}|$



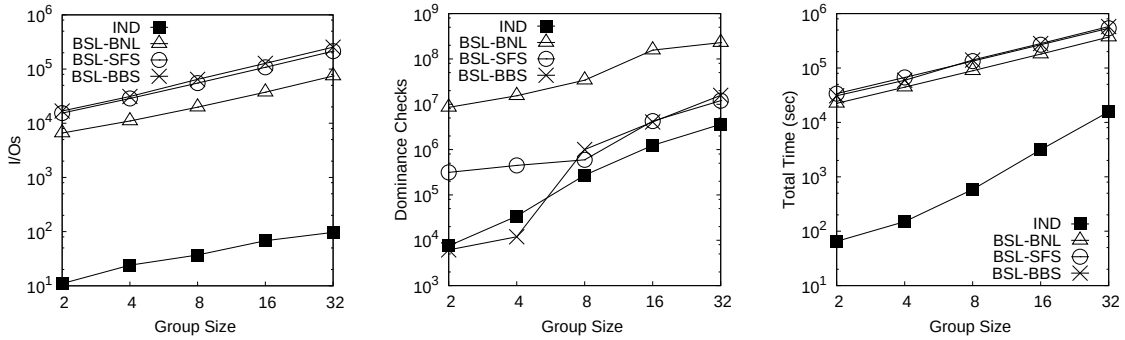
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.10: GMCO algorithms, RestaurantsF (Synthetic preferences): varying $|\mathcal{U}|$



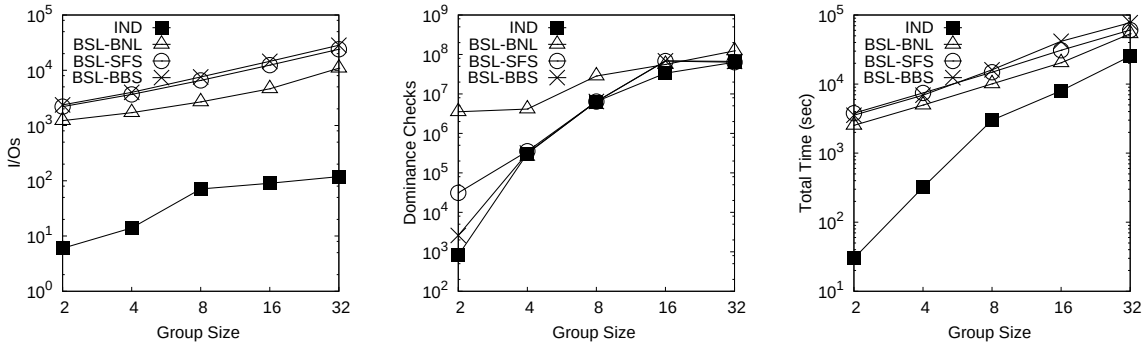
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.11: GMCO algorithms, ACM (Real preferences): varying $|\mathcal{U}|$



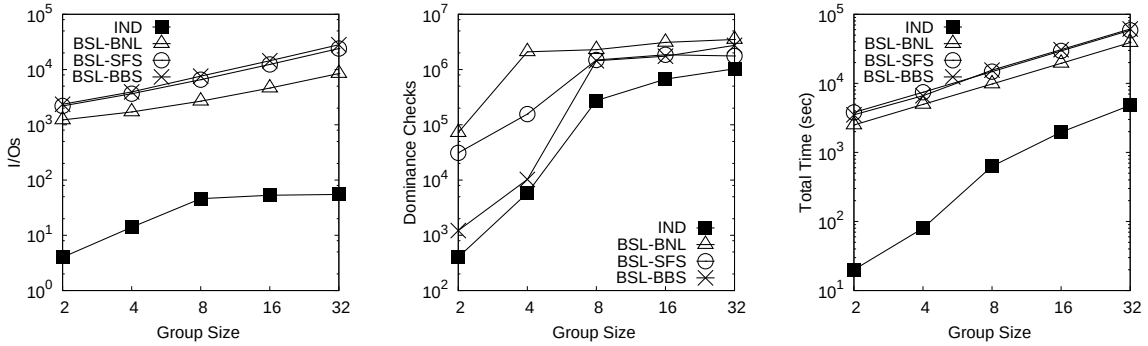
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.12: GMCO algorithms, ACM (Synthetic preferences): varying $|\mathcal{U}|$



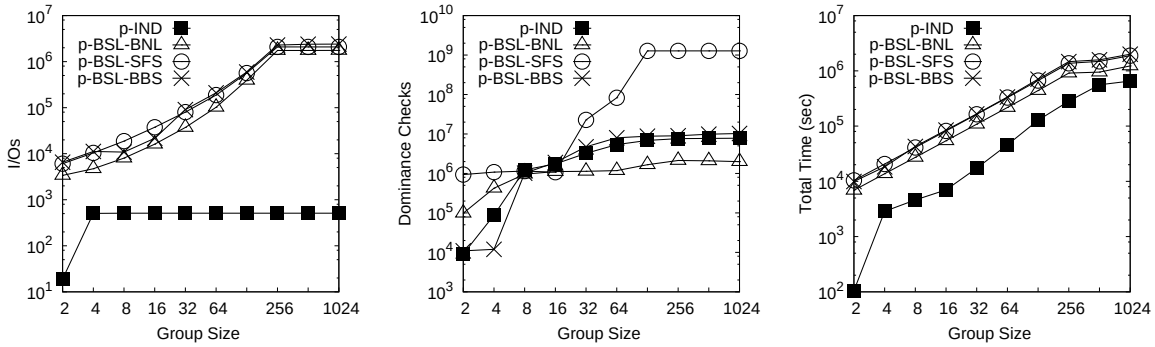
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.13: GMCO algorithms, Cars (Real preferences): varying $|\mathcal{U}|$



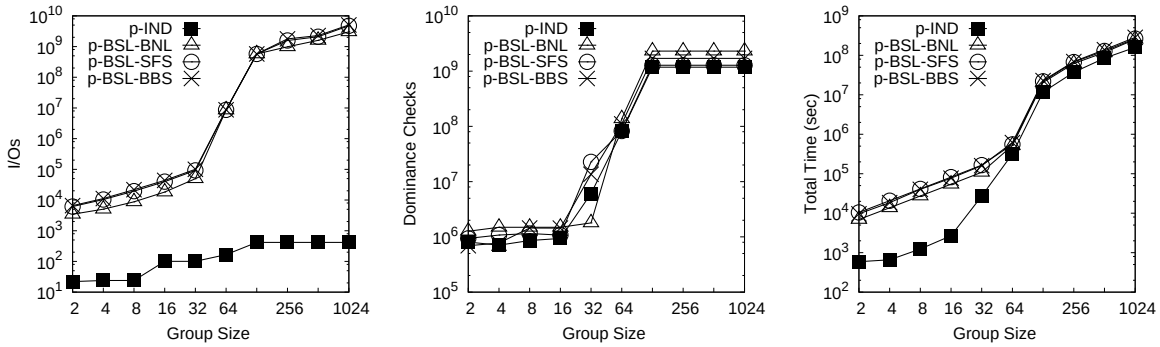
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.14: GMCO algorithms, Cars (Synthetic preferences): varying $|\mathcal{U}|$



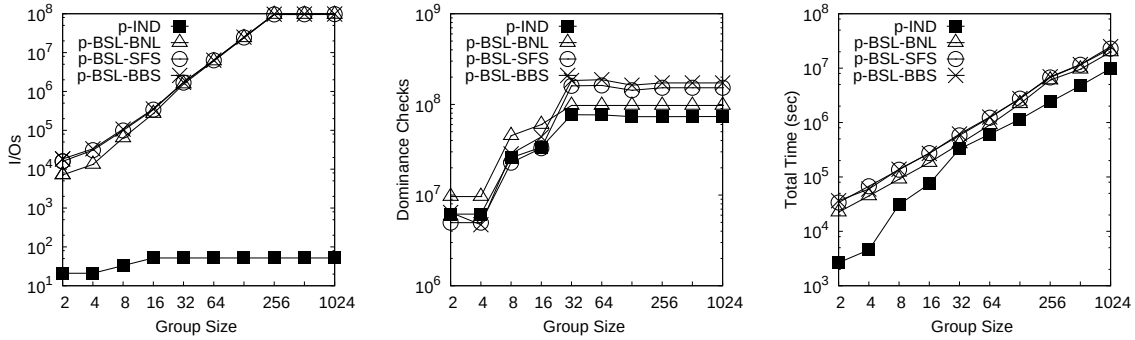
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.15: p-GMCO algorithms, RestaurantsF (Real preferences): varying $|\mathcal{U}|$



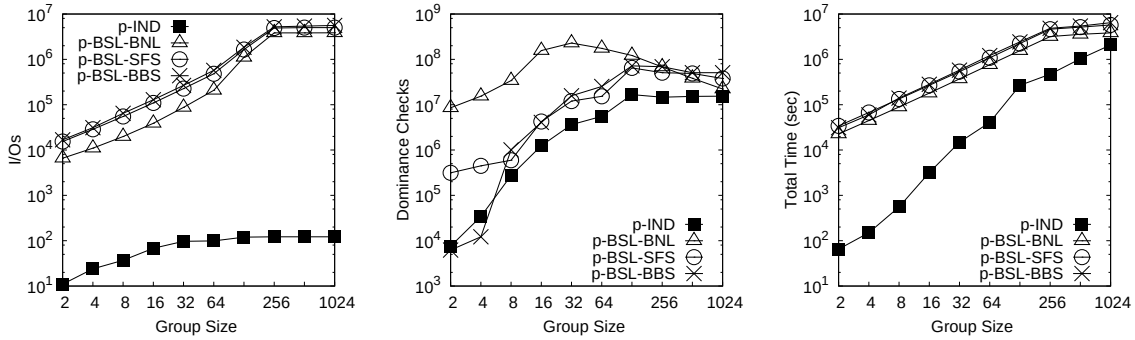
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.16: p-GMCO algorithms, RestaurantsF (Synthetic preferences): varying $|\mathcal{U}|$



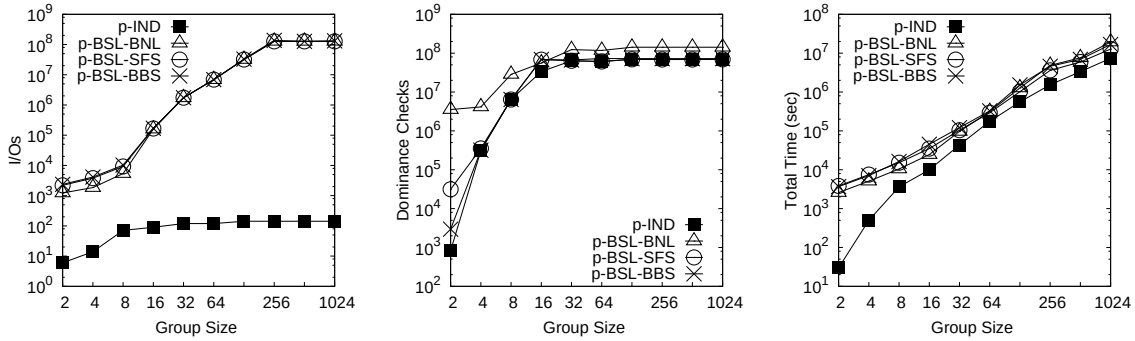
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.17: p-GMCO algorithms, ACM (Real preferences): varying $|\mathcal{U}|$



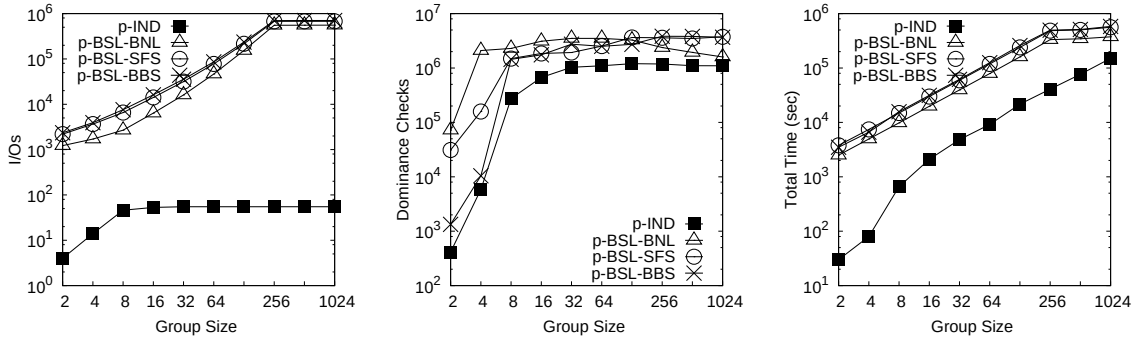
(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.18: p-GMCO algorithms, ACM (Synthetic preferences): varying $|\mathcal{U}|$



(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.19: p-GMCO algorithms, Cars (Real preferences): varying $|\mathcal{U}|$



(a) I/O Operations (b) Dom. Checks (c) Total Time

Figure 2.20: p-GMCO algorithms, Cars (Synthetic preferences): varying $|\mathcal{U}|$

2.7.4 Effectiveness of GRCO

In this section we study the effectiveness of the GRCO problem (Section 2.5). We compare our RANK-CM algorithm (Section 2.5.1) to nine popular aggregations strategies adopted by most group recommender systems [108]. Particularly, we implement the following aggregation strategies:

- *Additive* (ADD): adds the individual matching degrees.
- *Multiplicative* (MULT): multiplies the individual matching degrees.
- *Least Misery* (MISERY): considers the minimum of individual matching degrees.
- *Most Pleasure* (PLEASURE): considers the maximum of individual matching degrees.
- *Average Without Misery* (AVG_MISERY): takes the average matching degrees, excluding matching degrees below a threshold;
- *Average Without Misery Threshold-free* (AVG_MISERY+): is a strategy introduced here, similar to AVG_MISERY, with the difference that the threshold is set to the minimum of individual matching degrees.
- *Copeland Rule* (COPELAND): counts the number of times an object has higher individual matching degrees than the rest of the objects, minus the number of times the object has lower individual matching degrees.
- *Approval Voting* (APPROVAL): counts the number of individual matching degrees with values greater than or equal to a threshold.
- *Borda Count* (BORDA): adds the scores computed per matching degree according to its rank in a user's preference list (the matching degree with the lowest value gets a zero score, the next one point, and so on).

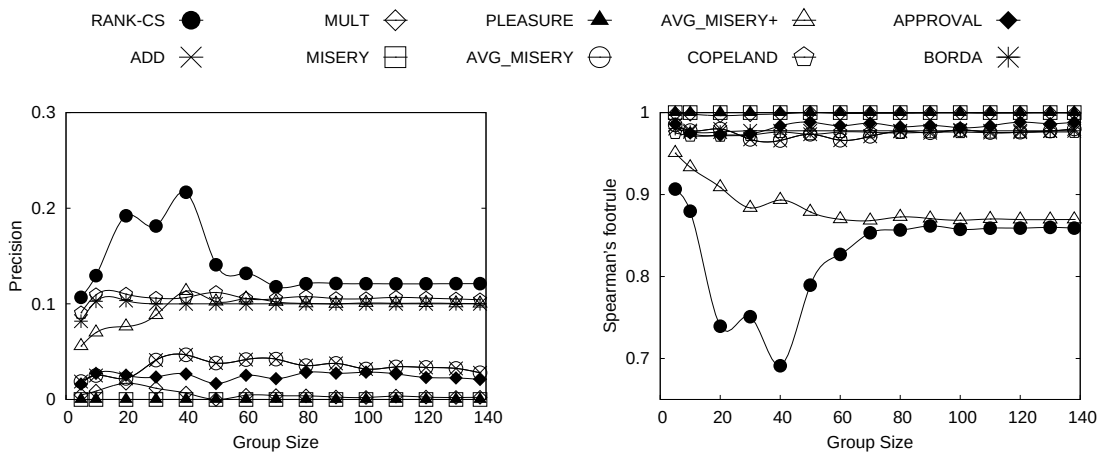
Note that, the threshold in AVG_MISERY and APPROVAL strategies is set to 0.5.

To gauge the effectiveness of our ranking scheme, we use the RestaurantsR dataset. We use the reviews from all users and extract a ranked list of the most popular restaurants to serve as the ground truth. Then, we compare the ranked lists returned by RANK-CM and the other aggregation strategies to the ground truth, computing *Precision* and the *Generalized Spearman's Footrule* [168], in several ranks and for different group sizes. In order to construct group of users, for each group size, we randomly select users, composing 500 groups of the same size. Hence, in each experiment the average measurements are presented.

Varying the group size. In the first experiment (Figures 2.21 & 2.22), we consider different group sizes, varying the number of users, from 5 to 138. We compute the precision and the Spearman's footrule for the ranked listed returned by all methods, compared to the ground truth list, at rank 10 (Figure 2.21) and rank 20 (Figure 2.22).

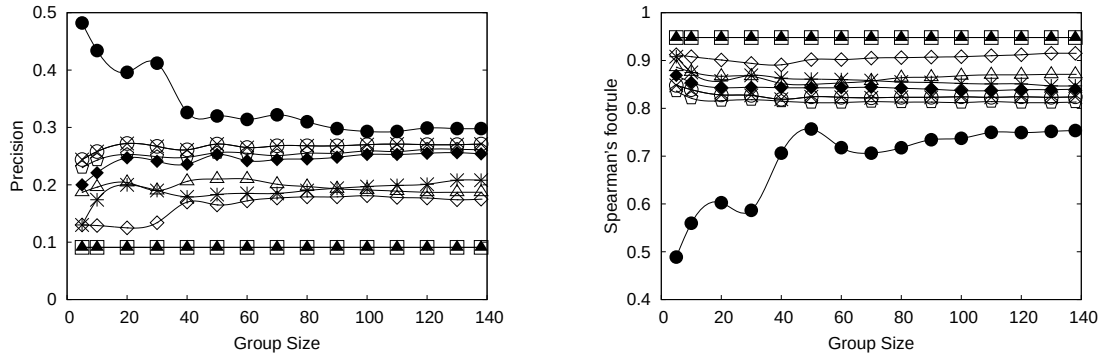
In Figure 2.21, we consider the first ten restaurants retrieved (i.e., at rank 10); the precision for each method is defined as the number of common restaurants between the ground truth list and the ranked list returned by each method, divided by ten. For example, in Figure 2.21a, for the groups of 20 users, RANK-CM has precision around 0.2; that is, among the first ten restaurants retrieved, RANK-CM retrieves in average two popular restaurants. On the other hand, BORDA and COPELAND retrieve in average one popular restaurant, and have precision around 0.1.

Regarding the results at rank 10, as we can observe from Figure 2.21, RANK-CM outperforms all other methods in both metrics. Note that, Spearman's footrule values range from 0 to 1, where lower values indicate a better match to the ground truth (0



(a) Precision (b) Spearman's footrule

Figure 2.21: RestaurantsR (Rank 10): varying $|\mathcal{U}|$



(a) Precision (b) Spearman's footrule

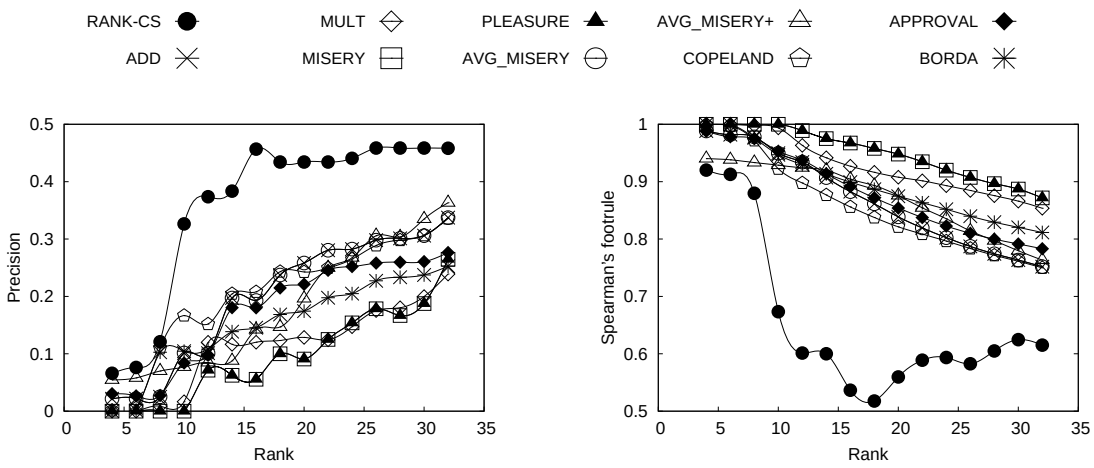
Figure 2.22: RestaurantsR (Rank 20): varying $|\mathcal{U}|$

means that the two lists are identical). Regarding the other aggregation strategies, the best results are provided by COPELAND, BORDA and AVG_MISERY+, while MISERY and PLEASURE performed the worst.

Similar results and observations hold at rank 20 (Figures 2.22), where RANK-CM outperforms all other methods, with COPELAND, ADD and AVG_MISERY being the best alternatives.

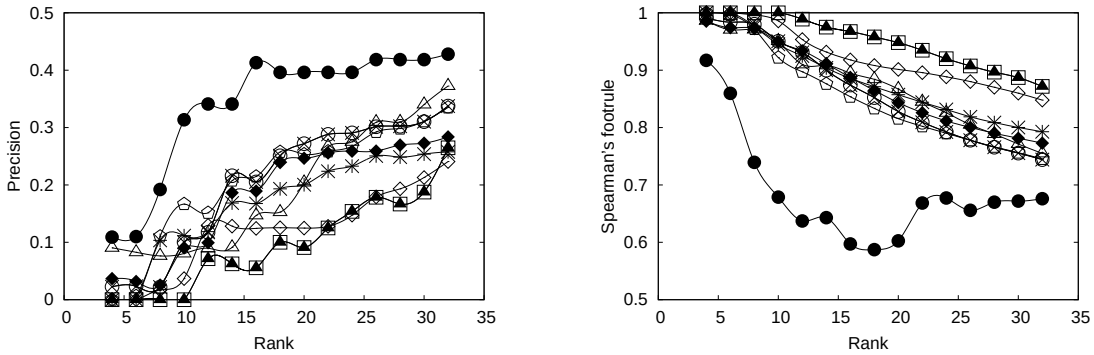
Overall, RANK-CM performs better in terms of precision and Spearman's footrule than the other strategies, in all cases. The COPELAND strategy seems to be the best alternative, while MISERY and PLEASURE the worst.

Varying rank. In this experiment, we consider three different group sizes (i.e., 10, 20, 30) and compute the precision and the Spearman's footrule from rank 4 to rank 32. As we can observe from Figures 2.23, 2.24 & 2.25, the performance of all methods is almost similar for the examined group sizes. The RANK-CM achieves better performance in terms of precision and Spearman's footrule in almost all examined ranks, with the exceptions at ranks 4 and 6 for group sizes 10 and 30, where COPELAND achieves almost the same performance with RANK-CM. Regarding the other methods, the best performance is from COPELAND, ADD and AVG_MISERY+.



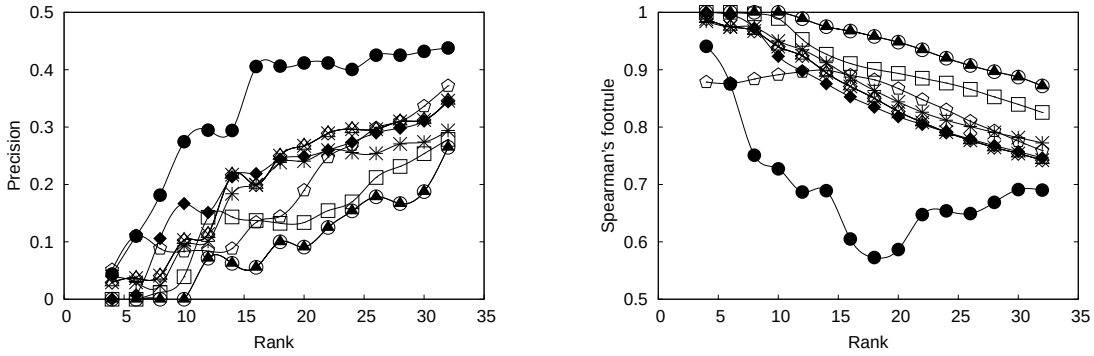
(a) Precision (b) Spearman's footrule

Figure 2.23: RestaurantsR ($|\mathcal{U}| = 10$): varying rank



(a) Precision (b) Spearman's footrule

Figure 2.24: RestaurantsR ($|\mathcal{U}| = 20$): varying rank



(a) Precision (b) Spearman's footrule

Figure 2.25: RestaurantsR ($|\mathcal{U}| = 30$): varying rank

2.8 Related Work

This section reviews work on *recommender systems* and algorithms for *skyline computation*.

2.8.1 Recommender Systems

There exist several techniques to specify *preferences* on objects [355, 252]. The *quantitative preferences*, e.g., [26, 211, 246], assign a numeric score to attribute values, signifying importance. For example, values a, b, c are assigned scores 0.9, 0.7, 0.1, respectively, which implies that a is more preferable than b, which in turn is more preferable than c. There also exist *qualitative preferences*, e.g., [237, 122], which are relatively specified using binary relationships. For example, value a is preferred over b and c, but b, c are indifferent. This work assumes the case of boolean quantitative preferences, where a single attribute value is preferred, while others are indifferent.

The general goal of *recommendation systems* [20, 92, 400, 230] is to identify those objects that are most aligned to a user's preferences. Typically, these systems provide a *ranking* of the objects by *aggregating* user preferences. Particularly, the work in [26] defines generic functions that merge quantitative preferences. The works in [114, 211] deal with linear combinations of preference scores and propose index and view based techniques for ranking tuples. For preferences in general, [237, 122] introduce a framework for composing or accumulating interests. Among the discussed methods is the Pareto composition, which is related to the skyline computation, discussed below.

Recently, several methods for *group recommendations* are proposed [219, 284, 108, 97]. These methods, recommend items to a group of users, trying to satisfy all the group members. The existing methods are classified into two approaches. In the first, the preferences of each group member are combined to create a virtual user; the recommendations to the group are proposed w.r.t. to the virtual user. In the second, individual recommendations for each member is computed; the recommendations of all members are merged into a single recommendation. A large number of group recommendation methods have been developed in several domains such as: music [350, 130, 309, 286, 115, 406], movies [298], TV programs [283, 401, 386, 52], restaurants [301, 285], sightseeing tours [179, 39, 235], vacation packages [288, 218], food [159], news [310], and online communities [182, 239, 48]. Finally, several works study the problem of rank aggregation in the context of group recommendations [326, 50, 66, 287, 297].

Several methods to combine different ranked lists are presented in the IR literature. There the *data fusion* problem is defined. Given a set of ranked lists of documents returned by different search engines, construct a single ranked list combining the individual rankings [157]. Data fusion techniques can be classified based on whether they require knowledge of the relevance scores [42]. The simplest method based solely on the documents' ranks is the Borda-fuse model. It assigns as score to each document the summation of its rank in each list. The Condorcet-fuse method [292] is based on a majoritarian voting algorithm, which specifies that a document d_1 is ranked higher in the fused list than another document d_2 if d_1 is ranked higher than d_2 more times than d_2 is ranked higher than d_1. The approach in [170], assumes that a document is ranked better than another if the majority of input rankings is in concordance with this fact and at the same time only a few input rankings refute it. When the relevance scores are available, other fusion techniques, including CombSUM, CombANZ and CombMNZ, can be applied [174]. In CombSUM, the fused relevance score of a document is the summation of the scores assigned by each source. In CombANZ (resp. CombMNZ), the final score of a document is calculated as that of CombSUM divided (resp. multiplied) by the number of lists in which the document appears.

2.8.2 Skyline Computation

The work of [98] rekindled interest in the problem of finding the *maximal objects* [251] and re-introduces it as the *skyline operator* (a.k.a. *skyline query*). Given a database of objects, the skyline query returns those objects which are not dominated, refereed as *skyline* or *maximal objects*. An object is *dominated* if there exists another object before it according to the partial order enforced by the Pareto-based aggregation. In other words, an object *dominates* another, if it has better values on all attributes, and strictly better value on at least one. Finding the skyline is also known as the *Pareto-optimal set*, or *maximal vectors* problem in multi-objective optimization research, where it has been studied extensively in the past, for in-memory computations [251, 64, 63].

External memory skyline algorithms can be classified into three categories: (1) *scan-based*, (2) *index-based*, and (3) *partitioning-based* algorithms.

The *scan-based* approaches perform multiple passes over the dataset and use a small window of candidate objects, which is used to prune dominated objects. The algorithms of this category can be further classified into two approaches: with and without pre-processing. Algorithms of the first category, directly process the set of objects, in the order in which they are stored, or produced (e.g., in the case of pipelining multiple operators). The *Block Nested Loops* (BNL) [98] and *Randomize multi-pass* (RAND) [332] algorithms lie in this category (for more details see Section 3.4). On the other hand, methods in the second category perform an external sort of the objects before, or parallel to the skyline computation. These methods are based on the fact that examining points according to a monotone (in all attributes) preference function reduces the average number of dominance checks. The *Sort Filter Skyline* (SFS) [123] and *Linear Elimination Sort for Skyline* (LESS) [188], belong to this category (for more details see Section 3.4). Other algorithm, include *Sort and Limit Skyline algorithm* (SaLSa) [53], which is similar to SFS and additionally introduces a condition for early terminating the input file scan, and *Skyline Operator on Anti-correlated Distributions* (SOAD) [338], which is also similar to SFS but uses different sorting functions for different sets of attributes.

In *index-based* approaches, various types of indices are used to guide the search for skyline points and prune large parts of the space. The most well-known and efficient method is the *Branch and Bound Skyline* (BBS) [300] algorithm. BBS employs an R-tree, and is shown to be I/O optimal with respect to this index. Similarly, the *Nearest Neighbor* algorithm (NN) [245] also uses an R-tree performing multiple nearest neighbor searches to identify skyline objects. A bitmap structure is used by *Bitmap* [362] algorithm to encode the input data. In the *Index* [362] algorithm, several B-trees are used to index the data, one per dimension. Other methods, e.g., [258, 266], employ a space-filling curve, such as the Z-order curve, and use a single-dimensional index. The *Lattice Skyline* (LS) algorithm [293] builds a specialized data structure for low-cardinality domains.

In *partitioning-based* approaches, algorithms divide the initial space into several partitions. The first algorithm in this category, *D&C* [98] computes the skyline objects adopting the divide-and-conquer paradigm. A similar approach with stronger theoretical guarantees is presented in [341, 340]. Recently, partitioning-based skyline algorithms which also consider the notion of incomparability are proposed in [404, 256]. *Object-based space partitioning scheme* (OSP) [404] attempts to reduce the number of checks between incomparable points by recursively partition the skyline points. *BSkyTree* [256] enhances [404] by considering both the notions of dominance and incomparability while partitioning the space.

Considering specific settings, several algorithms are proposed to efficiently compute the skyline over partially ordered domains [111, 394, 329, 405], metric spaces [119], non-metric spaces [299], or anticorrelated distributions [338].

Further, several lines of research attempt to address the issue that the size of skyline

cannot be controlled, by introducing new concepts and/or ranking the skyline (see [269] for a survey). [398] ranks tuples based on the number of records they dominate. [112] deals with high-dimensional skylines, and relaxes the notion of dominance to k-dominance, according to which a record is k-dominated if it is dominated in a subspace of k dimensions. [272] uses a skyline-based partitioning to rank tuples. The k most representative skyline operator is proposed in [263], which selects a set of k skyline points, so that the number of points dominated by at least one of them is maximized. In a similar spirit, [363] tries to select the k skyline points that best capture the trade-offs among the parameters. Finally, [257] attempts to find a small and focused skyline set. The size of the skyline is reduced by asking from users to state additional preferences.

2.9 Summary

This work addressed objective ranking techniques for a group of preferences over categorical attributes, where the goal is to rank objects based on what is considered ideal by all users. In particular, we study three related problems based on a double Pareto aggregation. The first is to return the set of objects that are unanimously considered ideal by the entire group. In the second problem, we relax the requirement for unanimity and only require a percentage of users to agree. Then, in the third problem, we devise an effective ranking scheme based on our double Pareto aggregation framework. The proposed methods take advantage of a transformation of the categorical attribute values in order to use a standard index structure. A detailed experimental study verified the efficiency and effectiveness of our techniques.

Chapter 3

External Memory Skyline Algorithms

Skyline queries return the set of non-dominated tuples, where a tuple is dominated if there exists another with better values on all attributes. In the past few years the problem has been studied extensively, and a great number of external memory algorithms have been proposed. We thoroughly study the most important scan-based methods, which perform a number of passes over the database in order to extract the skyline. Although these algorithms are specifically designed to operate in external memory, there are many implementation details which are neglected, as well as several design choices resulting in different flavors for these basic methods. We perform an extensive experimental evaluation using real and synthetic data. We conclude that specific design choices can have a significant impact on performance. We also demonstrate that, contrary to common belief, simpler skyline algorithm can be much faster than methods based on pre-processing.

3.1 Introduction

The *skyline query*, or skyline operator as it was introduced in [98], has in the past few years received great attention in the data management community. Given a database of objects, the skyline query returns those objects which are not dominated. An object *dominates* another, if it has better values on all attributes, and strictly better value on at least one. Finding the skyline is also known as the *Pareto-optimal set*, or *maximal vectors* problem in multi-objective optimization research, where it has been studied extensively in the past, but only for in-memory computations. For example the well-known divide and conquer algorithm of [251] has complexity $O(N \log^{d-2} N)$, for $d \geq 2$, where N is the number of objects, and d their dimensionality; the algorithm is optimal for $d = 3$.

The interest in external memory algorithms has sparked after the seminal work in [98]. The most efficient method in terms of worst-case Input/Output (I/O) operations is the algorithm in [341], which requires in the worst case $O\left((N/B)\log_{M/B}^{d-2}(N/B)\right)$ I/Os, where M is the memory size and B the block (minimum unit of transfer in an I/O operation) size in terms of objects. However, in practice, other external-memory algorithms proposed over the past years can be faster.

This work studies in detail an important class of practical algorithms, the *scan-based skyline algorithms*. An algorithm of this class performs multiple passes over an input file, where the input file in the first pass is the database, and in a subsequent pass it is the output of the previous pass. The algorithm terminates when the output file remains empty after a pass concludes. Generally speaking, during each pass, the algorithm maintains in main memory a small *window* of incomparable objects, which it uses to remove dominated

objects from the input file. Any object not dominated is written to the output file.

Although the studied algorithms are specifically designed to operate in external memory, little attention has been given to important implementation details regarding memory management. For example, all algorithms assume that the unit of transfer during an I/O operation is the object, whereas in a real system is the block, i.e., a set of objects. Our work addresses such shortcomings by introducing a more realistic I/O model that better captures performance in a real system. Furthermore, by thoroughly studying the core computational challenge in these algorithms, which is the management of the objects within the window, we introduce several novel potentially interesting policies.

Contributions. Summarizing, the contributions of our study are the following:

1. Based on a standard external memory model [22], we appropriately adapt four popular scan-based algorithms, addressing in detail neglected implementation details regarding memory management.
2. We focus on the core processing of scan-based algorithms, the management of objects maintained in the in-memory window. In particular, we introduce various policies for two tasks: traversing the window and evicting objects from the window. Both tasks can have significant consequences in the number of required I/Os and in the CPU time.
3. We experimentally evaluate concrete disk-based implementations, rather than simulations, of all studied algorithms and derive useful conclusions for synthetic and real datasets. In particular, we demonstrate that, in many cases and contrary to common belief, algorithms that pre-process (typically, sort) the database are not faster.
4. We perform an extensive study of our proposed policies, and reach the conclusion that in some settings (dimensionality and dataset distribution) these policies can reduce the number of dominance checks by more than 50%.

3.2 Preliminaries

3.2.1 Definitions

Let $\mathcal{O}$ be a set of *d-dimensional objects*. Each object $o \in \mathcal{O}$ is represented by its *attributes* $o = (o^1, o^2, \ldots, o^d)$. The *domain* of each attribute, is the positive real numbers set $\mathbb{R}^+$. Without loss of generality, we assume that an object o_1 is *better* than another object o_2 on an attribute j, iff $o_1^j < o_2^j$. An object o_1 *dominates* another object o_2, denoted by $o_1 \succ o_2$, iff (1) $\forall i \in [1,d]$, $o_1^i \leqslant o_2^i$ and (2) $\exists j \in [1,d]$, $o_1^j < o_2^j$. The *skyline* of an object set $\mathcal{O}$, denoted as $SL(\mathcal{O})$, is the set of objects in $\mathcal{O}$ that are not dominated by any other object of $\mathcal{O}$. Formally, $SL(\mathcal{O}) = \{o_i \in \mathcal{O} \mid \nexists o_k \in \mathcal{O} : o_k \succ o_i\}$.

3.2.2 External Memory I/O Model

This section describes an *external memory model*, similar to that of [22]. The unit of transfer between the main memory and the external memory (i.e., the disk) is a single *block*.[1] Any external memory algorithm, like the skyline methods, read/write blocks from/to disk *files*. We assume that files are stored contiguously on disk, and therefore a new block is written always at the end of a file.

We denote as $N = |\mathcal{O}|$ the size of the database, i.e., N is the total number of objects to be processed. We measure the fixed size B of a block in terms of objects (tuples). Similarly, main memory can fit M objects, with the requirements that $M < N$ (and often

[1] [22] assumes that P blocks can be transferred concurrently; in this work we set $P = 1$

much smaller) to justify the need for external memory algorithms, and $M > 2B$ to support basic in-memory operations.

We next discuss Input/Output (I/O) operations. We assume no input or output buffers, so that blocks from the disk are transferred directly to (resp. from) the disk from (resp. to) the main memory. Equivalently, the input/output buffers share the same memory of size M with the algorithm.

We categorize I/O operations in two ways. Naturally, a *read* transfers data from the disk, whereas a *write* transfers data to the disk. The second categorization is based on the number of blocks that are transferred. Note that a read (resp. write) operation transfers at least one block and at most $\lfloor \frac{M}{B} \rfloor$ blocks into main memory (resp. disk). We also remark that in disks, the *seek time*, i.e., the time it takes for the head to reach the exact position on the ever spinning disk where data is to be read or written, is a crucial parameter in disk performance. Reading or writing k consecutive blocks on the disk is much faster than reading or writing k blocks in arbitrary positions on the disk. The reason is that only one seek is required in the first case, compared to the k seeks for the second. Therefore, we distinguish between *sequential* and *random* I/Os. A random I/O incorporates the seek time, whereas a sequential I/O does not. For example, when a procedure reads k blocks sequentially from the disk, we say that it incurs 1 random read and $k-1$ sequential reads.

3.3 A Model for Scan-based Skyline Algorithms

All skyline algorithms maintain a set of objects, termed *window*, which consists of possible skyline objects, actual skyline objects, or some arbitrary objects in general. A common procedure found in all algorithms is the following. Given some candidate object not in the window, *traverse* the window and determine if the candidate object is dominated by a window object, and, if not, additionally determine the window objects that it dominates. Upon completion of the traversal and if the candidate is not dominated, the skyline algorithm may choose to insert it into the window, possible *evicting* some window objects.

In the aforementioned general procedure, we identify and focus on two distinct design choices. The first is the *traversal policy* that determines the order in which window objects are considered and thus dominance checks are made. This design choice directly affects the number of dominance checks performed and thus the running time of the algorithm. An ideal (but unrealistic) traversal policy would require only one dominance check in the case that the candidate is dominated, i.e., visit only a dominating window object, and/or visit only those window objects which the candidate dominates.

The second design choice is the *eviction policy* that determines which window object(s) to remove so as to make room for the candidate object. This choice essentially determines the dominance power of the window, and can thus indirectly influence both the number of future dominance checks and the number of future I/O operations.

We define three basic *window traversal policies*:

- The *sequential traversal policy* (sqT), where window objects are traversed *sequentially*, i.e., in the order they are stored. This policy is the one adopted by all existing algorithms.
- The *random traversal policy* (rdT), where window objects are traversed in *random* order. This policy is used to gauge the effect of others.
- The *entropy-based traversal policy* (enT), where window objects are traversed in ascending order of their *entropy* (i.e., $\sum_{i=1}^{d} \ln(o^i + 1)$) values. Intuitively, an object with a low entropy value has greater dominance potential as it dominates a large volume of the space.

In addition to basic traversal policies, we define various ranking schemes for objects, which will be discussed later. These schemes attempt to capture the dominance potential of an object, with higher ranks suggesting greater potential. Particularly, we also consider the following window traversal policies:

- The *ranked-based traversal policy* (rkT), where window objects are traversed in descending order based on their *rank* values. Moreover, we consider three hybrid random-, rank-based traversal policies.
- The *highest-random traversal policy* ($hgRdT$), where the k objects with the highest rank are traversed first, in descending order of their rank; then, the random traversal policy is adopted.
- The *lowest-random traversal policy* ($lwRdT$), where the k objects with the lowest rank are compared first, before continuing with a random traversal.
- The *recent-random traversal policy* ($rcRdT$), where the k most recently read objects are compared first, before continuing with a random traversal.

Moreover, we define three *eviction policies*:

- The *append eviction policy* (apE), where the *last* inserted object is removed. This is the policy adopted by the majority of existing algorithms.
- The *entropy-based eviction policy* (enE), where the object with the *highest entropy* value is removed.
- The *ranked-based eviction policy* (rkE), where the object with the *lowest rank* value is removed. In case of ties in entropy or rank values, the most recent object is evicted.

We next discuss ranking schemes used in the ranked-based traversal and eviction policies. Each window object is assigned a rank value, initially set to zero. Intuitively, the rank serves to identify "promising" objects with high dominance power, i.e., objects that dominate a great number of other objects. Then, the skyline algorithm can exploit this information in order to reduce the required dominance checks by starting the window traversal from promising objects, and/or evict non-promising objects.
We define the following *ranking schemes*:

- *r0R*: the rank of an object o at a time instance t, is equal to the number of objects that have been dominated by o until t. In other words, this ranking scheme counts the number of objects dominated by o.
- *r1R*: this ranking is similar to *r0R*. However, it also considers the number of objects that have been dominated by the objects that o dominates. Let $rank(o)$ denote the rank of an object o. Assume that object o_1 dominates o_2, Then, the rank of o_1 after dominating o_2 is equal to $rank(o_1) + rank(o_2) + 1$.
- *r2R*: this ranking assigns two values for each object o, its *r1R* value, as well as the number of times o is compared with another object and none of them is dominated (i.e., the number of incomparable dominance checks). The *r1R* value is primarily considered to rank window objects, while the number of incomparable check is only considered to solve ties; the more incomparable checks an object has, the lower its rank.

Finally, several scan-based skyline algorithm perform a preprocessing step in which the input objects are *sorted* according to a function monotone on all attributes. In this study, we consider the three most common *sorting functions*:

- The *entropy sorting function* (*Entr*) defined as $Entr(o) = \sum_{i=1}^{d} \ln(o^i + 1)$.
- The *sum sorting function* (*Sum*), defined as the sum of the object's attributes values.
- The *minimum sorting function (minC)*, which sorts objects ascending on their minimum attribute value; the *Sum* function is used to solve ties and guarantee monotonicity.

3.4 Algorithm Adaptations for the I/O Model

3.4.1 Block Nested Loop Algorithm (BNL)

The *Block Nested Loop* (BNL) [98] algorithm is one of the first external memory algorithms for skyline computation. All computations in BNL occur during the window traversal. Therefore, BNL uses a window as big as the memory allows. In particular, let W denote the number of objects stored in the window, and let O_b denote the number of objects scheduled for writing to disk (i.e., in the output buffer). The remaining memory of size $I_b = M - W - O_b$ serves as the input buffer, to retrieve objects from the disk. Note that the size of the I/O buffers I_b and O_b vary during the execution of BNL, subject to the restriction that the size of the input buffer is always at least one disk block, i.e, $I_b \geq B$, and that the output buffer never exceeds a disk block, i.e., $O_b \leq B$; we discuss later how BNL enforces this requirements.

We next describe memory management in the BNL algorithm. BNL performs a number of passes, where in each an input file is read. For the first pass, the input file is the database, whereas the input file in subsequent passes is created at the previous pass. BNL terminates when the input file is empty. During a pass, the input file is read in *chunks*, i.e., sets of blocks. In particular, each read operation transfers into main memory exactly $\lfloor \frac{I_b}{B} \rfloor$ blocks from disk, incurring thus 1 random and $\lfloor \frac{I_b}{B} \rfloor - 1$ sequential I/Os. On the other hand, whenever the output buffer fills, i.e., $O_b = B$, a write operation transfers into disk exactly 1 block and incurs 1 random I/O.

We now discuss what happens when a chunk of objects is transfered into the input buffer within the main memory. For each object o in the input buffer, BNL traverses the window, adopting the sequential traversal policy (sqT). Then, BNL performs a two-way dominance check between o and a window object w. If o is dominated by w, o is discarded and the traversal stops. Otherwise, if o dominates w, object w is simply removed from the window.

At the end of the traversal, if o has not been discarded, it is appended in the window. If W becomes greater than $M - O_b - B$, BNL needs to move an object from the window to the output buffer to make sure that enough space exists for the input buffer. In particular, BNL applies the append eviction policy (apE), and selects the last inserted object, which is o, to move into the output buffer. If after this eviction, the output buffer contains $O_b = B$ objects, its contents are written to the file, which will become the input file of the next pass.

A final issue is how BNL identifies an object o to be a skyline object, BNL must make sure that o is dominance checked with all surviving objects in the input file. When this can be guaranteed, o is removed from the window and returned as a result. This process is implemented through a *timestamp mechanism*; details can be found in [98].

3.4.2 Sort Filter Skyline Algorithm (SFS)

The *Sort Filter Skyline* (SFS) [123] algorithm is similar to BNL with one significant exception: the database is first sorted by an external sort procedure according to a monotonic scoring function. SFS can use any function defined in Section 3.3.

Similar to BNL, the SFS algorithm employs the sequential window traversal policy (sqT) and the append eviction policy (apE). There exist, however, two differences with respect to BNL. Due to the sorting, dominance checks during window traversal are one-way. That is an object o is only checked for dominance by a window object w. In addition, the skyline identification in SFS is simpler than BNL. At the end of each pass, all window objects are guaranteed to be results and are thus removed and returned.

3.4.3 Linear Elimination Sort for Skyline Algorithm (LESS)

The *Linear Elimination Sort for Skyline* (LESS) [188] algorithm improves on the basic idea of SFS, by performing dominance checks during the external sort procedure. Recall that standard external sort performs a number of passes over the input data. The so-called zero pass (or sort pass) brings into main memory M objects, sorts them in-memory and writes them to disk. Then, the k-th (merge) pass of external sort, reads into main memory blocks from up to $\lfloor M/B \rfloor - 1$ files created in the previous pass, merges the objects and writes the result to disk.

LESS changes the external sort procedure in two ways. First, during the zero pass, LESS maintains a window of size W_0 objects as an elimination filter to prune objects during sorting. Thus the remaining memory $M - W_0$ is used for the in-memory sorting. The window is initially populated after reading the first $M - W_0$ objects by selecting those with the lowest entropy scores. Then for each object o read from the disk and before sorting them in-memory, LESS performs a window traversal. In particular, LESS employs the sequential traversal policy (sqT) performing a one-way dominance check, i.e., it only checks if o is dominated. Upon comparing all input objects with the window, the object with the lowest entropy o_h is identified. Then, another sequential window traversal (sqT) begins, this time checking if o_h dominates the objects in the window. If o_h survives, it is appended in the window, evicting the object with the highest entropy score, i.e., the entropy-based eviction policy (enE) is enforced.

The second change in the external sort procedure is during its last pass, where LESS maintains a window of size W objects. In this pass, as well as any subsequent skyline processing passes, LESS operates exactly like SFS. That is the sequential traversal policy (sqT) is used, one-way dominance checks are made, and window objects are removed according to the append eviction policy (epE).

3.4.4 Randomized Multi-pass Streaming Algorithm (RAND)

In the *Randomized multi-pass streaming* (RAND) algorithm [332], each pass in RAND consists of three phases, where each scans the input file of the previous pass. Therefore, each pass essentially corresponds to three reads of the input file. In the first phase, the input file is read and a window of maximum size $W = M - B$ is populated with randomly sampled input objects (using reservoir sampling).

In the second phase, the input file is again read one block at a time, while the window of W objects remain in memory. For each input object o, the algorithm traverses the window in sequential order (sqT), performing one-way dominance checks. If a window object w is dominated by o, w is replaced by o. Note that, at the end of this phase, all window objects are skyline objects, and can be returned. However, they are not removed from memory.

In the third phase, for each input object o, RAND performs another sequential traversal of the window (sqT), this time performing an inverse one-way dominance check. If o is dominated by a window object w, or if o and w correspond to the same object, RAND discards o. Otherwise it is written on a file on the disk, serving as the input file for the next pass. At the end of this phase, the memory is cleaned.

Table 3.1: Parameters

Description	Parameter	Values
Number of Objects	N	50k, 100K, **500K**, 1M, 5M
Number of Attributes	d	3, **5**, 7, 9, 15
Memory Size	$M/N(\%)$	0.15%, 0.5% **1%**, 5%, 10%
Block Size (Bytes)	$B \cdot$ object size	1024, **2048**, 4096
Distribution	-	INT, CORR, ANT

3.5 Experimental Analysis

3.5.1 Setting

3.5.1.1 Datasets

Our experimental evaluation involves both synthetic and real datasets. To construct synthetic datasets, we consider the three standard distribution types broadly used in the skyline literature. In particular, the distributions are: anti-correlated (ANT), correlated (CORR), and independent (IND). The synthetic datasets are created using the generator developed by the authors of [98].

We also perform experiments on three real datasets. *NBA* dataset consists of $17,264$ objects, containing statistics of basketball players. For each player we consider 5 statistics (i.e., points, rebound, assist, steal blocks). *House* is 6-dimensional dataset consists of $127,931$ objects. Each object, represents the money spent in one year by an American family for six different types of expenditures (e.g., gas, electricity, water, heating, etc.). Finally, *Colour* is a 9-dimensional dataset, which contains $68,040$ objects, representing the first three moments of the RGB color distribution of an image.

3.5.1.2 Implementation

All algorithms, described in Section 3.3, were written in C++, compiled with gcc, and experiments were performed on a 2.6GHz CPU. In order to accurately convey the effect of I/O operations, we disable the operating system caching, and perform direct and synchronous I/O's.

The size of each object is set equal to 100 bytes, as was the case in the experimental evaluation of the works that introduced the algorithms under investigation. Finally, the default size of block is set to 2048 bytes; hence, in default setting, each block contains 20 object.

3.5.1.3 Metrics

To gauge efficiency of all algorithms, we measure: (1) the number of disk I/O operations, which are distinguished into four categories, read, write operations, performed during the pre-processing phase (i.e., sorting) if any, and read, write operations performed during the main computational phase; (2) the number of dominance checks; (3) the time spent solely on CPU processing denoted as CPU Time and measured in seconds; (4) the total execution time, denoted as Total Time and measured in seconds; In all cases the reported time values are the averages of 5 executions.

3.5.2 Algorithms Comparison

Table 3.1 lists the parameters and the range of values examined. In each experiment, we vary a single parameter and set the remaining to their default (bold) values. SFS and

LESS sort according to the entropy function. During pass zero in LESS, the window is set to one block.

3.5.2.1 Varying the number of objects

In this experiment, we vary the number of objects from 50K up to 5M and measure the total time, number of I/O's and dominance checks, and CPU time, in Figures 3.1~3.4.

The important conclusions from Figure 3.1 are two. First, RAND and BNL outperform the other methods in anti-correlated datasets. This is explained as follows. Note that the CPU time mainly captures the time spent for the following task: dominance checks, data sorting in case of LESS/SFS, and skyline identification, in case of BNL. From Figure 3.4 we can conclude that BNL spends a lot of CPU time in skyline identification. BNL requires the same or more CPU time than RAND, while BNL performs fewer dominance checks than RAND. This is more clear in the case of independent and correlated datasets where the cost for dominance checks is lower compared to the anti-correlated dataset. In these datasets, the BNL CPU time increased sharply as the cardinality increases.

The second conclusion is that, in independent and correlated datasets, the performance of BNL quickly degrades as the cardinality increases. This is due to the increase of the window size, which in turn makes window maintenance and skyline identification more difficult.

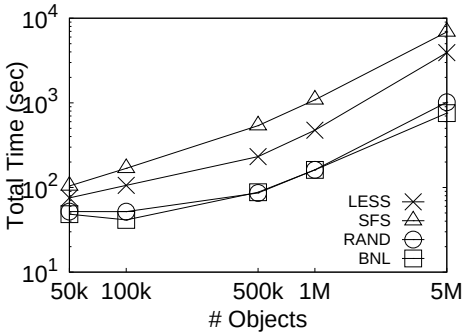
Figure 3.2 shows the I/O operations performed by the algorithms. We observe that BNL outperforms the other methods in almost all settings. Particularly, in the correlated dataset, LESS is very close to BNL. Also, we can observe that, in general, the percentage of write operations in LESS and SFS is much higher than in BNL and RAND. We should remark that, the write operations are generally more expensive compared to the read operations. Finally, for LESS and SFS, we can observe that the larger amount of I/O operations are performed during the sorting phase.

Regarding the number of dominance checks, shown in Figure 3.3, LESS and SFS perform the fewest, while RAND the most, in all cases. Figure 3.4 shows the CPU time spent by the methods. SFS spends more CPU time than LESS even though they perform similar number of dominance checks; this is because SFS sorts a larger number of object than LESS. Finally, as previously mentioned, BNL spends considerable CPU time for skyline identification.

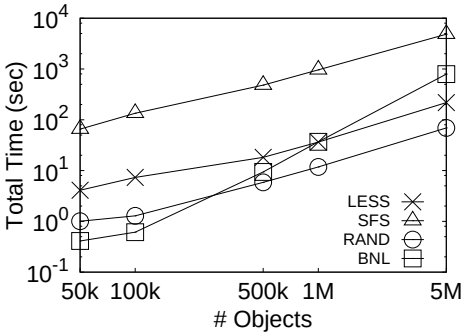
3.5.2.2 Varying the number of dimensions

In this experiment we investigate the performance as we vary the number of dimensions from 3 up to 15. In Figure 3.5 where the total time is depicted, the performance of all methods become almost the same for anti-correlated and independent datasets, as the dimensionality increases. In the correlated dataset, the skyline can fit in main memory, hence BNL and RAND require only a few passes, while SFS and LESS waste time sorting the data.

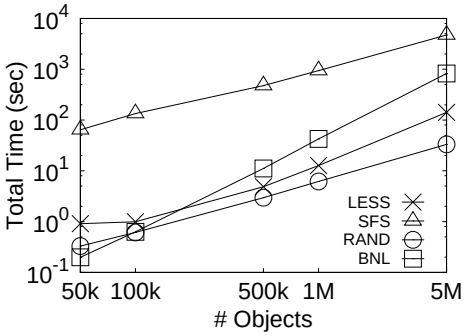
Regarding I/O's (Figure 3.6), BNL outperforms all other methods in all cases, while LESS is the second best method. Similarly, as in Figure 3.2, LESS and SFS performs noticeable more write operations compared to BNL and RAND. Figure 3.7 shows that LESS and SFS outperforms the other method, performing the same number of dominance checks. Finally, CPU time is presented in Figure 3.8, where once again the cost for skyline identification is noticeable for BNL.



(a) Anti-correlated

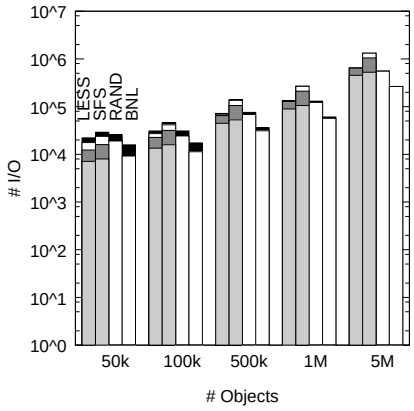


(b) Independent

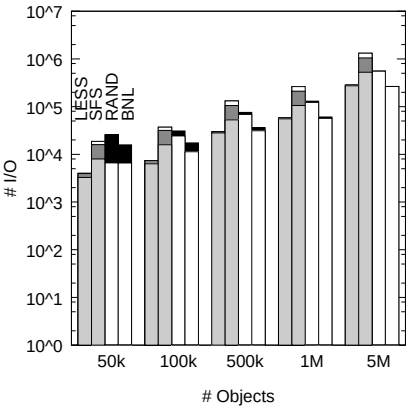


(c) Correlated

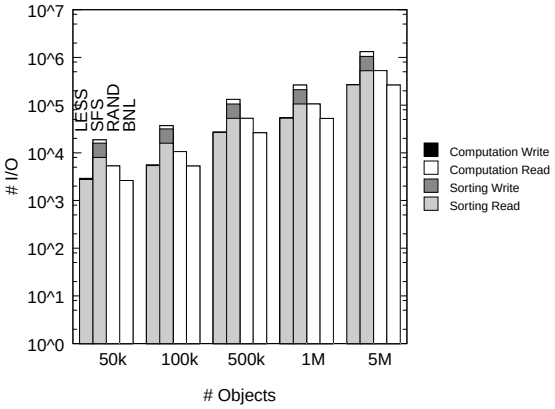
Figure 3.1: Total Time: varying number of objects



(a) Anti-correlated

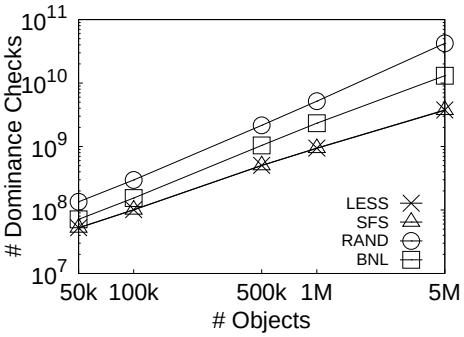


(b) Independent

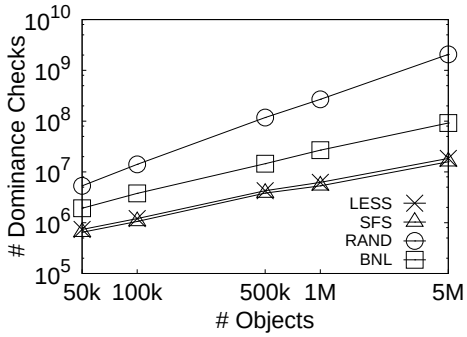


(c) Correlated

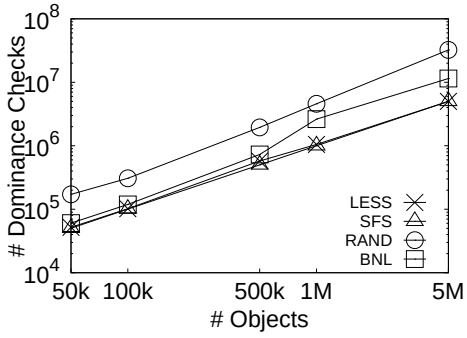
Figure 3.2: I/O Operations: varying number of objects



(a) Anti-correlated

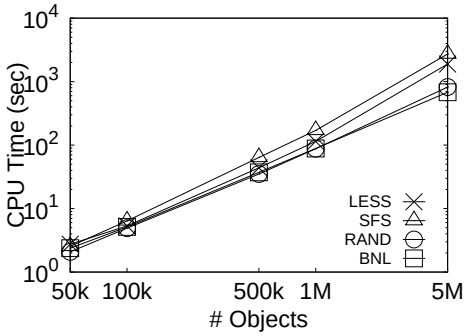


(b) Independent

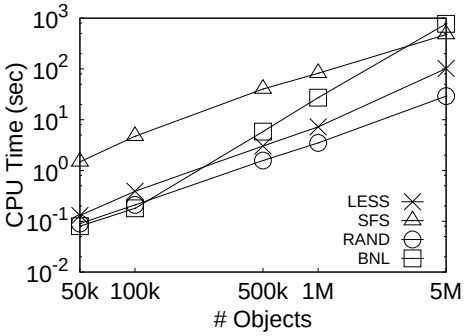


(c) Correlated

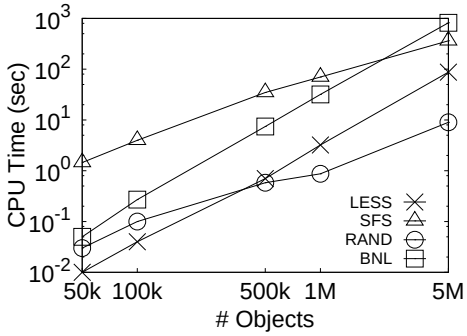
Figure 3.3: Dominance Checks: varying number of objects



(a) Anti-correlated

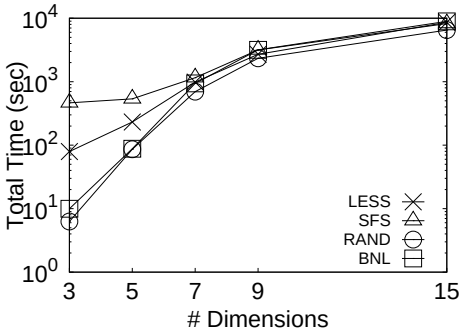


(b) Independent

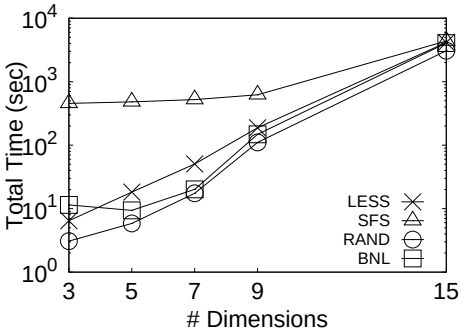


(c) Correlated

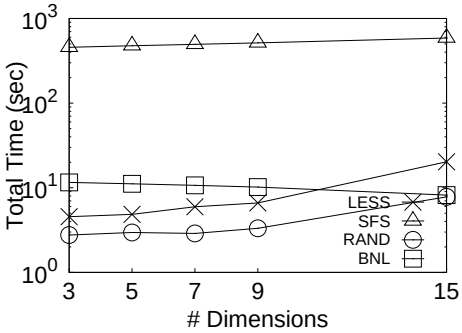
Figure 3.4: CPU Time: varying number of objects



(a) Anti-correlated

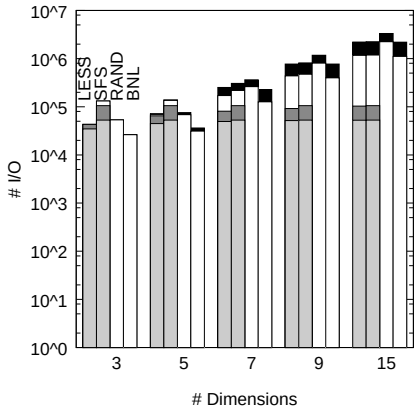


(b) Independent

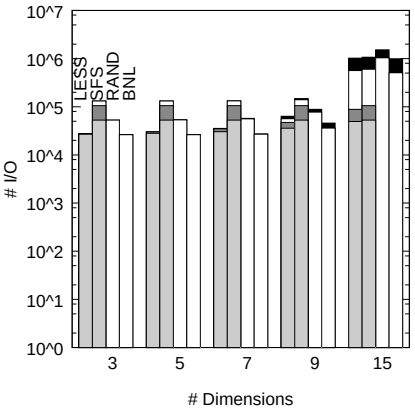


(c) Correlated

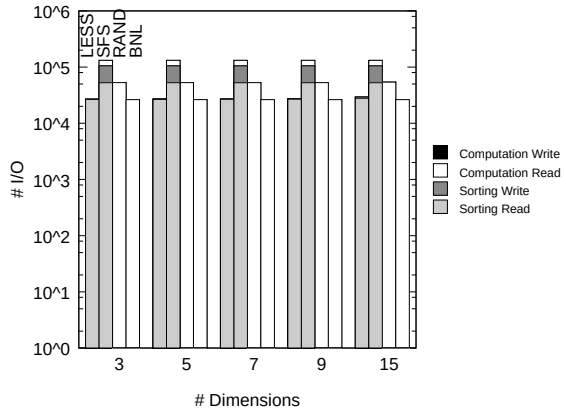
Figure 3.5: Total Time: varying number of attributes



(a) Anti-correlated

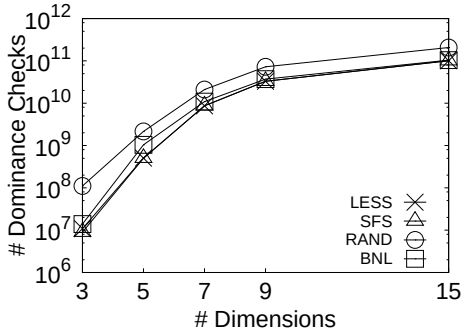


(b) Independent

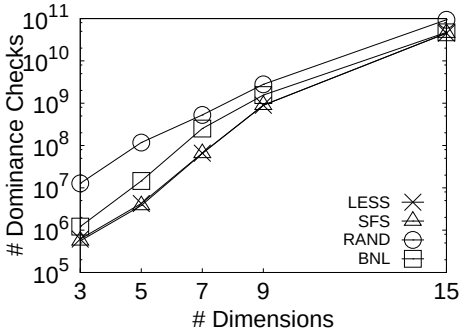


(c) Correlated

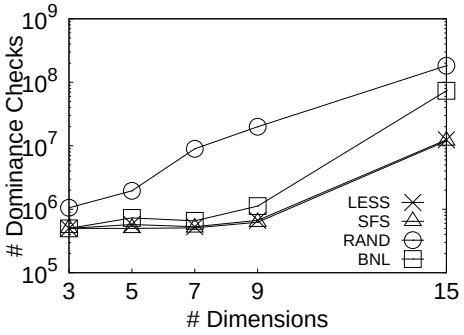
Figure 3.6: I/O Operations: varying number of attributes



(a) Anti-correlated

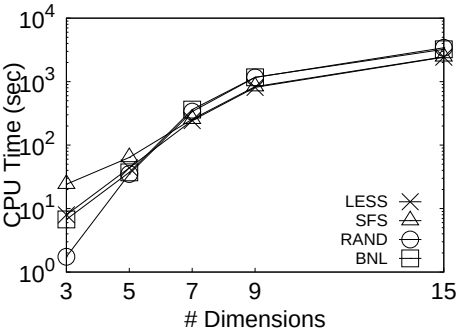


(b) Independent

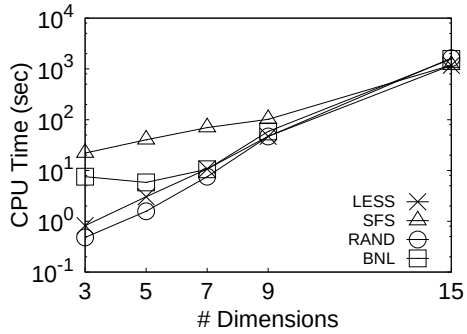


(c) Correlated

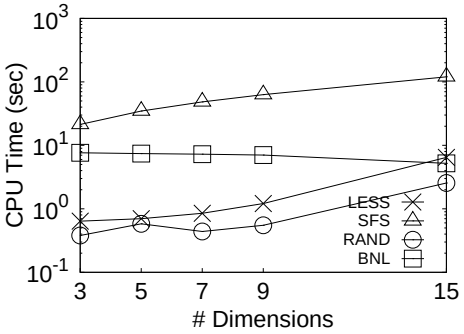
Figure 3.7: Dominance Checks: varying number of attributes



(a) Anti-correlated

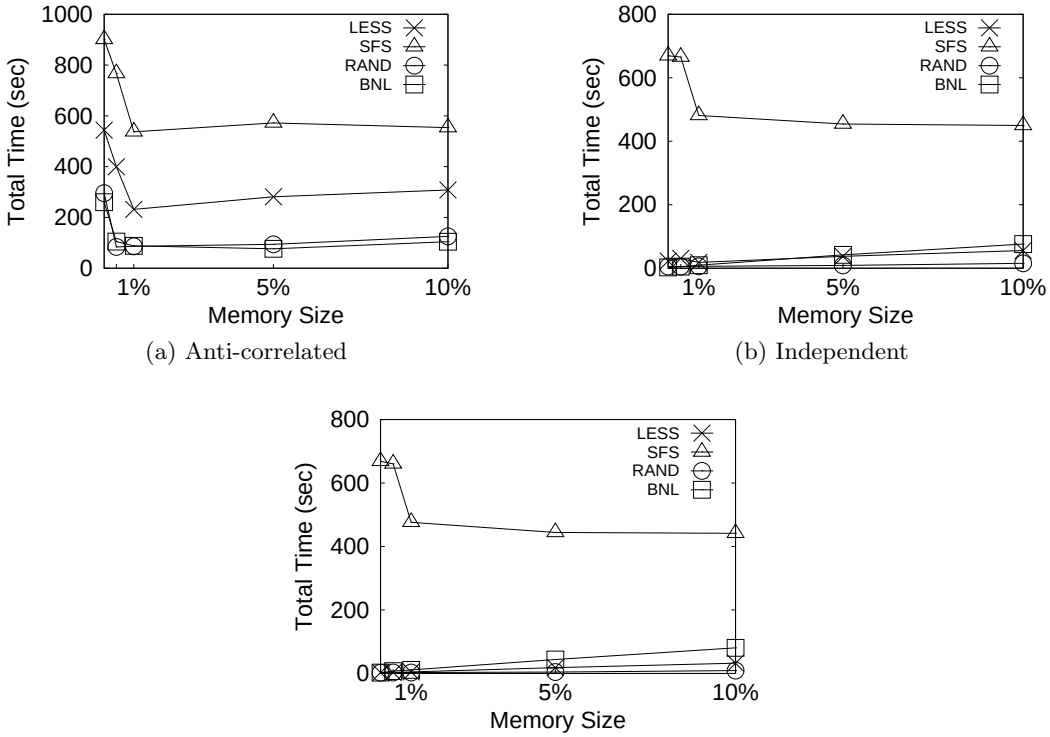


(b) Independent



(c) Correlated

Figure 3.8: CPU Time: varying number of attributes



(a) Anti-correlated

(b) Independent

(c) Correlated

Figure 3.9: Total Time: varying memory size

3.5.2.3 Varying the memory size

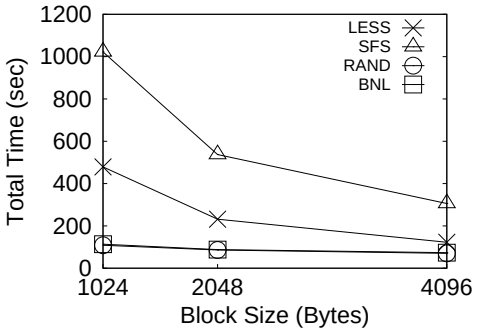
In Figure 3.9, we vary the size of the available memory. In general, the total time here, follows the trend of I/O operations. We observe that the required time of all methods decreased sharply for memory sizes up to 1%. However, beyond this point, the time is almost stable as the memory size increases, with the exception of BNL, where the time slightly increases (due to the skyline identification cost w.r.t. window size).

3.5.2.4 Varying the block size

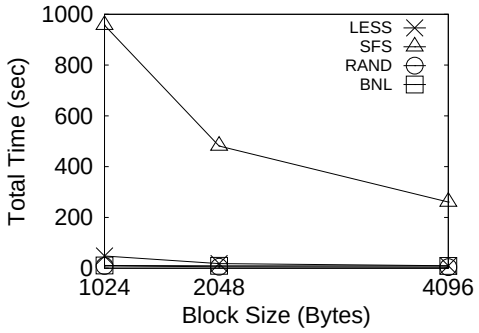
In this experiment we vary the block size, from 1024 up to 4096 bytes. Note that, as we previously mentioned the object size is equal to 100 bytes; as a result, a block of size 1024 bytes contains 1024/100 = 10 objects. Figure 3.10 shows the total time, which follows the I/O trend, since other factors remain stable.

3.5.2.5 Real Datasets

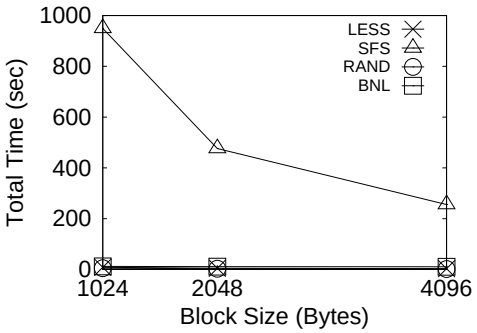
In this experiment, we evaluate our methods using the real datasets described in Section 3.5.1. Table 3.2 summarizes the results, presenting the total time required by all methods. We observe that BNL outperforms the other methods in all datasets in terms of total time. RAND outperforms the other methods in all cases, while SFS is the worst. Note that, in House and Colour datasets, RAND performs more dominance checks, and more I/O operations, than LESS. However, LESS requires more total time, due to larger number of write operations, and the CPU time spend for sorting.



(a) Anti-correlated



(b) Independent



(c) Correlated

Figure 3.10: Total Time: varying block size

Table 3.2: Real datasets: Total Time (sec)

Dataset	LESS	SFS	RAND	BNL
House	30.11	178.21	15.25	4.98
Colour	14.43	90.73	3.70	1.28
NBA	9.45	26.68	0.71	0.41

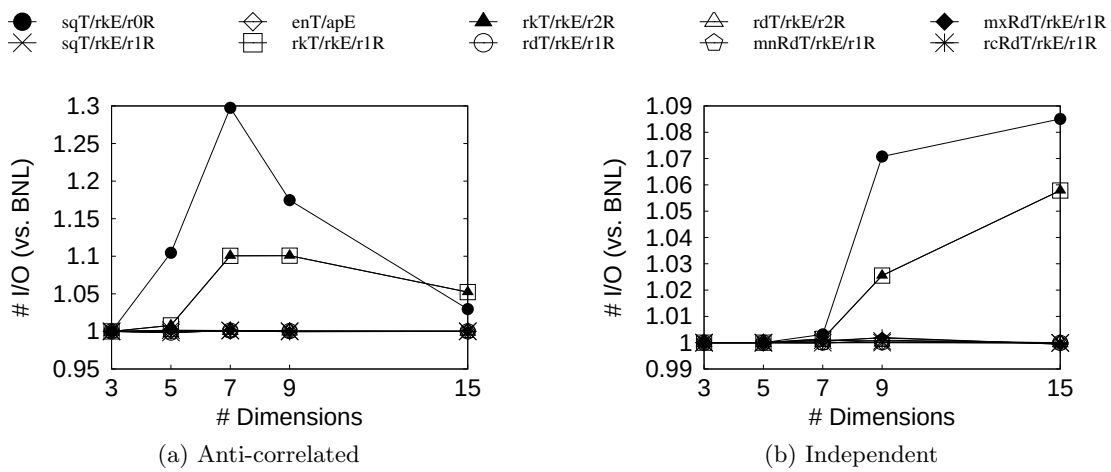
3.5.3 Policies Evaluation

In this experiment, we study the effect of different window policies in scan-based skyline algorithms. Particularly, we use BNL and SFS algorithms and we employ several traversal and eviction and policies, in conjunction with different ranking schemes. The effect of policies in LESS are similar to those in SFS and are not shown. Regarding RAND, only the window traversal policy affects its performance; its effect is not dramatic and hence it is also not shown.

All results are presented w.r.t. the original algorithms. That is, let m be a measurement for the original algorithm, and m' be the corresponding measurement for an examined variation. In this case, the measurement presented for the variation is $1 + (m' - m)/m$.

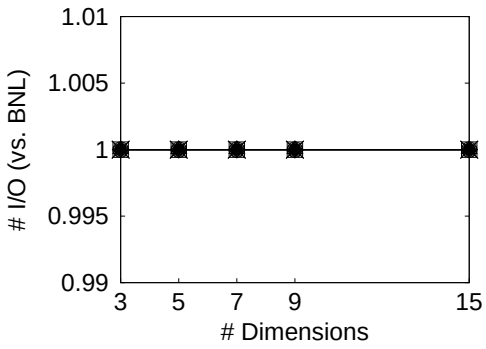
3.5.3.1 BNL

We first study the performance of BNL under the 10 most important policy and ranking scheme combinations. Figure 3.11 shows the I/O operations performed by the BNL flavors. As we can see, none of the examined variations performs significant better than the original algorithm. In almost all cases, the I/O performance of most variations is very close to the original. The reason is that the append eviction policy (apE), adopted by the original BNL already performs very well for two reasons. First, the apE policy always removes objects that have not dominated any other object. This way, the policy indirectly



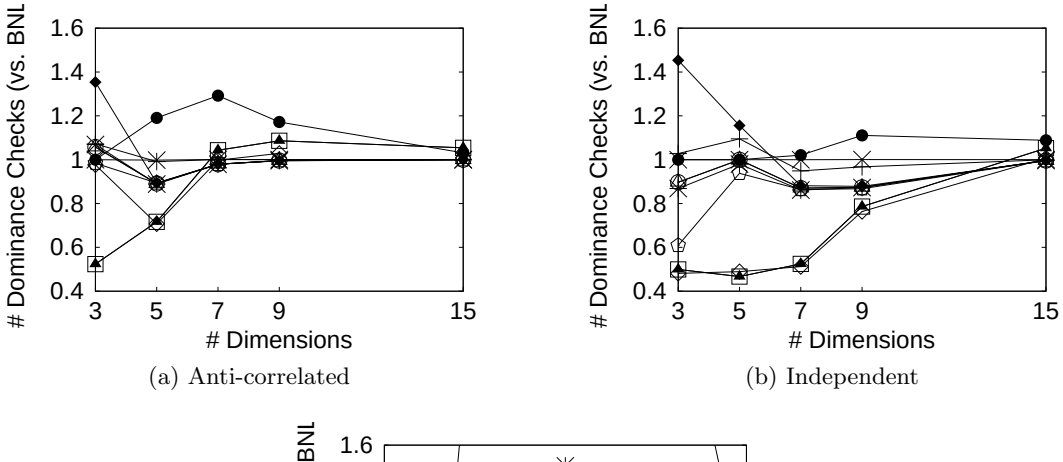
(a) Anti-correlated

(b) Independent



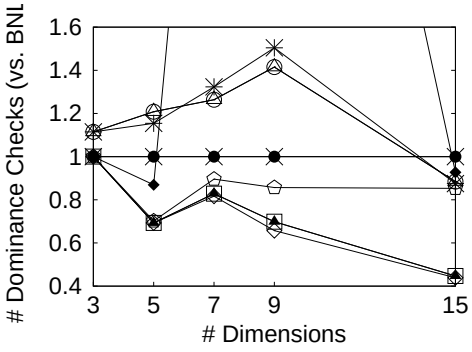
(c) Correlated

Figure 3.11: BNL Policies (I/O Operations): varying number of attributes



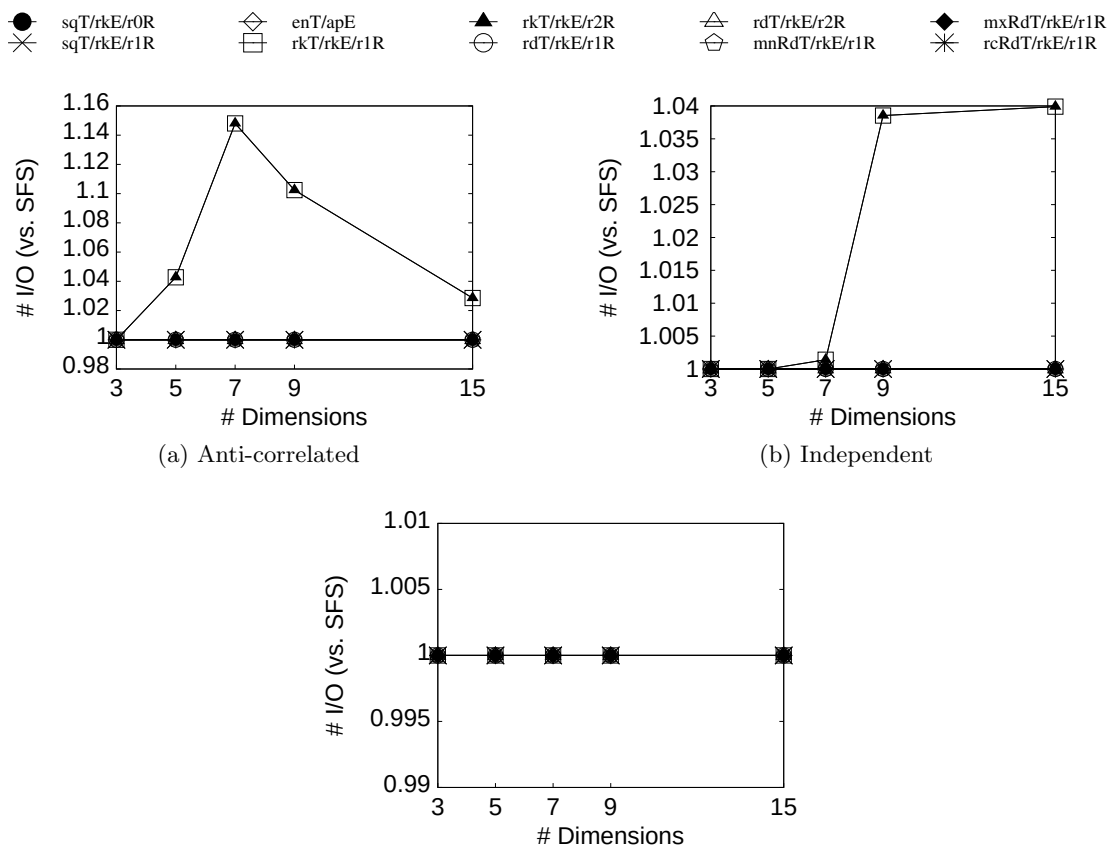
(a) Anti-correlated

(b) Independent



(c) Correlated

Figure 3.12: BNL Policies (Dominance Checks): varying number of attributes

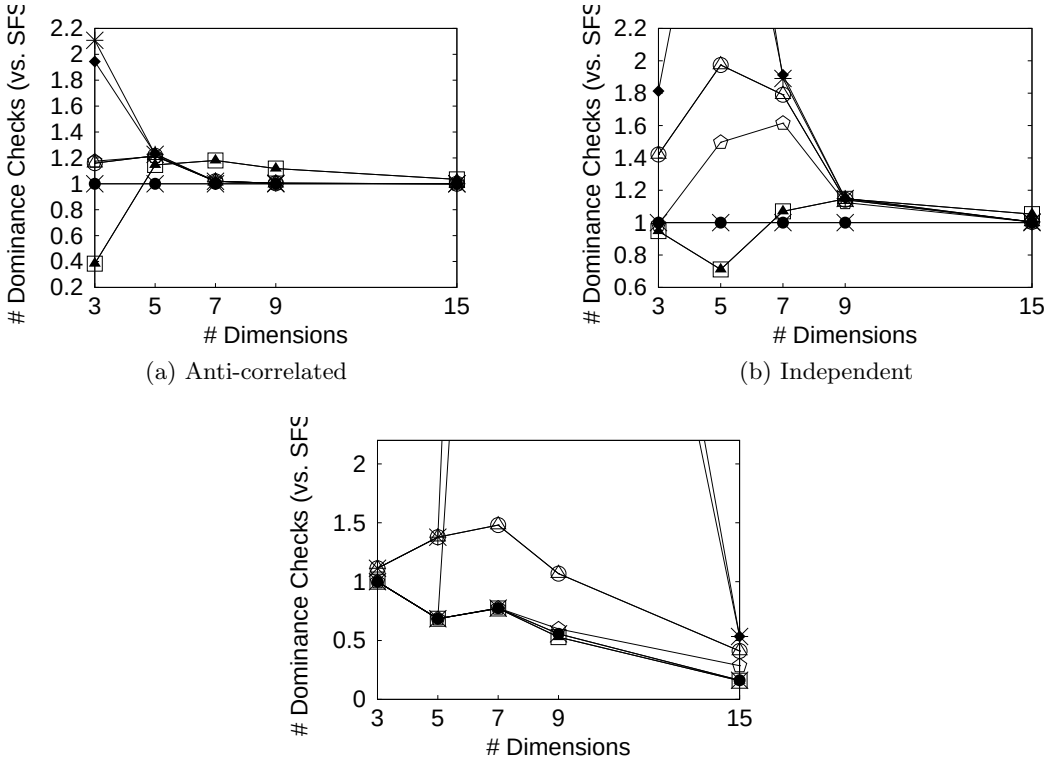


(a) Anti-correlated

(b) Independent

(c) Correlated

Figure 3.13: SFS Policies (I/O Operations): varying number of attributes

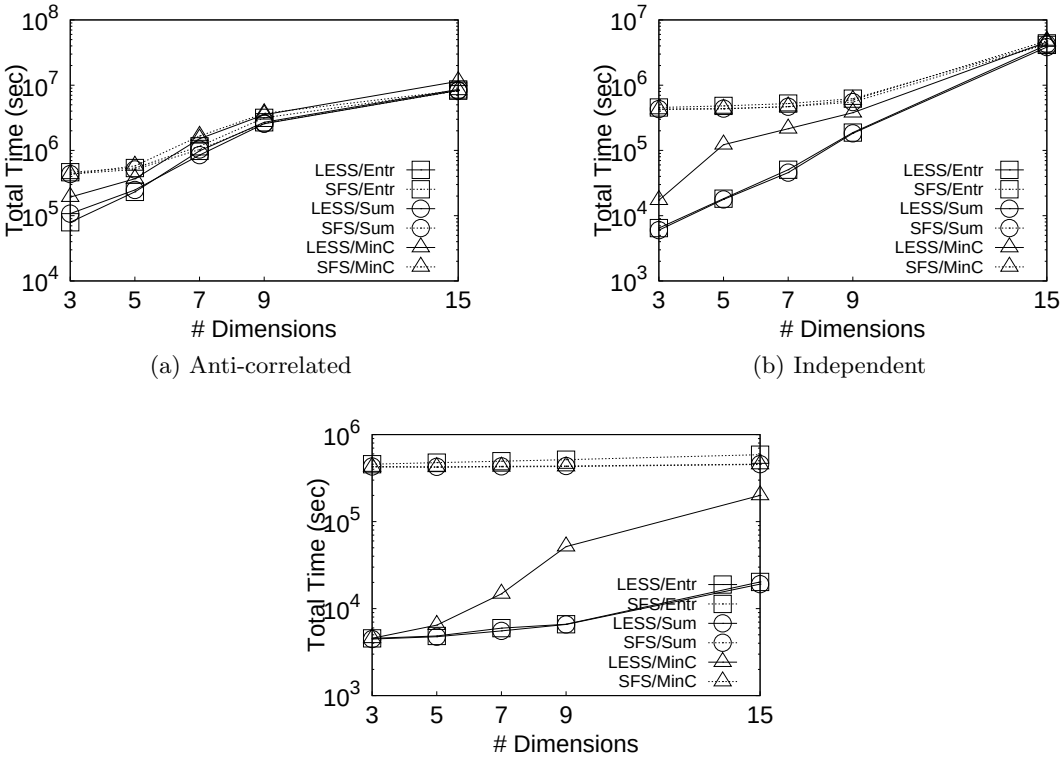


(a) Anti-correlated

(b) Independent

(c) Correlated

Figure 3.14: SFS Policies (Dominance Checks): varying number of attributes



(a) Anti-correlated

(b) Independent

(c) Correlated

Figure 3.15: Sorting Functions (Total Time): varying number of attributes

implements a dominance-oriented criterion. Second, the apE policy always removes the most recently read object, which is important for BNL. A just read object, requires the most time (compared to other objects in the window) in order to be identified as a skyline, thus propagated to the results and freeing memory. Hence, by keeping "older" objects we increase the probability of freeing memory in the near future. Still it is possible to marginally decrease the number of I/Os.

Figure 3.12 shows the number of dominance checks performed. We can observe that, in several cases, the variants that adopt rank-based traversal, perform significant fewer dominance checks than the original. Particularly, the rkT/rkE/r1R and rkT/rkE/r2R variants outperform the others in almost all cases, in independent and correlated datasets, by up to 50%. Similar results also hold for low dimensionalities in the anti-correlated dataset. However, this does not hold in more dimensions, due to the explosion of skyline objects in anti-correlated datasets.

3.5.3.2 SFS

Here, as in the previous experiment, we examine the performance of SFS algorithm adopting several policies. Similar to BNL, none of SFS variants perform noticeable fewer I/O operations (Figure 3.13). Regarding the dominance checks (Figure 3.14), in anti-correlated and independent datasets, most of variants have similar performance to the original algorithm. Only for correlated datasets, ranked-based policies exhibit significant performance gains.

3.5.4 Sorting Function Evaluation

In this experiment we study the performance of SFS and LESS algorithms considering different sorting functions. We use three sorting functions: *Entr*, *Sum* and *minC*, as

described in Section 3.3. As we can see in Figure 3.15, the sorting functions *Entr* and *Sum* have similar performance in all cases, for both SFS and LESS algorithms. Another interesting observation is that SFS, unlike LESS, has similar performance for all sorting functions.

3.5.5 Discussion

In an I/O-sensitive setting, i.e., when I/O operations cost significantly more than CPU cycles, BNL seems to be the ideal choice, as it performs less I/O operations than all other methods in almost all settings. Additionally, BNL and RAND perform less write operation than the other methods. On the other hand, in a CPU-sensitive setting, LESS and RAND seem to be good choices. LESS performs the fewest dominance checks, while RAND doesn't spend time for sorting the data, or for skyline identification. Finally, regarding the policies tested, the rank-based ones show significant gains but only in CPU-sensitive settings.

3.6 Summary

In this work we have studied an important class of external memory skyline algorithms. Particularly, we have assumed a standard external memory model and have addressed neglected details that arise in real system implementations. Moreover, we have focused on the core issue of managing in-memory objects, comprising the so-called window. Last but not least, we have performed an extensive experimental evaluation on real and synthetic data using real disk-based implementations, and have drawn insightful conclusions.

Part II

Exploratory Data Analysis

Chapter 4

Visual Exploration and Analysis over Large Datasets

Data exploration and visualization systems are of great importance in the Big Data era, in which the volume and heterogeneity of available information make it difficult for humans to manually explore and analyse data. Most traditional systems operate in an offline way, limited to accessing preprocessed (static) sets of data. They also restrict themselves to dealing with small dataset sizes, which can be easily handled with conventional techniques. However, the Big Data era has realized the availability of a great amount and variety of big datasets that are dynamic in nature; most of them offer API or query endpoints for online access, or the data is received in a stream fashion. Therefore, modern systems must address the challenge of on-the-fly scalable visualizations over large dynamic sets of data, offering efficient exploration techniques, as well as mechanisms for information abstraction and summarization. Finally, they must take into account different user-defined exploration scenarios and user preferences.

In this chapter two problems are considered. In the first one we study the problem of on-the-fly visual exploration over large sets of data. For this problem we present a generic model for personalized multilevel exploration and analysis. Our model is built on top of a lightweight tree-based structure which can be efficiently constructed on-the-fly for a given set of data. This structure aggregates input objects into a hierarchical multiscale model. Considering different exploration scenarios over large datasets, the proposed model enables efficient multilevel exploration, offering incremental construction and prefetching via user interaction, and dynamic adaptation of the hierarchies based on user preferences. A thorough theoretical analysis is presented, illustrating the efficiency of the proposed methods. The presented model is realized in a web-based prototype tool, called SynopsViz that offers multilevel visual exploration and analysis over Linked Data datasets. Finally, we provide a performance evaluation and a empirical user study employing real datasets.

The second problem considers the exploration and visualization of very large graphs. For this problem we present a novel platform which enables the user to interact with the visualized graph in a way that is very similar to the exploration of maps at multiple levels. Our approach involves an offline preprocessing phase that builds the layout of the graph by assigning coordinates to its nodes with respect to a Euclidean plane. The respective points are indexed with a spatial data structure, i.e., an R-tree, and stored in a database. Multiple abstraction layers of the graph based on various criteria are also created offline, and they are indexed similarly so that the user can explore the dataset at different levels of granularity, depending on her particular needs. Then, our system translates user operations into simple and very efficient spatial operations (i.e., window queries) in the backend. This technique allows for a fine-grained access to very large graphs with extremely low latency and memory requirements and without compromising

the functionality of the tool. Our web-based prototype supports four main operations: (1) interactive navigation, (3) multi-level exploration, (3) keyword search and (4) subgraph selection and manipulation.

4.1 Efficient Multilevel Exploration

Exploring, visualizing and analysing data is a core task for data scientists and analysts in numerous applications. Data exploration and visualization enable users to identify interesting patterns, infer correlations and causalities, and support sense-making activities over data that are not always possible with traditional data mining techniques [214, 139]. This is of great importance in the Big Data era, where the volume and heterogeneity of available information make it difficult for humans to manually explore and analyse large datasets.

One of the major challenges in visual exploration is related to the *large size* that characterizes many datasets nowadays. Considering the visual information seeking mantra: "*overview first, zoom and filter, then details on demand*" [343], gaining overview is a crucial task in the visual exploration scenario. However, offering an overview of a large dataset is an extremely challenging task. *Information overloading* is a common issue in large dataset visualization; a basic requirement for the proposed approaches is to offer mechanisms for information abstraction and summarization.

The above challenges can be overcome by adopting *hierarchical aggregation* approaches (for simplicity we also refer to them as hierarchical) [162]. Hierarchical approaches allow the visual exploration of very large datasets in a multilevel fashion, offering overview of a dataset, as well as an intuitive and usable way for finding specific parts within a dataset. Particularly, in hierarchical approaches, the user first obtains an overview of the dataset (both structure and a summary of its content) before proceeding to data exploration operations, such as roll-up and drill-down, filtering out a specific part of it and finally retrieving details about the data. Therefore, hierarchical approaches directly support the visual information seeking mantra. Also, hierarchical approaches can effectively address the problem of information overloading as it provides information abstraction and summarization.

A second challenge is related to the availability of API and query endpoints (e.g., SPARQL) for online data access, as well as the cases where that data is received in a stream fashion. The latter pose the challenge of handling large sets of data in a dynamic setting, and as a result, a preprocessing phase, such as traditional indexing, is prevented. In this respect, modern techniques must offer scalability and efficient processing for on-the-fly analysis and visualization of dynamic datasets.

Finally, the requirement for on-the-fly visualization must be coupled with the diversity of preferences and requirements posed by different users and tasks. Therefore, the proposed approaches should provide the user with the ability to customize the exploration experience, allowing users to organize data into different ways according to the type of information or the level of details she wishes to explore.

Considering the general problem of exploring big data [344, 202, 294, 395, 186], most approaches aim at providing appropriate summaries and abstractions over the enormous number of available data objects. In this respect, a large number of systems adopt *approximation techniques* (a.k.a. *data reduction* techniques) in which partial results are computed. Existing approaches are mostly based on: (1) sampling and filtering [173, 302, 238, 21, 216, 56] and/or (2) aggregation (e.g., binning, clustering) [162, 224, 223, 187, 268, 393, 55, 265, 19, 221]. Similarly, some modern database-oriented systems adopt approximation techniques using query-based approaches (e.g., query translation, query rewriting) [56, 224, 223, 385, 395]. Recently, incremental approximation techniques are adopted; in

these approaches approximate answers are computed over progressively larger samples of the data [173, 21, 216]. In a different context, an adaptive indexing approach is used in [408], where the indexes are created incrementally and adaptively throughout exploration. Further, in order to improve performance many systems exploit caching and prefetching techniques [365, 226, 220, 55, 113, 236, 151]. Finally, in other approaches, parallel architectures are adopted [160, 228, 227, 216].

Addressing the aforementioned challenges, in this work, we introduce a generic model that combines personalized multilevel exploration with online analysis of numeric and temporal data. At the core lies a lightweight hierarchical aggregation model, constructed on-the-fly for a given set of data. The proposed model is a tree-based structure that aggregates data objects into multiple levels of hierarchically related groups based on numerical or temporal values of the objects. Our model also enriches groups (i.e., aggregations/summaries) with statistical information regarding their content, offering richer overviews and insights into the detailed data. An additional feature is that it allows users to organize data exploration in different ways, by parameterizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, and so on. On top of this model, we propose three user exploration scenarios and present two methods for efficient exploration over large datasets: the first one achieves the incremental construction of the model based on user interaction, whereas the second one enables dynamic and efficient adaptation of the model to the user's preferences. The efficiency of the proposed model is illustrated through a thorough theoretical analysis, as well as an experimental evaluation. Finally, the proposed model is realized in a web-based tool, called *SynopsViz* that offers a variety of visualization techniques (e.g., charts, timelines) for multilevel visual exploration and analysis over Linked Data (LD) datasets.

Contributions. The main contributions of this work are summarized as follows.

1. We introduce a generic model for organizing, exploring, and analysing numeric and temporal data in a multilevel fashion.
2. We implement our model as a lightweight, main memory tree-based structure, which can be efficiently constructed on-the-fly.
3. We propose two tree structure versions, which adopt different approaches for the data organization.
4. We describe a simple method to estimate the tree construction parameters, when no user preferences are available.
5. We define different exploration scenarios assuming various user exploration preferences.
6. We introduce a method that incrementally constructs the hierarchy tree via user interaction.
7. We propose an efficient method that dynamically adapts an existing hierarchy to a new, considering user's preferences.
8. We present a thorough theoretical analysis, illustrating the efficiency of the proposed model.
9. We develop a prototype system which implements the presented model, offering multilevel visual exploration and analysis over LD.
10. We conduct a thorough performance evaluation and an empirical user study, using the DBpedia 2014 dataset.

4.1.1 The HETree Model

In this section we present HETree (**H**ierarchical **E**xploration **Tree**), a generic model for organizing, exploring, and analysing numeric and temporal data in a multilevel fashion. Particularly, HETree is defined in the context of multilevel (visual) exploration and analysis. The proposed model hierarchically organize arbitrary numeric and temporal data, without requiring it to be described by an hierarchical scheme. We should note that, our model is not bound to any specific type of visualization; rather it can be adopted by several "flat" visualization techniques (e.g., charts, timeline), offering scalable and multilevel exploration over non-hierarchical data.

In what follows, we present some basic aspects of our working scenario (i.e., visual exploration and analysis scenario) and highlight the main assumptions and requirements employed in the construction of our model. First, the input data in our scenario can be retrieved directly from a database, but also produced dynamically; i.e., either from a query or from data filtering (e.g., faceted browsing). Thus, we consider that data visualization is performed online; i.e., we do not assume an offline preprocessing phase in the construction of the visualization model. Second, users can specify different requirements or preferences with respect to the data organization. For example, a user prefers to organize the data as a deep hierarchy for a specific task, while for another task a flat hierarchical organization is more appropriate. Therefore, even if the data is not dynamically produced, the data organization is dynamically adapted to the user preferences. The same also holds for any additional information (e.g., statistical information) that is computed for each group of objects. This information must be recomputed when the groups of objects (i.e., data organization) are modified.

From the above, a basic requirement is that the model must be constructed on-the-fly for any given data and users preferences. Therefore, we implement our model as a lightweight, main memory tree structure, which can be efficiently constructed on-the-fly. We define two versions of this tree structure, following data organization approaches well-suited to visual exploration and analysis context: the first version considers fixed-range groups of data objects, whereas the second considers fixed-size groups. Finally, our structure allows efficient on-the-fly statistical computations, which are extremely valuable for the exploration and analysis scenario.

The basic idea of our model is to hierarchically group data objects based on values of one of their properties. Input data objects are stored at the leaves, while internal nodes aggregate their child nodes. The root of the tree represents (i.e., aggregates) the whole dataset. The basic concepts of our model can be considered similar to a simplified version of a static 1D R-Tree [191].

Regarding the visual representation of the model and data exploration, we consider that both data objects sets (leaf nodes contents) and entities representing groups of objects (leaf or internal nodes) are visually represented enabling the user to explore the data in a hierarchical manner. Note that our tree structure organizes data in a hierarchical model, without setting any constraints on the way the user interacts with these hierarchies. As such, it is possible that different strategies can be adopted, regarding the traversal policy, as well as the nodes of the tree that are rendered in each visualization stage.

In the rest of this section, preliminaries are presented in Section 4.1.1.1. In Section 4.1.1.2, we introduce the proposed tree structure. Sections 4.1.1.3 and 4.1.1.4 present the two versions of the structure. Finally, Section 4.1.1.5 discusses the specification of the parameters required for the tree construction, and Section 4.1.1.6 presents how statistics computations can be performed over the tree.

4.1.1.1 Preliminaries

In this work we formalize data objects as RDF triples. However, the presented methods are generic and can be applied to any data objects with numeric or temporal attributes. Hence, in the following, the terms triple and (data) object will be used interchangeably.

We consider an *RDF dataset* R consisting of a set of *RDF triples*. As *input data*, we assume a set of RDF triples D, where $D \subseteq R$ and triples in D have as objects either numeric (e.g., integer, decimal) or temporal values (e.g., date, time). Let tr be an RDF triple, $tr.s$, $tr.p$ and $tr.o$ represent, respectively, the *subject*, *predicate* and *object* of the RDF triple tr.

Given input data D, S is an *ordered set* of RDF triples, produced from D, where triples are sorted based on objects' values, in ascending order. Assume that $S[i]$ denotes the i-th triple, with $S[1]$ the first triple. Then, for each $i < j$, we have that $S[i].o \leq S[j].o$. Also, $D = S$, i.e., for each $tr, tr \in D$ iff $tr \in S$.

Figure 4.1 presents a set of 10 RDF triples, representing persons and their ages. In Figure 4.1, we assume that the subjects $p0$-$p9$ are instances of a class *Person* and the predicate *age* is a datatype property with integer range.

$p0$ *age* 35	$p5$ *age* 35
$p1$ *age* 100	$p6$ *age* 45
$p2$ *age* 55	$p7$ *age* 80
$p3$ *age* 37	$p8$ *age* 20
$p4$ *age* 30	$p9$ *age* 50

Figure 4.1: Running example input data (data objects)

Example 1. In Figure 4.1, given the RDF triple $tr = p0$ *age* 35, we have that $tr.s = p0$, $tr.p =$ *age* and $tr.o = 35$. Also, given that all triples comprise the input data D and S is the ordered set of D based on the object values, in ascending order; we have that $S = \{p8$ *age* 20, $p4$ *age* 30, $p0$ *age* 35, $p5$ *age* 35, $p3$ *age* 37, $p6$ *age* 45, $p9$ *age* 50, $p2$ *age* 55, $p7$ *age* 80, $p1$ *age* 100$\}$. Hence, $S[1] = p8$ *age* 20 and $S[10] = p1$ *age* 100. □

Assume an *interval* $I = [a, b]$, where $a, b \in \mathbb{R}$; then, $I = \{k \in \mathbb{R} \mid a \leq k \leq b\}$. Similarly, for $I = [a, b)$, we have that $I = \{k \in \mathbb{R} \mid a \leq k < b\}$. Let I^- and I^+ denote the lower and upper bound of the interval I, respectively. That is, given $I = [a, b]$, then $I^- = a$ and $I^+ = b$. The *length* of an interval I is defined as $|I^+ - I^-|$.

In this work we assume rooted trees. The number of the children of a node is its *degree*. Nodes with degree 0 are called *leaf nodes*. Moreover, any non-leaf node is called *internal node*. *Sibling nodes* are the nodes that have the same parent. The *level of a node* is defined by letting the root node be at level zero. Additionally, the *height of a node* is the length of the longest path from the node to a leaf. A leaf node has a height of 0.

The *height of a tree* is the maximum level of any node in the tree. The *degree of a tree* is the maximum degree of a node in the tree. An *ordered tree* is a tree where the children of each node are ordered. A tree is called an *m-ary tree* if every internal node has no more than m children. A *full m-ary tree* is a tree where every internal node has exactly m children. A *perfect m-ary tree* is a full m-ary tree in which all leaves are at the same level.

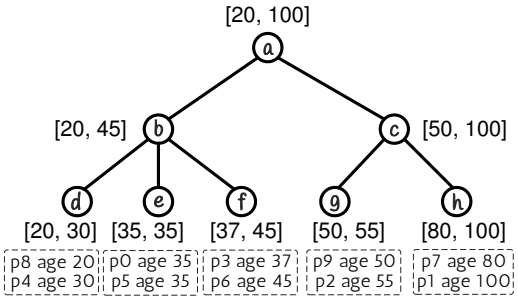


Figure 4.2: A Content-based HETree (HETree-C)

4.1.1.2 The HETree Structure

In this section, we present in more detail the *HETree* structure. HETree hierarchically organizes numeric and temporal[1] data into groups; intervals are used to represents these groups. HETree is defined by the tree degree and the number of leaf nodes[2]. Essentially, the number of leaf nodes corresponds to the number of groups where input data objects are organized. The tree degree corresponds to the (maximum) number of groups where a group is split in the lower level.

Given a set of data objects (RDF triples) D, a positive integer ℓ denoting the number of leaf nodes; and a positive integer d denoting the tree degree; an *HETree* (D, ℓ, d) is an *ordered d-ary tree*, with the following basic properties.

- The tree has exactly ℓ number of leaf nodes.
- All leaf nodes appear in the same level.
- Each leaf node contains a set of data objects, sorted in ascending order based on their values. Given a leaf node n, $n.data$ denote the data objects contained in n.
- Each internal node has at most d children nodes. Let n be an internal node, $n.c_i$ denotes the i-th child for the node n, with $n.c_1$ be the leftmost child.
- Each node corresponds to an interval. Given a node n, $n.I$ denotes the interval for the node n.
- At each level, all nodes are sorted based on the lower bounds of their intervals. That is, let n be an internal node, for any $i < j$, we have that $n.c_i.I^- \leq n.c_j.I^-$.
- For a leaf node, its interval is bounded by the values of the objects included in this leaf node. Let n be the leftmost leaf node; assume that n contains x objects from D. Then, we have that $n.I^- = S[1].o$ and $n.I^+ = S[x].o$, where S is the ordered object set resulted from D.
- For an internal node, its interval is bounded by the union of the intervals of its children. That is, let n be an internal node, having k child nodes; then, we have $n.I^- = n.c_1.I^-$ and $n.I^+ = n.c_k.I^+$.

Example 2. Given the set of RDF triples D from Figure 4.1. Figure 4.2 presents a HETree with five leaf nodes (i.e., $\ell = 5$) and degree equal to three (i.e., $d = 3$). Considering the leftmost leaf node d, we can see that it contains two triples in the following order $p8$ *age* 20, $p4$ *age* 30. As a result, the lower bound for the its interval, is equal to the

[1]Note that our structure handles numeric and temporal data in a similar manner. Also, other types of one-dimensional data may be supported, with the requirement that a total order can be defined over the data.

[2]Note that following a similar approach, the HETree can also be defined by specifying the tree height instead of degree or number of leaves.

value of the first triple object, i.e., $d.I^- = 20$; the upper bound is equal to the value of the last triple object, i.e., $d.I^+ = 30$. For the internal node b, its interval is bounded by the intervals of its children nodes (d, e, f); i.e., considering the lower bound of its leftmost child (i.e., 20) and the upper bound of its rightmost child (i.e., 45). □

Furthermore, we present two different approaches for organizing the data in the HETree. Assume the scenario in which a user wishes to (visually) explore and analyse the historic events from DBpedia [45], per decade. In this case, user orders historic events by their date and organizes them into groups of equal ranges (i.e., decade). In a second scenario, assume that a user wishes to analyse in the Eurostat dataset the gross domestic product (GDP) organized into fixed groups of countries. In this case, the user is interested in finding information like: the range and the variance of the GDP values over the top-10 countries with the highest GDP factor. In this scenario, the user orders countries by their GDP and organizes them into groups of equal sizes (i.e., 10 countries per group).

In the first approach, we organize data objects into groups, where the object values of each group covers equal range of values. In the second approach, we organize objects into groups, where each group contains the same number of objects. In the following sections, we present in detail the two approaches for organizing the data in the HETree.

4.1.1.3 A Content-based HETree (HETree-C)

In this section we introduce a version of the HETree, named HETree-C (*Content-based* HETree). This HETree version organizes data into equally sized groups. The basic property of the HETree-C is that each leaf node contains approximately the same number of objects and the content (i.e., objects) of a leaf node specifies its interval. For the tree construction, the objects are first assigned to the leaves and then the intervals are defined.

An *HETree-C* (D, ℓ, d) is an HETree, with the following extra property. Each leaf node contains λ or $\lambda - 1$ objects, where[3] $\lambda = \left\lceil \frac{|D|}{\ell} \right\rceil$. Particularly, the $\ell - (\lambda \cdot \ell - |D|)$ leftmost leaves contain λ objects, while the rest leaves[4] contain $\lambda - 1$. We can equivalently define the HETree-C by providing the number of objects per leaf λ, instead of the number of leaves ℓ.

Example 3. Figure 4.2 presents an HETree-C constructed by considering the set of objects D from Figure 4.1, $\ell = 5$ and $d = 3$. As we can observe, all the leaf nodes contain equal number of objects. Particularly, we have that $\lambda = \left\lceil \frac{10}{5} \right\rceil = 2$. Also, we have that the $5 - (2 \cdot 5 - 10) = 5$ leftmost leaves (i.e., all leaves in this case), will contain λ triples. As we can verify from Figure 4.2, all leaves (d, e, f, g, h) contain 2 triples. Note also that all nodes in HETree-C define closed intervals. □

4.1.1.3.1 The HETree-C Construction

We construct the HETree-C in a bottom-up way. Algorithm 6 describes the HETree-C construction. The algorithm takes as input: (1) a set of data objects D; (2) the number of leaf nodes ℓ; and (3) the tree degree d. Initially, the algorithm sort the object set D in ascending order, based on objects values (*line 1*). Then, the algorithm uses two procedures to construct the tree nodes. The first procedure, named constrLeaves-C creates the leaf nodes of the tree (*line 2*). The second procedure, named constrtInterNodes, creates the internal nodes of the tree (*line 3*). Finally, the root node of the constructed tree is returned (*line 4*).

[3] We assume that, the number of objects is at least as the number of leaves; i.e., $|D| \geq \ell$.

[4] As an alternative we can construct the HETree-C, so each leaf contains λ objects, except the rightmost leaf which will contain between 1 and λ objects.

Algorithm 6. createHETree-C/R (D, ℓ, d)

Input: D: set of objects; ℓ: number of leaf nodes; d: tree degree
Output: r: root node of the HETree tree

```
1  S ← sort D based on objects values
2  L ← constrLeaves-C/R(S, ℓ)
3  r ← constrtInterlNodes(L, d)
4  return r
```

Procedure 1: constrLeaves-C(S, ℓ)

Input: S: ordered set of objects; ℓ: number of leaf nodes
Output: L: ordered set of leaf nodes

```
1  λ ← ⌈|S|/ℓ⌉
2  k ← ℓ − (λ · ℓ − |S|)
3  beg ← 1
4  for i ← 1 to ℓ do
5      create an empty leaf node n
6      if i ≤ k then
7          num ← λ
8      else
9          num ← λ − 1
10     end ← beg + num
11     for t ← beg to end do
12         n.data ← S[t]
13     n.I⁻ ← S[beg].o
14     n.I⁺ ← S[end].o
15     L[i] ← n
16     beg ← end + 1
17 return L
```

Procedure 1 presents the pseudocode for the constrLeaves-C procedure. First, the procedure uses an ordered set of data objects S and creates ℓ leaf nodes containing the objects S. Then, the procedure computes the number of objects per leaf λ (*line 1*), as well as the number of leaves that contain λ triples (*line 2*). Then, ℓ leaf nodes are constructed (*lines 4–16*). For the first k leaves, λ objects are inserted, while for the rest leaves, $\lambda - 1$ objects are inserted (*lines 6–9*). The interval of each leaf is specified by the objects values of the first and last triple inserted in this leaf (*lines 13–14*). Finally, the set of created leaf nodes is returned (*line 17*).

The constrtInterNodes procedure (Procedure 2) builds the internal nodes in a recursive manner. Particularly, the procedure takes as input a set of nodes H, as well as the tree degree d. The basic idea of this procedure is the following. For the nodes H, their parents nodes P are created (*lines 4-16*); then, the procedure calls itself using as input the parent nodes P (*line 21*). The recursion terminates when the number of created parent nodes is equal to one (*line 17*); i.e., the root of the tree is created.

Computational Analysis. The computational cost for the HETree-C construction (Algorithm 6) is the sum of three parts. The first is sorting the input data, which can be done in the worst case in $O(|D|log|D|)$, employing a linearithmic sorting algorithm (e.g., merge-sort). The second part is the constrLeaves-C procedure, which requires $O(|D|)$ for scanning all data objects. The third part is the constrtInterNodes procedure, which requires $d \cdot (\lceil \frac{\ell}{d} \rceil + \lceil \frac{\ell}{d^2} \rceil + \lceil \frac{\ell}{d^3} \rceil + \ldots + 1)$, with the sum being the number of internal nodes in the tree. Note that the maximum number of internal nodes in a d-ary tree corresponds to the number of internal nodes in a perfect d-ary tree of the same height. Also, note the number of internal nodes of a perfect d-ary tree of height h is $\frac{d^h - 1}{d-1}$. In our case,

Procedure 2: constrtInterNodes(H, d)

Input: H: ordered set of nodes; d: tree degree
Output: r: root node for H
Variables: P: ordered set of H's parent nodes

```
1  p_num ← ⌈|H|/d⌉                                  //number of parents nodes
2  t ← d − (p_n · d − |H|)                          //last parent's number of children
3  c_beg ← 1                                        //first child node
4  for p ← 1 to p_num do
5      create an empty internal node n
6      if p = p_num then
7          c_num ← t                                //number of children
8      else
9          c_num ← d
10     c_end ← c_beg + c_num                        //last child node
11     for j ← c_beg to c_end do
12         n.c[j] ← H[j]
13     n.I⁻ ← H[c_beg].I⁻
14     n.I⁺ ← H[c_end].I⁺
15     P[p] ← n
16     c_beg ← c_end + 1
17 if p_num = 1 then
18     r ← P
19     return r
20 else
21     return constrtInterlNodes(P, d)
```

the height of our tree is $h = \lceil log_d \ell \rceil$. Hence, the maximum number of internal nodes is $\frac{d^{\lceil log_d \ell \rceil}-1}{d-1} \leq \frac{d \cdot \ell - 1}{d-1}$. Therefore, the constrtInterNodes procedure, in worst case requires $O(\frac{d^2 \cdot \ell - d}{d-1})$. Therefore, the overall computational cost for the HETree-C construction in the worst case is $O(|D|log|D| + |D| + \frac{d^2 \cdot \ell - d}{d-1}) = O(|D|log|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

4.1.1.4 A Range-based HETree (HETree-R)

The second version of the HETree is called HETree-R (*Range-based* HETree). HETree-R organizes data into equally ranged groups. The basic property of the HETree-R is that each leaf node covers an equal range of values. Therefore, in HETree-R, the data space defined by the objects values is equally divided over the leaves. As opposed to HETree-C, in HETree-R the interval of a leaf specifies its content. Therefore, for the HETree-R construction, the intervals of all leaves are first defined and then objects are inserted.

An *HETree-R* (D, ℓ, d) is an HETree, with the following extra property. The interval of each leaf node has the same length; i.e., covers equal range of values. Formally, let S be the sorted RDF set resulted from D, for each leaf node its interval has length ρ, where[5] $\rho = \frac{|S[1].o - S[|S|].o|}{\ell}$. Therefore, for a leaf node n, we have that $|n.I^- - n.I^+| = \rho$. For example, for the leftmost leaf, its interval is $[S[1].o, S[1].o + \rho)$. The HETree-R is equivalently defined by providing the interval length ρ, instead of the number of leaves ℓ.

Example 4. Figure 4.3 presents an HETree-R tree constructed by considering the set of objects D (Figrue 4.1), $\ell = 5$ and $d = 3$. As we can observe from Figure 4.3, each leaf node covers equal range of values. Particularly, we have that the interval of each leaf must have length $\rho = \frac{|20-100|}{5} = 16$. Hence, the leftmost leaf d has the interval $[20, 20+16)$. Based on the specified intervals, the d leaf node contains four triples, the e leaf three triples, while the rest leaves contain one triple. Note that, all nodes in HETree-R define

[5]We assume here that, there is at least one object in D with different value than the rest objects.

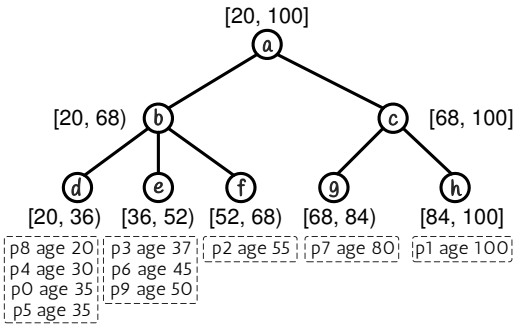


Figure 4.3: A Range-based HETree (HETree-R)

closed-open intervals, except the rightmost node in each level, which defines a closed interval. □

4.1.1.4.1 The HETree-R Construction

This section studies the construction of the HETree-R structure. The HETree-R is also constructed in a bottom-up fashion.

Similarly with the HETree-C version, Algorithm 6 is used for the HETree-R construction. The only difference is the constrLeaves-R procedure (*line 2*), which creates the leaf nodes of the HETree-R and is presented in Procedure 3.

The procedure takes as input an ordered set of data objects S, as well as the number of leaf nodes ℓ. First, it computes the range ρ of the leaves (*line 1*). The procedure constructs ℓ leaf nodes (*lines 2–9*) and assigns same intervals to all of them (*lines 4–8*), it traverses all objects in S (*lines 10–12*) and places them to the appropriate leaf node (*line 12*). Finally, it removes empty leaf nodes (*lines 13–15*) and returns the set of created leaves (*line 13*).

Procedure 3: constrLeaves-R(S, ℓ)

Input: S: ordered set of objects; ℓ: number of leaf nodes
Output: L: ordered set of leaf nodes

```
1  ρ ← (|S[1].o − S[|S|].o|) / ℓ
2  for i ← 1 to ℓ do
3      create an empty leaf node n
4      if i = 1 then
5          n.I⁻ ← S[1].o
6      else
7          n.I⁻ ← L[i − 1].I⁺
8      n.I⁺ ← n.I⁻ + ρ
9      L[i] ← n
10 for t ← 1 to |S| do
11     j ← ⌊(S[t].o − S[1].o) / ρ⌋ + 1
12     L[j].data ← S[t]
13 return L
```

Computational Analysis. The computational cost for the HETree-R construction (Algorithm 6) for sorting the input data (*line 1*) and creating the internal nodes (*line 3*) is the same as in the HETree-C case. The constrLeaves-R procedure (*line 2*) requires $O(\ell + |D|) = O(|D|)$ (since $|D| \geq \ell$). Using the computational costs for the first and the third part from Section 4.1.1.3.1, we have that in worst case, the overall computational cost for the HETree-R construction is $O(|D|log|D| + |D| + \frac{d^2 \cdot \ell - d}{d-1}) = O(|D|log|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

4.1.1.5 Estimating the HETree Parameters

In our working scenario, the user specifies the parameters required for the HETree construction (e.g., number of leaves ℓ). In this section, we describe our approach for automatically calculating the HETree parameters based on the input data, when no user preferences are provided. Our goal is to derive the parameters by the input data, such that the resulting HETree can address some basic guidelines set by the visualization environment. In what follows, we discuss in detail the proposed approach.

An important parameter in hierarchical visualizations is the minimum and maximum number of objects that can be effectively rendered in the most detailed level[6]. In our case, the above numbers correspond to the number of objects contained in the leaf nodes. The proper calculation of these numbers is crucial such that the resulting tree avoids overloaded and scattered visualizations.

Therefore, in HETree construction, our approach considers the minimum and the maximum number of objects per leaf node, denoted as λ_{min} and λ_{max}, respectively. Besides the number of objects rendered in the lowest level, our approach consider perfect m-ary trees, such that a more "uniform" structure (i.e., all the groups are divided into same number of groups) is resulted. The following example illustrates our approach to calculate the HETree parameters.

Table 4.1: Number of leaf nodes for perfect m-ary trees

	Degree				
Height	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6
2	4	9	16	25	36
3	8	27	64	625	216
4	16	81	256	3125	1296
5	32	243	1024	15625	7776
6	64	729	4048	78125	46656

Example 5. Assume that based on an adopted visualization technique, the ideal number of data objects to be rendered on a specific screen is between 25 and 50. Hence, we have that $\lambda_{min} = 25$ and $\lambda_{max} = 50$.

Now, let's assume that we want to visualize the object set D_1, using an HETree-C, where $|D_1| = 500$. Based on the number of objects and the λ bounds, we can estimate the bounds for the number of leaves. Let ℓ_{min} and ℓ_{max} denote the lower and the upper bound for the number of leaves. Therefore, we have that $\left\lceil \frac{|D_1|}{\lambda_{max}} \right\rceil \leq \ell \leq \left\lceil \frac{|D_1|}{\lambda_{min}} \right\rceil \Leftrightarrow \left\lceil \frac{500}{50} \right\rceil \leq \ell \leq \left\lceil \frac{500}{25} \right\rceil \Leftrightarrow 10 \leq \ell \leq 20$.

Hence, our HETree-C should have between $\ell_{min} = 10$ and $\ell_{max} = 20$ leaf nodes. Since, we consider perfect m-ary trees, from Table 4.1 we can identify the tree characteristics that conform to the number of leaves guideline. The candidate settings (i.e., leaf number and degree) are indicated in Table 4.1, using dark-grey colour. The setting with $d = 2$ is rejected since visualizing two groups of objects in each level, can be considered a small number under most visualization settings. Note that, in our work, in any case we assume settings with $d \geq 3$ and $height \geq 2$. Therefore, an HETree-C with $\ell = 16$ and $d = 4$ is a suitable structure for our case.

Now, let's assume that we want to visualize the object set D_2, where $|D_2| = 1000$. Following a similar approach, we have that $20 \leq \ell \leq 40$. The candidate settings are indicated in Table 4.1 using light-grey colour. Also, here the setting with $d = 2$, is

[6]Similar bounds can also be defined for other tree levels.

rejected. Hence, we have the following settings that satisfy the considered guideline: $S1$: $\ell = 27$, $d = 3$; $S2$: $\ell = 25$, $d = 5$; and $S3$: $\ell = 36$, $d = 6$.

In the case, where more than one setting satisfies the considered guideline, we select the preferable one according to following set of rules. From the candidate settings, we prefer the setting which results in the highest tree[7] (1st *Criterion*). In case that the highest tree is constructed by more than one settings, we consider the distance c, between ℓ and the centre of ℓ_{min} and ℓ_{max} (2nd *Criterion*); i.e., $c = |\ell - \frac{\ell_{min}+\ell_{max}}{2}|$. The setting with the lowest c value is selected. Note that, based on the visualization context, different criteria and preferences may be followed.

In our example, from the candidate settings, the setting $S1$ is selected, since it will construct the highest tree (i.e., $height = 3$). On the other hand, the settings $S2$ and $S3$ will construct trees with lower heights (i.e., $height = 2$).

Now, assume a scenario where only the settings $S2$ and $S3$ are candidates. In this case, since both settings result to trees with equal heights, the 2nd *Criterion* is considered. Hence, for the $S2$ setting we have $c_2 = |25 - \frac{20+40}{2}| = 5$. Similarly, for the $S3$ setting $c_3 = |36 - \frac{20+40}{2}| = 6$. Therefore, between the settings $S2$ and $S3$, the setting $S2$ is preferable, since $c_2 < c_3$.

In case of HETree-R, a similar approach is followed, assuming normal distribution over the values of the objects. □

4.1.1.6 Statistics Computations over HETree

Data statistics is a crucial aspect in the context of hierarchical visual exploration and analysis. Statistical informations over groups of objects offer rich insights into the underlying data. In this way, useful information regarding different set of objects with common characteristics is provided. Additionally, this information may also guide the users through their navigation over the hierarchy.

In this section, we present how statistics computation is performed over the nodes of the HETree. Statistics computations exploit two main aspects of the HETree structure: (1) the internal nodes aggregate their child nodes; and (2) the tree is constructed in bottom-up fashion. Statistics computation is performed during the tree construction; for the leaf nodes, we gather statistics from the objects they contain, whereas for the internal nodes we aggregate the statistics of their children.

For simplicity, here, we assume that each node contains the following extra fields, used for simple statistics computations, although more complex or RDF-related (e.g., most common subject, subject with the minimum value, etc.) statistics can be computed. Assume a node n, as $n.N$ we denote the *number* of objects covered by n; as $n.\mu$ and $n.\sigma^2$ we denote the mean and the variance of the objects' values covered by n, respectively. Additionally, we assume the minimum and the maximum values, denoted as $n.min$ and $n.max$, respectively.

Statistics computations can be easily performed in the construction algorithms (Algorithm 6) without any modifications. The follow example illustrates these computations.

Example 6. In this example we assume the HETree-C presented in Figure 4.2. Figure 4.4 shows the HETree-C with the computed statistics in each node. When all the leaf nodes have been constructed, the statistics for each leaf is computed. For instance, we can see from Figure 4.4, that for the rightmost leaf h we have: $h.N = 2$, $h.\mu = \frac{80+100}{2} = 90$ and $h.\sigma^2 = \frac{1}{2} \cdot ((80-90)^2 + (100-90)^2) = 100$. Also, we have $h.min = 80$ and $h.max = 100$. Following the above process, we compute the statistics for all leaf nodes.

[7]Depending on the scenario, the shortest tree may be preferable.

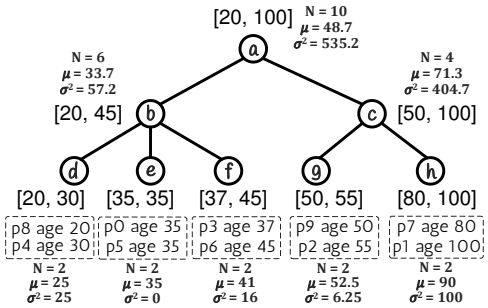


Figure 4.4: Statistics computation over HETree

Then, for each parent node we construct, we compute its statistics using the computed statistics of its child nodes. Considering the c internal node, with the child nodes g and h, we have that $c.min = 50$ and $c.max = 100$. Also, we have that $c.N = g.N + h.N = 2 + 2 = 4$. Now we will compute the mean value by combining the children mean values: $c.\mu = \frac{g.N \cdot g.\mu + h.N \cdot h.\mu}{g.N + h.N} = \frac{2 \cdot 52.5 + 2 \cdot 90}{2+2} = 71.3$. Similarly, for variance we have $c.\sigma^2 = \frac{g.N \cdot g.\sigma^2 + h.N \cdot h.\sigma^2 + g.N \cdot (g.\mu - c.\mu)^2 + h.N \cdot (h.\mu - c.\mu)^2}{g.N + h.N} = \frac{2 \cdot 6.25 + 2 \cdot 100 + 2 \cdot (52.5 - 71.3)^2 + 2 \cdot (90 - 71.3)^2}{2+2} = 404.7$.

The similar approach is also followed for the case of HETree-R. □

Computational Analysis. Most of the well known statistics (e.g., mean, variance, skewness, etc.) can be computed linearly w.r.t. the number of elements. Therefore, the computation cost over a set of numeric values S is considered as $O(|S|)$. Assume a leaf node n containing k objects, then the cost for statistics computations for n is $O(k)$. Also, the cost for all leaf nodes is $O(|D|)$. Let an internal node n, then the cost for n is $O(d)$; since the statistics in n are computed by aggregating the statistics of the d child nodes. Considering that $\frac{d \cdot \ell - 1}{d-1}$ is the maximum number of internal nodes (Section 4.1.1.3.1), we have that in the worst case the cost for the internal nodes is $O(\frac{d^2 \cdot \ell - d}{d-1})$. Therefore, the overall cost for statistics computations over an HETree is $O(|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

4.1.2 Efficient Multilevel Exploration

In this section, we exploit the HETree structure in order to efficiently handle different multilevel exploration scenarios. Essentially, we propose two methods for efficient hierarchical exploration over large datasets. The first method incrementally constructs the hierarchy via user interaction; the second one achieves dynamic adaptation of the data organization based on user's preferences.

4.1.2.1 Exploration Scenarios

In a typical multilevel exploration scenario, referred here as *Basic exploration scenario* (BSC), the user explores a dataset in a top-down fashion. The user first obtains an overview of the data through the root level, and then drills down to more fine-grained contents for accessing the actual data objects at the leaves. In BSC, the root of the hierarchy is the starting point of the exploration and, thus, the first element to be presented (i.e., rendered).

The described scenario offers basic exploration capabilities; however it does not assume use cases with user-specified starting points, other than the root, such as starting the exploration from a specific resource, or from a specific range of values.

Consider the following example, in which the user wishes to explore the *DBpedia* infoboxes dataset to find places with very large population. Initially, she selects the *populationTotal* property and starts her exploration from the root node, moves down the

right part of the tree and ends up at the rightmost leaf that contains the highly populated places. Then, she is interested in viewing the area size (i.e., *areaTotal* property) for one of the highly populated places and, also, in exploring places with similar area size. Finally, she decides to explore places based on the water area size (i.e., *areaWater*) they contain. In this case, she prefers to start her exploration by considering places that their water area size is within a given range of values.

In this example, besides BSC one we consider two additional exploration scenarios. In the *Resource-based exploration scenario* (RES), the user specifies a resource of interest (e.g., an IRI) and a specific property; the exploration starts from the leaf containing the specific resource and proceeds in a bottom-up fashion. Thus, in RES the data objects contained in the same leaf with the resource of interest are presented first. We refer to that leaf as *leaf of interest*.

The third scenario, named *Range-based exploration scenario* (RAN) enables the user to start her exploration from an arbitrary point in the hierarchy providing a range of values; the user starts from a set of internal nodes and she can then move up or down the hierarchy. The RAN scenario begins by rendering all sibling nodes that are children of the node covering the specified range of interest; we refer to these nodes as *nodes of interest*.

Note that, regarding the adopted rendering policy for all scenarios, we only consider nodes belonging to the same level. That is, sibling nodes or data objects contained in the same leaf, are rendered.

Regarding the "navigation-related" *operations*, the user can move down or up the hierarchy by performing a *drill-down* or a *roll-up* operation, respectively. A drill-down operation over a node n enables the user to focus on n and render its child nodes. If n is a leaf node, the set of data objects contained in n are rendered. On the other hand, the user can perform a *roll-up* operation on a set of sibling nodes S. The parent node of S along with the parent's sibling nodes are rendered. Finally, the roll-up operation when applied to a set of data objects O will render the leaf node that contains O along its sibling leaves, whereas a drill-down operation is not applied to a data object.

4.1.2.2 Incremental HETree Construction

In the Web of Data, the dataset might be dynamically retrieved by a remote site (e.g., via an SPARQL endpoint), as a result, in all exploration scenarios, we have assumed that the HETree is constructed on-the-fly at the time the user starts her exploration. In the previous DBpedia example, the user explores three different properties; although only a small part of their hierarchy is accessed, the whole hierarchies are constructed and the statistics of all nodes are computed. Considering the recommended HETree parameters for the employed properties, this scenario requires that 29.5K nodes will be constructed for *populationTotal* property, 9.8K nodes for the *areaTotal* and 3.3K nodes for the *areaWater*, amounting to a total number of 42.6K nodes. However, the construction of the hierarchies for large datasets poses a time overhead (as shown in the experimental section) and, consequently, increased response time in user exploration.

In this section, we introduce ICO (**I**ncremental HETree **Co**nstruction) method, which incrementally constructs the HETree, based on user interaction. The proposed method goes beyond the incremental tree construction, aiming at further reducing the response time during the exploration process by "pre-constructing" (i.e., prefetching) the parts of the tree that will be visited by the user in her next roll-up or drill-down operation. Hence, a node n is not constructed when the user visits it for the first time; instead, it has been constructed in a previous exploration step, where the user was on a node in which n can be reached by a roll-up or a drill-down operation. This way, our method offers incremental construction of the tree, tailored to each user's exploration. Finally, we show that, during an exploration scenario, ICO constructs the minimum number of HETree elements.

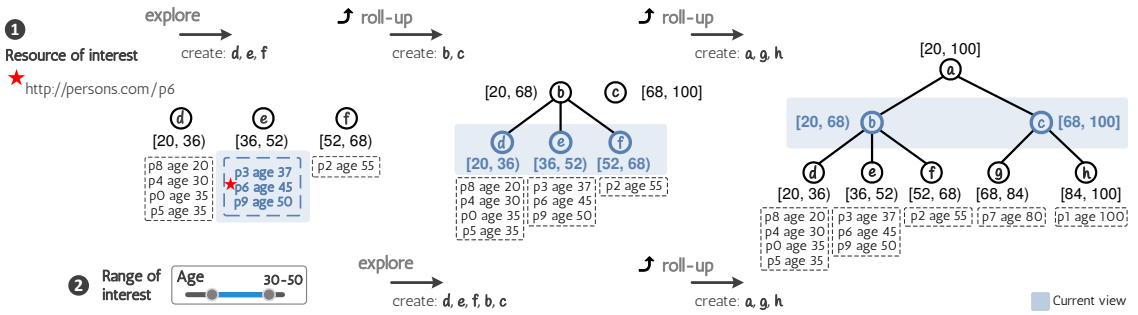


Figure 4.5: Incremental HETree construction example. ❶ Resource-based (RES) exploration scenario; ❷ Range-based (RAN) exploration scenario

Employing ICO method in the DBpedia example, the *populationTotal* hierarchy will only construct 76 nodes (the root along its child nodes and 9 nodes in each of the lower tree levels) and the *areaTotal* will construct 3 nodes corresponding to the leaf node containing the requested resource and its siblings. Finally, the *areaWater* hierarchy initially will contain either 6 or 15 nodes, depending on whether the user's input range corresponds to a set of sibling leaf nodes, or to a set of sibling internal nodes, respectively.

Example 7. We demonstrate the functionality of ICO through the following example. Assume the dataset used in our running examples, describing persons and their ages. Figure 4.5 presents the incremental construction of the HETree presented in Figure 4.3 for the RES and RAN exploration scenarios. Blue color is used to indicate the HETree elements that are presented (rendered) to the user, in each exploration stage.

In the RES scenario (upper flow in Figure 4.5), the user specifies "http://persons.com/p6" as her resource of interest; all data objects contained in the same leaf (i.e., e) with the resource of interest are initially presented to the user. The ICO initially constructs the leaf e, along with its siblings, i.e., leaves d and f. These leaves correspond to the nodes that the user can reach in a next (roll-up) step. Next, the user rolls up and the leaves d, e and f are presented to her. At the same time, parent node b and its sibling c are constructed. Note that all elements which are accessible to the user by moving either down (i.e., d, e, f data objects), or up (i.e., b, c nodes) are already constructed. Finally, when the user rolls up b and c nodes are rendered and parent node a, along with the children of c, i.e., g and h, are constructed.

In the RAN scenario (lower flow in Figure 4.5), the user specifies [20, 50] as her range of interest. The nodes covering this range (i.e., d, e) are initially presented along with their sibling f. Also, ICO constructs the parent node b and its sibling c because they are accessible by one exploration step. Then, the user performs a roll-up and ICO constructs the a, g, h nodes (as described in the RES scenario above). □

In the beginning of each exploration scenario, ICO constructs a set of *initial nodes*, which are the nodes initially presented, as well as the nodes potentially reached by the user's first operation (i.e., required HETree elements). The *required HETree elements* of an exploration step are nodes that can be reached by the user by performing one exploration operation. Hence, in the RES scenario, the initial nodes are the leaf of interest and its sibling leaves. In the RAN, the initial nodes are the nodes of interest, their children, and their parent node along with its siblings. Finally, in the BSC scenario the initial nodes are the root node and its children.

In what follows we describe the *construction rules* adopted by ICO through the user exploration process. These rules provide the correspondences between the types of elements presented in each exploration step and the elements that ICO constructs. Note that these rules are applied after the construction of the initial nodes, in all three exploration

scenarios. The correctness of these rules is verified later in Proposition 5.

Rule 1: *If a set of internal sibling nodes C is presented*, ICO constructs: (i) the parent node of C along with the parent's siblings, and (ii) the children of each node in C.

Rule 2: *If a set of leaf sibling nodes L is presented*, ICO does not construct anything (the required nodes have been previously constructed).

Rule 3: *If a set of data objects O is presented*, ICO does not construct anything (the required nodes have been previously constructed).

Remark 1. Each time ICO constructs a node (either as part of initial nodes or due to a construction rule), it also constructs all of its sibling nodes.

The following proposition shows that, in all case, the required HETree elements have been constructed earlier by ICO.

Proposition 5. In any exploration scenario, the HETree elements, a user can reach by performing one operation (i.e., required elements), have been previously constructed by ICO.

Proof. Considering the different cases of currently presented HETree elements and the available exploration operations, we have the following.

(1) *A set of (internal or leaf) sibling nodes S are presented and the user performs a roll-up action.* Here, the roll-up action will render the parent node of S along with parent's sibling nodes. In the case that S are the nodes of interest (RAN scenario), the rendered nodes have been constructed in the beginning of the exploration (as part of RAN initial nodes). Otherwise, the presented nodes have been previously constructed due to construction *Rule* $1(i)$.

(2) *A set of internal sibling nodes C are presented and the user performs a drill-down action over a node $c \in C$.* In this case, the drill-down will render c child nodes. If C are the nodes of interest (RAN scenario), then the child nodes of c have been constructed at the beginning of the exploration (as part of RAN initial nodes). Else, if C is the root node (BSC scenario), then again the child nodes of c have been constructed at the beginning of the exploration (as part of BSC initial nodes). Otherwise, the children of c, have been constructed before due to construction *Rule* $1(ii)$.

(3) *A set of leaf sibling nodes L are presented and the user performs a drill-down action over a leaf $l \in L$.* In this case the drill-down action will render data objects contained in l. Since a leaf is constructed together with its data objects, all data objects here have been previously constructed along with l.

(4) *A set of data objects O are presented and the user performs a roll-up action.* Here, the roll-up action will render the leaf that contains O along with the leaf's siblings. In RAN and BSC exploration scenarios, data objects are reachable only via a drill-down action over the leaf over the leaf that are contained, whereas in the RES scenario, the data objects, contained in the leaf of interest, are the first elements that are presented to the user.

In the general case, since O are reached only via a drill-down, their parent leaf has already been constructed. Based on Remark 1, all sibling nodes of this leaf have also been constructed. In the case of the RES scenario, where O includes the resource of interest, the leaf that contains O along with leaf's siblings have been constructed at the beginning of the exploration.

Thus, it is shown that, in all cases, the HETree elements that a user can reach by performing one operation, have been previously constructed by ICO. This concludes the proof of Proposition 1. ■

Also, the following theorem shows that over any exploration scenario ICO constructs only the required HETree elements.

Theorem 1. ICO constructs the minimum number of HETree elements in any exploration scenario.

PROOF. We will show that, during an exploration scenario, in any exploration step, ICO constructs only the required HETree elements. Considering an exploration scenario, ICO constructs nodes only either as initial nodes, or via construction rules. The initial nodes are constructed once, at the beginning of the exploration process; based on the definition of the initial nodes, these nodes are the required HETree elements for the first user operation.

During the exploration process, ICO constructs nodes only via the construction rules. Essentially, from construction rules, only the *Rule* 1 construct new nodes. Considering the part of the tree rendered when *Rule* 1 is applied, it is apparent that the nodes constructed by *Rule* 1 are only the required HETree elements.

Therefore, it is apparent that in any exploration step, ICO constructs only the required HETree elements. By considering all the steps comprising a user exploration scenario, the overall number of elements constructed is the minimum. This concludes the proof of Theorem 1. ■

4.1.2.2.1 ICO Algorithm

In this section, we present the incremental HETree construction algorithm. Note that, here we include the pseudocode only for the HETree-R version, since the only difference with the HETree-C version is in the way that the nodes' intervals are computed and that the dataset is initially sorted. In the analysis of the algorithms, both versions are studied.

Here, we assume that each node n contains the following extra fields. Let a node n, $n.p$ denotes the parent node of n, and $n.h$ denotes the height of n in the hierarchy. Additionally, given a dataset D, $D.minv$ and $D.maxv$ denote the minimum and the maximum value for all objects in D, respectively. The user preferences regarding the exploration's starting point are represented as an interval U. In the RES scenario, given that the value of the explored property for the resource of interest is o, we have $U^- = U^+ = o$. In the RAN scenario, given that the range of interest is R, we have that $U^- = \max(D.minv, R^-)$ and $U^+ = \min(D.maxv, R^+)$. In the BSC scenario, the user does not provide any preferences regarding the starting point, so we have $U^- = D.minv$ and $U^+ = D.maxv$. Finally, according to the definition of HETree, a node n *encloses* a data object (i.e., triple) tr if $n.I^- \geq tr.o$ and $n.I^+ \leq tr.o$.

The algorithm ICO-R (Algorithm 7) implements the incremental method for HETree-R. The algorithm uses two procedures to construct all required nodes. The first procedure constrRollUp-R (Procedure 4) constructs the nodes which can be reached by a roll-up operation, whereas constrDrillDown-R (Procedure 5) constructs the nodes which can be reached by a drill-down operation. Additionally, the aforementioned procedures exploit two secondary procedures: computeSiblingInterv-R (Procedure 6) and constrSiblingNodes-R (Procedure 7), which are used for nodes' intervals computations and nodes construction.

The ICO-R algorithm is invoked at the beginning of the exploration scenario, in order to construct the initial nodes, as well as every time the user performs an operation. The algorithm takes as input the dataset D, the tree parameters d and ℓ, the starting point U, the currently presented (i.e., rendered) elements *cur*, and the constructed HETree H. ICO-R begins with the currently presented elements *cur* equal to *null* (*lines 1-5*). Based on the starting point U, the algorithm computes the interval I_0 corresponding to the sibling nodes that are first presented to the user, as well as its hierarchy height h_0 (*line 3*).

Algorithm 7. ICO-R(D, ℓ, d, U, cur, H)

Input: D: set of objects; ℓ: number of leaf nodes; d: tree degree; U: interval representing user's starting point; cur: currently presented elements; H: currently created HETree-R
Output: H: updated HETree-R
Variables: len: the length of the leaf's interval

```
1  if cur = null then                                              //first ICO call
2      len ← (D.maxv − D.minv) / ℓ
3      from U compute I_0, h_0                                     //used for constructing initial nodes
4      cur, H ← constrSiblingNodes-R(I_0, null, D, h_0)
5      if RES then return H
6  if cur[1].p = null and D ≠ ∅ then
7      H ← constrRollUp-R(D, d, cur, H)
8      if cur[1].h > 0 then                                        //cur are not leaves
9          H ← constrDrillDown-R(D, d, cur, H)
10 return H
```

Procedure 4: constrRollUp-R(D, d, cur, H)

Input: D: set of objects; d: tree degree; cur: currently presented elements; H: currently created HETree-R
Output: H: updated HETree-R
//Computed in ICO-R: len: the length of the leaf's interval

```
1  create an empty node par                                        //cur parent node
2  par.h ← cur[1].h + 1
3  par.I⁻ ← cur[1].I⁻
4  par.I⁺ ← cur[|cur|].I⁺
5  for i ← 1 to |cur| do                                           //create parent-child relations
6      par.c[i] ← cur[i]
7      cur[i].p ← par
8  insert par into H
9  l_p ← par.I⁺ − par.I⁻                                           //par interval length
10 I⁻_ppar ← D.minv + d · l_p · ⌊(par.I⁻ − D.minv) / (d · l_p)⌋
11 I⁺_ppar ← min(D.maxv, I⁻_ppar + d · l_p)                        //compute interval for par parent, I_ppar
12 l_sp ← (len · d^(cur[1].h))                                     //interval length for a par sibling node
13 I_spar ← computeSiblingInterv-R(I⁻_ppar, I⁺_ppar, l_sp, d)      //compute intervals for all par sibling nodes
14 remove par.I from I_spar                                        //remove par interval, par already constructed
15 S ← constrSiblingNodes-R(I_spar, null, D, cur[1].h + 1)
16 insert S into H
17 return H
```

For sake of simplicity, the details for computing I_0 and h_0 are omitted. For example, the interval I for the leaf that contains the resource of interest with object value o, is computed as $I^- = D.minv + len \cdot \lfloor \frac{o - D.minv}{len} \rfloor$ and $I^+ = \min(D.maxv, I^- + len)$. Following a similar approach, we can easily compute I_0 and h_0.

Based on I_0, the algorithm constructs the sibling nodes that are first presented to the user (*line 4*). Then, the algorithm constructs the rest initial nodes (*lines 6-9*). In the RES case, as I_0 we consider the interval that includes the leaf that contains the resource of interest along with its sibling leaves. Hence, all the initial nodes are constructed in line 4 and the algorithm terminates (*line 5*) until the next user's operation.

After the first call, in each ICO execution, the algorithm initially checks if the parent node of the currently presented elements is already constructed, or if all the nodes that enclose data objects[8] have been constructed (*line 6*). Then, procedure constrRollUp-R (*line 7*) is used to construct the *cur* parent node, as well as the parent's siblings. In the

[8]Note that in the HETree-R version, we may have nodes that do not enclose any data objects.

Procedure 5: constrDrillDown-R(D, d, cur, H)

Input: D: set of objects; d: tree degree; cur: currently presented elements; H: currently created HETree-R
Output: H: updated HETree-R
//Computed in ICO-R: len: the length of the leaf's interval

1 $l_c = len \cdot d^{cur[1].h-1}$ //length of the children's intervals
2 **for** $i \leftarrow 1$ **to** $|cur|$ **do**
3 **if** $cur[i].c[0] =$ null **then continue** //nodes previously constructed
4 $I_{ch} \leftarrow$ computeSiblingInterv-R($cur[i].I^-, cur[i].I^+, l_c, d$) //compute intervals for $cur[i]$ children
5 $S \leftarrow$ constrSiblingNodes-R($I_{ch}, cur[i], cur[i].data, cur[1].h-1$)
6 **for** $k \leftarrow 1$ **to** $|S|$ **do**
7 $cur[i].c[k] \leftarrow S[k]$
8 insert S into H
9 **return** H

Procedure 6: computeSiblingInterv-R(low, up, len, n)

Input: low: intervals' lower bound; up: intervals' upper bound; len: intervals' length; n: number of siblings
Output: I: an ordered set with at most n equal length intervals

1 $I_t^-, I_t^+ \leftarrow low$
2 **for** $i \leftarrow 1$ **to** n **do**
3 $I_t^- \leftarrow I_t^+$
4 $I_t^+ \leftarrow \min(up, len + I_t^-)$
5 append I_t to I
6 **if** $I_t^+ = up$ **then break**
7 **return** I

case that *cur* are not leaf nodes or data objects (*line 8*), procedure constrDrillDown-R (*line 9*) is used to construct all *cur* children. Finally, the algorithm returns the updated HETree (*line 10*).

The constrRollUp-R (Procedure 4) initially constructs the *cur* parent node *par* (*lines 1-7*). Next, it computes the interval I_{ppar} corresponding to *par* parent node interval (*lines 10-11*). Using I_{ppar}, it computes the intervals for each of *par* sibling nodes (*line 13*). Finally, the computed sibling nodes' intervals I_{spar} are used for the nodes construction (*line 15*).

In the constrDrillDown-R (Procedure 5), for each node in *cur*, its children are constructed as follows (*line 2*). First, the procedure computes the intervals I_{ch} of each child and then it constructs all children (*line 5*). Finally, the child relations for the parent node $cur[i]$ are constructed (*line 6-7*).

4.1.2.2.2 Computational Analysis

Here we analyse the incremental construction for both HETree versions.

Number of Constructed Nodes. Regarding the number of *initial nodes* constructed in each scenario: in RES scenario, at most d leaf nodes are constructed; in RAN scenario, at most $2d + d^2$ nodes are constructed; finally in BSC scenario, $d + 1$ are constructed.

Regarding the maximum number of nodes constructed *in each operation* in RES and RAN scenarios: (1) A *roll-up operation* constructs at most $d + d \cdot (d-1) = d^2$ nodes. The d nodes are constructed in constrRollUp, whereas the $d \cdot (d-1)$ in constrDrillDown. (2) A *drill-down operation* constructs at most d^2 nodes in constrDrillDown. As for the BSC scenario: (1) A *roll-up operation* does not construct any nodes. (2) A *drill-down operation* constructs at most d^2 nodes in constrDrillDown.

Procedure 7: constrSiblingNodes-R(I, p, A, h)

Input: I: an ordered set with equal length intervals p: nodes' parent node; A: available data objects; h: nodes' height
Output: S: a set of HETree-R sibling nodes

```
1  l = I[1]^+ − I[1]^−                                   //intervals' length
2  T[ ] ← ∅
3  foreach tr ∈ A do                                     //indicate enclosed data for each node
4      j ← ⌊(tr.o − I[1]^−) / l⌋ + 1
5      if j ≥ 0 and j ≤ |I| then
6          insert object tr into T[j]
7          remove object tr from A
8  for i ← 1 to |I| do                                   //construct nodes
9      if T[i] = ∅ then continue
10     create a new node n
11     n.I^− ← I[i]^−
12     n.I^+ ← I[i]^+
13     n.p ← p
14     n.c ← null
15     n.data ← T[i]
16     n.h ← h
17     if h = 0 then                                     //node is a leaf
18         sort n.data based on objects values
19     append n to S
20 return S
```

Next, we analyse in details the worst case of ICO algorithm, i.e., when the construction cost is maximized.

The HETree-R Version. The worst case in HETree-R occurs when the whole dataset D is contained in a set of sibling leaf nodes L, where $|L| \leq d$.

Considering the above setting, in the RES scenario, the cost is maximized when ICO-R constructs L (as initial nodes). In this case, the cost is $O(|D| + |D|log|D|) = O(|D|log|D|)$.

In a RAN scenario, the cost is maximized when the parent node p of L along with p's sibling nodes are considered as nodes of interest. First, let's note that in this case p has no sibling nodes, since all the sibling nodes are empty (i.e., they do not enclose data). Hence, the p has to be constructed in ICO-R as initial nodes, as well as the L in constrDrillDown-R, and the parent of p in constrRollUp-R. The p construction in ICO-R requires $O(|D|)$. Also, the L construction in constrDrillDown-R requires $O(d + |D| + |D|log|D| + d)$. Finally, the construction of the parent of p in constrRollUp-R requires $O(1)$. Therefore, in RAN the overall cost in the worst case is $O(|D| + d + |D| + |D|log|D| + d) = O(|D|log|D|)$.

Finally, in BSC scenario, the cost is maximized when the L have to be constructed by constrDrillDown-R, which requires $O(d + |D| + |D|log|D| + d) = O(|D|log|D|)$.

The HETree-C Version. First let's note that in HETree-C version, the dataset is sorted at the beginning of the exploration and the leaves contain equal number of data objects. As a result, during a node construction, the data objects, enclosed by it, can be directly identified by computing its position over the dataset and without the need of scanning the dataset or the enclosed data values. However, in ICO we assume that the node's statistics are computed each time the node is constructed. Hence, in each node construction, we scan the data objects that are enclosed by this node. In RES scenario, the worst case occurs, when the user rolls up for the first time to the nodes at level 2 (i.e., two levels below the root). In this case, ICO has to construct the d nodes at level 1, as well the children for the $d - 1$ nodes in level 2. Note that the construction of the parent of the

nodes in level 2 does not require to process any data objects or construct children, since these nodes are already constructed. Now regarding the construction of the rest $d-1$ nodes at level 1, ICO will process at most the $\frac{d-1}{d}$ of all data objects[9]. Thus, the cost for constrRollUp-C is $O(d + \frac{d-1}{d}|D| + d - 1)$. Finally, for constructing the child nodes for the $d-1$ nodes in level 2, we are required to process at most the $\frac{d-1}{d^2}$ of all data objects. Hence, the cost for constrDrillDown-C is $O(d^2 + \frac{d-1}{d^2}|D| + d^2)$. Therefore, in RES the cost in worst case is $O(d + \frac{d-1}{d}|D| + d - 1 + d^2 + \frac{d-1}{d^2}|D| + d^2) = O(d^2 + \frac{d-1}{d}|D|)$.

In RAN scenario, the worst case occurs, when the user starts from any set of sibling nodes at level 2. Hence, the cost is maximized at the beginning of the exploration. In this case, ICO has to construct the d initial nodes at level 2, the d nodes at level 1, and the children for all the d nodes in level 2. First the d initial nodes at level 2 are constructed by ICO-R, which can be done in $O(d + |D| + d)$. Then, the d nodes at level 1 are constructed by constrRollUp-C. Similarly as in RES scenario, this can be done in $O(d + \frac{d-1}{d}|D| + d - 1)$. Finally, the construction of the child nodes for all the d nodes in level 2 requires to process $\frac{|D|}{d}$ data objects. Hence, the cost for constrDrillDown-C is $O(d^2 + \frac{|D|}{d} + d^2)$. Therefore, in RAN the cost, in the worst case, is $O(|D|+d+d+\frac{d-1}{d}|D|+d-1+d^2+\frac{|D|}{d}+d^2) = O(d^2+\frac{d-1}{d}|D|)$.

Finally, in BSC scenario, the worst case occurs, when the user visits for the first time any of the node at level 1. In this case, ICO has to construct the children for the d nodes in level 1. Hence, constrDrillDown-C has to process $|D|$ data objects in order to construct the d^2 child nodes. Therefore, in BSC the cost in worst case is $O(d^2 + |D| + d^2) = O(d^2 + |D|)$.

Discussion. The worst case for the computational cost is higher in HETree-R than in HETree-C, for all exploration scenarios. Particularly, in HETree-R worst case, ICO must build leaves that contain the whole dataset and the computational cost is $O(|D|log|D|)$ for all scenarios. In HETree-C, for the RES and RAN scenarios, the cost is $O(d^2 + \frac{d-1}{d}|D|)$, and for the BSC scenario the cost is $O(d^2 + |D|)$.

4.1.2.3 Adaptive HETree Construction

In a (visual) exploration scenario, users wish to modify the organization of the data by providing user-specific preferences for the whole hierarchy or part of it. The user can select a specific subtree and alter the number of groups presented in each level (i.e., the tree degree) or the size of the groups (i.e., number of leaves). In this case, a new tree (or a part of it) pertaining to the new parameters provided by the user should be constructed on-the-fly.

For example, consider the HETree-C of Figure 4.6 representing ages of persons[10]. A user may navigate to node b, where she prefers to increase the number of groups presented in each level. Thus, she modifies the degree of b from 2 to 4 and the subtree is adapted to the new parameter as depicted on the bottom tree of Figure 4.6. On the other hand, the user prefers exploring the right subtree (starting from node c) with less details. She chooses to increase the size of the groups by reducing (from 4 to 2) the number of leaves for the subtree of c. In both cases, constructing the subtree from scratch based on the user-provided parameters and recomputing statistics entails a significant time overhead, especially, when user preferences are applied to a large part of or the whole hierarchy.

In this section, we introduce ADA (**Ada**ptive HETree Construction) method, which dynamically adapts an existing HETree to a new set of user-defined parameters. Instead of both constructing the tree and computing the nodes' statistics from scratch, our method reconstructs the new part(s) of the hierarchy by exploiting the existing elements (i.e., nodes, statistics) of the tree. In this way, ADA achieves to reduce the overall construction

[9]This number can be easily computed by considering the number of leafs enclosed by these nodes.

[10]For simplicity, Figure 4.6 presents only the values of the objects.

Table 4.2: Summary of adaptive HETree construction*

		Modify Degree				**Modify Num. of Leaves**			
	Full Construction	$d' = d^k$	$d' = k \cdot d$	$d' = \sqrt[k]{d}$	*elsewhere*	$\ell' > \ell$	$\ell' = \frac{\ell}{d^k}$	$\ell' = \frac{\ell}{k}$	$\ell' = \ell - k$
Tree Construction									
Complexity	$O(mlogm+d'e)$	$O(mlog_{\sqrt[k]{d'}}m)$	$O(d'e)$	$O(d'^k r)$	$O(d'e)$	$O(m+d'e)$	$O(m)$	$O(m+d'e)$	$O(mlogm+d'e)$
#leaves$_0$	ℓ'	0	0	0	0	ℓ'	0	0	ℓ'
#leaves$_+$	0	0	0	0	0	0	ℓ'	ℓ'	0
#internals$_0$	e	0	e	$e-r$	e	e	0	e	e
#internals$_+$	0	0	0	0	0	0	0	0	0
Statistics Computations									
Complexity	$O(m+d'e)$	$O(1)$	$O(\frac{k\ell'}{d'}+d'e)$	$O(d'(e-r))$	$O(d'e)$	$O(m+d'e)$	$O(1)$	$O(m+d'e)$	$O(m+d'e-\ell'-k)$
#leaves$_0$	ℓ'	0	0	0	0	ℓ'	0	0	$\ell' - \frac{\ell'^2}{d'}$
#leaves$_+$	0	0	0	0	0	0	0	ℓ'	$\frac{\ell'^2}{d'}$
#internals$_0$	e	0	$e - \lceil \frac{\ell'}{d'} \rceil$	$e-r$	e	e	0	e	e
#internals$_+$	0	0	$\lceil \frac{\ell'}{d'} \rceil$	0	0	0	0	0	0

*$m = |D|$, $e = \frac{d'\ell'-1}{d'-1}$ (maximum number of internal nodes), and $r = \frac{d'^k\ell'-1}{d'^k-1}$

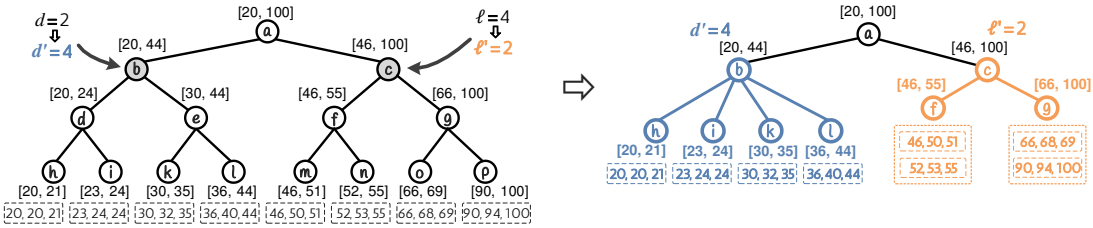


Figure 4.6: Adaptive HETree example

cost and enables the on-the-fly reorganization of the visualized data. In the example of Figure 4.6, the new subtree of b can be derived from the old one, just by removing the internal nodes d and e, while the new subtree of c results from merging leaves together and aggregating their statistics.

Let $\mathcal{T}(D, \ell, d)$ denote the existing HETree and $\mathcal{T}'(D, \ell', d')$ is the new HETree corresponding to the new user preferences for the tree degree d' and the number of leaves ℓ'. Note that $\mathcal{T}$ could also denote a subtree of an existing HETree (in the scenario where the user modifies only a part of it). In this case, the user indicates the *reconstruction root* of $\mathcal{T}$.

Then, ADA identifies the following elements of $\mathcal{T}$: (1) The elements of $\mathcal{T}$ that also exist in $\mathcal{T}'$. For example, consider the following two cases: the leaf nodes of $\mathcal{T}'$ are internal nodes of $\mathcal{T}$ in level x; the statistics of $\mathcal{T}'$ nodes in level x are equal to the statistics of $\mathcal{T}$ nodes in level y. (2) The elements of $\mathcal{T}$ that can be reused (as "building blocks") for constructing elements in $\mathcal{T}'$. For example, consider the following two cases: each leaf node of $\mathcal{T}'$ is constructed by merging x leaf nodes of $\mathcal{T}$; the statistics for the node n of $\mathcal{T}'$ can be computed by aggregating the statistics from the nodes q and w of $\mathcal{T}$.

Consequently, we consider that an element (i.e., node or node's statistics) in $\mathcal{T}'$ can be: (1) constructed/computed from scratch[11], (2) reused as is from $\mathcal{T}$ or (3) derived by aggregating elements from $\mathcal{T}$.

Table 4.2 summarizes the ADA reconstruction process. Particularly, the table includes: (1) the computational complexity for constructing $\mathcal{T}'$, denoted as *Complexity*; (2) the number of leaves and internal nodes of $\mathcal{T}'$ constructed from scratch, denoted as $\#leaves_0$ and $\#internals_0$, respectively; and (3) the number of leaves and internal nodes of $\mathcal{T}'$ derived from nodes of $\mathcal{T}$, denoted as $\#leaves_+$ and $\#internals_+$, respectively. The lower part of the table presents the results for the computation of node statistics in $\mathcal{T}'$. Finally, the second table column, denoted as *Full Construction*, presents the results of constructing $\mathcal{T}'$ from scratch.

The following example demonstrates the ADA results, considering a DBpedia exploration scenario.

Example 8. The user explores the *populationTotal* property of the DBpedia dataset. The default system organization for this property is a hierarchy with degree 3. The user modifies the tree parameters in order to fit better visualization results as following. First, she decides to render more groups in each hierarchy level and increases the degree from 3 to 9 (1st *Modification*). Then, she observes that the results overflow the visualization area and that a smaller degree fits better; thus she re-adjusts the tree degree to a value of 6 (2nd *Modification*). Finally, she navigates through the data values and decides to increase the groups' size by a factor of three (i.e., dividing by three the number of leaves) (3rd *Modification*). Again, she corrects her decision and readjusts the final group size to twice the default size (4th *Modification*).

Table 4.3 summarizes the number of nodes, constructed by a *Full Construction* and

[11]Note that it is possible for a from scratch constructed node in $\mathcal{T}'$ to aggregate statistics from nodes in $\mathcal{T}$.

ADA in each modification, along with the required statistics computations. Considering the whole set of modifications, ADA constructs only the 22% (15.4K vs. 70.2K) of the nodes that are created in the case of the full construction. Also, ADA computes the statistics for only 8% (5.6K vs. 70.2K) of the nodes. □

Table 4.3: Full Construction vs. ADA over DBpedia exploration scenario (cells values: Full / ADA)

	Modify Degree		**Modify Num. of Leaves**	
	1st Modification	2nd Modification	3rd Modification	4th Modification
Tree Construction				
#nodes	22.1K / 0	23.6K / 3.9K	9.8K / 6.6K	14.7K / 4.9K
Statistics Computations				
#nodes	22.1K / 0	23.6K / 659	9.8K / 0	14.7K / 4.9K

In the next sections, we present in detail the reconstruction process through the example trees of Figure 4.7. Figure 4.7a presents the initial tree $\mathcal{T}$ that is an HETree-C, with $\ell = 8$ and $d = 2$. Figures 4.7b ~ 4.7e present several reconstructed trees $\mathcal{T}'$. Blue colour is used to indicate the elements (i.e., nodes, edges, statistics) of $\mathcal{T}'$ which do not exist in $\mathcal{T}$. Regarding statistics, we assume that in each node we compute the mean value. In each $\mathcal{T}'$, we present only the mean values that are not known from $\mathcal{T}$. Also, in mean values computations with red colour, we highlight the values that are reused from $\mathcal{T}$.

4.1.2.3.1 Preliminaries

In order to perform traversal over the levels of the HETree (i.e., level-order traversal), we use an array $\mathcal{H}$ of pointers to the ordered set of nodes at each level, with $\mathcal{H}[0]$ referring to the set of leaf nodes and $\mathcal{H}[k]$ referring to the set of nodes at height k. Moreover, we consider the following simple procedures that are used for the ADA implementation:

- mergeLeaves(L, m), where L is an ordered set of leaf nodes and $m \in \mathbb{N}^+$, with $m > 1$. This procedure returns an ordered set of $\lceil \frac{L}{m} \rceil$ new leaf nodes, i.e., each new leaf merges m leaf nodes from L. The procedure traverses L, constructs a new leaf for every m nodes in L and appends the data items from the m nodes to the new leaf. This procedure requires $O(|L|)$.
- replaceNode(n_1, n_2), replaces the node n_1 with the node n_2; it removes n_1, and updates the parent of n_1 to refer to n_2. This procedure requires constant time, hence $O(1)$.
- createEdges(P, C, d), where P, C are ordered sets of nodes and d is the tree degree. It creates the edges (i.e., parent-child relations) from the parent nodes P to the child nodes C, with degree d. The procedure traverses over P and connects each node $P[i]$ with the nodes from $C[(i-1)d+1]$ to $C[(i-1)d+d]$. This procedure requires $O(|C|)$.

4.1.2.3.2 The User Modifies the Tree Degree

Regarding the modification of the degree parameter, we distinguish the following cases:

The user increases the tree degree. We have that $d' > d$; based on the d' value we have the following cases:

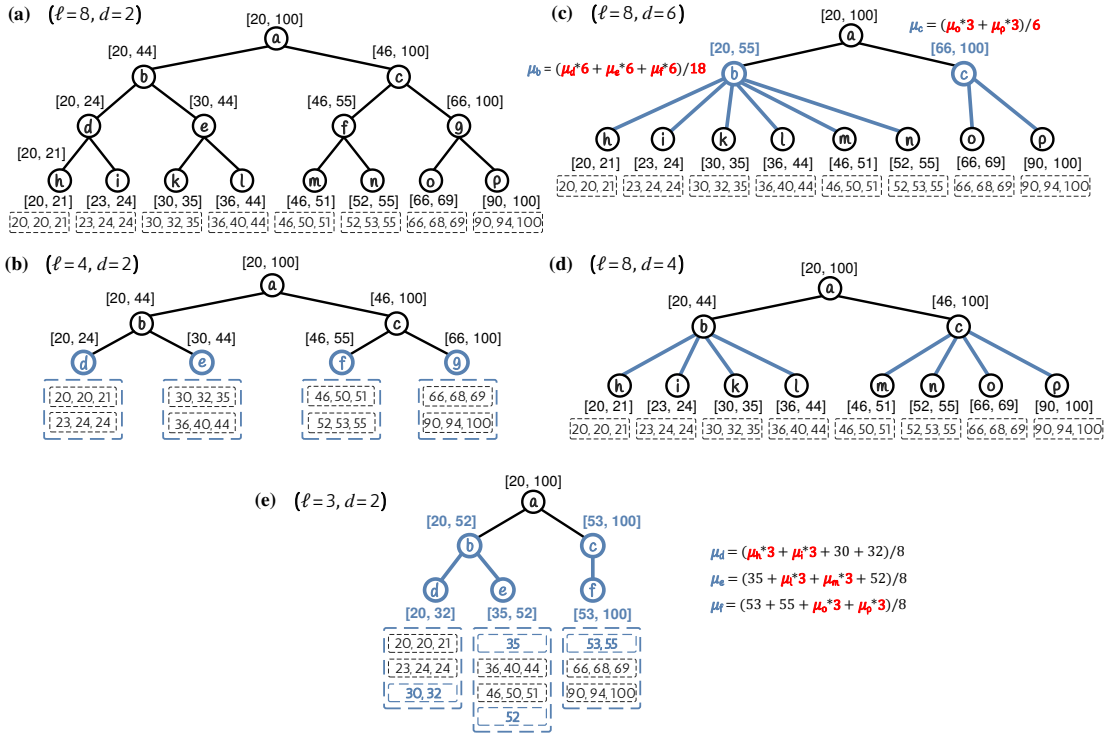


Figure 4.7: Adaptive HETree construction examples

(1) $d' = d^k$, with $k \in \mathbb{N}^+$ and $k > 1$

Figure 4.7a presents $\mathcal{T}$ with $d = 2$ and Figure 4.7d presents the reconstructed $\mathcal{T}'$ with $d' = 4$ (i.e., $k = 2$). $\mathcal{T}'$ results by simply removing the nodes with height 1 (i.e., d, e, f, g) and connecting the nodes with height 2 (i.e., b, c) with the leaves. In general, $\mathcal{T}'$ results from $\mathcal{T}$ by simply removing tree levels from $\mathcal{T}$. Additionally, there is no need for computing any new statistics, since the statistics for all nodes of $\mathcal{T}'$ remain the same as in $\mathcal{T}$.

Tree Construction. For the $\mathcal{T}'$ construction, we perform a reverse level-order traversal over $\mathcal{T}$, using the $\mathcal{H}$ vector. Starting from the leaves ($\mathcal{H}[0]$), we skip (i.e., remove) $k-1$ levels of nodes. Then, for the nodes of the above level ($\mathcal{H}[k]$), we create child relations with the (non-skipped) nodes in the level below. The above process continues until we reach the root node of $\mathcal{T}$.

Hence, in this case all nodes in $\mathcal{T}'$ are obtained "directly" from $\mathcal{T}$. Particularly, $\mathcal{T}'$ is constructed using the root node of $\mathcal{T}$, as well the $\mathcal{T}$ nodes from $\mathcal{H}[j \cdot k], j \in \mathbb{N}^0$.

The $\mathcal{T}'$ construction requires the execution of createEdges procedure, j times. For computing j, we have that $j \cdot k \leq |\mathcal{H}| \Leftrightarrow j \cdot k \leq log_d \ell$. Considering that $d' = d^k$, we have that $k = log_d d'$. Hence, $j \cdot log_d d' \leq log_d \ell \Leftrightarrow j \leq log_d(\ell - d')$. So, considering that worst case complexity for createEdges is $O(\ell)$, we have that the overall complexity is $O(\ell \cdot log_d(\ell - d'))$. Since we have that $\ell \leq |D|$, then in worst case the $\mathcal{T}'$ can be constructed in $O(|D| log_d(|D|)) = O(|D| log_{\sqrt[k]{d'}}(|D|))$.

Statistics Computations. In this case there is no need for computing any new statistics.

(2) $d' = k \cdot d$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^\nu$ where $\nu \in \mathbb{N}^+$

An example with $k = 3$ is presented in Figure 4.7c, where we have $d' = 6$. In this case, the leaves of $\mathcal{T}$ (Figure 4.7a) remain leaves in $\mathcal{T}'$ and all internal nodes up to the reconstruction root of $\mathcal{T}$ are constructed from scratch. As for the node statistics, we can compute the mean values for $\mathcal{T}'$ nodes with height 1 (i.e., μ_b, μ_c) by aggregating already computed

mean values (e.g., μ_d, μ_e, etc.) from $\mathcal{T}$.

In general, except for the leaves, we construct all internal nodes from scratch. For the internal nodes of height 1, we compute their statistics by aggregating the statistics of $\mathcal{T}$ leaves, whereas for internal nodes of height greater than 1, we compute from scratch their statistics.

Tree Construction. As in $\mathcal{T}'$ the leaves remain the same as in $\mathcal{T}$, we only use the constrInterNodes (Procedure 2) to build the rest of the tree. Therefore, in the worst case, the complexity for constructing the $\mathcal{T}'$ is $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

Statistics Computations. The statistics for $\mathcal{T}'$ nodes of height 1 can be computed by aggregating statistics from $\mathcal{T}$. Particularity, in $\mathcal{T}'$ the statistics computations for each internal node of height 1, require $O(k)$ instead of $O(d')$, where $k = \frac{d'}{d}$. Hence, considering that there are $\lceil \frac{\ell}{d'} \rceil$ internal nodes of height 1 in $\mathcal{T}'$, the cost for their statistics is $O(k \cdot \lceil \frac{\ell}{d'} \rceil) = O(\frac{k \cdot \ell}{d'} + k)$.

Regarding the cost of recomputing them from scratch, consider that there are $\frac{\ell-1}{d'-1}$ internal nodes[12] with heights greater than 1; the statistics computations for these nodes require $O(\frac{d' \cdot \ell - d'}{d'-1})$. Therefore, the overall cost for statistics computations is $O(\frac{k \cdot \ell}{d'} + k + \frac{d' \cdot \ell - d'}{d'-1})$.

(3) *elsewhere*

In any other case where the user increases the tree degree, all internal nodes in $\mathcal{T}'$ except for the leaves are constructed from scratch. In contrast with the previous case, the leaves' statistics from $\mathcal{T}$ can not be reused and, thus, for all internal nodes in $\mathcal{T}'$ the statistics are recomputed.

Tree Construction. Similar to the previous case, the $\mathcal{T}'$ construction requires $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

Statistics Computations. In this case, the statistics should be computed from scratch for all internal nodes in $\mathcal{T}'$. Therefore, the complexity is $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

The user decreases the tree degree. Here we have that $d' < d$; based on the d' value we have the following two cases:

(1) $d' = \sqrt[k]{d}$, with $k \in \mathbb{N}^+$ and $k > 1$

Assume that now Figure 4.7d depicts $\mathcal{T}$, with $d = 4$, while Figure 4.7a presents $\mathcal{T}'$ with $d' = 2$. We can observe that $\mathcal{T}'$ contains all nodes of $\mathcal{T}$, as well as a set of extra internal nodes (i.e., d, e, f, g). Hence, $\mathcal{T}'$ results from $\mathcal{T}$ by constructing some new internal nodes.

Tree Construction. For the $\mathcal{T}'$ construction we perform a reverse level-order traversal over $\mathcal{T}$ using the $\mathcal{H}$ vector and starting from the nodes having height of 1. In each level, for each node n we call the constrInterNodes (Procedure 2) using as input the d child nodes of n and the new degree d'. Note that, in this reconstruction case, the constrInterNodes does require to construct the root node; the root node here is always corresponding to the node n. Hence, the complexity of constrInterNodes for one call is $O(d)$. Considering that, we perform a procedure call for all the internal nodes, as well as that the maximum number of internal nodes is $\frac{d \cdot \ell - 1}{d-1}$, we have that, in the worst case the $\mathcal{T}'$ can be constructed in $O(\frac{d^2 \cdot \ell - d}{d-1}) = O(\frac{d'^{2k} \cdot \ell - d'^k}{d'^k - 1})$.

Regarding the number of internal nodes that we have to construct from scratch. Since $\mathcal{T}'$ has all the nodes of $\mathcal{T}$, for $\mathcal{T}'$ we have to construct from scratch $\frac{d' \cdot \ell - 1}{d'-1} - \frac{d \cdot \ell - 1}{d-1}$ new internal

[12] Take into account that the maximum number of internal nodes (considering all levels) is $\frac{d^{\lceil log_d \ell \rceil} - 1}{d-1}$.

nodes, where the first part corresponds to the number of internal nodes of $\mathcal{T}'$, and the second part corresponds to $\mathcal{T}$. Considering that, $d' = \sqrt[k]{d}$, we have to build $\frac{d' \cdot \ell - 1}{d' - 1} - \frac{d'^k \cdot \ell - 1}{d'^k - 1}$ internal nodes.

Statistics Computations. Statistics should be computed only for the new internal nodes of $\mathcal{T}'$. Hence, the cost here is $O(d' \cdot (\frac{d' \cdot \ell - 1}{d' - 1} - \frac{d'^k \cdot \ell - 1}{d'^k - 1}))$

(2) *elsewhere*

This case is similar to the previous case (3) where the user increases the tree degree.

4.1.2.3.3 The User Modifies the Number of Leaves

Regarding the modification of the number of leaves parameter, we distinguish the following cases:

The user increases the number of leaves. In this case we have that $\ell' > \ell$; hence, each leaf of $\mathcal{T}$ is split into several leaves in $\mathcal{T}'$ and the data objects contained in a $\mathcal{T}$ leaf must be reallocated to the new leaves in $\mathcal{T}'$. As a result, all nodes (both leaves and internal nodes) in $\mathcal{T}'$ have different contents compared to nodes in $\mathcal{T}$ and must be constructed from scratch along with their statistics.

In this case, constructing $\mathcal{T}'$ requires $O(|D| + \frac{d^2 \cdot \ell' - d}{d - 1})$ (by avoiding the sorting phase).

The user decreases the number of leaves. In this case we have that $\ell' < \ell$; based on the ℓ' value we have the following three cases:

(1) $\ell' = \dfrac{\ell}{d^k}$, with $k \in \mathbb{N}^+$

Considering that Figure 4.7a presents $\mathcal{T}$ with $\ell = 8$ and $d = 2$. A reconstruction example of this case with $k = 1$, is presented in Figure 4.7b, where we have $\mathcal{T}'$ with $\ell' = 4$. In Figure 4.7b, we observe that the leaves in $\mathcal{T}'$ result from merging d^k leaves of $\mathcal{T}$. For example, the leaf d of $\mathcal{T}'$ results from merging the leaves h and i of $\mathcal{T}$. Then, $\mathcal{T}'$ results from $\mathcal{T}$, by replacing the $\mathcal{T}$ nodes with height k (i.e., b, e, f, g), with the $\mathcal{T}'$ leaves. Finally, the nodes of $\mathcal{T}$ with height less than k are not included in $\mathcal{T}'$.

Therefore, in this case, $\mathcal{T}'$ is constructed by merging the leaves of $\mathcal{T}'$ and removing the internal nodes of $\mathcal{T}'$ having height less or equal to k. Also, we do not recompute the statistics of the new leaves of $\mathcal{T}'$ as these are derived from the statistics of the removed nodes with height k.

Tree Construction. In this case, each leaf in $\mathcal{T}'$ is resulted by merging d^k leaves from $\mathcal{T}$. Hence, $\mathcal{T}'$ leaves are constructed by calling $\mathsf{mergeLeaves}(\ell, d^k)$. So, considering the $\mathsf{mergeLeaves}$ complexity, in worst case the new leaves construction requires $O(|D|)$. Then, each leaf of $\mathcal{T}'$ replace an internal nodes of $\mathcal{T}$ having height of k. Therefore, in worst case ($k = 1$), we call $\lceil \frac{\ell}{d} \rceil$ times the $\mathsf{replaceNode}$ procedure, which requires $O(\lceil \frac{\ell}{d} \rceil)$. Therefore, the overall cost for constructing $\mathcal{T}'$ in worst case is $O(|D| + \lceil \frac{\ell}{d} \rceil) = O(|D|)$. Note that in this, as well as in the following case, we assume that $\mathcal{T}$ is a perfect tree. In case where $\mathcal{T}$ is not perfect, we can use as $\mathcal{T}$ the perfect tree that initially proposed by our system.

Statistics Computations. In this case there is no need for computing any new statistics.

(2) $\ell' = \frac{\ell}{k}$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^\nu$, where $\nu \in \mathbb{N}^+$

As in the previous case, the leaves in $\mathcal{T}'$ are constructed by merging leaves from $\mathcal{T}$ and their statistics are computed based on the statistics of the merged leaves. In this case, however, all internal nodes in $\mathcal{T}'$ have to be constructed from scratch.

Tree Construction. In this case, each leaf in $\mathcal{T}'$ is resulted by merging k leaves from $\mathcal{T}$. Hence, the $\mathcal{T}'$ leaves are constructed by calling the mergeLeaves(ℓ, k), which in worst case requires $O(|D|)$. Then, the rest of the tree is constructed from scratch using the constrInterNodes. Therefore, the overall cost for $\mathcal{T}'$ construction is $O(|D| + \frac{d^2 \cdot \ell' - d}{d-1})$.

Statistics Computations. The statistics for all internal nodes have to be computed from scratch. Regarding the leaves, the statistics for each leaf in $\mathcal{T}'$ are computed by aggregating the statistics of the $\mathcal{T}$ leaves it includes. Essentially, for computing the statistics in each leaf in an HETree-C, we have to process k values instead of $\frac{|D|}{\ell'}$. However, in the worst case (i.e., $\ell = |D|$), we have that $k = \frac{|D|}{\ell'}$. Therefore, in the worst case (for both HETree versions) the complexity is the same as computing statistics from scratch.

(3) $\ell' = \ell - k$, with $k \in \mathbb{N}^+$, $k > 1$ and $\ell' \neq \frac{\ell}{\nu}$, where $\nu \in \mathbb{N}^+$

The two previous cases describe that each leaf in $\mathcal{T}'$ *fully* contains k leaves from $\mathcal{T}$. In this case, a leaf in $\mathcal{T}'$ may *partially* contains leaves from $\mathcal{T}$. A leaf in $\mathcal{T}'$ fully contains a leaf from $\mathcal{T}$ when the $\mathcal{T}'$ leaf contains all data objects belonging to the $\mathcal{T}$ leaf. Otherwise, a leaf in $\mathcal{T}'$ partially contain a leaf from $\mathcal{T}$ when the $\mathcal{T}'$ leaf contains a subset of the data objects from the $\mathcal{T}$ leaf.

An example of this case is shown in Figure 4.7e that depicts a reconstructed $\mathcal{T}'$ resulted from the $\mathcal{T}$ presented in Figure 4.7a. The d leaf of $\mathcal{T}'$ fully contains leaves h, i of $\mathcal{T}$ and partially leaf k for which value 35 belongs to a different leaf (i.e., e).

Due to this partial containment, we have to construct all leaves and internal nodes from scratch and recalculate their statistics. Still, the statistics of the fully contained leaves of $\mathcal{T}$ can be reused, by aggregating them with the individual values of the data objects included in the leaves. For example, as we can see in Figure 4.7e, the mean value μ_d of the leaf d is computed by aggregating the mean values μ_h and μ_i corresponding to the fully contained leaves h and i, with the individual values 30, 32 of the partially contained leaf k.

Tree Construction. In order to construct $\mathcal{T}'$ we have to construct all nodes from scratch, which in the worst case requires $O(|D|log|D| + \frac{d^2 \cdot \ell' - d}{d-1})$.

Statistics Computations. The statistics of the $\mathcal{T}$ leaves that are fully contained in $\mathcal{T}'$ can be used for calculating the statistics of the new leaves. The worst case is when the number of leaves that are fully contained in $\mathcal{T}'$ is minimized. For HETree-C (resp. HETree-R), this occurs when the size of leaves in $\mathcal{T}'$ is $\lambda' = \lambda + 1$ (resp. of length $\rho' = \rho + \frac{\rho}{\ell}$). In this case, for every λ (resp. ρ) leaves from $\mathcal{T}$ that are used to construct the $\mathcal{T}'$ leaves, at least 1 leaf is fully contained. Hence, when we process all ℓ leaves, at least $\frac{\ell^2}{|D|}$ leaves are fully contained in $\mathcal{T}'$.

Hence in HETree-C, in statistics computations over the leaves, instead of processing $|D|$ values, we process at most $|D| - \frac{\ell^2}{|D|} \cdot \lambda = |D| - \ell$. The same also holds in HETree-R, if we assume normal distribution over the values in D. Therefore, the cost for computing the leaves statistics is $O(|D| - \ell) = O(|D| - \ell' - k)$.

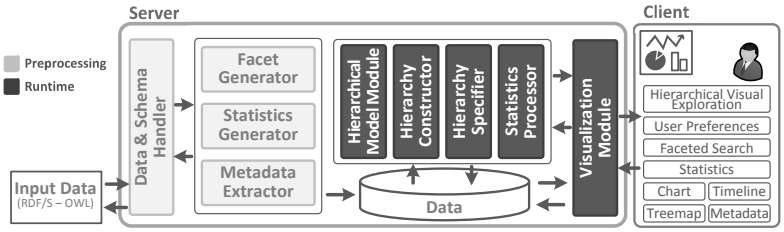


Figure 4.8: System architecture

4.1.3 The SynopsViz Tool

Based on the proposed hierarchical model, we have developed a Web-based prototype called *SynopsViz*[13]. The key features of SynopsViz are summarized as follows: (1) It supports the aforementioned *hierarchical model* for RDF data visualization, browsing and analysis. (2) It offers *automatic* on-the-fly hierarchy construction, as well as *user-defined* hierarchy construction based on users' preferences. (3) Provides *faceted* browsing and filtering over classes and properties. (4) Integrates *statistics with visualization*; visualizations have been enriched with useful statistics and data information. (5) Offers several visualization techniques (e.g., timeline, chart, treemap). (6) Provides a large number of dataset's *statistics* regarding the: *data-level* (e.g., number of sameAs triples), *schema-level* (e.g., most common classes/properties), and *structure level* (e.g., entities with the larger in-degree). (7) Provides numerous *metadata* related to the dataset: licensing, provenance, linking, availability, undesirability, etc. The latter can be considered useful for assessing data quality [402].

In the rest of this section, Section 4.1.3.1 describes the system architecture, Section 4.1.3.2 demonstrates the basic functionality of the SynopsViz. Finally, Section 4.1.3.3 provides technical information about the implementation.

4.1.3.1 System Architecture

The architecture of SynopsViz is presented in Figure 4.13. Our scenario involves three main parts: the Client UI, the SynopsViz, and the Input data. The *Client* part, corresponds to the system's front-end offering several functionalities to the end-users. For example, hierarchical visual exploration, facet search, etc. (see Section 4.1.3.2 for more details). SynopsViz consumes RDF data as *Input data*; optionally, OWL-RDF/S vocabularies/ontologies describing the input data can be loaded. Next, we describe the basic components of the SynopsViz.

In the preprocessing phase, the *Data and Schema Handler* parses the input data and inferes schema information (e.g., properties domain(s)/range(s), class/ property hierarchy, type of instances, type of properties, etc.). *Facet Generator* generates class and property facets over input data. *Statistics Generator* computes several statistics regarding the schema, instances and graph structure of the input dataset. *Metadata Extractor* collects dataset metadata. Note that the model construction does not require any preprocessing, it is performed online, according to user interaction.

During runtime the following components are involved. *Hierarchy Specifier* is responsible for managing the configuration parameters of our hierarchy model, e.g., the number of hierarchy levels, the number of nodes per level, and providing this information to the Hierarchy Constructor. *Hierarchy Constructor* implements our tree structure. Based on the selected facets, and the hierarchy configuration, it determines the hierarchy of groups and the contained triples. *Statistics Processor* computes statistics about the groups included in the hierarchy. *Visualization Module* allows the interaction between the user and the

[13] synopsviz.imis.athena-innovation.gr

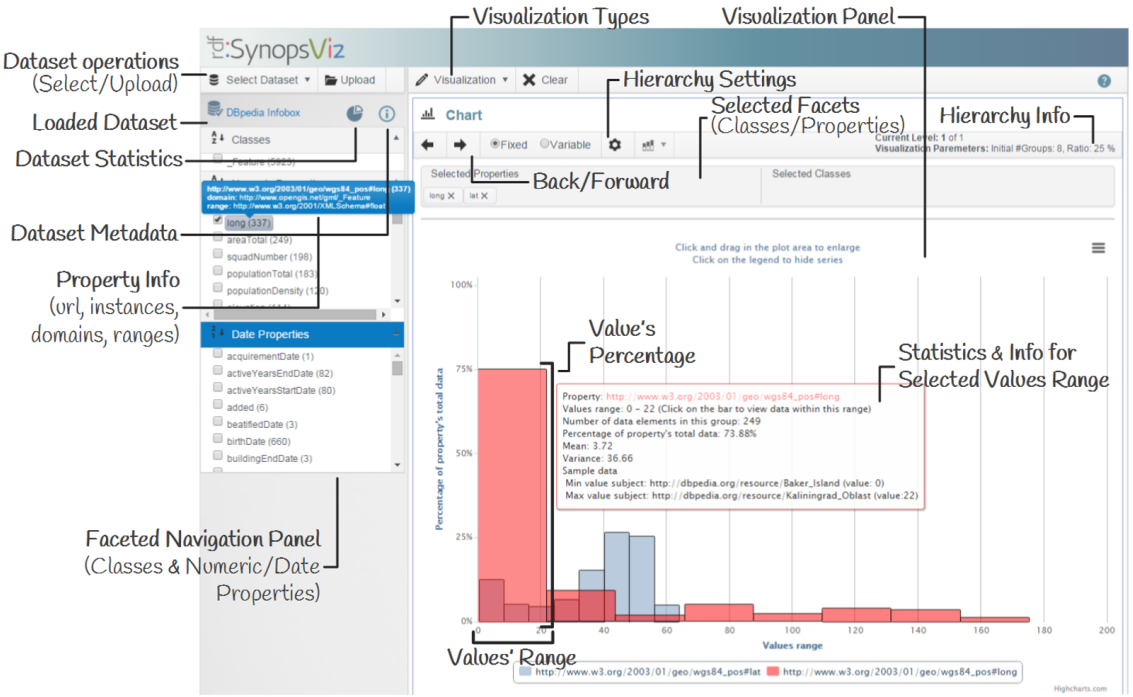


Figure 4.9: Web user interface

back-end, allowing several operations (e.g., navigation, filtering, hierarchy specification) over the visualized data. Finally, the *Hierarchical Model Module* maintains the in-memory tree structure for our model and communicates with the Hierarchy Constructor for the model construction, the Hierarchy Specifier for the model customization, the Statistics Processor for the statistics computations, and the Visualization Module for the visual representation of the model.

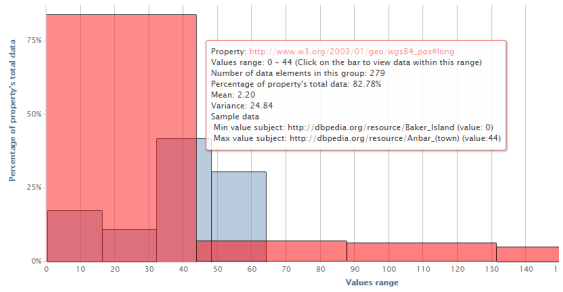
4.1.3.2 SynopsViz In-Use

In this section we outline the basic functionality of SynopsViz prototype. Figure 4.9 presents the Web user interface of the main window. SynopsViz UI consists of the following main panels: *Facets panel*: presents and manages facets on classes and properties; *Input data control panel*: enables the user to import and manage input datasets; *Visualization panel*: is the main area where interactive charts and statistics are presented; *Configuration panel*: handles visualization settings.

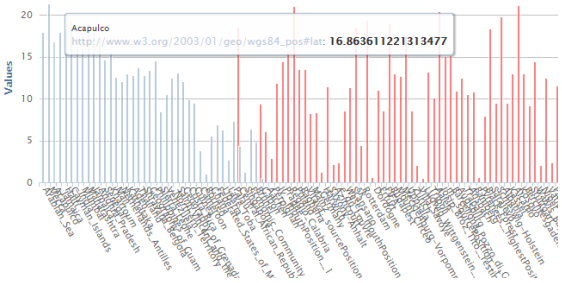
Initially, users are able to select a dataset from a number of offered real-word LD datasets (e.g., DBpedia, Eurostat) or upload their own. Then, for the selected dataset, the users are able to examine several of the *dataset's metadata*, and explore several *datasets's statistics*.

Using the *facets panel*, users are able to navigate and *filter* data based on classes, numeric and date properties. In addition, through facets panel several information about the classes and properties (e.g., number of instances, domain(s), range(s), IRI, etc.) are provided to the users through the UI.

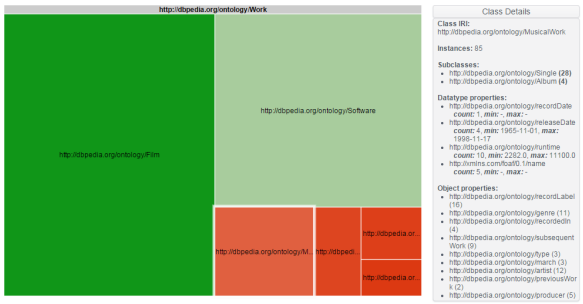
Users are able to visually explore data by considering properties' values. Particularly, *area charts* and *timeline-based area charts* are used to visualize the resources considering the user's selected properties. Classes' facets can also be used to *filter* the visualized data. Initially, the top level of the hierarchy is presented providing an *overview* of the data, organized into top-level groups; the user can interactively *drill-down* (i.e., zoom-in) and *roll-up* (i.e., zoom-out) over the group of interest, up to the actual values of the input data (i.e., LD resources). At the same time, statistical information concerning



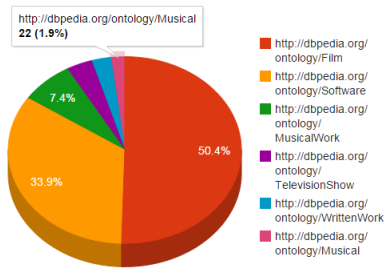
(a) Groups of numeric RDF data (Area chart)



(b) Numeric RDF data (Column chart)



(c) Class hierarchy (Treemap chart)



(d) Class hierarchy (Pie chart)

Figure 4.10: Numeric data & class hierarchy visualization examples

the hierarchy groups as well as their contents (e.g., mean value, variance, sample data, range) is presented through the UI (Figure 4.10a). Regarding the most detailed level (i.e., LD resources), several visualization types are offered; i.e., area, column, line, spline and areaspline (Figure 4.10b).

In addition, users are able to visually explore data, through class hierarchy. Selecting one or more classes, users can interactively navigate over the class hierarchy using treemaps (Figure 4.10c) or pie charts (Figure 4.10d). Properties' facets can also be used to filter the visualized data. In SynopsViz the treemap visualization has been enriched with schema and statistical information. For each class, schema metadata (e.g., number of instances, subclasses, datatype/object properties) and statistical information (e.g., the cardinality of each property, min, max value for datatype properties) are provided.

Finally, users can interactively modify the hierarchy specifications. Particularly, they are able to increase or decrease the level of abstraction/detail presented, by modifying both the number of hierarchy levels, and number of nodes per level.

A video presenting the basic functionality of our prototype is available at `youtu.be/n2ctdH5PKA0`.

4.1.3.3 Implementation

SynopsViz is implemented on top of several open source tools and libraries. The back-end of our system is developed in Java, Jena framework is used for RDF data handing and Jena TDB is used for disk-based RDF storing. The front-end prototype, is developed using HTML and Javascript. Regarding visualization libraries, we use Highcharts, for the area, column, line, spline, areaspline and timeline-based charts and Google Charts for treemap and pie charts.

4.1.4 Experimental Analysis

In this section we present the evaluation of our approach. In Section 4.1.4.1, we present the dataset and the experimental setting. Then, in Section 4.1.4.2 we present the performance

results and in Section 4.1.4.3 the user evaluation we performed.

4.1.4.1 Experimental Setting

In our evaluation, we use the well known *DBpedia* 2014 LD dataset. Particularly, we use the *Mapping-based Properties (cleaned)* dataset[14] which contains high-quality data, extracted from Wikipedia Infoboxes. This dataset contains 33.1M triples and includes a large number of numeric and temporal properties of varying sizes. The largest numeric property in this dataset has 534K triples, whereas the largest temporal property has 762K.

Regarding the methods used in our evaluation, we consider our HETree hierarchical approaches, as well as a simple non-hierarchical visualization approach, referred as *FLAT*. FLAT is considered as a competitive method against our hierarchical approaches. It provides one level visualizations, rendering only the actual data objects; i.e., it is the same as the visualization provided by SynopsViz at the most detailed level. In more detail, FLAT approach corresponds to a column chart in which the resources are sorted in ascending order based on their object values, the horizontal axis contains the resources' names (i.e., triples' subjects), and the vertical axis corresponds to objects' values. By hovering over a resource, a tooltip appears including the resource's name and object value.

Regarding the HETree approaches, the tree parameters (i.e., number of leaves, degree and height) are automatically computed following the approach described in Section 4.1.1.5. In our experiments, the lower and the upper bound for the objects rendered at the most detailed level have been set to λ_{min} = 10 and λ_{max} = 50, respectively. Considering the visualizations provided by the default Highcharts settings, these numbers are reasonable for our screen size and resolution.

Finally, our backend system is hosted on a server with a quad-core CPU at 2GHz and 8GB of RAM running Windows Server 2008. As client, we used a laptop with i5 CPU at 2.5GHz with 4G RAM, running Windows 7, Firefox 38.0.1 and ADSL2+ internet connection. Additionally, in the user evaluation, the client is employed with a 24" (1920×1200) screen.

4.1.4.2 Performance Evaluation

In this section, we study the performance of the proposed model, as well as the behaviour of our tool, in terms of construction and response time, respectively. Section 4.1.4.2.1 describes the setting of our performance evaluation, and Section 4.1.4.2.2 presents the evaluation results.

4.1.4.2.1 Setup

In order to study the performance, a number of numeric and temporal properties from the employed dataset are visualized using the two hierarchical (i.e., HETree-C/R) and the FLAT approach. We select one set from each type of properties; each set contains 15 properties with varying sizes, starting from small properties having 50-100 triples up to the largest properties.

In our experiment, for each of the three approaches, we measure the tool response time. Additionally, for the two hierarchical approaches we also measure the time required for the HETree construction.

Note that in hierarchical approaches through user interaction, the server sends to the browser only the data required for rendering the current visualization level (although the whole tree is constructed at the backend). Hence, when a user requests to generate a visualization we have the following workflow. Initially, our system constructs the tree;

[14] downloads.dbpedia.org/2014/en/mappingbased_properties_ cleaned_en.nt.bz2

then, the data regarding the top-level groups (i.e., root node children) are sent to the browser which renders the result. Afterwards, based on user interactions (i.e., drill-down, roll-up), the server retrieves the required data from the tree and sends it to the browser. Thus, the tree is constructed the first time a visualization is requested for the given input dataset; for any further user navigation over the hierarchy, the response time does not include the construction time. Therefore, in our experiments, in hierarchical approaches as response time we measure the time required by our tool to provide the first response (i.e., render the top-level groups), which corresponds to the slower response in our visual exploration scenario. Thus, we consider the following measures in our experiments:

Construction Time: the time required to build the HETree structure. This time includes (1) the time for sorting the triples; (2) the time for building the tree; and (3) the time for the statistics computations.

Response Time: the time required to render the charts, starting from the time the client sends the request. This time includes (1) the time required by the server to compute and build the response. In hierarchical approaches, this time corresponds to the *Construction Time*, plus the time required by the server to build the JSON object sent to the client. In FLAT approach, it corresponds to the time spent in sorting the triples plus the time for the JSON construction; (2) the time spent in the client-sever communication; and (3) the time required by the visualization library to render the charts on the browser.

4.1.4.2.2 Results

Table 4.4 presents the evaluation results regarding the numeric (upper half) and the temporal properties (lower half). The properties are sorted in ascending order of the number of triples. For each property, the table contains the number of triples, the characteristics of the constructed HETree structures (i.e., number of leaves, degree, height, and number of nodes), as well as the construction and the response time for each approach. The presented time measurements are the average values from 50 executions.

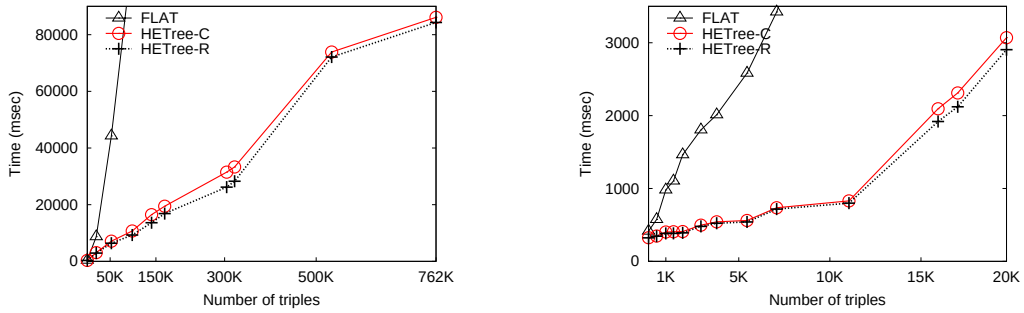
Regarding the comparison between the HETree and the FLAT, the FLAT approach can not provide results for properties having more than 305K triples, indicated in the last rows for both numeric and temporal properties with "–" in the FLAT response time. For the rest properties, we can observe that the HETree approaches clearly outperform the FLAT in all cases, even in the smallest property (i.e., *rankingWin*, 50 triples). As the size of properties increases, the difference between the HETree approaches and the FLAT increases, too. In more detail, for properties having more than 1.400 triples (i.e., the numeric properties larger than the *hsvCoordinateValue* -5th row-, and the temporal properties larger than the *lastAirDate* -4th row-), the HETree approaches outperform the FLAT by about one order of magnitude. Finally, for the largest property that FLAT can handle (i.e., *populationTotal*, 305K triples), the difference between the HETree and the FLAT is about two order of magnitude.

Regarding the time required for the construction of the HETree structure, from Table 4.4 we can observe the following. The performance of both HETtree structures is very close for most of the examined properties, with the HETree-R performing slightly better than the HETree-C (especially in the relatively small numeric properties). Furthermore, we can observe that the response time follows a similar trend as the construction time. This is expected since the communication cost, as well as the times required for constructing and rendering the JSON object are almost the same for all cases. Particularly, in our HETree setting, the Highchart requires approximately 90 msec for rendering the charts in the browser.

Regarding the comparison between the construction and the response time in the HETree approaches, from Table 4.4 we can observe the following. For properties having up to 5.5K triples (i.e., the numeric properties smaller than the *width* -8th row-, and the

Table 4.4: Performance results for numeric & temporal properties

Property (#Triples)	Tree Characteristics				HETree-C		HETree-R		FLAT
	#Leaves	Degree	Height	#Nodes	Construction Time (msec)	Response Time (msec)	Construction Time (msec)	Response Time (msec)	Response Time (msec)
Numeric Properties									
rankingWins (50)	9	3	2	13	5	324	1	323	415
distanceToBelfast (104)	9	3	2	13	7	337	4	329	419
waistSize (241)	16	4	2	21	10	346	9	336	440
fileSize (492)	27	3	3	40	18	347	16	345	575
hsvCoordinateValue (995)	81	3	4	121	74	403	50	383	980
lineLength (1,923)	81	3	4	121	77	409	55	391	1,463
powerOutput (5,453)	243	3	5	364	234	560	217	540	2,583
width (11,049)	729	3	6	1,093	506	830	467	799	6,135
numberOfPages (21,743)	729	3	6	1,093	2,888	3,219	2,403	2,722	12,669
inseeCode (36,780)	2,187	3	7	3,280	4,632	4,962	4,105	4,436	19,119
areaWater (40,564)	2,187	3	7	3,280	4,945	5,134	5,274	5,457	29,538
populationDensity (52,572)	2,187	3	7	3,280	6,803	7,127	6,080	6,404	44,262
areaTotal (140,408)	6,561	3	8	9,841	16,158	16,482	13,298	13,627	219,018
populationTotal (304,522)	19,683	3	9	29,524	31,141	31,473	25,866	26,196	1,523,675
lat (533,900)	19,683	3	9	29,524	73,528	73,862	71,784	72,106	—
Temporal Properties									
retired (155)	9	3	2	13	8	330	4	327	425
endDate (341)	27	3	3	40	17	339	16	339	468
lastAirDate (704)	64	4	3	85	34	359	30	359	853
buildingStartDate (1,415)	81	3	4	121	73	406	53	384	1,103
latestReleaseDate (2,925)	243	3	5	364	162	496	146	480	1,804
orderDate (3,788)	243	3	5	364	210	542	195	523	2,011
decommissioningDate (7,082)	243	3	5	364	405	735	383	717	3,423
shipLaunch (15,938)	729	3	6	1,093	1,772	2,094	1,595	1,919	6,935
completionDate (17,017)	729	3	6	1,093	1,987	2,311	1,793	2,121	7,814
foundingDate (19,694)	729	3	6	1,093	2,745	3,069	2,583	2,905	8,699
added (44,227)	2,187	3	7	3,280	5,912	5,943	6,244	6,265	33846
activeYearsStartDate (98,160)	6,561	3	8	9,841	10,368	10,702	8,952	9,282	107,587
releaseDate (169,156)	6,561	3	8	9,841	19,122	19,451	16,526	16,856	950,545
deathDate (321,883)	19,683	3	9	29,524	32,990	33,313	27,936	28,271	—
birthDate (761,830)	59,049	3	10	88,573	85,797	86,120	83,982	84,314	—



(a) Properties sizes: 50 to 762K triples (b) Properties sizes: 50 to 20K triples

Figure 4.11: Response Time w.r.t. the number of triples

temporal properties smaller than the *decommissioningDate* -7th row-), the response time is dominated by the communication cost, and the time required for the JSON construction and rendering. For properties with only a small number of triples (i.e., *waistSize*, 241 triples), only the 1.5% of the response time is spent on constructing the HETree. Moreover, for a property with a larger number of triples (i.e., *buildingStartData*, 1.415 triples), 18% of the time is spent on constructing the HETree. Finally, for the largest property for which the construction time is dominated by the other costs (i.e., *powerOutput*, 5.453 triples), 42% of the time is spent on constructing the HETree.

Figure 4.11 summarizes the results from Table 4.4, presenting the response time for all approaches w.r.t. the number of triples. Particularly, Figure 4.11a includes all properties sizes, whereas Figure 4.11b focus on the properties having up to 20K triples. We observe also here that the HETree-R performs slightly better than the HETree-C. Additionally, from Figure 4.11b we can indicate that for up to 10K triples the performance of the HETree approaches is almost the same. We can also observe the significant difference between the FLAT and the HETree approaches. Finally, an important observation is that, in practice both the construction and the response time, and thus the overall performance of our tool, grow sub-linearly to the number of triples.

Overall, our hierarchical approaches exhibit linear time performance w.r.t. number of input objects. In addition, it is shown that the hierarchical approaches clearly outperform the employed non-hierarchical approach.

Although our method clearly outperforms the non-hierarchical method, as we can observe from the above results, the construction of the whole hierarchy can not provide an efficient solution for datasets containing more than 10K objects. As discussed in Section 4.1.2.2, for efficient exploration over large datasets an incremental hierarchy construction is required. In incremental exploration scenario, the number of hierarchy nodes that have to be processed and constructed is significant fewer compared to the non-incremental. For example, adopting an non-incremental construction in *populationTotal* (305K triples), 29.6K nodes are to be initially constructed (along with their statistics), while the incremental approach constructions in the worst case, 15 nodes.

4.1.4.3 User Study

In this section we present the user evaluation of our tool, where we have employed three approaches: the two hierarchical and the FLAT. Section 4.1.4.3.1 describes the user tasks, Section 4.1.4.3.2 outlines the evaluation procedure and setup, Section 4.1.4.3.3 summarizes the evaluation results, and Section 4.1.4.3.4 discusses issues related to the evaluation process.

4.1.4.3.1 Tasks

In this section we describe the different types of tasks that are used in the user evaluation process.

Type 1 [Find resources with specific value]: This type of tasks requests the resources having value v (as object). For this task type, we define task T1 by selecting a value v that corresponds to 5 resources. Given this task, the participants are asked to provide the number of resources that pertain to this value. In order to solve this task, the participants first have to find a resource with value v and then check which of the nearby resources also have the same value.

Type 2 [Find resources in a range of values]: This type of tasks requests the resources having value greater than v_{min} and less than v_{max}. We define two tasks of this type, by selecting different combinations of v_{min} and v_{max} values, such that tasks which consider different number of resources are defined. Particularly, in the first task, named T2.1, we specify the values v_{min} and v_{max} such that a small set of (approximately 10) resources are included, whereas the second task, T2.2, considers a larger set of (approximately 50) resources. Given these tasks, the participants are asked to provide the number of resources included in the given range. This task can be solved by first finding a resource with a value included in the given range, and then explore the nearby resources in order to identify the resources in the given range.

Type 3 [Compare distributions]: This type of tasks requests from the participant to identify whether more resources appear above or below a given value v. For this type, we define task T3, by selecting the value v near to median. Given this task, the participants are asked to provide the number of resources appearing either above or below the value v. The answer for this tasks requires from the participants to indicate the value v and determine the number or resources appearing either before or after this value.

4.1.4.3.2 Setup

In order to study the effect of the property size in the selected tasks, we have selected two properties of different sizes from the employed dataset (Section 4.1.4.1). The *hsv-CoordinateHue* numeric property containing 970 triples, is referred as *Small*, and the *maximumElevation* numeric property, containing 37.936 triples, is referred as *Large*. The first one corresponds to a hierarchy of height 4 and degree 3, and the latter corresponds to a hierarchy of height 7 and degree 3. We should note here, that through the user evaluation, the hierarchy parameters were fixed for all the tasks, and the participants were not allowed to modify them, such that the setting has been the same for everyone.

In our evaluation, 10 participants took part. The participants were computer science graduate students and researchers. At the beginning of the evaluation, each participant has introduced to the system by an instructor who provided a brief tutorial over the features required for the tasks. After the instructions, the participants familiarized themselves with the system. Note that we have integrated in the SynopsViz the FLAT approach along with the HETree approaches.

During the evaluation, each participant performed the previously described four tasks, using all approaches (i.e., HETree-C/R and FLAT), over both the small and large properties. In order to reduce the learning effects and fatigue we defined three groups. In the first group, the participants start their tasks with the HETree-C approach, in the second with HETree-R, and in the third with FLAT. Finally, the property (i.e., small, large) first used in each task was counterbalanced among the participants and the tasks. The entire evaluation did not exceed 75 minutes.

Furthermore, for each task (e.g., T2.1, T.3), three task instances were specified by slightly modifying the task parameters. As a result, given a task, a participant has to solve a different instance of this task, in each approach.

For example, in task T2.1, for the HETree-R, the selected v corresponds to a solution of 11 resources, in HETree-C, to 9 resources, whereas for FLAT v corresponded to a solution of 8 resources. The task instance assigned to each approach varied among the participants.

During the evaluation the instructor measured the time required for each participant to complete a task, as well as the number of incorrect answers. Table 4.5 presents the average time required for the participants to complete each task. The table contains the measurements for all approaches, and for both properties. Although we acknowledge that the number of participants in our evaluation is small, we have computed the statistical significance of the results. Essentially, for each property, the p-value of each task is presented in the last column. The p-value is computed using one-way repeated measures ANOVA.

In addition, the results regarding the number of tasks that were not correctly answered are presented in Table 4.6. Particularly, the table presents the percentage of incorrect answers for each task and property, referred as *error rate*. Additionally, for each task and property, the table includes the p-value. Here, the p-value has been computed using *Fisher's exact test*.

4.1.4.3.3 Results

Task T1. Regarding the first task, as we can observe from Table 4.5, HETree approaches outperform the FLAT, in both property sizes. Note that the time results on T1 are statistical significant ($p < 0.01$).

As expected, all approaches require more time for the Large property compared to the Small one. This overhead in FLAT is caused by from the larger number of resources that the participants have to scroll over and examine, until they indicate the requested resource's value. On the other hand, in HETree, the overhead is caused by the larger number of levels that the Large property hierarchy has. Hence, the participants have to perform more drill-down operations and examine more groups of objects, until they reach the LD resources.

We can also observe that in this task, the HETree-R outperforms the HETree-C in both property sizes. This is due to the fact that, in HETree-R structure, resources having the same value are always contained in the same leaf. As a result, the participants had to inspect only one leaf. On the other hand, in HETree-C this does not always hold, hence the participants could have explored more than one leaf.

Finally, as we can observe from Table 4.6, in all cases only correct answers have been provided. However, none of those results are statistically significant ($p > 0.05$).

Task T2.1. In the next task, where the participants had to indicate a small set of resources in a range of values, the FLAT performance is very close to the HETree, especially in the Small property (Table 4.5). In addition, we can observe that the HETree-C approach performs slightly better than the HETree-R. Finally, regarding the statistical significance of the results, in Small property we have $p > 0.05$, while in Large we have $p < 0.005$.

The poor performance of the HETree approaches in this task can be explained by the small set of resources requested and the HETree parameters adopted in the user evaluation. In this setting, the resources contained in the task solution are distributed over more than one leaves. Hence, the participants had to perform several roll-up and drill-down operations in order to find all the resources. On the other hand, in FLAT, once the participants had indicated one of the requested resources, it was very easy for them to find out the rest of the solution's resources. To sum up, in FLAT, most of the

Table 4.5: Average task completion time (sec)

	Small Property				Large Property			
	FLAT	HETree-C	HETree-R	p	FLAT	HETree-C	HETree-R	p
T1	54	29	28	★★	85	52	47	★★
T2.1	63	57	64	◆	74	60	69	★
T2.2	120	69	74	★★	128	72	77	★★
T3	262	41	40	★★	—	64	62	—

★★ ($p < 0.01$) ★ ($p < 0.05$) ◆ ($p > 0.05$)

Table 4.6: Error rate (%)

	Small Property				Large Property			
	FLAT	HETree-C	HETree-R	p	FLAT	HETree-C	HETree-R	p
T1	0	0	0	◆	0	0	0	◆
T2.1	0	0	0	◆	0	0	0	◆
T2.2	20	0	0	◆	20	0	10	◆
T3	70	0	0	★★	—	0	0	—

★★ ($p < 0.01$) ★ ($p < 0.05$) ◆ ($p > 0.05$)

time is spent on identifying the first of the resources, while in HETree the first resource is identified very quickly. Regarding the difference in performance between the HETree approaches we have the following. In HETree-C due to the fixed number of objects in each leaf, the participants had to visit at most one or two leaves in order to solve this task. On the other hand, in HETree-R, the number of objects in each leaf is varied, so in most times the participants had to inspect more than two leaves in order to solve the task. Finally, again, for this task, only correct answers were given (Table 4.6).

Task T2.2. In this task the participants had to indicate a larger set (compared to the previous task) of resources given a range of values. HETree approaches noticeable outperform the FLAT approach with statistical significance ($p < 0.01$), while similar results are observed in both properties.

In FLAT approach a considerable time was spent to identify and navigate over a large number of resources. On the other hand, due to the large number of resources involved in the task's solution, there are groups in the hierarchy that explicitly contain solution's resources (i.e., they do not contain resources not included in the solution). As a result, the participants in HETree could easily indicate and compute the whole solution by combining the information related to the groups (i.e., number of enclosed resources) and individual resources. Due to the same reasons stated in the previous task (i.e., T2.1), similarly in T2.2 the HETree-C performs slightly better than the HETree-R. Finally, we can observe from Table 4.6 (but without statistical significance), that it was more difficult for participants to solve correctly this task with FLAT than with HETree.

Task T3. In the last task, participants were requested to find which of the two ranges contained more resources. As expected, from Table 4.5 shows that the HETree approaches clearly outperform the FLAT approach with statistical significance in the Small property. This is due to the fact that the participants in FLAT approach had to overview and navigate over almost half of the dataset. As a result, apart from the long time required for this process, it was also very difficult to find the correct solution. This is also verified by Table 4.6 on a statistically significant level. On the other hand, in HETree approaches, the participants could easily find out the answer by considering the resources enclosed by several groups.

Regarding the Large property, as it is expected, it was impossible for participants to

solve this task with FLAT, since this required to parse over and count about 19K resources. As a result, none of the participants completed this task using FLAT (indicated with "–" in Table 4.5), considering the 5 minute time limit used in this task.

4.1.4.3.4 Discussion

The user evaluation showed that the hierarchical approaches can be efficient (i.e., require short time in solving tasks) and effective (i.e., have lower error rate) in several cases. In more detail, the HETree approaches performed very well on indicating specific values over a dataset, and given the appropriate parameter setting are marginally affected by the dataset size. Also note that, due to the "vertical-based" exploration, the position (e.g., towards the end) of the requested value in the dataset does not affect the efficiency of the approach. Furthermore, it is shown that the hierarchical approaches can efficiently and effectively handle visual exploration tasks that involve large numbers of objects.

At the end of the evaluation, the participants gave us valuable feedbacks on possible improvements of our tool. Most of the participants criticized several aspects in the interface; since our tool is an early prototype. Also, several participants mentioned difficulties in obtaining their "position" (e.g., which is the currently visualized range of values, or the previously visualized range of values) during the exploration. Finally, some participants mentioned that some hierarchies contained more levels than needed. As previously mentioned, the adopted parameters are not well suited for the evaluation, since hierarchies with larger than 3 degree (and as result less levels) are required.

Finally, additional tasks for demonstrating the capabilities of our model can be considered. However, most of these tasks were not selected in this evaluation, because it was not possible for the participants to perform them with the FLAT approach. An indicative set includes: (1) Find the number of resources (and/or statistics) in the 1st and 3rd quartile; (2) Find statistics (e.g., mean value, variance) for the top-10 or 50 resources; (3) Find the decade (i.e., temporal data) in which most events take place; etc.

4.1.5 Related Work

This section reviews works related to our approach on visualization and exploration in the Web of Data (WoD). Section 4.1.5.1 presents systems and techniques for visualization and exploration WoD, Section 4.1.5.2 discusses techniques on WoD statistical analysis, Section 4.1.5.3 present hierarchical data visualization techniques, and finally, Section 4.1.5.4 discusses works on data structures & processing related to our HETree data structure.

In Table 4.7 we provide an overview and compare several visualization systems that offer similar features to our SynopsViz system. The *WoD* column indicates systems that target the Semantic Web and Linked Data area (i.e., RDF, RDF/S, OWL). The *Hierarchical* column indicates systems that provide hierarchical visualization of non-hierarchical data. The *Statistics* column captures the provision of statistics about the visualized data. The *Recomm.* column indicates systems, which offer recommendation mechanisms for visualization settings (e.g., appropriate visualization type, visualization parameters, etc.). The *Incr.* column indicate systems that provide incremental visualizations. Finally, the *Preferences* column captures the ability of the users to apply data (e.g., aggregate) or visual (e.g., increase abstraction) operations.

4.1.5.1 Exploration & Visualization Systems

A large number of works studying issues related to WoD visual exploration and analysis have been proposed in the literature [140, 282, 29]. In what follows, we classify these works into the following categories: (1) Browsers and exploratory systems, (2) Generic

Table 4.7: Visualization systems overview

System	WoD	Hierarchical	Data Types*	Vis. Types**	Statistics	Recomm.	Incr.	Preferences	Domain	App. Type
Rhizomer [103]	✓		N, T, S, H, G	C, M, T, TL		✓			generic	Web
Payola [242]	✓		N, T, S, H, G	C, CI, G, M, T, TL, TR					generic	Web
LDVM [102]	✓		S, H, G	B, M, T, TR		✓			generic	Web
Vis Wizard [374]	✓		N, T, S	B, C, M, P, PC, SG		✓		✓	generic	Web
LDVizWiz [43]	✓		S, H, G	M, P, TR		✓			generic	Web
LinkDaViz [368]	✓		N, T, S	B, C, S, M, P		✓		✓	generic	Web
VizBoard [387]	✓		N, H	C,S, T		✓		✓	generic	Web
SemLens [204]	✓		N	S				✓	generic	Web
LODeX [62]	✓		G	G, M, P	✓				generic	Web
LODWheel [358]	✓		N, S, G	C, G, M, P					generic	Web
RelFinder [203]	✓		G	G				✓	generic	Web
Fenfire [201]	✓		G	G					generic	Desktop
Lodlive [106]	✓		G	G				✓	generic	Web
IsaViz [308]	✓		G	G				✓	generic	Desktop
graphVizdb [76]	✓		G	G				✓	generic	Web
ViCoMap [321]	✓		N, T, S	M	✓	✓		✓	generic	Web
EDT [280]			N, T, H	C, CM, T, SP	✓				OLAP	Desktop
Polaris [356]		✓	N, T, S, H	C, M, S	✓		✓	✓	OLAP	Desktop
XmdvTool [392, 176]		✓	N	DS, PC, S, ST					generic	Desktop
GrouseFlocks [37]		✓	G	G			✓	✓	generic	Desktop
GMine [221]		✓	G	G					generic	Desktop
Gephi [54]		✓	G	G	✓			✓	generic	Desktop
CGV [373]		✓	G	G			✓	✓	generic	Desktop
SynopsViz	✓	✓	N, T, H	C, P, T, TL [1]	✓	✓	✓	✓	generic	Web

* N: Numeric, T: Temporal, S: Spatial, H: Hierarchical (tree), G: Graph (network)

** B: *bubble chart*, C: *chart*, CI: *circles*, CM: *colormap*, DS: *dimensional stacking*, G: *graph*, M: *map*, P: *pie*, PC: *parallel coordinates*, S: *scatter*, SG: *streamgraph*, SP: *solarplot*, ST: *star glyphs*, T: *treemap*, TL: *timeline*, TR: *tree*

[1] The HETree model is not restricted to these visualization types.

visualization systems, (3) Domain, vocabulary & device-specific visualization systems, (4) Graph-based visualization systems, (5) Ontology visualization systems, and (6) Visualization libraries.

4.1.5.1.1 Browsers & Exploratory Systems

WoD browsers have been the first systems developed for WoD utilization and analysis [140, 29]. Similarly to the traditional ones, WoD browsers provide the functionality for link navigation and usable representation of WoD resources and their properties; thus enabling browsing and exploration of WoD in a most intuitive way. WoD browsers mainly use tabular views and links to provide navigation over the WoD resources.

Haystack [315] is one of the first WoD browsers, it exploits stylesheets in order to customize the data presentation. Similarly, *Disco*[15] renders all information related to a particular RDF resource as HTML table with property-value pairs. *Noadster* [327] performs property-based data clustering in order to structure the results. *Piggy Bank* [213] is a Web browser plug-in, that allows users to convert HTML content into RDF. *LESS* [47] allows users to create their own Web-based templates in order to aggregate and display WoD. *Tabulator* [68] is another WoD browser, additionally provides maps and timeline visualizations. *Explorator* [35] is an WoD exploratory system that allows users to browse a dataset by combining search and facets. *VisiNav* [199] is a system that allows users to pose expressive exploratory-based queries. The system is built on top of following concepts: keyword search, object focus, path traversal, and facet selection. *Information Workbench* (IWB) [193] is a generic platform for semantic data management offering several back-end (e.g., triple store) and front-end systems. Regarding the front-end, IWB offers a flexible user interface for data exploration and visualization. *Marbles*[16] formats RDF triples using the Fresnel vocabulary (a vocabulary for rendering RDF resources as HTML). Also, it retrieves information about a resource by accessing Semantic Web indexes and search engines. Finally, *URI Burner*[17] is a service which retrieves data about resources. For the requested resources, it generates an RDF graph by exploiting existing ontologies and other knowledge from the Web.

4.1.5.1.2 Generic Visualization Systems

In the context of WoD visual exploration, there is a large number of generic visualization frameworks, that offer a wide range of visualization types and operations. Next, we outline the best known systems in this category.

Rhizomer [103] provides WoD exploration based on a overview, zoom and filter workflow. Rhizomer offers various types of visualizations such as maps, timelines, treemaps and charts. *VizBoard* [387, 388] is an information visualization workbench for WoD build on top of a mashup platform. VizBoard presents datasets in a dashboard-like, composite, and interactive visualization. Additionally, the system provides visualization recommendations. *Payola* [242] is a generic framework for WoD visualization and analysis. The framework offers a variety of domain-specific (e.g., public procurement) analysis plugins (i.e., analyzers), as well as several visualization techniques (e.g., graphs, tables, etc.). In addition, Payola offers collaborative features for users to create and share analyzers. In Payola the visualizations can be customized according to ontologies used in the resulting data.

The *Linked Data Visualization Model* (LDVM) [102] provides an abstract visualization process for WoD datasets. LDVM enables the connection of different datasets with various

[15] www4.wiwiss.fu-berlin.de/bizer/ng4j/disco
[16] mes.github.io/marbles
[17] linkeddata.uriburner.com

kinds of visualizations in a dynamic way. The visualization process follows a four stage workflow: Source data, Analytical abstraction, Visualization abstraction, and View. A prototype based on LDVM considers several visualization techniques, e.g., circle, sunburst, treemap, etc. Finally, the LDVM has been adopted in several use cases [243]. *Vis Wizard* [374] is a Web-based visualization system, which exploits data semantics to simplify the process of setting up visualizations. Vis Wizard is able to analyse multiple datasets using brushing and linking methods. Similarly, *Linked Data Visualization Wizard* (LDVizWiz) [43] provides a semi-automatic way for the production of possible visualization for WoD datasets. In a same context, *LinkDaViz* [368] finds the suitable visualizations for a give part of a dataset. The framework uses heuristic data analysis and a visualization model in order to facilitate automatic binding between data and visualization options.

Balloon Synopsis [335] provides a WoD visualizer based on HTML and JavaScript. It adopts a node-centric visualization approach in a tile design. Additionally, it supports automatic information enhancement of the local RDF data by accessing either remote SPARQL endpoints or performing federated queries over endpoints using the Balloon Fusion service. Balloon Synopsis offers customizable filters, namely ontology templates, for the users to handle and transform (e.g., filter, merge) input data. *SemLens* [204] is a visual system that combines scatter plots and semantic lenses, offering visual discovery of correlations and patterns in data. Objects are arranged in a scatter plot and are analysed using user-defined semantic lenses. *LODeX* [62] is a system that generates a representative summary of a WoD source. The system takes as input a SPARQL endpoint and generates a visual (graph-based) summary of the WoD source, accompanied by statistical and structural information of the source. *LODWheel* [358] is a Web-based visualizing system which combines JavaScript libraries (e.g., MooWheel, JQPlot) in order to visualize RDF data in charts and graphs. *Hide the stack* [141] proposes an approach for visualizing WoD for mainstream end-users. Underlying Semantic Web technologies (e.g., RDF, SPARQL) are utilized, but are "hidden" from the end-users. Particularly, a template-based visualization approach is adopted, where the information for each resource is presented based on its rdf:type.

4.1.5.1.3 Domain, Vocabulary & Device-specific Visualization Systems

In this section, we present systems that target visualization needs for specific types of data and domains, RDF vocabularies or devices.

Several systems focus on visualizing and exploring geo-spatial data. *Map4rdf* [260] is a faceted browsing tool that enables RDF datasets to be visualized on an OSM or Google Map. *Facete* [352] is an exploration and visualization tool for SPARQL accessible data, offering faceted filtering functionalities. *SexTant* [65] and *Spacetime* [382] focus on visualizing and exploring time-evolving geo-spatial data. The *LinkedGeoData Browser* [351] is a faceted browser and editor which is developed in the context of LinkedGeoData project. Finally, in the same context *DBpedia Atlas* [381] offers exploration over the DBpedia dataset by exploiting the dataset's spatial data. Furthermore, in the context of linked university data, *VISUalization Playground* (VISU) [32] is an interactive tool for specifying and creating visualizations using the contents of linked university data cloud. Particularly, VISU offers a novel SPARQL interface for creating data visualizations. Query results from selected SPARQL endpoints are visualized with Google Charts.

A variety of tools target multidimensional WoD modelled with the Data Cube vocabulary. *CubeViz* [164, 331] is a faceted browser for exploring statistical data. The tool provides data visualizations using different types of charts (i.e., line, bar, column, area and pie). The *Payola Data Cube Vocabulary* [205] adopts the LDVM stages [102] in order to visualize RDF data described by the Data Cube vocabulary. The same types of charts as in CubeViz are provided in this tool. The *OpenCube Toolkit* [225] offers several tools

related to statistical WoD. For example, *OpenCube Browser* explores RDF data cubes by presenting a two-dimensional table. Additionally, the *OpenCube Map View* offers interactive map-based visualizations of RDF data cubes based on their geo-spatial dimension. The *Linked Data Cubes Explorer* (LDCE) [229] allows users to explore and analyse statistical datasets. Finally, [305] offers several map and chart visualizations of demographic, social and statistical linked cube data.

Regarding device-specific systems, *DBpedia Mobile* [57] is a location-aware mobile application for exploring and visualizing DBpedia resources. *Who's Who* [107] is an application for exploring and visualizing information focusing on several issues that appear in the mobile environment. For example, the application considers the usability and data processing challenges related to the small display size and limited resources of the mobile devices.

4.1.5.1.4 Graph-based Visualization Systems

A large number of systems visualize WoD datasets adopting a *graph-based* (a.k.a., node-link) approach. *RelFinder* [203] is a Web-based tool that offers interactive discovery and visualization of relationships (i.e., connections) between selected WoD resources. *Fenfire* [201] and *Lodlive* [106] are exploratory tools that allow users to browse WoD using interactive graphs. Starting from a given URI, the user can explore WoD by following the links. *IsaViz* [308] allows users to zoom and navigate over the RDF graph, and also it offers several "edit" operations (e.g., delete/add/rename nodes and edges). In the same context, *graphVizdb* [76] is built on top of spatial and database techniques offering interactive visualization over very large (RDF) graphs. *ZoomRDF* [403] employs a space-optimized visualization algorithm in order to increase the number of resources which are can displayed. *Trisolda* [150] proposes a hierarchical RDF graph visualization. It adopts clustering techniques in order to merge graph nodes. More details regarding hierarchical graph visualization can be found in Section 4.1.5.3. *Paged Graph Visualization* (PGV) [146] utilizes a Ferris-Wheel approach to display nodes with high degree. *RDF graph visualizer* [333] adopts a node-centric approach to visualize RDF graphs. Rather than trying to visualize the whole graph, nodes of interest (i.e., staring nodes) are discovered by searching over nodes labels; then the user can interactively navigate over the graph. Finally, *RDF-Gravity*[18] visualizes RDF and OWL data. It offers filtering, keyword search and editing the the graph layout. Also, the nodes can be displayed in different colors and shapes based on their RDF types.

4.1.5.1.5 Ontology Visualization Systems

The problems of *ontology visualization and exploration* have been extensively studied in several research areas (e.g., biology, chemistry). In what follows we focus on graph-based ontology visualization systems that have been developed in the WoD context [175, 156, 192, 255, 234]. In most systems, ontologies are visualized following the node-link paradigm [271, 270, 210, 295, 95, 169, 212, 262, 30, 250, 357] [19,20]. On the other hand, *CropCircles* [391] uses a geometric containment approach, representing the class hierarchy as a set of concentric circles. Furthermore, hybrids approaches are adopted in other works. *Knoocks* [247] combines containment-based and node-link approaches. In this work, ontologies are visualized as nested blocks where each block is depicted as a rectangle containing a sub-branch shown as tree map. Finally, *OntoTrix* [49] and *NodeTrix* [206] use node-link and adjacency matrix representations.

[18] semweb.salzburgresearch.at/apps/rdf-gravity
[19] protegewiki.stanford.edu/wiki/OntoGraf
[20] protegewiki.stanford.edu/wiki/OWLViz

4.1.5.1.6 Visualization Libraries

Finally, there is a variety of Javascript libraries which allow WoD visualizations to be embedded in Web pages. *Sgvizler* [347] is a JavaScript wrapper for visualizing SPARQL results. Sgvizler allows users to specify SPARQL Select queries directly into HTML elements. Sgvizler uses Google Charts to generate the output, offering numerous visualizations types such as charts, treemaps, graphs, timelines, etc. *Visualbox* [189] provides an environment where users can build and debug SPARQL queries in order to retrieve WoD; then, a set of visualization templates is provided to visualize results. Visualbox uses several visualization libraries like Google Charts and D3 [99], offering 14 visualization types.

4.1.5.1.7 Discussion

In contrast to the aforementioned approaches, our work does not focus solely on proposing techniques for WoD visualization. Instead, we introduce a generic model for organizing, exploring and analysing numeric and temporal data in a multilevel fashion. The underlying model is not bound to any specific type of visualization (e.g., chart); rather it can be adopted by several "flat" techniques and offer multilevel visualizations over non-hierarchical data. Also, we present a prototype system that employs the introduced hierarchical model and offers efficient multilevel visual exploration over WoD datasets, using charts and timelines.

4.1.5.2 Statistical Analysis in the Web of Data

A second area related to the analysis features of the proposed model deals with WoD statistical analysis. *RDFStats* [254] calculates statistical information about RDF datasets. *LODstats* [46] is an extensible framework, offering scalable statistical analysis of WoD datasets. *RapidMiner LOD Extension* [320, 303] is an extension of the data mining platform RapidMiner[21], offering sophisticated data analysis operations over WoD. *SparqlR*[22] is a package of the R[23] statistical analysis platform. SparqlR executes SPARQL queries over SPARQL endpoints and provides statistical analysis and visualization over SPARQL results. Finally, *ViCoMap* [321] combines WoD statistical analysis and visualization, in a Web-based tool, which offers correlation analysis and data visualization on maps.

4.1.5.2.1 Discussion

In comparison with these systems, our work does not focus on new techniques for WoD statistics computation and analysis. We are primarily interested on enhancing the visualization and user exploration functionality by providing statistical properties of the visualized datasets and objects, making use of existing computation techniques. Also, we demonstrate how in the proposed structure, computations can be efficiently performed on-the-fly and enrich our hierarchical model. The presence of statistics provides quantifiable overviews of the underlying WoD resources at each exploration step. This is particularly important in several tasks when you have to explore a large number of either numeric or temporal data objects. Users can examine next levels' characteristics at a glance, this way are not enforced to drill down in lower hierarchy levels. Finally, the statistics over the different hierarchy levels enables analysis over different granularity levels.

[21] rapidminer.com
[22] cran.r-project.org/web/packages/SPARQL/index.html
[23] www.r-project.org

4.1.5.3 Hierarchical Visual Exploration

The wider area of data and information visualization has provided a variety of approaches for hierarchical analysis and presentation.

Treemaps [342] visualize tree structures using a space-filling layout algorithm based on recursive subdivision of space. Rectangles are used to represent tree nodes, the size of each node is proportional to the cumulative size of its descendant nodes. Finally, a large number of treemaps variations have been proposed. For example, *Cushion Treemaps* [383] and *Squarified Treemaps* [101] use shade in order to provide insight into the hierarchical structure. *Ordered Treemaps* [345] ensures that items near each other in the given (i.e., input) order, will be near each other in the treemap layout. Finally, *Quantum Treemaps* [59] have been proposed for laying out images within the generated rectangles.

Moreover, hierarchical visualization techniques have been extensively employed to visualize very large graphs using the node-link paradigm. In these techniques the graph is recursively decomposed into smaller sub-graphs that form a hierarchy of abstraction layers. In most cases, the hierarchy is constructed by exploiting clustering and partitioning methods [261, 38, 36, 19, 44, 54, 221, 373]. In other works, the hierarchy is defined with hub-based [264] and density-based [407] techniques. *GrouseFlocks* [37] supports ad-hoc hierarchies which are manually defined by the users. Finally, there also some *edge bundling* techniques which join graph edges to bundles. The edges are often aggregated based on clustering techniques [177, 165, 306], a mesh [253, 134] or explicitly by a hierarchy [209].

In the context of data warehousing and *online analytical processing* (OLAP), several approaches provide hierarchical visual exploration, by exploiting the predefined hierarchies in the dimension space. [280] proposes a class of OLAP-aware hierarchical visual layouts; similarly, [367] uses OLAP-based hierarchical stacked bars. *Polaris* [356] offers visual exploratory analysis of data warehouses with rich hierarchical structure.

Further, several hierarchical techniques have been proposed in the context of ontology visualization and exploration; e.g., [391, 247] (see Section 4.1.5.1.5)

Finally, in the context of hierarchical navigation, [233] organizes query results using the MeSH concept hierarchy. In [109] a hierarchical structure is dynamically constructed to categorize numeric and categorical query results. Similarly, [120] constructs personalized hierarchies by considering diverse users preferences.

4.1.5.3.1 Discussion

In contrast to above approaches that target graph-based or hierarchically-organized data, our work focuses on handling arbitrary numeric and temporal data, with out requiring it to be described by an hierarchical schema. As an example of hierarchically-organized data, consider class hierarchies or multidimensional data organized in multilevel hierarchical dimensions (e.g., in OLAP context, temporal data is hierarchically organized based on years, months, etc.). In contrast to aforementioned approaches, our work dynamically constructs the hierarchies from raw numeric and temporal data. Thus the proposed model can be combined with "flat" visualization techniques (e.g., chart, timeline), in order to provide multilevel visualizations over non-hierarchical data. In that sense, our approach can be considered more flexible compared to the techniques that rely on predefined hierarchies, as it can enable exploratory functionality on dynamically retrieved datasets, by (incrementally) constructing hierarchies on-the-fly, and allowing users to modify these hierarchies.

4.1.5.4 Data Structures & Data Processing

In this section we present the data structures and the data (pre-)processing techniques which are the most relevant to our approach.

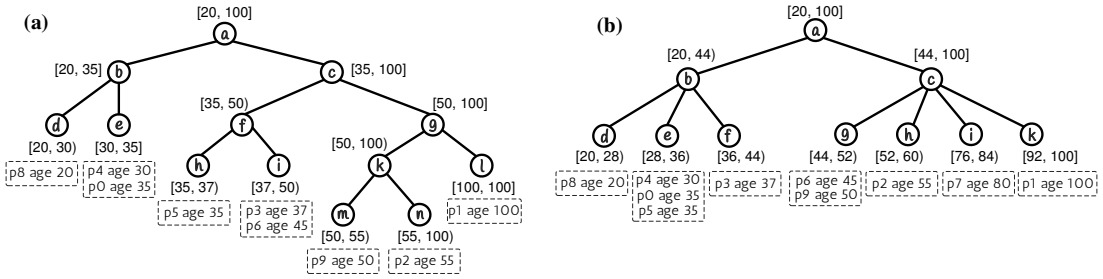


Figure 4.12: Hierarchies generated from different approaches. a) based on [126] b) based on [196]

R-Tree [191] is disk-based multi-dimensional indexing structure, which has been widely used in order to efficiently handle spatial queries. R-Tree adopts the notion of minimum bounding rectangles (MBRs) in order to hierarchical organize multi-dimensional objects.

Data discretization [181, 152] is a process where continuous attributes are transformed into discrete. A large number of methods (e.g., supervised, unsupervised, univariate, multivariate) for data discretization have been proposed. *Binning* is a simple unsupervised discretization method in which a predefined number of bins is created. Widely known binning methods are the *equal-width* and *equal-frequency*. In equal-width approach, the range of an attribute is divided into intervals that have equal width and each interval represents a bin. In equal-frequency approach, an equal number of values are placed in each bin.

By recursively applying discretization techniques, a hierarchical discretization of attribute's values can be produced (a.k.a. *concept/generalization hierarchies*). [339] proposes a dynamic programming algorithm for generating numeric concept hierarchies. The algorithm attempts to maximize both the similarity between the objects stored in the same hierarchy's node, as well as the dissimilarity between the objects stored in different nodes. The generated hierarchy is a balanced tree where different nodes may have different number of children. [196] constructs hierarchies based on data distribution. Essentially, both the leaf and the interval nodes are created in such a way that an even distribution is achieved. The hierarchy construction considers also a threshold specifying the maximum number of distinct values enclosed by nodes in each hierarchy level. Finally, binary concept hierarchies (with degree equal to two) are generated in [126]. Starting from the whole dataset, it performs a recursive binary partitioning over the dataset's values; the recursion is terminated when the number of distinct values in the resultant partitions is less than a pre-specified threshold.

Using the data objects from our running example (Figure 4.1), Figure 4.12 shows the hierarchies generated from the aforementioned approaches. Figure 4.12(a) presents the hierarchy resulted from [126] and Figure 4.12(b) depicts the result using the method from [196]. The parameters in each method are set, so that the resulting hierarchies are as much as possible similar to our hierarchies (Figures 4.2 & 4.3) . Hence, the threshold in (a) is set to 3, and in (b) is set to 2.

4.1.5.4.1 Discussion

The basic concepts of HETree structure can be considered similar to a simplified version of a static 1D R-Tree. However, in order to provide efficient query processing in disk-based environment, R-Tree considers a large number of I/O-related issues (e.g., space coverage, nodes overlaps, fill guarantees, etc.). On the other hand, we introduce a lightweight, main memory structure that efficiently constructed on-the-fly. Also, the proposed structure

aims at organizing the data in a practical manner for a (visual) exploration scenario, rather than for disk-based indexing and querying efficiency.

Compared to discretization techniques, our tree model exhibits several similarities, namely, the HETree-C version can be considered as a hierarchical version of the equal-frequency binning, and the HETree-R of the equal-width binning. However, the goal of data organization in HETree is to enable visualization and hierarchical exploration capabilities over dynamically retrieved non-hierarchical data. Hence, compared to the binning methods we can consider the following basic differences. First, in contrast with binning methods that require from the user to specify some parameters (e.g., the number/size of the bins, the number of distinct values in each bin, etc); our approach is able to automatically estimate the hierarchy parameters and adjust the visualization results by considering the visualization environment characteristics. Second, in hierarchical approaches the user is not always allowed to specify the hierarchy characteristics (e.g., degree). For example, the hierarchies in [126] have always degree equal to two (Figure 4.12(a)), while in [196] the nodes have varying degrees (Figure 4.12(b)). On the other hand, in our approach the hierarchy characteristics can be specified precisely. In addition, when not specific hierarchy characteristics are requested, our approach generates perfect trees (Section 4.1.1.5), offering a "uniform" hierarchy structure. Third, the computational complexity in some of the hierarchical approaches (e.g., [339]) is prohibitive (i.e., at least cubic) for using them in practise; especially in settings where the hierarchies have to constructed on-the-fly. Fourth, the proposed tree structure is exploited in order to allow efficient statistics computations over different groups of data; then, the statistics are used in order to enhance the overall exploration functionality. Finally, the construction of the model is tailored to the user interaction and preferences; our model offers incremental construction considering the user interaction, as well as efficiently adaptation to the users preferences.

4.1.6 Summary

In this section we have presented HETree, a generic model that combines personalized multilevel exploration with online analysis of numeric and temporal data. Our model is built on top of a lightweight tree-based structure, which can be efficiently constructed on-the-fly for a given set of data. We have presented two variations for constructing our model: the HETree-C structure organizes input data into fixed-size groups, whereas the HETree-R structure organizes input data into fixed-range groups. In that way the users can customize the exploration experience, allowing them to organize data into different ways, by parametrizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, etc. We have also provided a way for efficiently computing statistics over the tree, as well as a method for automatically deriving from the input dataset the best-fit parameters for the construction of the model. Regarding the performance of multilevel exploration over large datasets, our model offers incremental HETree construction and efficient HETree adaptation based on user's preferences. Based on the introduced model, a Web-based prototype system, called SynopsViz, has been developed. Finally, the efficiency and the effectiveness of the presented approach are demonstrated via a thorough performance evaluation and an empirical user study.

4.2 Scalable Graph Exploration

Graph visualization is a core task in various applications such as scientific data management, social network analysis, and decision support systems. With the wide adoption of the RDF data model and the recent *Linked Open Data* initiative, graph data are almost everywhere. Visualizing these data as graphs provides the non-experts with an intuitive means to explore the content of the data, identify interesting patterns, etc. Such operations require interactive visualizations (as opposed to a static image) in which graph elements are rendered as distinct visual objects; e.g., DOM objects in a web browser. This way, the user can manipulate the graph directly from the UI, e.g., click on a node or an edge to get additional information (metadata), highlight parts of the graph, rearrange some nodes on the plane, etc. Given that graphs in many real-world scenarios are huge, the aforementioned visualizations pose significant technical challenges from a data management perspective.

First of all, the visualization must be feasible without the need to load the whole graph in main memory. These "holistic" approaches [54, 201] result in prohibitive memory requirements, and usually rely on dedicated client-server architectures which are not always affordable by enterprises, especially start-ups. Then, the visualization tool must ensure extremely low response time, even in multi-user environments built upon commodity machines with limited computational resources. Finally, the visualization must be flexible and meaningful to the user, allowing her to explore the graph in different ways and at multiple levels of detail.

State-of-the-art works in the field [19, 37, 44, 222, 264, 373, 407] tackle with the previous problems through a hierarchical visualization approach. In a nutshell, hierarchical visualizations merge parts of the graph into abstract nodes (recursively) in order to create a tree-like structure of abstraction layers. This results in a decomposition of the graph into much smaller (nested) sub-graphs which can be separately visualized and explored in a "vertical" fashion, i.e., by clicking on an abstract node to retrieve the enclosed sub-graph of the lower layer. In most cases, the hierarchy is constructed by exploiting clustering and partitioning methods [19, 44, 54, 222, 373]. In other works, the hierarchy is defined with hub-based [264] and density-based [407] techniques. [37] supports ad-hoc hierarchies which are manually defined by the users. A different approach has been adopted in [359] where sampling techniques are exploited. Finally, in the context of the Web of Data, there is a large number of tools that visualize RDF graphs [80]; however, all these tools require the whole graph to be loaded on the UI. Although the hierarchical approaches provide fancy visualizations with low memory requirements, they do not support intuitive "horizontal" exploration (e.g., for following paths in the graph). Further, with hierarchical approaches it is not easy to explore dense parts of the graph in full detail (i.e., without using an abstract representation). Finally, the applicability of hierarchical approaches is heavily based on the particular characteristics of the dataset; for example, the existence of small and coherent clusters [19, 37, 44, 222, 373] or the distribution of node degrees [264, 407]. An extended review of relate works is presented in Section 4.1.5.

In this work we introduce a generic platform for scalable multi-level visualizations that do not necessarily depend on the specific graph characteristics considered in previous works. The proposed platform can easily support various visualizations, including all ad-hoc approaches in the literature, and bases its efficiency on a novel technique for indexing and storing the graph at multiple levels of abstraction. In particular, our approach involves an offline preprocessing phase that builds the layout of the input graph by assigning coordinates to its nodes with respect to a Euclidean plane. The same offline procedure is followed for all levels of abstraction each one of which corresponds to a graph that is produced by (recursively) applying an abstraction method to the input graph. The

respective points are then indexed with a spatial data structure (i.e., R-tree) and stored in a database. This way, our system maps user operations into efficient spatial operations (i.e., window queries) in the backend. The prototype we demonstrate here is a proof of concept that interactive visualizations can be effective on commodity hardware, still, allowing the user to perform intuitive navigations on the plane (e.g., follow interesting paths in the graph) at any level of abstraction and regardless the size of the graph.

Contributions. The main contributions of this work are summarized as follows.

1. We propose a new paradigm for efficient exploration over large visualized graphs, similar to this used in maps exploration.
2. We introduce a partition-based method for visualizing very large graph.
3. We present a greedy algorithm for organizing the visualized partitions on a "global" plane.
4. We present a scheme for disk-based indexing and storing the graph at multiple levels of abstraction.
5. We outline the translation of user interactions to spatial operations.
6. We develop a web-based prototype system which support four main operations: (1) interactive navigation, (2) multilevel exploration, (3) subgraph selection and manipulation, and (4) keyword search.

4.2.1 System Architecture

The architecture of our platform, graphVizdb, is depicted in Figure 4.13. It consists of three main parts: (a) the *Client*, (b) the graphVizdb *Core* module, and (c) the *Database*. The *Client* is the frontend UI that offers several functionalities to the users, e.g., an interactive canvas, search features, multi-level exploration, etc. The *Core* module contains the *Preprocessing* module and the *Query Manager* that is responsible for the communication between the *Client* and the *Database*. The *Database* contains all data needed for the visualization along with the necessary indexes. Details for each part are provided in the following sections.

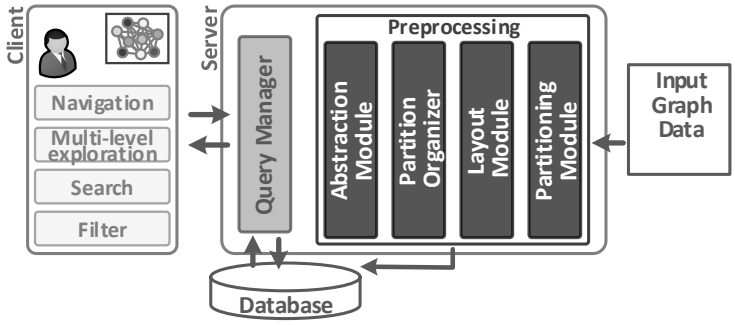


Figure 4.13: System architecture

4.2.2 Preprocessing Phase

In our approach, the layout of the input graph is built on the server side once, during the preprocessing phase, and this can be done with any of the existing layout algorithms. The result of this process is the assignment of coordinates to the nodes of the graph with respect to a Euclidean plane. The state-of-art layout algorithms provide layouts of

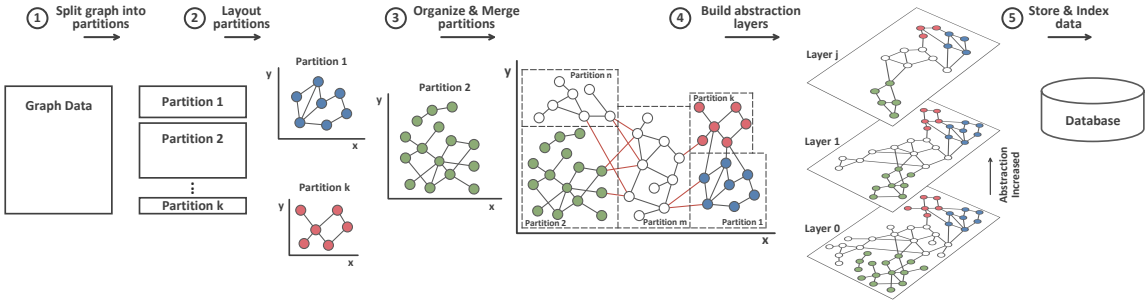


Figure 4.14: Preprocessing overview

high quality, however, they require large amounts of memory in practice, even for graphs with few thousands of nodes and edges. In order to overcome this problem, we adopt a partition-based approach as shown in Figure 4.14.

Step 1. The graph is divided by the *Partitioning* module into a set of k distinct sub-graphs, where k is proportional to the total graph size and the available memory of the machine. This is a *k-way* partitioning that aims at minimizing the number of edges between the different sub-graphs [232].

Step 2. The *Layout* module applies the layout algorithm to each partition independently, and assigns coordinates to the nodes of each sub-graph without considering the edges that cross different partitions. Note that any layout algorithm can be used in this step, e.g., circle, star, hierarchical, etc.

Step 3. The *Partition Organizer* arrange and combine the visualized partitions into a "global" plane taking into account the edges connecting different partitions.

Step 4. Multiple abstraction layers of the input graph are constructed by the *Abstraction* module.

Step 5. At final step, the input graph along with the abstract graphs are indexed and stored in the *Database*.

In the following sections, we provide more details on *Step* 3, 4, and 5.

4.2.2.1 Organizing Partitions

Partitions are organized on the "global" plane using a greedy algorithm whose goal is twofold. First, it ensures that the distinct sub-graphs do not overlap on the plane, and at the same time it tries to minimize the total length of the edges between different partitions (crossing edges).

Initially, the algorithm counts the number of crossing edges for each partition. Then, it selects the partition with the largest number of crossing edges (to all other partitions), and places it at the center of the plane, i.e., it updates the coordinates of its nodes with respect to the "global" plane. This is the m-th partition in Figure 4.14 which has 9 such edges (denoted with red color). The remaining partitions are kept in a priority queue, sorted on the number of the common crossing edges they have with the partitions that exist on the plane (in descending order). At each subsequent step, the algorithm assigns the first partition from the queue to an empty area on the plane so that the total length of the crossing edges between this partition and all other partitions on the plane is minimized. Then, the partition is removed from the queue and the coordinates of its nodes are updated with respect to the assigned area. The order of the partitions in the queue is also updated accordingly, and the algorithm proceeds to the next step. The above process terminates when the priority queue is empty. Intuitively, the efficiency of the algorithm is guaranteed

by the small number of partitions (k), and also by the small size of the area we have to check for the best assignment at each step; this area lies around the non-empty areas from the previous steps.

4.2.2.2 Building Abstraction Layers

After arranging the partitions, a number of abstraction layers is constructed for the initial graph, as shown in Figure 4.14. A layer i ($i > 0$) corresponds to a new graph that is produced by applying an abstraction method to the graph at layer $i-1$. Hence, the overall hierarchy of layers is constructed in a bottom-up fashion, starting from the initial graph at layer 0. Each time we create a new graph at layer i, its layout is based on the layout of the graph at layer $i-1$. The abstraction method can be any algorithm that produces a more condense form of the input graph, either by merging parts of the graph into single nodes (like the graph summarization methods we mentioned in the introduction) or by filtering parts of the graph according to a metric, e.g., a node ranking criterion like PageRank. We emphasize that our approach does not pose any restrictions to the number of layers or the size of the graph at each layer. Finally, all layers are kept as separate graphs in the database as we explain below.

index	B-tree	fulltext	R-tree		B-tree	fulltext
attribute	$Node_1$ ID	$Node_1$ Label	Edge Geometry	Edge Label	$Node_2$ ID	$Node_2$ Label
type	*int*	*text*	*geometry*	*text*	*int*	*text*

Figure 4.15: Storage scheme

4.2.2.3 Storage Scheme

Our database includes a single relational table per abstraction layer that stores all information about the graph of this layer. All these tables have the same schema as depicted in Figure 4.15. Intuitively, each graph is stored as a set of triples of the form ($node_1$, edge, $node_2$). A row in the table of Figure 4.15 contains the following attributes: (1) the unique ID of the first node ($Node_1$ ID), (2) the label of the first node ($Node_1$ Label), (3) the geometry of the connecting edge (Edge Geometry) which is an binary object that represents the line between $node_1$ and $node_2$ on the plane, (4) the label of the edge (Edge Label), (5) the unique ID of the second node ($Node_2$ ID), and (6) the label of the second node ($Node_2$ Label). When the edge is directed, $node_1$ is always the source node whereas $node_2$ is the target node. This information is encoded in the binary object that represents the geometry of the edge.

B^+-trees are built on attributes (1) and (5) to retrieve all information about a node efficiently. The full text indexes shown in Figure 4.15 correspond to tries, and they are used to support fast keyword search on the graph metadata. Finally, an R-tree is used to index the geometries of the edges on the plane. Note that each such geometry is internally defined by the coordinates of the first and the second node whose IDs and labels are stored in the same row of the table.

4.2.3 Exploration Operations

On the client side, our platform provides three main visual exploration operations:

4.2.3.1 Interactive Navigation

The user navigates on the graph by moving the viewing window ("horizontal" navigation). When the window is moved, its new coordinates with respect to the whole canvas are tracked on the client side, and a spatial range query (i.e., a window query) is sent to the server. This query retrieves all elements of the graph (nodes and edges) that overlap with the current window. The query is evaluated with a lookup in the R-tree of Figure 4.15, and the respective part of the graph is fetched from the database and sent to the client.

After the part of the graph is rendered on the canvas, the user can start the exploration. By clicking on a node, an *Information Panel* shows the complete label of the selected node along with the labels of its neighbours (adjacent nodes) and also the labels of the connecting edges. This information is retrieved through asynchronous requests to the server using the unique ID assigned to each node. The user can also highlight a node's neighbourhood by hiding all other nodes except the selected one and its adjacent nodes.

4.2.3.2 Multilevel Exploration

The user moves up or down at different abstraction layers of the graph through a *Layer Panel* ("vertical" navigation). When changing a level of abstraction, the graph elements are fetched through spatial range queries on the appropriate table that corresponds to the selected layer. Vertical navigation can be combined with traditional zoom in/out operations in order to give the impression of a lower/higher perspective. In this case, the size of the window (rectangle) that is sent to the server is decreased/increased proportionally according to the zoom level.

4.2.3.3 Keyword-based Exploration

Finally, the user searches the graph using keywords through a *Search Panel*. In this case, a keyword query is sent to the server and it is evaluated on the whole set of node labels which are indexed with tries. The result of this query is a list of nodes whose labels contain the given keyword. By clicking on a node from the list, the user's window focuses on the position of this node. In this case, the spatial query sent to the server uses as window the rectangle whose size is equal to the size of the client's window and whose center has the same coordinates with the selected node from the list.

4.2.4 The graphVizdb Platform

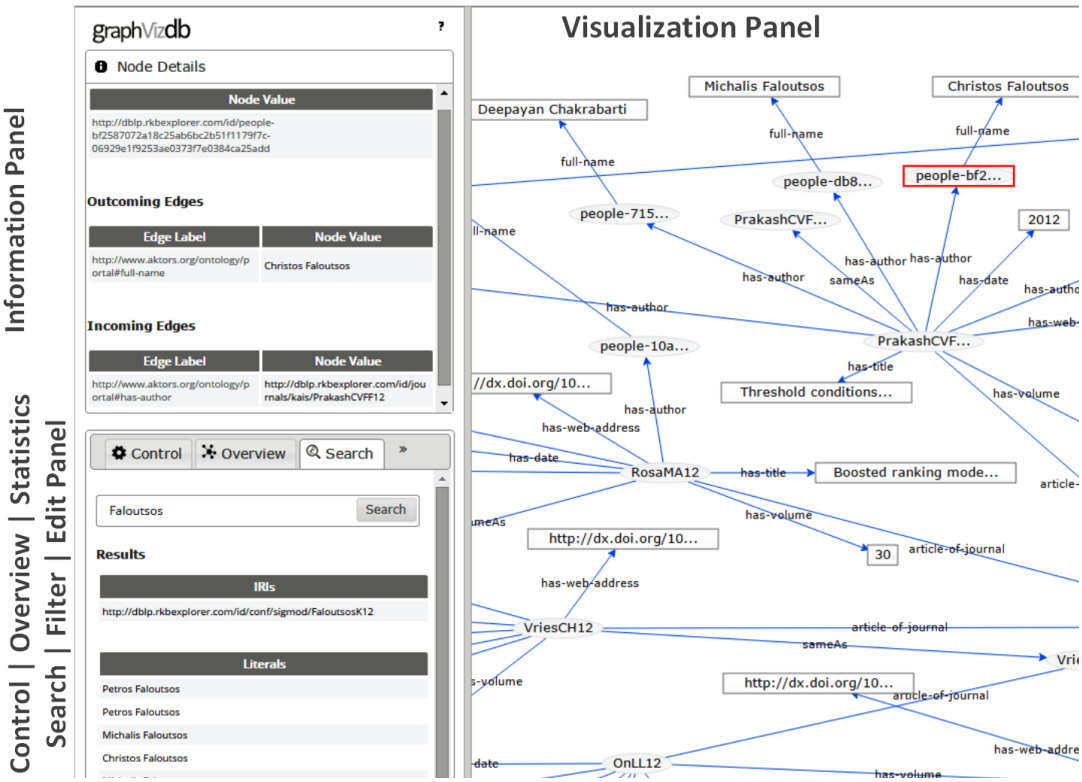
4.2.4.1 Implementation

graphVizdb[24] is implemented on top of several open-source tools and libraries. The *Core* module of our system is developed in Java 1.7, and the database we use is MySQL 5.6.12. The partitioning of the graph, during the preprocessing phase, is done with Metis 5.1.0[25] whereas the layout of each partition is built with Graphviz 2.38.0[26]. The web-based frontend is entirely based on HTML and JavaScript. For the interactive visualization of the graph on the client side, we use mxGraph 3.1.2.1[27].

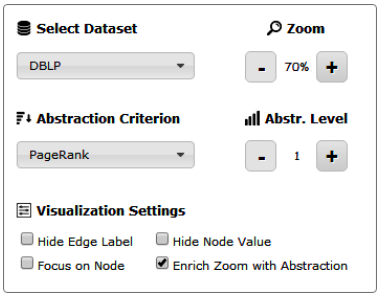
4.2.4.2 Web User Interface

The UI shown in Fig. 4.16a consists of the following panels: (1) *Visualization*, i.e., the interactive canvas, (2) *Information* that provides information about a selected node (meta-

[24] graphvizdb.imis.athena-innovation.gr
[25] glaros.dtc.umn.edu/gkhome/views/metis
[26] www.graphviz.org
[27] www.jgraph.com



(a) Main window



(b) Control panel

(c) Birdview

Figure 4.16: Web user interface

data), (3) *Control* (Fig. 4.16b), (4) *Birdview* (Fig. 4.16c), i.e., a large-scale image of the whole graph on the plane, (5) *Search* that offers keyword search functionalities, (6) *Statistics* that provides basic statistics for the graph (e.g., average node degree, density, etc.) and the current window, (7) *Filter* that offers filtering operations on the canvas (i.e., hide edges/nodes), and (8) *Edit* that allows the user to store in the database the graph modifications made through the canvas.

4.2.4.3 graphVizdb In-Use

In this section we outline the basic functionality of graphvVizdb prototype. The graphVizdb allow users to efficiently and effectively interact, navigate and explore large graphs through a web browser.

Initially, the users are able to select a dataset from a number of real-word datasets (e.g., DBpedia, DBLP, ACM, Notre Dame web graph). Then, users are able to have a quick glance on the graph using various navigation methods such as panning, selecting a specific

Table 4.8: Time for each preprocessing step (min)

Dataset	#Edges	#Nodes	Step 1	Step 2	Step 3	Step 4	Step 5
Wikidata	151M	146M	1.8	4.5	25.5	16.5	670.1
Patent	16.5M	3.8M	5.1	2.8	9.7	8.2	41.2

part of the graph in the birdview panel, etc. Through the navigation, the information panel can provide useful informations regarding the concepts (i.e., nodes) and relations (i.e., edges) appear in the dataset. Further, the users are able to filter (i.e., hide) edges and/or nodes of specific types (e.g., leaf nodes), as well as to zoom in/out over the graph. For example, in the ACM dataset, a user interested in exploring the citations between articles, will be able to filter out irrelevant edges (e.g., *has-author*, *has-title*) and visualize only the *cite* edges.

Additionally, the users are able to explore the "Focus on node" mode, which is suitable for pathway navigation, as well as for helping users to further understand the relations amongst the nodes of interest. In this mode, only the selected node and its neighbours are visible. The user interested in exploring the scientific collaborations of an author will be able to use keywords in order to search for this person, e.g., Christos Faloutsos. Then, using the "Focus on node", the user can quickly explore all Faloutsos' collaborations by following the "Christos Faloutsos · has-author · article · has-author" paths.

Beyond simple navigation, users are able to perform a multi-level graph exploration. In particular, they are be able to modify the abstraction level as well as the abstraction criteria (e.g., Node degree, PageRank, HITS). For example, by selecting either PageRank or HITS as the abstraction criterion in a web graph, the users will be able to view different layers of the graph that contain only the "important" nodes (e.g., sites whose PageRank score is above a certain threshold).

A video presenting the basic functionality of our prototype is available at: `vimeo.com/117547871`.

4.2.5 Experimental Analysis

In this section, we study the performance of the proposed platform and we present the results of our experimental evaluation using two real graph datasets.

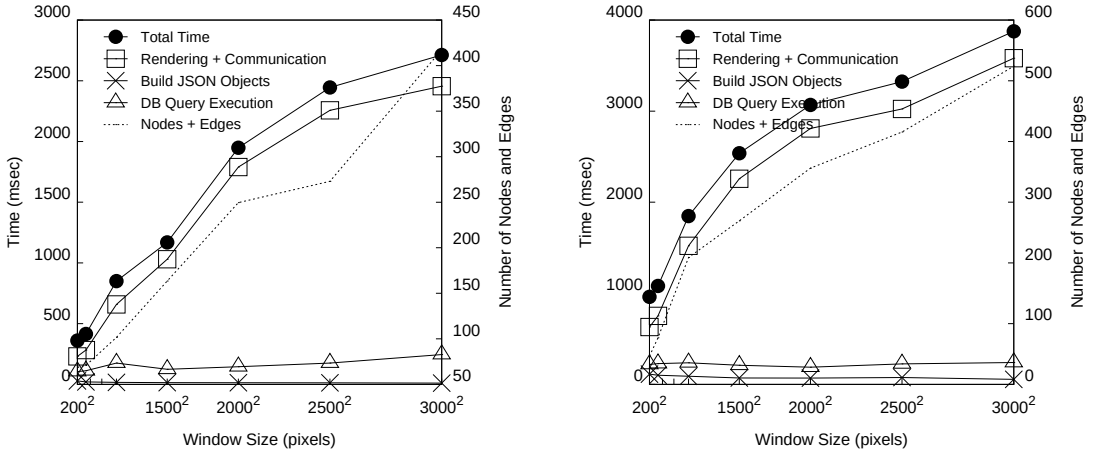
4.2.5.1 Setting

The experiments we present here were conducted on the Okeanos cloud[28] using a VM with a quad-core CPU at 2GHz and 8GB of RAM running Linux. For the client application, we used Google Chrome on a laptop with an i7 CPU at 1.8GHz and 4GB of RAM. The cache size of MySQL on the server side was set to 6GB.

4.2.5.2 Datasets

To evaluate the response time of our system, we used several real graph datasets with rather different characteristics. Here we present only the results for two datasets: the *Wikidata*[29] RDF dataset, and the *Patent*[30] citation graph. The first one is an RDF export of Wikidata having 151M edges and 146M nodes. Its average node degree is 2.1 whereas its density is $1.4E$-8. The second dataset is taken from the SNAP repository[31] of

[28] okeanos.grnet.gr
[29] tools.wmflabs.org/wikidata-exports/rdf/exports/20150223
[30] snap.stanford.edu/data/cit-Patents.html
[31] snap.stanford.edu



(a) Wikidata (b) Patent

Figure 4.17: Time vs. Window size

large network datasets. It contains 16.5M edges and 3.8M nodes with average degree 8.8. Its density is $2.3E$-6.

4.2.5.3 Preprocessing Phase

Table 4.8 presents the preprocessing time for each step of Fig. 4.14. These times are higher for Wikidata since it is much bigger than the Patent dataset. The only exception is the time spent in Step 1 for applying the *k-way* partitioning; this process takes longer for Patent due to the higher average node degree. Note that the most expensive part of the preprocessing is the indexing step; however, the presented times correspond to the total time spent in indexing 5 layers of each dataset, one after the other. In practice, we can speed up this step by distributing the layers to different nodes of the cluster and perform the indexing in parallel. In this case, the time spent in Step 5 equals the time for indexing the input graph (layer 0), that is, 274.5 and 17.4 minutes for Wikidata and Patent respectively.

4.2.5.4 Exploration

Our experimental scenario includes the evaluation of window queries with different sizes. These queries are evaluated by the server and sent to the client for visualization. In particular, we used window queries whose size varies from 200^2 to 3000^2 pixels, and we evaluated them on the initial graph of each dataset, i.e., on the bottom layer of abstraction (i.e., most-detailed level).

For each window size, we generated 100 random queries. The results we present in Fig. 4.17 correspond to the following average times per query (msecs): (1) *DB Query Execution*: the time spent to evaluate the query in the database, (2) *Build JSON Objects*: the time required for the server to process the query result and build the JSON objects that are sent to the client, (3) *Communication + Rendering*: the time spent in the client-sever communication plus the time needed to render the graph on the browser, and (4) *Total Time*: the sum of the above times. The *Nodes + Edges* in Fig. 4.17 refer to the average number of nodes and edges included in the 100 random windows of each size.

The first observation is that the performance of our approach scales linearly with the window size and the total number of objects in it. This behaviour is similar for both datasets. As we can see in Fig. 4.17, the overall response time of the system is dominated

by the time spent in *Communication + Rendering*. We do not present these two operations separately because the part of the graph included in the window of the user is sent from the server to the client in small pieces, i.e., in a streaming fashion; hence, the respective times cannot be easily distinguished. As a final comment, the time spent to evaluate the query in the database is negligible and increases slightly as the size of the window increases.

4.2.6 Summary

In this section we have presented graphVizdb, a generic and scalable platform for the interactive visual exploration of very large graphs at multiple levels. The presented platform introduce a new paradigm to explore visualized graphs, similar to this followed in map exploration. A scheme for graph disk-based indexing and storing have been presented. Further, for enabling very large graph visualization, a partition-based visualization technique is proposed. Finally, the platform has been developed as a web-based prototype system which support four main operations: (1) interactive navigation, (2) multilevel exploration, (3) subgraph selection and manipulation, and (4) keyword search.

Part III

Semantic Data Analysis

Chapter 5

Interoperability between the XML and Semantic Web Worlds

The *Web of Data* is an open environment consisting of a great number of large inter-linked RDF datasets from various domains. In this environment, organizations and companies adopt the *Linked Data* practices utilizing *Semantic Web* (SW) technologies, in order to publish their data and offer SPARQL endpoints (i.e., SPARQL-based search services). On the other hand, the dominant standard for information exchange in the Web today is XML. Additionally, many international standards (e.g., *Dublin Core*, *MPEG-7*, *METS*) in several domains (e.g., Digital Libraries, GIS, Multimedia) have been expressed in XML Schema. The aforementioned have led to an increasing emphasis on XML data, accessed using the XQuery query language. The SW and XML worlds and their developed infrastructures are based on different data models, semantics and query languages. Thus, it is crucial to develop interoperability mechanisms that allow the Web of Data users to access XML datasets, using SPARQL, from their own working environments. It is unrealistic to expect that all the existing legacy data (e.g., Relational, XML) will be transformed into SW data. Therefore, publishing legacy data as Linked Data and providing SPARQL endpoints over them has become a major research challenge. In this direction, we introduce the *SPARQL2XQuery Framework* which creates an interoperable environment, where SPARQL queries are automatically translated to XQuery queries, in order to access XML data across the Web. The SPARQL2XQuery Framework provides a mapping model for the expression of OWL–RDF/S to XML Schema mappings as well as a method for SPARQL to XQuery translation. To this end, our Framework supports both manual and automatic mapping specification between ontologies and XML Schemas. In the automatic mapping specification scenario, the SPARQL2XQuery exploits the XS2OWL component which transforms XML Schemas into OWL ontologies. Finally, extensive experiments have been conducted in order to evaluate the schema transformation, mapping generation, query translation and query evaluation efficiency, using both real and synthetic datasets.

5.1 Introduction

The *Linked-Open Data*[1], *Open-Government*[2] and *Linked Life Data*[3] initiatives have played a major role in the development of the so called *Web of Data* (WoD). In the WoD, a large number of organizations, institutes and companies (e.g., DBpedia, GeoNames, PubMed, Data.gov) adopt the *Linked Data* practices. Utilizing the *Semantic Web* (SW) technologies

[1] linkeddata.org
[2] www.whitehouse.gov/open
[3] linkedlifedata.com

[82], they publish their data and offer SPARQL endpoints (i.e., SPARQL-based search services). Nowadays, there are hundreds of large inter-linked RDF datasets from various domains which comprise the WoD. It is challenging though, to make information that is stored in non-RDF data sources (e.g., Relational databases, XML repositories) available in the WoD.

The SW infrastructure supports the management of RDF datasets [100, 279, 289], accessed by the SPARQL query language [314]. Since the WoD applications and services have to coexist and interoperate with the existing applications that access legacy systems, it is essential for the WoD infrastructure to provide transparent access to information stored in heterogeneous legacy data sources. Publishing legacy data that adopt the Linked Data practices and offer SPARQL endpoints over it, has become a major research and development objective for many organizations.

In the current Web infrastructure the XML/XML Schema [138, 178, 360] are the dominant standards for information exchange as well as for the representation of semi-structured information. As a consequence, many international standards in several domains (e.g., Digital Libraries, GIS, Multimedia, e-Learning, Government, Commercial) have been expressed in XML Schema syntax. For example, the *Dublin Core* [1] and *METS* [6] standards are used by digital libraries, the *MPEG-7* [9] and *MPEG-21* [8] standards are utilized for multimedia content and service description, the *MARC 21* [4], *MODS* [7], *TEI* [15], *EAD* [2] and *VRA Core* [16] standards are used by cultural heritage institutions (e.g., libraries, archives, museums) and the *IEEE LOM* [3] and *SCORM* [12] standards are exploited in e-learning environments. The universal adoption of XML for web data exchange and the expression of several standards using XML Schema, have resulted in a large number of XML datasets accessed using the XQuery query language [346]. For example, Oracle has at least 7000 customers using the XQuery feature in its products [51].

Since the SW and XML worlds have different data models, different semantics and use different query languages to access data [82], it is crucial to develop frameworks, including models and adaptable software based on them, as well as methodologies that will provide interoperability between the SW and the XML infrastructures, thus facilitating transparent XML querying in the WoD using SW technologies.

The scenario of transforming all the legacy data into SW data is clearly unrealistic due to: (a) The different data models adopted and enforced by different standardization bodies (e.g., consortiums, organizations, institutions); (b) Ownership issues; (c) The existence of systems that access the legacy data; (d) Scalability requirements (large volumes of data involved); and (e) Management requirements, e.g., support of updates. Thus, a realistic integration of the two worlds has to be established.

The W3C community has realized the need to bridge different worlds (e.g., Relational, XML, SW) under several scenarios. Tim Berners Lee introduced the *Double Bus Architecture*[4], a W3C Design Issue. The Double Bus Architecture assumes that the WoD users and applications use the SPARQL query language to ask for content from the underlying XML and Relational data sources. In the context of the relational and SW worlds, the W3C RDB2RDF working group [11] has been established, which is attempting to bridge the relational and SW worlds [330, 143]. In addition, a large number of approaches has been proposed for bridging the relational databases with the SW through SPARQL to SQL translation [248, 337, 89, 379, 325, 105, 118, 275, 116, 161, 166, 316]. In the context of the SW and XML worlds, two W3C working groups (*GRDDL* [137] and *SAWSDL* [171]) focus on transforming XML data to RDF data (and vice versa). Moreover, W3C investigates the *XSPARQL* approach for merging XQuery and SPARQL for transforming XML to RDF data (and vice versa).

The recent efforts in bridging the SW and XML worlds focus on data transformation

[4] www.w3.org/DesignIssues/diagrams/sw-double-bus.png

(i.e., XML data to RDF data and vice versa). However, despite the significant body of related work on SPARQL to SQL translation, to the best of our knowledge, there is no work addressing the SPARQL to XQuery translation problem. Given the high importance of XML and the related standards in the Web, this is a major shortcoming in the state of the art. Finally, as far as the Linked Data context is concerned, publishing legacy data and offering SPARQL endpoints over them, has recently become a major research challenge. In spite of the fact that several systems (e.g., *D2R Server* [89], *SparqlMap* [379], *Quest* [325], *Virtuoso* [105], *TopBraid Composer*[5]) offer SPARQL endpoints over relational data, to the best of our knowledge, there is no system supporting XML data.

This work presents SPARQL2XQuery, a framework that provides transparent access over XML in the WoD. Using the SPARQL2XQuery Framework, XML datasets can be turned into SPARQL endpoints. The SPARQL2XQuery Framework provides a method for SPARQL to XQuery translation, with respect to a set of predefined mappings between ontologies and XML Schemas. To this end, our Framework supports both manual and automatic mapping specifications between ontologies and XML Schemas, as well as a schema transformation mechanism.

5.1.1 Motivating Example

Here, we outline two scenarios in order to illustrate the need for bridging the SW and XML worlds in several circumstances. In our examples, three hypothetically autonomous partners are involved: (a) *Digital Library X* (which belongs to an institution or a company), (b) *Organization A* and (c) *Organization Z*. Each has adopted different technologies to represent and manage their data. Assume that, Digital Library X has adopted XML-related technologies (i.e., XML, XML Schema, and XQuery) and its contents are described in XML syntax, while both organizations have chosen SW technologies (i.e., RDF/S, OWL, and SPARQL).

1st Scenario. Consider that Digital Library X wants to publish their data in the WoD using SW technologies, a common scenario in the Linked Data era. In this case, a schema transformation and a query translation mechanism are required. Using the schema transformation mechanism, the XML Schema of Digital Library X will be transformed to an ontology. Then, the query translation mechanism will be used to translate the SPARQL queries posed over the generated ontology, to XQuery queries over the XML data.

2nd Scenario. Consider WoD users and/or applications that express their queries or have implemented their query APIs using the ontologies of Organization A and/or Organization Z. These users and applications should be able to have direct access to Digital Library X from the SW environment, without changing their working environment (e.g., query language, schema, API). In this scenario, a mapping model and a query translation mechanism are required. In such a case, an expert specifies the mappings between the Organization ontologies and the XML Schema of Digital Library X. These mappings are then exploited by the query translation mechanism, in order to translate the SPARQL queries posed over the Organization ontologies, to XQuery queries to be evaluated over the XML data of Digital Library X. It should be noted that in most real-world situations, an XML Schema may be mapped to more than two ontologies.

Note that in the first scenario, Digital Library X may want to publish its data in the WoD, using existing, well accepted vocabularies (e.g., FOAF, SIOC, SKOS). The same may hold for the second scenario, where the queries or the APIs may be expressed over well-known vocabularies (which are manually mapped to the XML Schema of Digital Library X).

[5] www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition

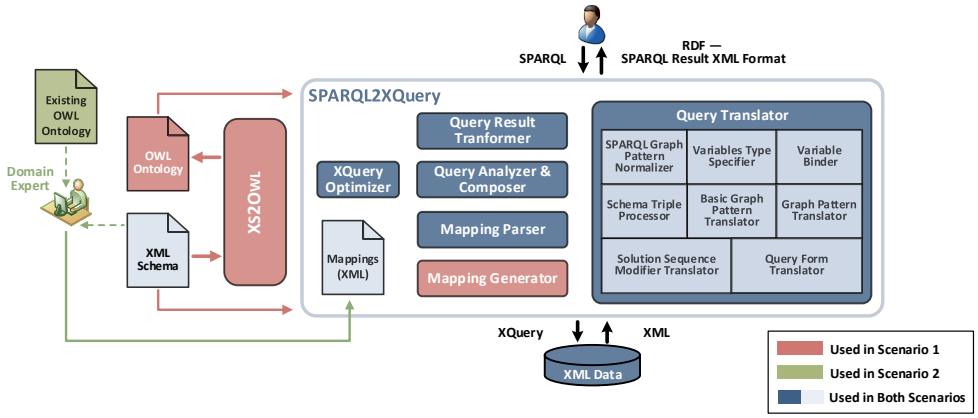


Figure 5.1: SPARQL2XQuery architectural overview. In the first scenario, the XS2OWL component is used to create an OWL ontology from the XML Schema. The mappings are automatically generated and stored. In the second scenario, a domain expert specifies the mapping between existing ontologies and the XML Schema. In both scenarios, SPARQL queries are processed and translated into XQuery queries for accessing the XML data. The results are transformed in the preferred format and returned to the user.

5.1.2 Framework Overview

In this chapter, we present the SPARQL2XQuery Framework, which bridges the heterogeneity gap and creates an interoperable environment between the SW (OWL/RDF/SPARQL) and XML (XML Schema/XML/XQuery) worlds. An overview of the system architecture of the SPARQL2XQuery Framework is presented in Figure 6.1. As shown in Figure 6.1, our working scenarios involve existing XML data that follow one or more XML Schemas. Moreover, the SPARQL2XQuery Framework supports two different scenarios:

1st Scenario: Querying XML data based on automatically generated ontologies. This is achieved through the XS2OWL component [375] that we have developed and integrated in the SPARQL2XQuery Framework. In particular, the XS2OWL component automatically generates OWL ontologies that capture the XML Schema semantics. Then, the SPARQL2XQuery Framework automatically detects, generates and maintains mappings between the XML Schemas and the OWL ontologies generated by XS2OWL. In this case, the following steps take place:

(a) Using the XS2OWL component, the XML Schema is expressed as an OWL ontology.

(b) The Mapping Generator component takes as input the XML Schema and the generated ontology, and automatically generates, maintains and stores the mappings between them in XML format.

(c) The SPARQL queries posed over the generated ontology are translated by the Query Translator component to XQuery expressions.

(d) The query results are transformed by the Query Result Transformer component into the desired format (SPARQL Query Result XML Format [136] or RDF format).

In this context, our approach can be viewed as a fundamental component of hybrid ontology-based integration [390] frameworks (e.g., [276, 277]), where the schemas of the XML data sources are represented as OWL ontologies and these ontologies, possibly along with other ontologies, are further mapped to a global ontology.

2nd Scenario: Querying XML data based on existing ontologies. In this scenario, XML Schema(s) are manually mapped by an expert to existing ontologies, resulting in the mappings that are used in the SPARQL to XQuery translation. In this case the following steps take place:

(a) An XML Schema is manually mapped to an existing RDF/S–OWL ontology.

(b) The SPARQL queries posed over the ontology are translated to XQuery expressions.

(c) The query results are transformed in the desired format.

In both scenarios, the systems and the users that pose SPARQL queries over the ontology are not expected to know the underlying XML Schemas or even the existence of XML data. They express their queries only in standard SPARQL, in terms of the ontology that they are aware of, and they are able to retrieve XML data. Our Framework is an essential component in the WoD environment that allows setting SPARQL endpoints over the existing XML data.

The SPARQL2XQuery Framework supports the following operations:

(a) *Schema Transformation.* Every XML Schema can be automatically transformed in an OWL ontology, using the XS2OWL component.

(b) *Mapping Generation.* The mappings between the XML Schemas and their OWL representations can be automatically detected and stored as XML documents.

(c) *Query Translation.* Every SPARQL query that is posed over the OWL representation of the XML Schemas (first scenario), or over the existing ontologies (second scenario), is translated in an XQuery query.

(d) *Query Result Transformation.* The query results are transformed in the preferred format.

5.1.3 Contributions

The main contributions of this work are summarized as follows:

1. We introduce the XS2OWL Transformation Model, which facilitates the transformation of XML Schema into OWL ontologies. As far as we know, this is the first work that fully captures the XML Schema semantics.
2. We introduce a mapping model for the expression of mappings from RDF/S–OWL ontologies to XML Schemas, in the context of SPARQL to XQuery translation.
3. We propose a method and a set of algorithms that provide a comprehensive SPARQL to XQuery translation. To the best of our knowledge, this is the first work addressing this issue.
4. We integrate the SPARQL2XQuery Framework with the XS2OWL component, thus facilitating the automatic generation and maintenance of the mappings exploited in the SPARQL to XQuery translation.
5. We propose a small number of XQuery rewriting/optimization rules which are applied on the XQuery expressions produced by the translation, aiming at the generation of more efficient XQuery expressions. In addition, we experimentally study the effect of these rewriting rules on the XQuery performance.
6. We describe an extension of the SPARQL2XQuery Framework in the context of supporting the SPARQL 1.1 update operations.
7. We conduct a thorough experimental evaluation, in terms of: (a) schema transformation time; (b) mapping generation time; (c) query translation time; and (d) query evaluation time, using both real and synthetic datasets.

5.2 Related Work

A large number of *data integration* [274] and *data exchange* (also known as data transformation/translation) [167] systems have been proposed in the existing literature. In the context of XML, the first research efforts have attempted to provide interoperability and integration between the relational and XML worlds [290, 312, 399, 145, 369, 231, 267, 249, 244]. In addition, several approaches have focused on data integration and exchange over heterogeneous XML data sources [148, 93, 194, 195, 364, 217, 40, 27, 96].

In the context of interoperability support between the SW and XML worlds [82], numerous approaches for transforming XML Schemas to ontologies, and/or XML data to RDF data and vice versa have been proposed. The most recent ones combine SW and XML technologies in order to transform XML data to RDF and vice versa. Among the published results, the most relevant to our approach are those that utilize the SPARQL query language.

In the rest of this section, we present an overview of the published research that is concerned with the interoperability and integration between the SW and XML worlds (Section 5.2.1). The latest approaches are described in Section 5.2.2. Finally, a discussion about the drawbacks and the limitations of the current approaches is presented in Section 5.2.3.

5.2.1 Bridging the Semantic Web and XML worlds — An Overview

In this section, we summarize the literature related to interoperability and integration issues between the SW and XML worlds. We categorize these systems into *data integration systems* (Table 6.2) and *data exchange systems* (Table 6.3).

Table 6.2 provides an overview of the data integration systems in terms of the *Environment Characteristics* and the supported *Operations*. The environment characteristics include the *Data Models* of the underlying data sources, the involved *Schema Definition Languages* and the supported *Query Languages*. The operations include the *Query Translation* and the *Schema Transformation*. Regarding the schema transformation, if the method support schema transformation, the value is "✓". Notice that the last row of each table describes our SPARQL2XQuery Framework. Note that the SPARQL2XQuery Framework does not deal with the problem of integrating data form different XML data sources; thus, it should be considered as an interoperability system or a core component of integration systems. Hence, it fits better in Table 6.2 than Table 6.3.

Table 6.3 provides an overview of the data exchange systems and is structured in a similar way with Table 6.2. If the value of the fifth column (*Use of an Existing Ontology*) is "✓", the method supports mappings between XML Schemas and existing ontologies and, as a consequence the XML data are transformed according to the mapped ontologies.

The data integration systems (Table 6.2) are generally older and they do not support the current standard technologies (e.g., XML Schema, OWL, RDF, SPARQL). Notice also, that, although the data exchange systems shown in Table 6.3 are more recent, they do not support an integration scenario neither they provide query translation methods. Instead, they focus on data and schema transformation, exploring how the RDF data can be transformed in XML syntax and/or how the XML Schemas can be expressed as ontologies and vice versa.

5.2.2 Recent Approaches

In this section, we present the latest approaches related to the support of interoperability and integration between the SW and XML worlds. These approaches utilize the current W3C standard technologies (e.g., XML Schema, RDF/S, OWL, XQuery, SPARQL). Most

Table 5.1: Overview of the data integration systems in the SW and XML worlds

Data Integration Systems					
System	**Environment Characteristics**			**Operations**	
	Data Models	**Schema Definition Languages**	**Query Languages**	**Query Translation**	**Schema Transformation**
STYX (2002) [33, 67]	XML	DTD / Graph	OQL / XQuery	OQL → XQuery	
ICS–FORTH SWIM (2003) [125, 124, 244]	Relational / XML	DTD / Relational / RDF Schema	SQL / XQuery / RQL	RQL → SQL & RQL → XQuery	
PEPSINT (2004) [397, 396, 133, 132]	XML	XML Schema / RDF Schema	XQuery / RDQL	RDQL → Xquery	XML Schema → RDF Schema
Lehti & Fankhauser (2004) [259]	XML	XML Schema / OWL	XQuery / SWQL	SWQL → XQuery	XML Schema → OWL
SPARQL2XQuery	XML	XML Schema / OWL	XQuery / SPARQL	SPARQL → XQuery	XML Schema → OWL (XS2Owl)

Table 5.2: Overview of the data exchange systems in the SW and XML worlds

Data Integration Systems					
System	**Environment Characteristics**		**Operations**		
	Data Models	**Schema Definition Languages**	**Schema Transformation**	**Use Existing Ontology**	**Data Transformation**
Klein (2002) [273]	XML / RDF	XML Schema / RDF Schema			XML → RDF
WEESA (2004) [318]	XML / RDF	XML Schema / OWL		✓	XML → RDF
Ferdinand et al. (2004) [172]	XML / RDF	XML Schema / OWL–DL	XML Schema → OWL–DL		XML → RDF
Garcia & Celma (2005) [180]	XML / RDF	XML Schema / OWL–FULL	XML Schema → OWL–FULL		XML → RDF
Bohring & Auer (2005) [94]	XML / RDF	XML Schema / OWL–DL	XML Schema → OWL–DL		XML → RDF
Gloze (2006) [328]	XML / RDF	XML Schema / OWL			XML ↔ RDF
JXML2OWL (2006 & 2008) [323, 324]	XML / RDF	XML Schema / OWL		✓	XML → RDF
GRDDL (2007) [137]	XML / RDF	not specified			XML ↔ RDF ⋆
SAWSDL (2007) [171]	XML / RDF	not specified			XML ↔ RDF ⋆
Thuy et al. (2007 & 2008) [371, 372]	XML / RDF	DTD / OWL–DL	DTD → OWL–DL ⋆		XML → RDF ⋆
Janus (2008 & 2011) [60, 61]	XML / RDF	XML Schema / OWL–DL	XML Schema → OWL–DL		
Deursen et al. (2008) [147]	XML / RDF	XML Schema / OWL		✓	XML → RDF ⋆
XSPARQL (2008) [28, 87, 86]	XML / RDF	not specified			XML ↔ RDF ⋆
Droop et al. (2007 & 2008) [155, 154, 153]	XML / RDF	not specified			XML → RDF ⋆
Cruz & Nicolle (2008) [131]	XML / RDF	XML Schema / OWL		✓	XML → RDF
XSLT+SPARQL (2008) [69]	XML / RDF	not specified			RDF → XML
DTD2OWL (2009) [370]	XML / RDF	DTD / OWL–DL	DTD → OWL–DL		XML → RDF
Corby et al. (2009) [128]	XML / RDF / Relational	not specified			XML → RDF ⋆ Relational → RDF
TopBraid Composer (Maestro Edition) – TopQuadrant (Commercial Product) [5]	XML / RDF	not specified / OWL	XML → OWL		XML ↔ RDF ⋆
XS2OWL	XML / RDF	XML Schema 1.1 / OWL 2	XML Schema → OWL		XML ↔ RDF

⋆The transformation is performed in a semi-automatic way that requires user intervention.

of the latest efforts (Table 6.3) focus on combining the XML and the SW technologies in order to provide an interoperable environment. In particular, they merge SPARQL, XQuery, XPath and XSLT features to transform XML data to RDF and vice versa.

The W3C *Semantic Annotations for WSDL* (SAWSDL) Working Group [171] uses XSLT to convert XML data into RDF, and uses a combination of SPARQL and XSLT for the inverse transformation. In addition, the W3C *Gleaning Resource Descriptions from Dialects of Languages* (GRDDL) Working Group [137] uses XSLT to extract RDF data from XML.

XSPARQL [28, 87, 86] combines SPARQL and XQuery in order to achieve the transformation of XML into RDF and back. In the XML to RDF scenario, XSPARQL uses a combination of XQuery expressions and SPARQL Construct queries. The XQuery expressions are used to access XML data, and the SPARQL Construct queries are used to convert the accessed XML data into RDF. In the RDF to XML scenario, XSPARQL uses a combination of SPARQL and XQuery clauses. The SPARQL clauses are used to access RDF data, and the XQuery clauses are used to format the results in XML syntax. Similarly, in [128] XPath, XSLT and SQL are embedded into SPARQL queries in order to transform XML and relational data to RDF. In XSLT+SPARQL [69] the XSLT language is extended in order to embed SPARQL SELECT and ASK queries. The SPARQL queries are evaluated over RDF data and the results are transformed to XML using XSLT expressions.

In some other approaches, SPARQL queries are embedded into XQuery and XSLT queries [190]. In [155, 154, 153], XPath expressions are embedded in SPARQL queries. These approaches attempt to process XML and RDF data in parallel, and benefit from the combination of the SPARQL, XQuery, XPath and XSLT language characteristics. Finally, a method that transforms XML data into RDF and translates XPath queries into SPARQL, has been proposed in [155, 154, 153].

5.2.3 Discussion

In this section we discuss the existing approaches, and we highlight their main drawbacks and limitations. The existing data integration systems (Table 6.2) do not support the current standard technologies (e.g., XML Schema, OWL, RDF, SPARQL). On the other hand, the data exchange systems (Table 6.3) are more recent and support the current standard technologies, but do not support integration scenarios and query translation mechanisms. Instead, they focus on data transformation and do not provide mechanisms to express XML retrieval queries using the SPARQL query language.

The recent approaches [137, 171, 28, 190, 155, 154, 153, 69, 128] however present severe usability problems for the end users. In particular, the users of these systems are forced to: (a) be familiar with both the SW and XML models and languages; (b) be aware of both ontologies and XML Schemas in order to express their queries; and (c) be aware of the syntax and the semantics of each of the above approaches in order to express their queries. In addition, each of these approaches has adopted its own syntax and semantics by modifying and/or merging the standard technologies. These modifications may also result in compatibility, usability, and expandability problems. It is worth noting that, as a consequence of the scenarios adopted by these approaches, they have only been evaluated over very small data sets.

Compared to the recent approaches, in the SPARQL2XQuery Framework introduced in this work the users (a) work only on SW technologies; (b) are not expected to know the underlying XML Schema or even the existence of XML data; and (c) they express their queries only in standard (i.e., without modifications) SPARQL syntax. Finally, the SPARQL2XQuery Framework has been evaluated over large datasets.

Moreover, with the high emphasis in the Linked Data infrastructures, publishing legacy

data and offering SPARQL endpoints has become a major research challenge. Although several systems (e.g., *D2R Server* [89], *SparqlMap* [379], *Quest* [325], *Virtuoso* [105], *TopBraid Composer*[5]) offer virtual SPARQL endpoints over relational data, to the best of our knowledge there is no system offering SPARQL endpoints over XML data. Finally, in contrast with the SPARQL to XQuery translation, the SPARQL to SQL translation has been extensively studied [330, 349, 143, 248, 337, 89, 379, 325, 105, 118, 275, 116, 161, 166, 316]. The SPARQL2XQuery Framework introduced here can offer SPARQL endpoints over XML data and it also proposes a method for SPARQL to XQuery translation.

The interoperability Framework presented in this work includes the XS2OWL component which offers the functionality needed for automatically transforming XML Schemas and data to SW schemas and data. As such, the XS2OWL component is related to the data exchange systems (Table 6.2). The major difference between our work and existing approaches in data exchange systems that provide schema transformation mechanisms is that the latter do not support: (a) the XML Schema identity constraints (i.e., key, keyref, unique); (b) the XML Schema user-defined simple datatypes; and (c) the new constructs introduced by XML Schema 1.1 [178]. These limitations have been overcome by the XS2OWL component, which is integrated with the other components of the SPARQL2XQuery Framework to offer comprehensive interoperability functionality. To the best of our knowledge, this is the first work that fully captures the XML Schema semantics and supports the XML Schema 1.1 constructs. Finally, this Framework is now completely integrated with the other components of the SPARQL2XQuery Framework. Some preliminary ideas regarding the SPARQL2XQuery Framework have been presented in [74].

5.3 Schema Transformation

In this section, we describe the schema transformation process (Figure 5.2) which is exploited in the first usage scenario, in order to automatically transform XML Schemas into OWL ontologies. Following the automatic schema transformation, mappings between the XML Schemas and the OWL ontologies are also automatically generated and maintained by the SPARQL2XQuery Framework. These mappings are later exploited by other components of the SPARQL2XQuery Framework, for automatic SPARQL to XQuery translation.

Figure 5.2: The XS2OWL schema transformation process

The schema transformation is accomplished using the XS2OWL component [375, 354], which implements the *XS2OWL Transformation Model*. The XS2OWL transformation model allows the automatic expression of the XML Schema in OWL syntax. Moreover, it allows the transformation of XML data in RDF format and vice versa. The new version of the XS2OWL Transformation Model which is presented here, exploits the OWL 2 semantics in order to achieve a more accurate representation of the XML Schema constructs in OWL syntax. In addition, it supports the latest versions of the standards (i.e., XML Schema 1.1 and OWL 2). In particular, the XML Schema identity constraints (i.e., *key*, *keyref*, *unique*), can now be accurately represented in OWL 2 syntax (which was not feasible with OWL 1.0). This overcomes the most important limitation of the previous versions of the XS2OWL Transformation Model.

An overview of the XS2OWL transformation process is provided in Figure 5.2. As is shown in Figure 5.2, the XS2OWL component takes as input an XML Schema XS and

Table 5.3: Correspondences between the XML Schema and OWL constructs, according to the XS2OWL Transformation Model

XML Schema Construct	OWL Construct
Complex Type	Class
Simple Datatype	Datatype Definition
Element	(Datatype or Object) Property
Attribute	Datatype Property
Sequence	Unnamed Class – Intersection
Choice	Unnamed Class – Union
Annotation	Comment
Extension, Restriction	subClassOf axiom
Unique (*Identity Constraint*)	HasKey axiom *
Key (*Identity Constraint*)	HasKey axiom – ExactCardinality axiom *
Keyref (*Identity Constraint*)	In the Backwards Compatibility Ontology
Substitution Group	SubPropertyOf axioms
Alternative $^{+}$	In the Backwards Compatibility Ontology
Assert $^{+}$	In the Backwards Compatibility Ontology
Override, Redefine $^{+}$	In the Backwards Compatibility Ontology
Error $^{+}$	Datatype

Note. The $^{+}$ indicates the new XML Schema constructs introduced by the XML Schema 1.1 specification. The * indicates the OWL 2 constructs.

generates: (a) An OWL Schema ontology O_S that captures the XML Schema semantics; and (b) A Backwards Compatibility ontology O_{BC} which keeps the correspondences between the O_S constructs and the XS constructs. O_{BC} also captures systematically the semantics of the XML Schema constructs that cannot be directly captured in O_S (since they cannot be represented by OWL semantics).

The OWL Schema Ontology O_S, which directly captures the XML Schema semantics, is exploited in the first scenario supported by the SPARQL2XQuery Framework. In particular, O_S is utilized by the users while forming the SPARQL queries. In addition, the SPARQL2XQuery Framework processes O_S and XS and generates a list of mappings between the constructs of O_S and XS.

The ontological infrastructure generated by the XS2OWL component, additionally supports the transformation of XML data into RDF format and vice versa [376]. For transforming XML data to RDF, O_S can be exploited to transform XML documents structured according to XS into RDF descriptions structured according to O_S. However, for the inverse process (i.e., transforming RDF documents to XML) both O_S. and O_{BC} should be used, since the XML Schema semantics that cannot be captured in O_S. are required. For example, the accurate order of the XML sequence elements should be preserved; but this information cannot be captured in O_S.

In the rest of this section, we outline the XS2OWL Transformation Model (Section 5.3.1) and we present an example that illustrates the transformation of XML Schema into OWL ontology (Section 5.3.2).

5.3.1 The XS2OWL Transformation Model

In this section, we outline the XS2OWL Transformation Model. A formal description of the XS2OWL Transformation Model and implementation details can be found in [84]. A listing of the correspondences between the XML Schema constructs and the OWL constructs, as they are specified in the XS2OWL Transformation Model, is presented in Table 6.4.

The major difficulties that we have encountered throughout the development of the XS2OWL Transformation Model have arisen from the fact that that the XML Schema and

the OWL have adopted different data models and semantics. In order to resolve some of these heterogeneity issues, we have employed the *Backwards Compatibility ontology* O_{BC} which encodes XML Schema information that cannot be captured by OWL semantics. This information includes: (a) *Identification information*; (b) *Structural information*; and (c) *"Orphan" construct information.*

Identification Information. The OWL semantics do not allow different resources to have the same identifier (rdf:ID), while the XML Schema allows instances of different XML Schema constructs to have the same name (for example, an XML Schema element may have the same name with an XML Schema attribute, two elements of different type may also have the same name, etc.). In order to resolve this issue, the XS2OWL component generates automatically unique identifiers for the OWL constructs in the *Schema ontology* O_S[6]. The correspondence between the names of the XML Schema constructs and the Schema ontology constructs is encoded in the Backwards Compatibility ontology.

Structural Information. The XML Schema data model describes ordered hierarchical structures, while the OWL data model allows the specification of directed unordered graph structures. As a consequence, the ordering information which is essential for some XML Schema constructs like the sequences, cannot be captured in the Schema ontology. This information is encoded in the Backwards Compatibility ontology (see [84] for details).

"Orphan" Construct Information. Since the XML/XML Schema and the OWL/RDF have adopted different data models and semantics, there exist "orphan" XML Schema constructs that can not be accurately represented by OWL constructs. Examples of "orphan" XML Schema constructs are the abstract and final attributes of the XML Schema type. In the context of the XS2OWL, information about the "orphan" XML Schema constructs is encoded in the Backwards Compatibility ontology.

5.3.2 XML Schema Transformation Example

We present here a concrete example that demonstrates the expression of an XML Schema in OWL using the XS2OWL component.

We introduce here an XML Schema (referred in the rest of this chapter as the *Persons* XML Schema), which will be used in the rest of this chapter. The Persons XML Schema is presented in Figure 6.2 and describes the personal information of a sequence of persons (which may be students). The root element *Persons* may contain any number of Person elements of type *Person_Type*, and any number of Student elements of type *Student_Type*. The complex type Person_Type represents persons and contains the *SSN* attribute and several simple elements (i.e., *LastName*, *FirstName*, *validAgeType* and *Email*). The complex type Student_Type extends the complex type Person_Type and represents students. In addition to the elements and attributes defined in the context of Person_Type, the complex type Student_Type has the *Dept* element. The simple type *validAgeType* is a restriction of the float type. Finally, the top-level element *Nachname* is an element that may substitute the LastName element, as is specified in its *substitutionGroup* attribute.

The constructs of the Schema ontology O_S that is automatically generated by the XS2OWL for the Persons XML Schema (referred in the rest of this chapter as the *Persons* Ontology) are presented in Table 6.5 and Table 6.6. In particular:

[6] This is achieved by the *identity generation rules* implemented in the XS2OWL transformation model. The identity generation rules verify the generation of unique identifiers for all the OS OWL constructs. These rules exploit the hierarchical structure of the XML Schema, as well as the types of the XML Schema constructs to generate unique identifiers. More details can be found in [84].

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="Person_Type">
    <xs:sequence>
      <xs:element ref="LastName" minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="FirstName" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="Age" type="validAgeType" minOccurs="1" maxOccurs="1" />
      <xs:element name="Email" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="SSN" type="xs:integer"/>
  </xs:complexType>

  <xs:complexType name="Student_Type">
    <xs:complexContent>
      <xs:extension base="Person_Type">
        <xs:sequence>
          <xs:element name="Dept" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Persons">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Person" type="Person_Type" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Student" type="Student_Type" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="LastName" type="xs:string"/>

  <xs:element name="Nachname" substitutionGroup="LastName" type="xs:string"/>

  <xs:simpleType name="validAgeType" >
    <xs:restriction base="xs:float">
        <xs:minInclusive value="0.0"/>
        <xs:maxInclusive value="150.0"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Figure 5.3: An XML Schema describing Persons (*Persons* XML Schema)

- Information about the classes is provided in Table 6.5. The table includes: (a) the name of the corresponding XML Schema complex type (*XML Schema Complex Types* column); (b) the class rdf:ID (*rdf:ID*); and (c) the superclass rdf:IDs (*rdfs:subClassOf*).
- Information about the datatype properties (DTP) and the object properties (OP) is provided in Table 6.6. The table includes (a) the name of the corresponding XML Schema element or attribute (*XML Schema Elements & Attributes* column); (b) the property type, i.e., DTP or OP (*Type*); (c) the property rdf:ID (*rdf:ID*); (d) the rdf:IDs of the superproperties (*rdfs:subPropertyOf*); (e) the property domains (*rdfs:domain*); and (f) the property ranges (*rdfs:range*).

The constructs of the Backwards Compatibility ontology generated by the XS2OWL are available in [84]. The XML Schema of Figure 6.2 and the Schema ontology O_S generated by XS2OWL are depicted in Figure 6.3.

5.4 Mapping Model

In the SW, the OWL–RDF/S have been adopted as schema definition languages; in the XML world, the XML Schema language is used. The proposed mapping model is defined in the context of the SPARQL to XQuery translation, for the definition of mappings between ontologies and XML Schemas. In particular, the SPARQL2XQuery mapping

Table 5.4: Representation of the Persons XML Schema complex types in the Schema Ontology (O_S)

XML Schema Complex Types	Ontology Classes	
	rdf:ID	rdfs:subClassOf
Person_Type	Person_Type	owl:Thing
Student_Type	Student_Type	Person_Type
Persons (*unnamed complex type*)	NS_Persons_UNType	owl:Thing

Table 5.5: Representation of the Persons XML Schema elements and attributes in the Schema Ontology (O_S)

XML Schema Elements & Attributes	Ontology Properties				
	Type	rdf:ID	rdfs:subPropertyOf	rdfs:domain	rdfs:range
LastName	DTP	LastName__xs_string	—	Person_Type	xs:string
FirstName	DTP	FirstName__xs_string	—	Person_Type	xs:string
Age	DTP	Age__validAgeType	—	Person_Type	validAgeType
Nachname	DTP	Nachname__xs_string	LastName__xs_string	Person_Type	xs:string
Email	DTP	Email__xs_string	—	Person_Type	xs:string
SSN	DTP	SSN__xs_integer	—	Person_Type	xs:integer
Dept	DTP	Dept__xs_string	—	Student_Type	xs:string
Person	OP	Person__Person_Type	—	NS_Persons_UNType	Person_Type
Student	OP	Student__Student_Type	—	NS_Persons_UNType	Student_Type
Persons	OP	Persons__NS_Persons_UNType	—	owl:Thing	NS_Persons_UNType

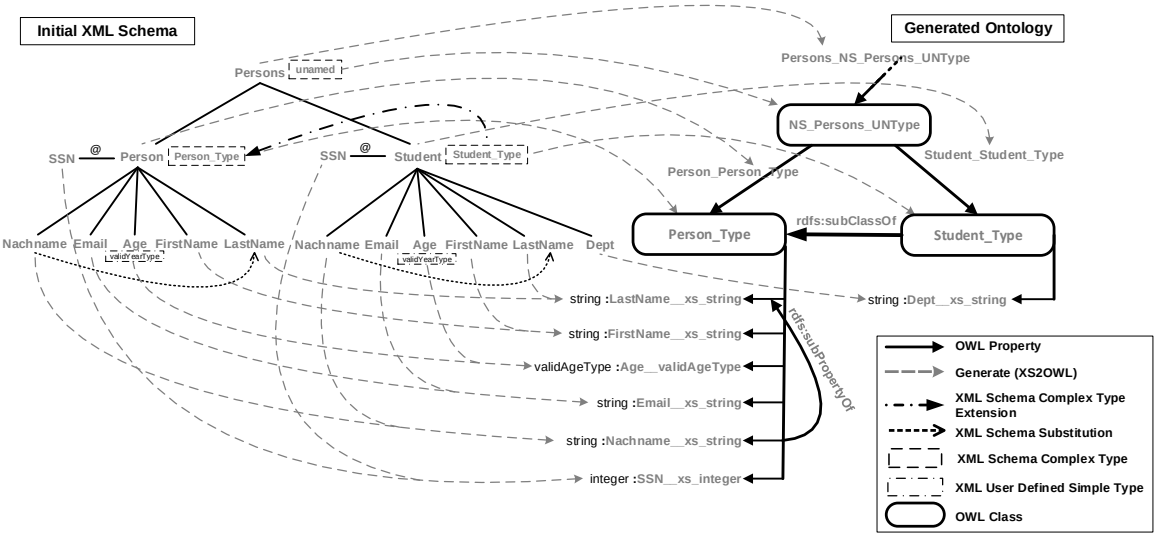


Figure 5.4: The Persons XML Schema of Figure 6.2 and the Persons Schema Ontology generated by XS2OWL with their correspondences drawn in dashed grey lines

model specifies: (a) the supported mappings; (b) the mapping representation; and (c) the necessary operators for formal mapping manipulation.

Mapping *conceptualization*, *definition* and *representation* have been extensively studied under several scenarios (e.g., schema integration, schema matching, data integration, data exchange). In each scenario, these concepts (i.e., conceptualization, definition, etc.) differ based on the scenario settings. For example, in the classical data integration scenario [274], the local sources are defined as views over a global schema (i.e., *local-as-view* – LAV), or the global schema is defined as a collection of views over the local schemas (i.e., *global-as-view* – GAV). In addition, several similar approaches (e.g., *global-local-as-view* – GLAV) have been extensively studied and used in data integration systems. Furthermore, in a

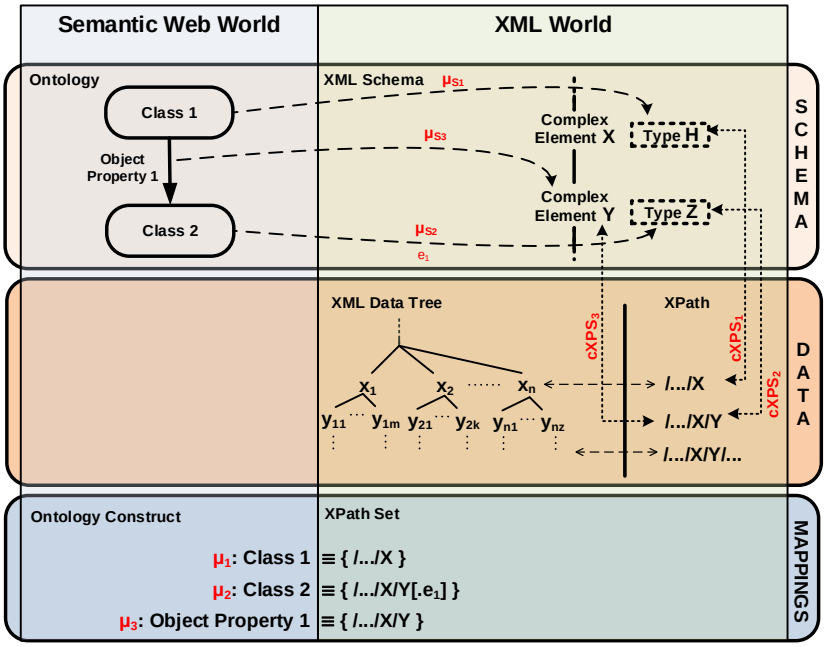


Figure 5.5: Associations between the SW and XML worlds. At the *Schema level*, associations between ontology constructs and XML Schema constructs are obtained. At the *Data level*, the XML data follows the XML Schema and every XML node can be addressed using XPath expressions. Based on the associations between the ontology and the XML Schema, the ontology constructs are associated with the corresponding XPath expressions. In the figure, μ_{S_i} represents a *schema mapping*, $cXPS_i$ a *correspondence between an XML Schema and XPath Sets*, μ_i a *mapping representation* and e_1 a *mapping condition*.

typical data exchange setting [167], mappings that specify the relations between a source and a target schema are defined as sets of *source-to-target tuple-generating-dependencies* (st-tgds). The mappings are used in order to generate instances of the target schema, based on the source data. Nevertheless, our work is not concerned neither with defining views over heterogeneous XML sources nor with defining dependencies for data transformations as is the case in XML data integration and exchange systems (e.g., [148, 93, 194, 195, 364, 217, 40, 27, 96]). Our mappings can be considered as an interoperability layer between the SW and XML worlds, aiming to provide formal, flexible and precise mapping definitions, as well as generation of efficient XQuery queries. Note that in this work we do not consider the problem of integrating data from different XML data sources.

We define our mapping model in the context of providing transparent XML querying in the SW world. In the proposed model, the mappings can be simply considered as pairs of ontology constructs (i.e., classes, properties) and path expressions over the XML data (i.e., XPath). The defined mappings are used for translating the SPARQL queries to XQuery expressions. The adoption of the XPath [17] notion in our mapping model, besides the wide acceptability of XPath, aims to benefit from several XPath properties (e.g., flexibility, expressivity), which are outlined below.

Using XPath expressions we can *precisely* indicate the involved XML nodes. For instance, consider a mapping that aims to indicate the persons whose age is between 20 and 30 (the person definitions follow the Persons schema of Figure 6.2). Using XPath, this mapping can be expressed as $/Persons/Person[./age > 20 \; and \; ./age < 30]$. Moreover, the XPath *expressivity* enhanced with the large XPath library of built-in functions and operators [18], allows our mapping model to support flexible and expressive mapping expressions.

The expression of mappings as XPath expressions allows us to *include both schema*

and data information. As schema information, we consider the hierarchical structure of data imposed by the XPath expressions. As data information, we consider conditions over data values (e.g., $age > 20$). The exploitation of the data structuring allows minimizing the number of the considered mappings, resulting in the creation of non redundant or irrelevant queries.

For example, consider a mapping that maps an ontology property name to the Persons XML schema of Figure 6.2. Assume that the ontology property name is mapped to the XPaths: $/Persons/Person/name$ and $/Persons/Student/name$. Consider now an ontology query aiming to return the names (i.e., the values of the name ontology property) of the persons indicated by the mapping of the previous example (i.e., persons with age between 20 and 30). By examining the property mappings, we can easily notice that the second XPath expression is not relevant to our query. Thus, in this case, the only relevant mapping is the path $/Persons/Person/name$.

Finally, the adoption of XPath expressions allows the definition of mappings using other mappings (as "building blocks"). This feature can be exploited in the XQuery expressions for (a) associating different variables and/or (b) for using already evaluated results. The aforementioned can lead to the generation of efficient XQuery queries. For instance, consider an XQuery variable $\$v$, that contains the results of the evaluation of the persons mappings over an XML dataset. Using the $\$v$ variable, we can easily "construct" the mappings for the name property as $\$v/name$. In this way, each person can be associated with his name(s) using For XQuery clauses.

Figure 6.4 outlines the associations between the SW (left side) and XML (right side) worlds. In particular, it presents an ontology, an XML Schema and the associations among them, in both the schema and data levels. At the *schema level* (Ontology/XML Schema), associations between the ontology constructs (i.e., classes, properties, etc.) and the XML Schema constructs (i.e., elements, complex types) are obtained. Moreover, at the *data level*, the XML data follow the XML Schema. As a result, we can identify the occurrences of the XML Schema constructs in the XML data, and address them using a set of XPath expressions. Finally, the *mappings in the context of SPARQL to XQuery translation* can be simply considered as associations between ontology constructs and XPath expressions (in the bottom layer of Figure 6.4).

In the rest of this section, we introduce the XPath Set notion (Section 5.4.1), we define the schema mappings (Section 5.4.2), we present the association between the schema and data levels (Section 5.4.3), we define the mapping representation (Section 5.4.4), and finally we outline the automatic mapping generation process (Section 5.4.5).

5.4.1 Preliminaries

In our mapping model, XPath expressions are exploited in order to address XML nodes at the data level. In this section, we provide the basic notions regarding the XPath expressions (Section 5.4.1.1) and we introduce operators for handling sets of XPath expressions (Section 5.4.1.2). Finally, Section 5.4.1.3 specifies the basic XML Schema and Ontology constructs involved in mapping model.

5.4.1.1 Basic XPath Notions

Let $xp \in \mathbf{XP}$ be an XPath expression, where $\mathbf{XP}$ is the set of the XPath expressions. xp is expressed using a fragment of the XPath language, which involves: (a) a set of *node names* $\mathbf{N} = \{n_1, \ldots n_i\}$; (b) the *child operator* (/); (c) the *predicate operator* ([ ]); (d) the *wildcard operator* (*); (e) the *attribute access operator* (@); (f) the XPath *comparison and set operators* $\mathbf{XPO} = \{!=, <, <=, >, >=, |, =, union, intersect\}$; (g) the XPath *built-in functions* $\mathbf{XPF} = \{empty, exists, length, \ldots\}$; and (h) a set of *constants* $\mathbf{C}$.

The *root* node is the first node of an XPath expression. A node a is a *leaf* node if it has no successors (i.e., it is the last XPath node). For example, in the XPath expression $xp = /n_1/n_2/.../n_v$, the nodes n_1 and n_v correspond to the *root* and *leaf* nodes respectively. Moreover, n_1 is *parent* of n_2 and n_2 is *child* of n_1. The *length* of an XPath is the number of the successive nodes when traversing the path from the beginning (the length of an XPath including only the root node is one). The function *length*: $\mathbf{XP} \longrightarrow \mathbb{N}^*$ assigns a length $z \in \mathbb{N}^*$ to an XPath $xp \in \mathbf{XP}$. The function *leaf*: $\mathbf{XP} \longrightarrow \mathbf{N}$ assigns the name of the leaf node $n \in \mathbf{N}$ to an XPath $xp \in \mathbf{XP}$. For example, let the XPath $xp = /n_1/n_2/.../n_n$, then $length(xp) = n$ and $leaf(xp) = n_n$. For the XPath expression xp with $length(xp) = n$ we define as $xp(i)$ $(1 \leq i \leq n)$ the ith node xp, with $xp(1)$ being the root node. In case of predicate existences in the ith node, $x(i)$ refers both to the ith node and to the predicates. As an example, let the XPath $xp = /a/b/c[./d = 10]/@e$, then, $x(1) = a$, $x(2) = b$, $x(3) = c$, $x(4) = c[./d = 10]$ and $x(5) = @e$.

In what follows, we introduce the notions required in order to specify the semantics of the wildcards (*) and predicates ([ ]) operators while handling the XPath expression.

Definition 1. (Loosely Equal Nodes) Two XPath nodes v and w are defined to be *loosely equal*, denoted as $v \doteq w$ if and only if: (a) v' and w' result, respectively, from v and w if the predicates [ ] are removed; and (b) $(v' = w')$ or $(v' = * \text{ or } w' = *)$.

Intuitively, two XPath nodes are *loosely equal nodes* if they are the same when we do not consider their predicates, or at least one of them is the wildcard (*) node.

Definition 2. (Loosely Equal XPaths) Two XPaths x and y are defined to be *loosely equal*, denoted as $x \approx y$ if and only if: (a) they have equal lengths: $length(x) = length(y) = n$; and (b) $\forall i \in 1, ..., n \Rightarrow x(i) \doteq y(i)$.

Intuitively, two XPath nodes are, *loosely equal* XPaths if they have the same length and all their nodes are *loosely equal nodes*.

Definition 3. (Prefix XPath) An XPath x is defined to be a *prefix* of an XPath y, denoted as $x \tilde{\subset} y$ if and only if: $\exists i : i \leq l$ and $x(j) \doteq y(j)$, $\forall j \in 1, \ldots, i$ with $l = length(x)$ where $length(x) \leq length(y)$.

Intuitively, an XPath x is *prefix* of another XPath y, if a part of x starting from the beginning of x is a *loosely equal* XPath of a path starting from the beginning of y.

Definition 4. (k–Prefix XPath) An XPath x is defined to be a *k–prefix* of an XPath y, denoted as $x \overset{k}{\tilde{\subset}} y$ if and only if: $\exists k : k \leq l$ and $x(i) \doteq y(i)$, $\forall i \in 1, \ldots, k$ with $l = length(x)$ where $length(x) \leq length(y)$.

Intuitively, an XPath x is *k–prefix* of another XPath y, if a part of *length* k of x starting from the beginning of x is a *loosely equal* XPath to a part of y (of k *length*) starting from the beginning of y.

Finally, we introduce the XPath Set notion.

Definition 5. (XPath Set) The set $\mathbf{XPS} = \{xp_1, xp_2, \ldots, xp_n\}$, where $xp_i \in \mathbf{XP}$ is defined to be an *XPath Set*.

5.4.1.2 XPath Set Operators

In this section, we introduce and formally define a collection of XPath Set operators used for handling XPath Sets.

Common Ancestors Operator. The *Common Ancestors* operator is a binary operator written as $\mathbf{X} \lessdot \mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are XPath Sets. The result of this operator is the XPath Set that contains the members (XPaths) of the left set $\mathbf{X}$, which are prefixes of members (i.e., have the same ancestors) of the right set $\mathbf{Y}$. The operator is formally defined as:

$$\mathbf{X} \lessdot \mathbf{Y} = \{z : z = x_i \mid \exists y_j \in \mathbf{Y} : x_i \overset{k_i}{\tilde{\subset}} y_j\}, \text{ where } x_i \in \mathbf{X} \text{ and } length(x_i) = k_i$$

Descendants of Common Ancestors Operator. The *Descendants of Common Ancestors* operator is a binary operator written as $\mathbf{X} \gtrdot \mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are XPath Sets. The result of this operator is the XPath Set that contains the members (XPaths) of the right set $\mathbf{Y}$, the prefix XPaths of which are members of the left set $\mathbf{X}$. The operator is formally defined as:

$$\mathbf{X} \gtrdot \mathbf{Y} = \{z : z = y_j \mid \exists x_i \in \mathbf{X} : x_i \overset{k_i}{\tilde{\subset}} y_j\}, \text{ where } x_i \in \mathbf{X} \text{ and } length(x_i) = k_i$$

Suffixes of Common Ancestors Operator. The *Suffixes of Common Ancestors* operator is a binary operator written as $\mathbf{X} \gg \mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are XPath Sets. The result of this operator is the XPath Set that contains the suffix parts of the members of the right set $\mathbf{Y}$, the prefix XPaths of which are contained in the left set $\mathbf{X}$ (i.e., XPaths contained in $\mathbf{Y}$ with their ancestors contained in $\mathbf{X}$). A suffix part of a $\mathbf{Y}$ member is formed by removing the XPath parts corresponding to the lengthiest prefix XPath included in $\mathbf{X}$. The operator is formally defined as:

$$\mathbf{X} \gg \mathbf{Y} = \{z : z = /y_j(k_i+1)/y_j(k_i+2)/.../y_j(k_j) \mid \exists x_i \in \mathbf{X} : x_i \overset{k_i}{\tilde{\subset}} y_j \text{ and}$$
$$\nexists x'_i \in \mathbf{X} : x'_i \overset{k'_i}{\tilde{\subset}} y_j, k_i \leq k'_i\}, \text{ where } x_i \in \mathbf{X}, y_j \in \mathbf{Y}, \text{ and } length(x_i) = k_i,\ length(x_j) = k_j,$$
$$k_i < k_j$$

XPath Set Union Operator. The *XPath Set Union* operator is a binary operator written as $\mathbf{X} \overline{\bigcup} \mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are XPath Sets. The result of this operator differs from the result of the classic set theory Union operator when a member of $\mathbf{X}$ and/or $\mathbf{Y}$ includes the wildcard operator (*) or predicates ([ ]). In these cases the more specific XPaths are excluded from the result set.

In order to formally define the XPath Set Union operator, we firstly introduce some special union operators: (a) the *Node Union* operator among XPath nodes; and (b) the *Loose XPath Union* operator among loosely equal XPaths (Definition 2). These operators are going to be exploited in the definition of the XPath Set Union operator among XPath Sets.

(a) The *Node Union* operator is a binary operator written as $v \dot{\vee} w$, where v and w are nodes. Let e, $e1$ and $e2$ be XPath expressions. The operator is formally defined as:

$$v \dot{\vee} w = \begin{cases} * & \text{if } (v = *) \text{ or } (w = *) \\ k & \text{if } (v = k) \text{ or } (w = k) \\ k[e] & \text{if } (v = k[e] \text{ or } w \neq *) \text{ or } (w = k[e] \text{ or } v \neq *) \\ k[e1 \mid e2] & \text{if } (v = k[e1]) \text{ or } (w = k[e2]) \end{cases}$$

(b) The *Loose XPath Union* operator is a binary operator written as $x \tilde{\vee} y$ and is applied to $x, y \in \mathbf{XP}$ when x and y are loosely equal i.e., $x \approx y$. The operator is formally defined as:

$$x \tilde{\vee} y = \{z : z = \text{`/'}x(1) \dot{\vee} y(1)\text{`/'}x(2) \dot{\vee} y(2)\text{`/'} \ldots \text{`/'}x(n) \dot{\vee} y(n), \text{ where}$$
$$n = length(x) = length(y)\}$$

Finally, the *XPath Set Union* operator is formally defined as:

$$\mathbf{X} \bar{\bigcup} \mathbf{Y} = \{z : z = x \tilde{\vee} y \text{ if } x \approx y\} \bigcup \{x \in \mathbf{X} \text{ if not } x \approx y, \forall y \in \mathbf{Y}\} \bigcup \{y \in \mathbf{Y} \text{ if not } y \approx x \forall x \in \mathbf{X}\}$$

XPath Set Intersection Operator. The *XPath Set Intersection* operator is a binary operator written as

In order to formally define the XPath Set Intersection operator, we firstly introduce some special intersection operators: (a) the *Node Intersection* operator among XPath nodes; and (b) the *Loose XPath Intersection* operator among loosely equal XPaths (Definition 2). These operators are going to be exploited in the definition of the XPath Set Intersection operator among XPath Sets.

(a) The *Node Intersection* operator is a binary operator written as $v \dot{\wedge} w$, where v and w are nodes. Let e, $e1$and $e2$ be XPath expressions. Formally the operator is defined as:

$$v \dot{\wedge} w = \begin{cases} * & \text{if } (v = *) \text{ or } (w = *) \\ k & \text{if } (v = k \text{ or } w \neq k[e]) \text{ or } (w = k \text{ or } v \neq k[e]) \\ k[e1] & \text{if } (v = k[e1] \text{ or } w \neq k[e2]) \text{ or } (w = k[e1] \text{ or } v \neq k[e2]) \\ k[e1][e2] & \text{if } (v = k[e1]) \text{ or } (w = k[e2]) \end{cases}$$

(b) The *Loose XPath Intersection* operator is a binary operator written as $x \tilde{\wedge} y$, is applied to x, $y \in \mathbf{XP}$ when x, and y are loosely equal i.e., $x \approx y$. The operator is formally defined as:

$$x \tilde{\wedge} y = \{z : z = \text{`/'} x(1) \dot{\wedge} y(1) \text{`/'} x(2) \dot{\wedge} y(2) \text{`/'} \ldots \text{`/'} x(n) \dot{\wedge} y(n), \text{ where } n = length(x) = length(y)\}$$

Finally, the *XPath Set Intersection* operator is formally defined as:

$$\mathbf{X} \bar{\bigcap} \mathbf{Y} = \begin{cases} x \tilde{\wedge} y & \text{if } x \approx y \\ \varnothing & \text{elsewhere} \end{cases}$$

XPath Set Concatenation Operator. The *XPath Set Concatenation* operator is a binary operator written as $\mathbf{X} \oplus \mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are XPath Sets. The result of this operator is the set that contains the XPaths formed by appending a member of $\mathbf{Y}$ on every member of $\mathbf{X}$. The operator is formally defined as:

$$\mathbf{X} \oplus \mathbf{Y} = \{z : z = x \textit{ concatenate } y, \forall x \in \mathbf{X}, \forall y \in \mathbf{Y}\}.$$

5.4.1.3 Basic XML Schema & Ontology Constructs

Here, we specify the basic XML Schema and ontology constructs involved in the proposed mapping model.

Let an XML Schema XS; (a) $\mathbf{XT}$ is the set of the (complex and simple) *Types* defined in XS. Let $\mathbf{XST}$ be the set of the *Simple Types* of XS and $\mathbf{XCT}$ be the set of the *Complex Types* of XS. Then, $\mathbf{XT} = \mathbf{XST} \cup \mathbf{XCT}$; (b) $\mathbf{XE}$ is the set of the *Elements* defined in XS; and (c) $\mathbf{XAttr}$ is the set of the *Attributes* defined in XS. As *XML Schema Constructs* we defined the set $\mathbf{XC} = \mathbf{XT} \cup \mathbf{XE} \cup \mathbf{XAttr}$. Let xc_1, $xc_2 \in \mathbf{XC}$ be XML constructs. We denote by $xc_1.xc_2$ that the definition of xc_2 is nested in the definition of xc_1.

Let also an OWL Ontology OL; (a) $\mathbf{C}$ is the set of the OL *Classes*; (b) $\mathbf{DT}$ is the set of the OL *Datatypes*; and (c) $\mathbf{Pr}$ is the set of the (datatype and object) OL *Properties*. Let $\mathbf{OP}$ be the set of the OL *Object Properties* and $\mathbf{DTP}$ be the set of the OL *Datatype Properties*. Then, $\mathbf{Pr} = \mathbf{DTP} \bigcup \mathbf{OP}$. As *Ontology Constructs* we define the set $\mathbf{OC} = \mathbf{C} \bigcup \mathbf{DT} \bigcup \mathbf{Pr}$.

In addition, we define a function $Domain : \mathbf{Pr} \longrightarrow \mathcal{P}(\mathbf{C})$, which assigns the powerset (i.e., $\mathcal{P}$) of $\mathbf{C}$ as domain to a property $pr \in \mathbf{Pr}$. We also define a function $Range : \mathbf{Pr} \longrightarrow \mathbf{A}$, which assigns the range $\mathbf{r} \subseteq \mathbf{A}$ to an ontology property $pr \in \mathbf{P}$ where $\mathbf{A} = \mathbf{DT}$ if $pr \in \mathbf{DTP}$ and $\mathbf{A} = \mathcal{P}(\mathbf{C})$ if $pr \in \mathbf{OP}$. The image (i.e., range) of the functions Range and Domain of a Property pr (i.e., $Domain(pr)$ and $Range(pr)$), are denoted, for the sake of simplicity, as $pr.domain$ and $pr.range$.

5.4.2 Schema Mappings

In this section we define the *Schema Mappings*, which are used to define associations between disparate schema structures over ontologies and XML Schemas in the context of SPARQL to XQuery translation. In our mapping model, the schema mappings may be also enriched with data level information (e.g., conditions over data values), resulting into precise and flexible mappings. Note that since in our context SPARQL queries expressed over ontologies are translated to XQuery queries expressed over XML Schemas, the schema mappings are defined in a directional way from ontologies to XML Schemas.

Given an ontology OL and an XML Schema XS, let $\mathbf{oc}$ be a set of OL constructs and $\mathbf{xc}$ a set of XS constructs. A *Schema Mapping* (μ_S) between OL and XS is an expression of the form:

$$\mu_S : OE \overset{\mathbf{E}}{\mapsto} XE$$

where OE is an expression containing $\mathbf{oc}$ constructs, conjunctions ($\bigwedge$) or/and disjunctions ($\bigvee$), XE is an expression containing $\mathbf{xc}$ constructs, conjunctions or/and disjunctions and $\mathbf{E}$ is a set of conditions applied over the $\mathbf{xc}$ members.

A schema mapping represents a association among $\mathbf{oc}$ and $\mathbf{xc}$ under the conditions specified in $\mathbf{E}$. We can simply say that the $\mathbf{oc}$ members are mapped to the $\mathbf{xc}$ members under the conditions specified in $\mathbf{E}$. The $\mathbf{E}$ conditions can be simply considered as tree expressions applied over the $\mathbf{xc}$ constructs.

In more detail, a mapping condition $e \in \mathbf{E}$ is a *tree expression* referring to XS constructs and/or XML data that follow XS. In particular, a mapping condition e is *applied* on a set of XML Schema constructs $xca \subseteq \mathbf{xc}$ and it may also *refer* (i.e., include) to several constructs independent on xca. In addition, a condition e may contain (a) tree paths, (b) operators and functions (e.g., *intersection*, *union*, <, >, =, ≠, *ends-with*, *concat*), as well as (c) constants (e.g., 25, 3.4, "John"). It is remarkable that every XML Schema construct can be referred in a condition expression. Moreover, a mapping condition e may be applied to specific constructs or may be applied to the whole XE expression. To sum up, a schema mapping condition e could be any condition which can be expressed in XPath syntax [17]; this way, the high expressiveness of the XPath expressions (including the built-in functions [18]) may be exploited in a mapping condition, and, together with the flexibility of applying independent conditions over different XML constructs, it leads to rich, flexible and expressive schema mappings.

For example, let c be an ontology class and w, z be XML Schema complex types. In addition, let the conditions e_1 and e_2 be applied, respectively, over w and z (denoted as $w\langle\!\langle e_1 \rangle\!\rangle$, $z\langle\!\langle e_2 \rangle\!\rangle$) and a condition e_3 applied over the whole XE expression (not over a specific construct). A schema mapping μ_S of the class c to the disjunction of the complex types w and z under the conditions e_1 and e_2, respectively on w and z, and both under the condition e_3 is denoted as: $\mu_S : c \overset{\{e_3,\, w\langle\!\langle e_1 \rangle\!\rangle,\, z\langle\!\langle e_2 \rangle\!\rangle\}}{\mapsto} w \bigvee z$, where, according

to the schema mapping definition, c is the OE expression, $w \bigvee z$ is the XE expression and $\{e_3, w\langle\langle e_1 \rangle\rangle, z\langle\langle e_2 \rangle\rangle\}$ is the condition set **E**. Since the condition e_3 is not applied over a specific construct (i.e., it is applied over all the constructs included in XE), it holds that $\mathbf{E} = \{w\langle\langle e_1 \bigwedge e_3 \rangle\rangle, z\langle\langle e_2 \bigwedge e_3 \rangle\rangle\}$.

Regarding the ontology properties, let pr be an ontology property and q be an XML Schema element or attribute. The schema mapping μ_S:$pr \mapsto q$ corresponds to $pr.domain \mapsto d$ and $pr.range \mapsto q$, where d is the (complex) XML element in which q is defined. In addition, the domain and range of an ontology property pr, might be individually mapped to different XML Schema elements/attributes. For instance, let q, v be XML Schema elements/attributes, then μ_{S_1}:$pr.domain \mapsto q$ and μ_{S_2}:$pr.range \mapsto v$.

We can also observe from Figure 6.4 that the following three schema mappings are obtained: μ_{S_1}:$Class\ 1 \mapsto Type\ H$, μ_{S_2}:$Class\ 2 \overset{\{Z\langle\langle e_1 \rangle\rangle\}}{\mapsto} Type\ Z$ and μ_{S_3}:$Object\ Property\ 1 \mapsto Complex\ Element\ Y$.

5.4.2.1 Schema Mapping Specification

In the first SPARQL2XQuery scenario, where the XS2OWL component is exploited, the schema mappings between the constructs of the XML Schemas and the generated ontologies are automatically specified through the XS2OWL transformation process (Section 5.3.1). Note that in this case, none of the schema mappings is conditional (i.e., the condition set **E** is equal to the empty set).

We have presented in Figure 6.3 (with dashed grey lines) the automatically specified schema mappings of the schema transformation example of Section 5.3.2. Note that the arrows represent the schema transformation process and the schema mappings follow the inverse direction. In this example (Figure 6.3), we can observe several schema mappings, for instance: μ_{S_1}:$Person\_Type \mapsto Person$, μ_{S_2}:$Dept\_xs\_string \mapsto Student.Dept$, μ_{S_3}:$SSN\_xs\_integer \mapsto Person.SSN \bigvee Student.SSN$, etc. For each of these schema mappings, the condition set **E** is equal to the empty set and is omitted.

In the second SPARQL2XQuery scenario, an existing ontology is manually mapped to an XML Schema by a domain expert. The mapping process is guided by the language level correspondences (summarized in Table 6.4), which have also been adopt-ed by the XS2OWL transformation model. For example, ontology classes can associated with XML Schema complex types, ontology object properties with XML elements of complex type, etc. Then, at the schema level, schema mappings between the ontology and XML Schema constructs have to be manually specified (e.g., the person class is mapped to the *person_type* complex type), following the language level correspondences.

Example 9. In Figure 5.6, we present an example of the manual mapping specification scenario, where two existing ontologies that describe the data of two organizations (*Organization A* and *Organization Z*) have been manually mapped to an XML Schema. The mappings are presented with dashed grey lines.

In this example, the XML Schema is an extension of the previously presented Persons XML Schema (Figure 6.2). Here, the Persons schema has been extended by adding the complex element *Courses* of type *Couses_Type* as a sub-element of the Student element. The Courses element contains two simple sub-elements, *ID* and *Grade*, of type xs:integer and xs:float respectively. These extensions were made in order to be able to define more complex manual mappings in our examples.

Regarding the involved ontologies, the ontology of Organization A has the *AGUFIL Group* class, where AGUFIL stands for "Adult Gmail Users with the First name Identical to Last name". Moreover, the ontology of Organization Z has the *MIT CS Student* class which describes the Computer Science Students of the MIT institute. Each of these ontologies has several (self-explained) properties.

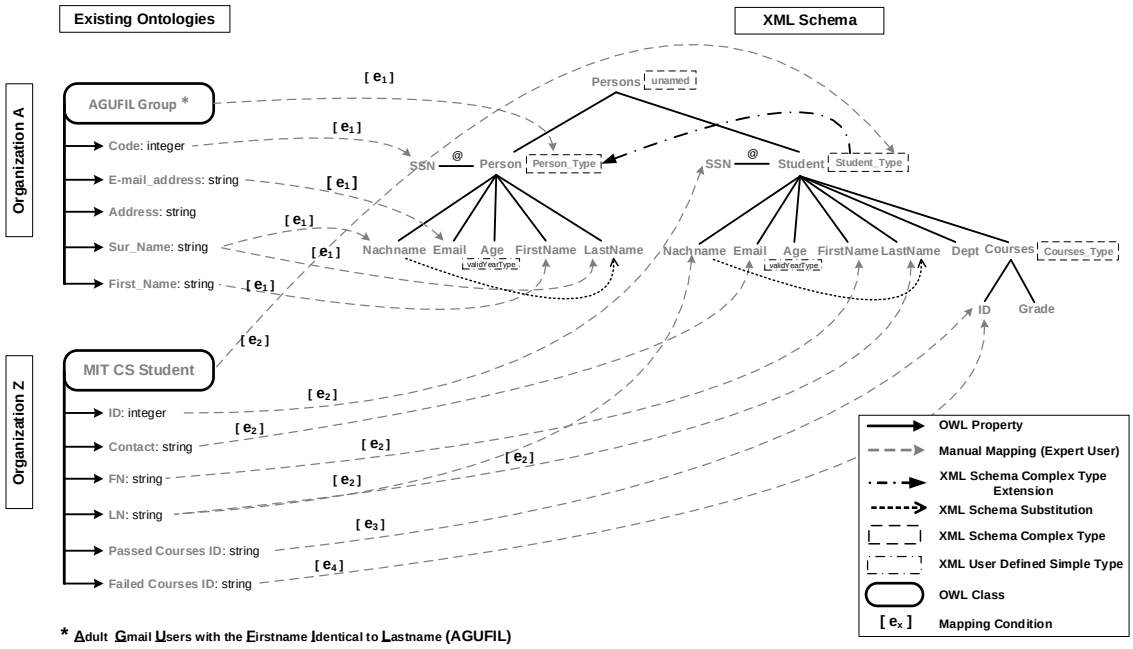


Figure 5.6: Manually specified schema mappings between existing ontologies and an XML Schema (extension of the Persons XML Schema). A domain expert has manually mapped the ontologies to the XML Schema. The mappings are drawn with dashed grey lines.

We can observe from Figure 5.6 that several schema mappings can be obtained. For instance, the class AGUFIL Group from Organization A can be mapped to the Person_Type XML complex type (see μ_{S_1} above), under the e_1 condition that restricts the persons to those who are older than 18 years old (i.e., are adults), their first name is the same with their last name and their email account is on the Gmail domain (i.e., ends with gmail.com).

$$\mu_{S_1}: AGUFIL\ Group \overset{\{e_1\}}{\mapsto} Person\_Type,$$
$$\text{where } e_1 \equiv Age > 18 \bigwedge FirstName = Lastname \bigwedge email.ends\text{-}with(\text{“}gmail.com\text{”})$$

In a similarly way, the class MIT CS Student from Organization Z can be mapped to the Student_Type XML complex type (μ_{S_2}), under the e_2 condition (see above) that restricts the students to those who have the CS as department and their email account is on the MIT domain (i.e., ends with mit.edu).

$$\mu_{S_2}: MIT\ CS\ Student \overset{\{e_2\}}{\mapsto} Student\_Type,$$
$$\text{where } e_2 \equiv Dept = \text{“}CS\text{”} \bigwedge email.ends\text{-}with(\text{“}mit.edu\text{”})$$

In Organization A, the ontology property Code can be mapped to the SSN attribute of the Person class under the e_1 condition (μ_{S_3}). Similarly, the property ID can be mapped to the SSN attribute of the Student class under the e_2 condition (μ_{S_4}). The Sur_Name property can be mapped to the union of the LastName and Nachname subelements of the Person element under the e_1 condition (μ_{S_5}). Finally, the same holds for the LN property and the LastName and Nachname subelements of the Person element (μ_{S_6}).

$$\mu_{S_3}: Code \overset{\{e_1\}}{\mapsto} Person.SSN$$
$$\mu_{S_4}: ID \overset{\{e_2\}}{\mapsto} Student.SSN$$
$$\mu_{S_5}: Sur\_Name \overset{\{e_1\}}{\mapsto} Person.LastName \bigvee Person.Nachname$$
$$\mu_{S_6}: LN \overset{\{e_2\}}{\mapsto} Student.LastName \bigvee Student.Nachname$$

Similarly, in Organization Z, the property Passed_Courses_ID can be mapped to the Course ID sub-element of the Student element (μ_{S_7}), under the condition e_3 that restricts the Course IDs to those belonging to students from the CS department whose email account is on the MIT domain (i.e., ends with mit.edu) and also refer to courses having a passing grade (i.e., equal or greater than 5.0). A similar mapping (μ_{S_8}) has been defined for the Failed_Courses property and the condition e_4.

$$\mu_{S_7}: Passed\_Courses\_ID \overset{\{e_3\}}{\mapsto} Student.Course.ID,$$
$$\text{where } e_3 \equiv Dept = \text{“}CS\text{”} \wedge email.ends\text{-}with(\text{“}mit.edu\text{”}) \wedge Grade \geq 5.0$$
$$\mu_{S_8}: Failed\_Courses\_ID \overset{\{e_4\}}{\mapsto} Student.Course.ID,$$
$$\text{where } e_4 \equiv Dept = \text{“}CS\text{”} \wedge email.ends\text{-}with(\text{“}mit.edu\text{”}) \wedge Grade < 5.0$$

□

5.4.3 Correspondences between XML Schema Constructs and XPath Sets — Associating Schema and Data

We have already defined the schema mappings between ontology constructs and XML Schema constructs (Section 5.4.2.1). Since we want to translate SPARQL queries into XQuery expressions that are evaluated over XML data, we should identify the correspondences between the ontology constructs (referred in the SPARQL queries) and the XML data, with respect to the predefined schema mappings. In this section we attempt to express the associations that hold between the XML Schema constructs and the XML data nodes using XPath Set expressions.

At the data level, the XML data is valid with respect to the XML Schema(s) it follows. As a result, for each XML Schema construct we can identify its corresponding XML data nodes and address them using XPath expressions. In this way, we can define the associations between XML schema constructs and XML data.

Given a SPARQL query, for all the ontology constructs referred in the query: (a) we identify the XML Schema constructs based on the predefined schema mappings; and (b) we determine the corresponding XPath Sets for the identified XML Schema constructs. As a result, the ontology constructs referred in the SPARQL query are directly associated with XML data through XPaths.

Formally, let D be an XML dataset, valid with respect to an XML Schema XS. A correspondence of an XML Schema construct xc to an XPath Set xps is a function $cXPS$: $\mathbf{XC} \longrightarrow \mathbf{XPS}$ that assigns the XPath Set $xps \in \mathbf{XPS}$ to the XML construct $xc \in \mathbf{XC}$, where xps addresses all the corresponding XML nodes of xc in D.

For example, we can observe from Figure 6.4 that we have the following three XML Schema Constructs to XPath Set Correspondences: $cXPS(Type\ H) = \{/\ldots/X\}$, $cXPS(Type\ Z) = \{/\ldots/X/Y\}$ and $cXPS(Complex\ Element\ Y) = \{/\ldots/X/Y\}$.

Table 6.7 summarizes the correspondences between the “XML part” (i.e., referring to the XML Schema constructs) of the schema mapping expressions and XPath Sets. In the left column of the Table 6.7, w and z are XML Schema constructs, e and p XML conditions and ce an expression comprised of XML conditions e_i, with $i \geq 1$ and possibly conjunctions and disjunctions. In the right column, xe, xp and xce are the XPath expressions corresponding to the e, p and ce conditions respectively. Notice that the conditions are applied over the XPaths using the XPath predicates “[ ]”.

5.4.4 Schema Mapping Representation

In the previous sections we have defined the associations at the schema level (Schema Mappings – Section 5.4.2), as well as the associations between the XML Schema and the XML data (Correspondences between XML Schema Constructs and XPath Sets –

Table 5.6: Correspondences between schema mapping expressions ("XML part") and XPath Sets

Schema Mapping Expression	XPath Set Correspondences
w	$cXPS(w)$
$e \bigvee p$	$xe \overline{\bigcup} xp$
$e \bigwedge p$	$xe \overline{\bigcap} xp$
$w\langle\langle ce \rangle\rangle$	$cXPS(w)[xce]$
$w \bigvee z$	$cXPS(w) \overline{\bigcup} cXPS(z)$
$w \bigwedge z$	$cXPS(w) \overline{\bigcap} cXPS(z)$
$(w \bigvee z)\langle\langle ce \rangle\rangle$	$cXPS(w)[xce] \overline{\bigcup} cXPS(z)[xce]$
$(w \bigwedge z)\langle\langle ce \rangle\rangle$	$cXPS(w)[xce] \overline{\bigcap} cXPS(z)[xce]$

Section 5.4.3). Here we exploit these associations in order to define and represent the schema mappings in the context of SPARQL to XQuery translation.

In particular, we specify the association of the ontology constructs with XPath Sets through the exploitation of (a) the predefined *schema mappings* between the ontology and XML Schema constructs; and (b) the determined XPath Set for the mapped XML constructs (i.e., *correspondences between XML Schema constructs and XPath Sets*).

To sum up, a mapping in the context of SPARQL to XQuery translation (or simply a *mapping*) is represented as the association of an ontology construct with XPath Sets. Thus, this *mapping* representation forms a "direct association" between the ontology constructs and the XML data using XPath expressions.

Formally, given an ontology OL and an XML Schema XS, let **oc** be a set of OL constructs, **xc** a set of XS constructs and μ_S a schema mapping between **oc** and **xc**. A *Mapping* (μ) between OL and XS in the context of the SPARQL to XQuery translation is an expression of the form:

μ: $oc \equiv sxps$, where $sxps$ is an XPath Set corresponding to **xc** constructs under the schema mapping μ_S.

In the rest of the chapter, for every ontology class c, the associated XPath Set is denoted as $\mathbf{X}_c$ (*Class XPath Set*). In addition, for every ontology property *pr*, the associated XPath Set is denoted as $\mathbf{X}_{pr}$ (*Property XPath Set*). Furthermore, for the *pr* domains and ranges, the associated XPath Sets are denoted as $\mathbf{X}_{prD}$ (*Property Domains XPath Set*) and $\mathbf{X}_{prR}$ (*Property Ranges XPath Set*) respectively.

Example 10. Consider the schema mappings μ_{S_2}, μ_{S_3}, μ_{S_5}, μ_{S_6}, μ_{S_7} and μ_{S_8} of Example 9 between the ontology and the XML Schema presented in Figure 5.6. The representations of these schema mappings in the context of SPARQL to XQuery translation are listed below:

$$\mu_{S_2}: MIT\ CS\ Student \equiv X_{MIT\ CS\ Student} = \{/Persons/Student[./Dept = \text{“}CS\text{”} \text{ and } ends\text{-}with(./email, \text{“}mit.edu\text{”})]\}$$

$$\mu_{S_3}: Code \equiv X_{Code} = \{/Persons/Person[./Age > 18 \text{ and } ./firstName = ./Lastname \text{ and } ends\text{-}with(./email, \text{“}gmail.com\text{”})]/@SSN\}$$

$$\mu_{S_5}: Sur\_Name \equiv X_{Sur\_Name} = \{/Persons/Person[./Age > 18 \text{ and } ./firstName = ./Lastname \text{ and } ends\text{-}with(./email, \text{“}gmail.com\text{”})]/LastName\}$$

$$\mu_{S_6}: LN \equiv X_{LN} = \{/Persons/Student[./Dept = \text{“}CS\text{”} \text{ and } ends\text{-}with(./email, \text{“}mit.edu\text{”})]/LastName\}$$

$$\mu_{S_7}: Passed\_Courses\_ID \equiv X_{Passed\_Courses\_ID} = \{/Persons/Student[./Dept = \text{“}CS\text{”} \text{ and } ends\text{-}with(./email, \text{“}mit.edu\text{”}) \text{ and } ./Courses/Grade \geq 5.0]/Courses/ID\}$$

$$\mu_{S_8}: Failed\_Courses\_ID \equiv X_{Failed\_Courses\_ID} = \{/Persons/Student[./Dept = \text{“}CS\text{”} \text{ and } ends\text{-}with(./email, \text{“}mit.edu\text{”}) \text{ and } ./Courses/Grade < 5.0]/Courses/ID\}$$

□

5.4.5 Automatic Mapping Generation

In the first SPARQL2XQuery scenario, the mappings are automatically generated. In particular, the generation of the mappings is carried out by the *Mapping Generator* component, which takes as input an XML Schema and the OWL ontology generated by XS2OWL for this XML Schema. In the first phase, the Mapping Generator component parses the input files and obtains the schema mappings between the XML Schema and the generated ontology by exploiting the XS2OWL Transformation Model. Then, using the XML Schema, the Mapping Generator component determines the *XML Schema construct to XPath Set Correspondences* for all the XML constructs. Finally, the component generates an XML document that contains the associations of all the ontology constructs with the XPath Sets. In particular, it generates the sets $\mathbf{X}_c$, $\mathbf{X}_{pr}$, $\mathbf{X}_{prD}$ and $\mathbf{X}_{prR}$ for all the ontology classes and properties.

Example 11. Consider the XML Schema of Figure 6.2 and the corresponding ontology generated by XS2OWL (Table 6.5 and Table 6.6). Based on the automatically specified schema mappings (Figure 6.3), the Mapping Generator component generates the mapping representations listed below. It should be mentioned that in this case the mappings are trivial, since the ontology is an OWL representation of the XML Schema.

Generated Mappings between the XML Schema and the Ontology of Figure 4

Classes:

Person_Type ≡ X_{Person_Type} = { /Persons/Person }

Student_Type ≡ $X_{Student\_Type}$ = { /Persons/Student }

NS_Persons_UNType ≡ $X_{NS\_Persons\_UNType}$ = { /Persons }

Object Properties:

Persons__NS_Persons_UNType ≡ $X_{Persons\_\_NS\_Persons\_UNType}$ = { /Persons }
Persons__NS_Persons_UNType.domain ≡ $X_{Persons\_\_NS\_Persons\_UNTypeD}$ = = { /Persons }
Persons__NS_Persons_UNType.range ≡ $X_{Persons\_\_NS\_Persons\_UNTypeR}$ = = { /Persons }

Person__Person_Type ≡ $X_{Person\_\_Person\_Type}$ = { /Persons/Person }
Person__Person_Type.domain ≡ $X_{Person\_\_Person\_TypeD}$ = { /Persons }
Person__Person_Type.range ≡ $X_{Person\_\_Person\_TypeR}$ = { /Persons/Person }

Student__Student_Type ≡ $X_{Student\_\_Student\_Type}$ = { /Persons/Student }
Student__Student_Type.domain ≡ $X_{Student\_\_Student\_Type\ D}$ = { /Persons }
Student__Student_Type.range ≡ $X_{Student\_\_Student\_TypeR}$ = = { /Persons/Student }

Datatype Properties:

FirstName__xs_string ≡ $X_{FirstName\_\_xs\_string}$={ /Persons/Person/FirstName, /Persons/Student/FirstName }
FirstName__xs_string.domain ≡ $X_{FirstName\_\_xs\_stringD}$={ /Persons/Person, /Persons/Student }
FirstName__xs_string.range ≡ $X_{FirstName\_\_xs\_stringR}$={ /Persons/Person/FirstName, /Persons/Student/FirstName }

LastName__xs_string ≡ $X_{LastName\_\_xs\_string}$ = { /Persons/Person/LastName, /Persons/Student/LastName }
LastName__xs_string.domain ≡ $X_{LastName\_\_xs\_stringD}$ = { /Persons/Person, /Persons/Student }
LastName__xs_string.range ≡ $X_{LastName\_\_xs\_stringR}$ ={ /Persons/Person/LastName, /Persons/Student/LastName }

Age__xs_integer ≡ $X_{Age\_\_xs\_integer}$ = { /Persons/Person/Age, /Persons/Student/Age}
Age__xs_integer.domain ≡ $X_{Age\_\_xs\_integerD}$ = { /Persons/Person, /Persons/Student }
Age__xs_integer.range ≡ $X_{Age\_\_xs\_integerR}$ = { /Persons/Person/Age, /Persons/Student/Age}

Email__xs_string ≡ $X_{Email\_\_xs\_string}$ = { /Persons/Person/Email, /Persons/Student/Email}
Email__xs_string.domain ≡ $X_{Email\_\_xs\_stringD}$ = { /Persons/Person, /Persons/Student }
Email__xs_string.range ≡ $X_{Email\_\_xs\_stringR}$ = { /Persons/Person/Email, /Persons/Student/Email }

Nachname__xs_string ≡ $X_{Nachname\_\_xs\_string}$ = { /Persons/Person/Nachname, /Persons/Student/Nachname }
Nachname__xs_string.domain ≡ $X_{Nachname\_\_xs\_stringD}$ = { /Persons/Person, /Persons/Student }
Nachname__xs_string.range ≡ $X_{Nachname\_\_xs\_stringR}$ ={ /Persons/Person/Nachname, /Persons/Student/Nachname }

SSN__xs_integer ≡ $X_{SSN\_\_xs\_integer}$ = { /Persons/Person/@SSN, /Persons/Student/@SNN}
SSN__xs_integer.domain ≡ $X_{SSN\_\_xs\_integerD}$ = { /Persons/Person, /Persons/Student }
SSN__xs_integer.range ≡ $X_{SSN\_\_xs\_integerR}$ ={ /Persons/Person/@SSN, /Persons/Student/@SNN}

Dept__xs_string ≡ $X_{Dept\_\_xs\_string}$ = { /Persons/Student/Dept }
Dept__xs_string.domain ≡ $X_{Dept\_\_xs\_stringD}$ = { /Persons/Student }
Dept__xs_string.range ≡ $X_{Dept\_\_xs\_stringR}$ = { /Persons/Student/Dept }

□

5.5 Introducing the Query Translation Process

In this section, we give an overview of the SPARQL query language (Section 5.5.1), we introduce several basic notions (Section 5.5.2), and finally we summarize the query translation process (Section 5.5.3).

5.5.1 SPARQL Query Language Overview

SPARQL [34] is a W3C recommendation and it is today the standard query language for RDF data. The evaluation of a SPARQL query is based on graph pattern matching. The SPARQL *Where* clause consists of a *Graph Pattern*. The Graph Pattern is defined recursively and contains *Triple patterns* and SPARQL operators. The operators of the SPARQL algebra that can be applied on Graph Patterns are: AND, UNION, OPTIONAL and FILTER. Triple patterns are just like RDF triples but each of the *subject*, *predicate* and *object* parts may be a variable.

SPARQL allows four query forms: *Select*, *Ask*, *Construct* and *Describe*. In addition, SPARQL provides various solution sequence modifiers that can be applied on the initial solution sequence in order to create another, user desired, sequence. The supported SPARQL solution sequence modifiers are: *Distinct*, *Order By*, *Reduced*, *Limit*, and *Offset*. Finally, the SPARQL query results may be *RDF Graphs*, *SPARQL solution sequences* and *Boolean* values.

5.5.1.1 RDF and SPARQL Syntax

In this section, we provide a set of formal definitions of the syntax of RDF and SPARQL (based on [304] and [314]). Let **I** be the set of the IRIs (Internationalized Resource Identifiers), **L** the set of the RDF Literals, and **B** be the set of the Blank nodes. In addition, assume the existence of an infinite set **V** of variables disjoint from the previous sets (**I**, **B**, **L**).

Definition 6. (RDF Triple) A triple $\langle s, p, o\rangle \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is called *RDF triple*. s, p and o represent, respectively, the *subject*, *predicate* and *object* of an RDF triple. The subject s can either be an IRI or a Blank node. The predicate p must be an IRI. The object o can be an IRI, a Blank node or an RDF Literal.

Definition 7. (RDF Dataset) An *RDF Dataset* (or RDF Graph) is a set of RDF triples.

Definition 8. (RDF Triple) A triple $\langle s, p, o\rangle \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$ is called *Triple pattern*.

In the rest of the chapter, when we refer to variables, we also refer to Blank nodes, since they are semantically equivalent.

Definition 9. (Graph Pattern) A *Graph Pattern* (GP) is a SPARQL graph pattern expression defined recursively as follows: (a) A triple pattern is a graph pattern. (b) If P_1 and P_2 are graph patterns, then the expressions (P_1 AND P_2), (P_1 OPT P_2), and (P_1 UNION P_2) are graph patterns. (c) If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern.

Note that a SPARQL built-in condition (or else Filter expression) is constructed using IRIs, RDF literals, variables and constants, as well as operators (e.g., &&, ||, !, =, ! =, >, ≤, +, *bound*, *lang*, *regex*, etc.) (Refer to [314] for a complete list). With $var(gp)$ we denote the set of variables occurring in a graph pattern gp.

Definition 10. (Union–Free Graph Pattern) A SPARQL graph pattern that does not contain UNION operators is a *Union–Free Graph Pattern* (UF–GP).

Definition 11. (Basic Graph Pattern) A finite sequence of conjunctive triple patterns and possible Filters is called *Basic Graph Pattern* (BGP).

5.5.2 Query Translation Preliminaries

Here, we introduce some essential query translation notions. Let $\mathbf{I}_{RDF}$ be the set containing the IRIs of the RDF vocabulary (e.g., rdf:type, rdf:Property), $\mathbf{I}_{RDFS}$ the set containing the IRIs of the RDF Schema vocabulary (e.g., rdfs:subClassOf, rdfs:domain) and $\mathbf{I}_{OWL}$ the set containing the IRIs of the OWL vocabulary (e.g., owl:equivalentClass, owl:FunctionalProperty). Moreover, let $\mathbf{I}_{CL}$ be the set containing the IRIs of the classes of an ontology and $\mathbf{I}_{PR}$ the set containing the IRIs of the properties of an ontology.

From the above sets, we define the set $\mathbf{I}_{VC}$, containing all the IRIs of the RDF/S and OWL vocabularies $\mathbf{I}_{VC} = \mathbf{I}_{RDF} \bigcup \mathbf{I}_{RDFS} \bigcup \mathbf{I}_{OWL}$. Moreover, we define the set $\mathbf{I}_{OL}$, containing the IRIs that refer to ontology classes and properties $\mathbf{I}_{OL} = \mathbf{I}_{CL} \bigcup \mathbf{I}_{PR}$.

Definition 12. (Schema Triple Pattern) A *Schema Triple Pattern* is a triple pattern which refers to the ontology structure and/or semantics. In particular, a Schema Triple Pattern is a triple pattern that contains concepts and properties of the RDF/S and OWL vocabularies, or, a triple pattern having IRIs that refer to ontology classes or properties. Formally, a Schema Triple Pattern is defined as follows:

A triple $\langle s, p, o \rangle \in (\mathbf{I}_{VC} \bigcup \mathbf{I}_{OL} \bigcup \mathbf{B} \bigcup \mathbf{V}) \times (\mathbf{I}_{VC} \bigcup \mathbf{I}_{OL}) \times (\mathbf{I}_{VC} \bigcup \mathbf{I}_{OL} \bigcup \mathbf{B} \bigcup \mathbf{L} \bigcup \mathbf{V})$ is called *Schema Triple Pattern* (or simply *Schema Triple*).

We use $schemaTr(gp)$ to denote the set of Schema Triples occurring in a graph pattern gp.

In what follows we introduce the notion of semantically corresponding queries. Let SL_1 and SL_2 be schema definition languages, and, s_1 and s_2 be schemas expressed in SL_1 and SL_2, respectively. Let $\mathbf{M}$ be a set of mappings between s_1 and s_2. Let D_1 be a set of instances (i.e., dataset) over s_1. A *Data Transformation* (DTr) from a set of instances D_1 to a set of instances D_2 w.r.t. $\mathbf{M}$, is the transformation of D_1 into instances of s_2 w.r.t. $\mathbf{M}$; resulting D_2. Thus, the Data Transformation can be considered a function $DTr : \{D_1\} \times \mathbf{M} \longrightarrow D_2$, where D_1 and D_2 are sets of instances over the schemas s_1 and s_2, and, s_1 and s_2 are expressed in SL_1 and SL_2 schema definition languages.

Let SL_1 and SL_2 be schema definition languages, and, s_1 and s_2 be schemas expressed in SL_1 and SL_2, respectively. Let $\mathbf{M}$ be a set of mappings between s_1 and s_2. Let D_1 be a set of instances over s_1. Let $D_2 = DTr(D_1, \mathbf{M})$ the data transformation of the set of instances D_1 w.r.t. $\mathbf{M}$, where D_2 is a set of instances over s_2. Let QL_1 and QL_2 be query languages, and, Q_1 and Q_2 be queries expressed in QL_1 and QL_2, respectively. We say that Q_1 is *semantically correspondent* to Q_2 w.r.t. $\mathbf{M}$ if and only if the solutions returned from the evaluation of Q_1 over D_1 are the same as the evaluation of Q_2 over D_2.

In our problem, SL_1 and SL_2 are, respectively, the XML Schema and OWL schema definition languages. Moreover, QL_1 and QL_2 are, respectively, the XQuery and SPARQL query languages.

5.5.3 Query Translation Overview

In this section we present an overview of the SPARQL to XQuery query translation process, which is performed by the Query Translator component. The Query Translator takes as input a SPARQL query and the mappings between an ontology and an XML Schema and translates the SPARQL query to semantically corresponding XQuery expressions w.r.t. the mappings.

The query translation process is based on a generic method and a set of algorithms for translating SPARQL queries to XQuery expressions following strictly the SPARQL semantics. The translation covers all the syntax variations of the SPARQL grammar [314]; as a result, it can handle every SPARQL query. In addition, the translation process is generic and scenario independent, since the mappings are represented in an abstract formal form as XPath Sets. The mappings may be automatically generated or manually specified.

The objectives for the development of the query translation process have been the following: (a) Development of a generic method for the SPARQL to XQuery translation; (b) Capability of translating every query compliant to the SPARQL grammar; (c) Obeying strictly the SPARQL semantics; (d) Independence from query engines and storage environments; (e) Production of as simple XQuery expressions as possible; (f) Construction of XQuery expressions so that their correspondence with SPARQL can be easily understood;

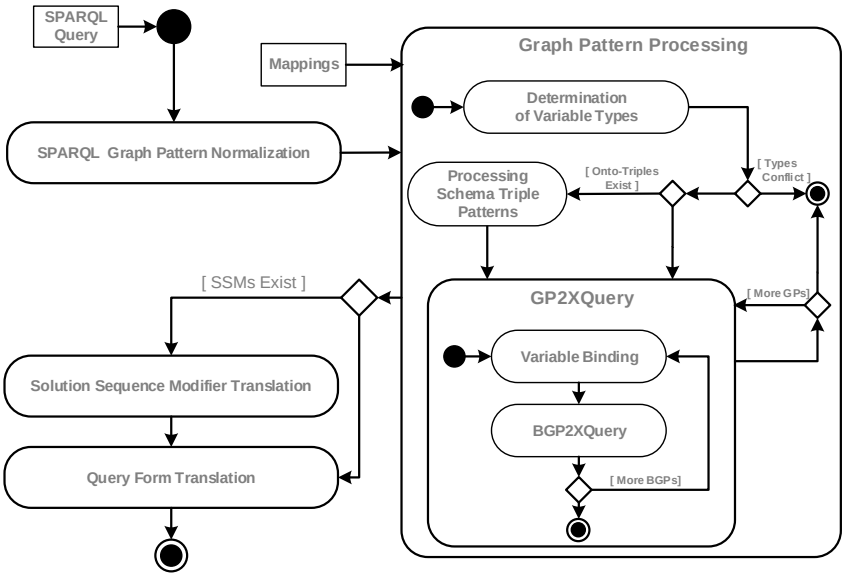


Figure 5.7: UML activity diagram describing the SPARQL to XQuery translation process. The activity takes as input a SPARQL query and the mappings between an OWL ontology and an XML Schema and generates the semantically corresponding XQuery expressions.

and (g) Construction of XQuery expressions that produce results that do not need any further processing.

Figure 5.7 presents, in the form of a UML diagram, the entire translation process. As is shown in Figure 5.7, the translation process takes as input a SPARQL query and the mappings between an ontology and an XML Schema. *The SPARQL Graph Pattern Normalization* activity (Section 5.6.1) rewrites the Graph Pattern (GP) of the SPARQL query in an equivalent normal form, resulting into a simpler and more efficient translation process. The *Graph Pattern Processing* follows, in order to translate the Graph Pattern of the SPARQL query (the query Where clause) to XQuery expressions. Afterwards, the solution sequence modifiers (SSMs) that may be contained in the query are translated (Section 5.9.1). Finally, based on the SPARQL query form, the generated XQuery is enhanced with appropriate expressions in order to achieve the desired structure of the results (e.g., either construct an RDF graph or return a SPARQL result sequence) (Section 5.9.2).

The Graph Pattern Processing is a composite activity with various sub-activities (Figure 5.7). Initially, an activity identifies the types of the SPARQL variables, in order to determine the form of the results, as well as to perform consistency checking in variable usage (Section 5.6.2). Afterwards, an activity (Section 5.6.3) processes the Schema Triples (Definition 12) that may exist in the pattern and determines the variable bindings for them (i.e., assigns the appropriate XPaths to variables). These bindings are going to be used in the next steps as initial variable bindings. Finally, a translation algorithm (*GP2XQuery*) that translates GPs to XQuery expressions is exploited (Section 5.8.2). Throughout the GP2XQuery translation, for each Basic Graph Pattern (BGP) contained in the GP, a Variable Binding phase (Section 5.7) and a BGP to XQuery translation (Section 5.8.3) are preformed.

5.6 Query Normalization, Variable Types & Schema Triples

5.6.1 SPARQL Graph Pattern Normalization

In this section, we describe the *SPARQL Graph Pattern normalization* phase, which rewrites the graph pattern (GP) of a SPARQL query, and transforms it to an equivalent

normal form. The SPARQL graph pattern normalization is based on the GP expression equivalences proved in [304] and on query rewriting techniques.

Definition 13. (Well Designed Graph Pattern) [304] A union–free graph pattern P is *well designed* if for every sub-pattern $P' = (P_1 \text{ OPT } P_2)$ of P and for every variable $?X$ occurring in P, the following condition holds: if $?X$ occurs both inside P_2 and outside P' then it also occurs in P_1.

The graph pattern equivalences differ for the well designed GPs and the non-well designed GPs[7]. Thus, in case of OPT existence, it is essential for this phase to identify if the GP is well designed or not (if OPT does not exist, GP is always well designed). This clarification is performed by validating the well design condition over the GP. Finally, every GP is transformed to a *normal form* formally described as follows:

$$\begin{gathered} P_1 \text{ UNION } P_2 \text{ UNION } P_3 \text{ UNION } \ldots \text{ UNION } P_n, \\ \text{where } Pi \ (1 \leq i \leq n) \text{ is a union-free graph pattern.} \end{gathered} \tag{1}$$

The new GP normal form allows an easier and more efficient translation process, as well as the creation of more efficient XQuery queries since: (a) The normal form contains a sequence of union–free graph patterns, each of which can be processed independently. (b) The normal form contains larger Basic Graph Patterns. The larger basic graph patterns result in a more efficient translation process, since they reduce the number of the variable bindings, as well as the BGP to XQuery translation processes that are required (more details can be found in Section 5.8). (c) The larger basic graph patterns result in more sequential conjunctions (i.e., ANDs) intrinsically handled by XQuery expressions, thus more efficient XQuery queries (more details in can be found Section 5.8). Note that in almost all cases, the "real-world" (i.e., user defined) SPARQL graph patterns are initially expressed in normal form [307], thus this phase is often avoided.

5.6.2 Variable Type Determination

In this section we describe the *variable type determination* phase. This phase identifies the type of every SPARQL variable referenced in a union–free graph pattern (UF–GP). The determined variable types are used to specify the form of the results and, consequently, the syntax of the Return XQuery clause. Moreover, the variable types are exploited for generating more efficient XQuery expressions. In particular, the variable types are exploited by the processing Schema Triple patterns and the variable binding phases, in order to reduce the possible bindings by pruning the redundant bindings. Finally, through the variable type determination, a consistency check is performed in variable usage, in order to detect possible conflicts (i.e., the same variable may be determined with different types in the same UF–GP). In such a case, the UF–GP can not be matched against any RDF dataset, thus, this UF–GP is pruned and is not translated, resulting into more efficient XQuery expressions that speed up the translation process. In Table 5.7 we define the variable types that may occur in triple patterns.

5.6.2.1 Variable Type Determination Rules

Here we describe the rules that are used for the determination of the variable types. Let OL be an ontology, UF–GP be a Union–Free Graph Pattern expressed over OL, **mDTP** (*Mapped Data Type Properties Set*) be the set of the mapped datatype properties of OL, **mOP** (*Mapped Object Properties Set*) be the set of the mapped object properties of OL,

[7] A graph pattern that is not compatible with Definition 13 is called a *non-well designed graph pattern.*

Table 5.7: Variable Types

Notation	Name	Description
CIVT	Class Instance Variable Type	Represents class instance variables
LVT	Literal Variable Type	Represents literal value variables
UVT	Unknown Variable Type	Represents unknown type variables
DTPVT	Data Type Predicate Variable Type	Represents data type predicate variables
OPVT	Object Predicate Variable Type	Represents object predicate variables
UPVT	Unknown Predicate Variable Type	Represents unknown predicate variables

$\mathbf{V}_{UFGP}$ (*UF–GP Variables Set*) be the set of the variables that are defined in the UF–GP[8] and $\mathbf{L}_{UFGP}$ (*UF–GP Literal Set*) be the set of the literals referenced in the UF–GP.

The variable type determination is a function $VarType$: $\mathbf{V}_{UFGP} \longrightarrow \mathbf{VT}$ that assigns a variable type $vt \in \mathbf{VT}$ to every variable $v \in \mathbf{V}_{UFGP}$, where $\mathbf{VT} = \{CIVT, LVT, UVT, DTPVT, OPVT, UPVT\}$ includes all the variable types. The relation between the domain and range of the function $VarType$ is defined by the determination rules presented below.

Here, we enumerate the determination rules that are applied iteratively for each triple in the given UF–GP. The final result of the rules is not affected by the order in which the rules are applied neither by the order in which the triple patterns are parsed. As T_x is denoted the type of a variable x.

Given a (non-Schema) triple pattern $t \in \langle s, p, o \rangle$, where s is the subject part, p the predicate part and o the object part, we define the following rules:

Rule 1: If $s \in \mathbf{V}_{UFGP} \Longrightarrow T_s = CIVT$. If the subject is a variable, then the variable type is *Class Instance Variable Type* (CIVT).

Rule 2: If $p \in \mathbf{DTP}$, and $o \in \mathbf{V}_{UFGP} \Longrightarrow T_o = LVT$. If the predicate is a datatype property and the object is a variable, then the type of the object variable is *Literal Variable Type* (LVT).

Rule 3: If $p \in \mathbf{mOP}$, and $o \in \mathbf{V}_{UFGP} \Longrightarrow T_o = CIVT$. If the predicate is an object property and the object is a variable, then the type of the object variable is *Class Instance Variable Type* (CIVT).

Rule 4: $T_p = DTPVT \Longleftrightarrow T_o = LVT \mid p, o \in \mathbf{V}_{UFGP}$. If the predicate variable type is Data Type Predicate Variable Type (DTPVT), then the type of the object variable is *Literal Variable Type* (LVT). The inverse also holds.

Rule 5: $T_p = OPVT \Longleftrightarrow T_o = CIVT \mid p, o \in \mathbf{V}_{UFGP}$. If the predicate variable type is Object Predicate Variable Type (OPVT), then the type of the object variable is *Class Instance Variable Type* (CIVT). The inverse also holds.

Rule 6: If $o \in \mathbf{L}_{UFGP}$ and $p \in \mathbf{V}_{UFGP} \Longrightarrow Tp = DTPVT$. If the object is a literal value, then the type of the predicate variable is *Data Type Predicate Variable Type* (DTPVT).

The unknown variable types UTV and UPTV do not result in conflicts in case that a variable has been also defined to have another type since they can be just ignored. All the variable types are initialized to the Unknown Predicate Variable Type (UPVT) if they appear in the predicate part of a triple; otherwise, they are initialized to the Unknown Variable Type (UVT).

As a result of the variable initialization, the following rule holds: If $s, p, o \in \mathbf{V}_{UFGP}$ and $T_p = UPVT$ and $T_o = UVT \Rightarrow T_p = UPVT$ and $T_o = UVT$. If a triple has subject,

[8]The $\mathbf{V}_{UFGP}$ set does not include the variables that occur only in Schema triple patterns, since the Schema triple patterns are omitted from the variables type determination phase.

predicate and object variables, the predicate variable type is Unknown Predicate Variable Type (UPVT) and the object variable type is Unknown Variable Type (UVT), no change is needed since they cannot be specified.

The variable type determination phase, including the variable initialization, the determination rules and the conflict check is also presented as an algorithm in [84].

5.6.2.2 Variable Result Form

For the formation of the result set we follow the Linked Data principles for publishing data. The resources are identified using Uniform Resource Identifiers (URI) in order to have a unique and universal name for every resource. The form of the results depends on the variable types. The following result forms are adopted for each variable type: (a) For CIVT variables, every result item is a combination of the URI of the XML Document that contains the node assigned to the variable with the XPath of the node itself (including the node context position). In XML, every element and/or attribute can be uniquely identified using XPath expressions and document–specific context positions. For example: http://www.music.tuc.gr/xmlDoc.xml#/Persons/Student[3]. (b) For DTPVT, OPVT and UPVT variables, every result item consists of the XPath of the node itself (without the position of the node context). For example: /Persons/Student/FirstName. (c) For LVT variables, every result item is the text representation of the node content. (d) For UVT variables, two cases are distinguished: (i) If the assigned node corresponds to a simple element, then the result form is the same with that of the LVT variables; and (ii) If the assigned node corresponds to a complex element, the result form is the same with that of the CIVT variables.

For the construction of the proper result form, XQuery functions (e.g., *func:CIVT( )*) formed using standard XQuery expressions, are used in the Return XQuery clauses.

5.6.3 Schema Triple Pattern Processing

In this section we present the *schema triple pattern processing*. This phase is performed in order to support schema-based queries. As schema-based queries are considered queries which contain triple patterns that refer to the ontology structure and/or semantics (i.e., *Schema Triple Patterns*, Definition 12). In the schema triple pattern processing context, the Schema Triple Patterns contained in the query are processed against the ontology so that the schema information can be used throughout the translation.

At first, ontology constructs are bound to the variables contained in the Schema Triples. Then, using the predefined mappings, the ontology constructs are replaced with the corresponding XPath Sets. As a result of this processing, XPaths are bound to the variables contained in the Schema Triples. These bindings will be used as initial bindings by the variable binding phase (Section 5.7). Note that as specified in Definition 12, triple patterns having a variable on their predicate part are not defined as schema triples, since they can deal either with data or with schema info. Hence, these triples are considered as non-schema triple patterns.

The schema triple patterns can be analyzed over the ontology, using a query or an inference engine. It should be noted that, in our approach we do not consider the semantics (e.g., entailment, open/close world assumptions, etc.) adopted in the evaluation of schema triples over the ontology. Since, the schema triple processing uses the results (i.e., ontology constructs) of the schema triple evaluation. Here, we have adopted simple entailment semantics (like the current SPARQL specification [314]). However, inferred results adhering to the RDFS or OWL entailments can be used if the SPARQL engine performs a query expansion step before evaluating the schema triples query, or an RDFS/OWL reasoner has been used. Currently, W3C works on defining the entailment regimes in the forthcoming

SPARQL 1.1 [185], which specify exactly what answers we get for several common entailment relations such as *RDFS entailment* or *OWL Direct Semantics entailment*. Finally, note that the SW is based on the *Open World Assumption* (OWA), while the XML world is based on the *Closed World Assumption* (CWA). This means that in the SW whatever is not explicitly stated is considered to be *unknown*, while in the XML world whatever is not explicitly stated in considered to be *false*.

5.7 Variable Binding

In this section, we describe the *variable binding* phase. In our context the term "variable bindings" is used to describe the assignment of XPaths to the variables referenced in a given BGP, thus enabling the translation of the BGPs in XQuery expressions.

Intuitively, this phase considers the graph structure(s) constructed by the triples patterns defined in the BGP, as well as the mappings, in order to determine the appropriate set of bindings. This set of bindings is going to be used in the construction of the XQuery expressions. It should be noted that, due to the form of the mappings (i.e., XPaths Sets) the (hierarchical) structure of XML data is also considered by the variable binding phase.

Additional schema information and/or semantics possibly expressed in the SPARQL query are exploited in the variable binding phase by using the bindings determined in the Schema Triple processing phase (Section 5.6.3). For this reason, the Schema Triples are omitted (i.e., pruned) from this phase and the determined Schema Triple bindings are used as initial bindings.

The variable binding algorithm is presented in Section 5.7.1, the variable binding rules are described in Section 5.7.2 and the XPath Set relations for triple patterns are discussed in Section 5.7.3.

5.7.1 Variable Binding Algorithm

5.7.1.1 Preliminaries

An RDF triple $\langle s, p, o \rangle$ is a sub graph of the directed RDF graph, where s, o are graph nodes and p is a directed graph edge, directed from s to o. As $\mathbf{X}_s$, $\mathbf{X}_p$ and $\mathbf{X}_o$ we denote the XPath Set correspond to subject, predicate and object XPath Sets respectively. Moreover, let $\mathbf{X}_{pD}$ and $\mathbf{X}_{pR}$ be the XPath Sets corresponding, respectively, to the predicate domains and ranges.

Considering the hierarchical structure of XML data and the structure of the directed RDF graph, the following relations must hold for the XPath Sets of the triple pattern parts:

(a) $\exists x_s \in \mathbf{X}_s$ and $\exists x_{pD} \in \mathbf{X}_{pD} : x_s \tilde{\sqsubset} x_{pD}$. The subject XPath Set ($\mathbf{X}_s$) contains XPaths that prefix the XPaths contained in the predicate domains XPath Set ($\mathbf{X}_{pD}$).

(b) $\exists x_{pD} \in \mathbf{X}_{pD}$ and $\exists x_{pR} \in \mathbf{X}_{pR} : x_{pD} \tilde{\sqsubset} x_{pR}$. The predicate domains XPath Set ($\mathbf{X}_{pD}$) contains XPaths that prefix the XPaths contained in the predicate ranges XPath Set ($\mathbf{X}_{pR}$).

(c) $\exists x_{pR} \in \mathbf{X}_{pR}$ and $\exists x_o \in \mathbf{X}_o : x_{pR} \tilde{\sqsubset} x_o$. The predicate ranges XPath Set ($\mathbf{X}_{pR}$) contains XPaths that prefix the XPaths contained in the object XPath Set ($\mathbf{X}_o$).

Thus, from (a), (b) and (c), we conclude to the *Subject–Predicate–Object Relation*, formally defined in (2):

$$\exists x_s \in \mathbf{X}_s, \exists x_{pD} \in \mathbf{X}_{pD}, \exists x_{pR} \in \mathbf{X}_{pR}, \exists x_o \in \mathbf{X}_o : x_s \tilde{\sqsubset} x_{pD} \tilde{\sqsubset} x_{pR} \tilde{\sqsubset} x_o \tag{2}$$

The Subject–Predicate–Object Relation must holds for every single triple pattern. Thus, the variable binding algorithm uses this relation in order to determine the appropriate bindings for the entire set of the conjunctive triple patterns (i.e., BGP), starting from the bindings of any single triple pattern part (subject, predicate, object).

Definition 14. (Shared Variable) A variable contained in a Union–Free Graph Pattern is called a *Shared Variable* when it is referenced in more than one triple patterns of the same Union–Free Graph Pattern regardless of its position in those triple patterns.

In case of shared variables, the algorithm uses the XPath Set Operators (i.e., $\preccurlyeq$, $\succcurlyeq$, $\overline{\bigcup}$), in order to determine the maximum set of bindings that satisfy the Subject– Predicate–Object Relation for the entire set of triple patterns (i.e., the entire BGP). As a result, all the XML nodes that satisfy the BGP are identified.

The variable binding algorithm does not determine the XPaths for Literal Variable Type (LVT) shared variables, since the literal equality (e.g., string equality, integer equality, etc.) is independent of the XML structure (i.e., XPath expressions). For example, consider that we want to identify the students with the same First Name and Last Name values. In this case, let the XPaths be $/Persons/Student/FirstName$ and $/Persons/Student/LastName$. Thus, the bindings for variables of LVT type cannot be determined at this step. Instead, they will be handled by the BGP2XQuery algorithm (Section 5.8.3), which exploits a combination of the mappings and the determined variable bindings.

For this phase we introduce the "special" XPath set value "$\ominus$". The value "$\ominus$", can be considered as the not initialized value, similar to the *null* value, however, different than the empty set $\varnothing$. Regarding the "$\ominus$" value, (a) the Intersection ($\overline{\bigcap}$), (b) the Descendants of Common Ancestors ($\succcurlyeq$) and (c) the Common Ancestors ($\preccurlyeq$) operators have the following semantics. Let the XPath Set $\mathbf{A}$, where $\ominus \notin \mathbf{A}$ and $\mathbf{A} \neq \varnothing$ and the XPath Set $\mathbf{e} = \{\ominus\}$. We have: **(a)** Intersection: (i) $\mathbf{A} \overline{\bigcap} \mathbf{e} = \mathbf{e} \overline{\bigcap} \mathbf{A} = \mathbf{A}$ (ii) $\mathbf{e} \overline{\bigcap} \mathbf{e} = \mathbf{e}$; **(b)** Descendants of Common Ancestors: (i) $\mathbf{A} \succcurlyeq \mathbf{e} = \mathbf{e}$ (ii) $\mathbf{e} \succcurlyeq \mathbf{A} = \mathbf{A}$ (iii) $\mathbf{e} \succcurlyeq \mathbf{e} = \mathbf{e}$; **(c)** Common Ancestors: (i) $\mathbf{A} \preccurlyeq \mathbf{e} = A$, (ii) $\mathbf{e} \preccurlyeq \mathbf{A} = \mathbf{e}$ (iii) $\mathbf{e} \preccurlyeq \mathbf{e} = \mathbf{e}$.

5.7.1.2 Algorithm Overview

Here we outline the *Variable Binding* algorithm (Algorithm 1), which takes as input (a) a Basic Graph Pattern (BGP); (b) a set of initial bindings ($\mathbf{X}^{Sch}$); (c) the types of variables that are present in the BGP ($varTypes$); and (d) the mappings of the BGP ontology constructs ($\mathbf{M}$). The variable types are determined by the determining of variable types phase and the initial bindings are those resulting from the Schema Triple processing.

In the beginning (*lines 1~7*), the variables that are not included in any Schema Triple, thus, no binding has been previously determined (from the Schema Triple Processing phase), are initialized here with the "special" value "$\ominus$" (*line 5*). The rest of the variables (included in a Schema Triple) are initialized to the initial bindings (*line 3*). Then, the algorithm performs an iterative process (*lines 11~21*) where it determines, at each step, the bindings of the entire BGP (triple by triple). The determination of the bindings of a single triple is performed using binding rules (*lines* 13, 16 & 19). Each part of the triple (subject–predicate–object) uses a binding rule (Section 5.7.2).This iterative process continues until the bindings for all the variables found in the successive iterations are equal (*line 23*). This implies that no further modifications in the variable bindings are to be made and that the current bindings are the final ones. Thus, the variable binding algorithm ensures that all the variables have been bound to the largest XPath sets with respect to: (a) the structure of the RDF data; (b) the structure of the XML data; and (c) the mappings between them. Note that $\mathbf{X}_w^i$ denotes the determined XPath Set at the i^{th} iteration of the algorithm for the w triple part.

Algorithm 1: Variable Binding Algorithm

Input: Basic Graph Pattern *BGP*, Initial Bindings $\mathbf{X}^{Sch}$,
Variable Types *varTypes*, Mappings **M**
Output: Variable Bindings $\mathbf{X}_v$

1. **for each** variable *v* **in** *BGP* *//initialize the bindings*
2. **if** $v \in var(schemaTr(BGP))$ *//if the variable v are included at schema triples*
3. $\mathbf{X}_v^0 = \mathbf{X}_v^{Sch}$
//initialize the bindings from the bindings determined the from schema triple processing
4. **else**
5. $\mathbf{X}_v^0 = \{\ominus\}$ *//initialize with the "special" value "⊖"*
6. **end if**
7. **end for**
8. it = 0 *//iteration counter initialization*
9. **repeat**
11. **for each** triple *t* **in** *BGP* *//loop over all the BGP triples*
12. **if** $s \in \mathbf{V}$ *//if the subject is a variable*
13. $\mathbf{X}_s^{i+1} = B_s(\ t, \mathbf{X}_s^i, \mathbf{X}_{pD}^i, \mathbf{X}_o^i, \mathbf{M}\ )$
//determine the subject bindings of the current iteration (i.e., t+1)
14. **end if**
15. **if** $p \in \mathbf{V}$ *//if the predicate is a variable*
16. $\mathbf{X}_p^{i+1} = B_p(\ t, \mathbf{X}_s^i, \mathbf{X}_p^i, \mathbf{M}, varTypes\ )$
//determine the predicate bindings of the current iteration (i.e., i+1)
17. **end if**
18. **if** $o \in \mathbf{V}$ *//if the object is a variable*
19. $\mathbf{X}_o^{i+1} = B_o(\ t, \mathbf{X}_s^i, \mathbf{X}_p^i, \mathbf{X}_o^i, \mathbf{M}, varTypes\ )$
//determine the object bindings of the current iteration (i.e., i+1)
20. **end if**
21. **end for**
22. i = i + 1 *//increase the counter*
23 **until** $(\forall\ v \in var(BGP) \Rightarrow \mathbf{X}_v^i = \mathbf{X}_v^{i-1})$
//loop until the bindings of the previous iteration are equal with the bindings of this iteration
24. **return** $\mathbf{X}_v\ \forall\ v \in var(BGP)$ *//return all the variable bindings for this basic graph pattern*

5.7.2 Variable Binding Rules

In this section we present the Variable Binding Rules applied by the variable binding algorithm (*lines* 13, 16 & 19) in order to determine the bindings for all the parts (i.e., subject, predicate and object) of a single triple pattern.

Initially, in order to define the binding rules we distinguish the Triple Pattern Types. According to the specified types we have defined the variable binding rules presented above.

Let the sets **V**, **L**, **I**, **B** (as defined in Section 5.5.1.1). We define four different types of triple patterns: (a) a triple pattern $\langle s, p, o\rangle \in (\mathbf{V} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{L}$, where the subject part s is a variable or a blank node, the predicate part p is an IRI and the object part o is a literal, is defined to be of *Type* 1; (b) a triple pattern $\langle s, p, o\rangle \in (\mathbf{V} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{V} \cup \mathbf{B})$, where the subject and object parts s, o are variables or blank nodes and the predicate part p is an IRI, is defined to be of *Type* 2; (c) a triple pattern $\langle s, p, o\rangle \in (\mathbf{V} \cup \mathbf{B}) \times (\mathbf{V} \cup \mathbf{B}) \times \mathbf{L}$, where the subject and predicate parts s, p are variables or blank nodes and the object part o is a literal, is defined to be of *Type* 3; (d) a triple pattern $\langle s, p, o\rangle \in (\mathbf{V} \cup \mathbf{B}) \times (\mathbf{V} \cup \mathbf{B}) \times (\mathbf{V} \cup \mathbf{B})$, where all the parts s, p, o are variables or blank nodes, is defined to be of *Type* 4.

Given a triple pattern $t\ \langle s, p, o\rangle$, where s is the subject part, p the predicate part and o the object part, we define the following binding rules depending on the triple pattern types. The following rules are applied to the triple pattern parts that are variables or

blank nodes. Firstly, the subject binding rule is applied, then the predicate binding rule, and finally the object binding rule, in order to determine the new XPath Set for every variable or blank node part.

In what follows, $\mathbf{X}_w^i$ denotes the determined XPath Set at the i^{th} iteration of the binding algorithm for the w triple part. In particular, $\mathbf{X}_s^i$ denotes the XPath Set corresponding to the subject part s, $\mathbf{X}_p^i$ denotes the XPath Set corresponding to the predicate part p, $\mathbf{X}_{pD}^i$ denotes the XPath Set corresponding to the domains of the predicate part p and $\mathbf{X}_o^i$ denotes the XPath Set corresponding to the object part o.

5.7.2.1 Subject Binding Rule

Here we present the binding rule B_S (3), which is applied in order to determine the XPath Set of the subject part ($\mathbf{X}_s^{i+1}$). The subject binding rule takes as input (a) the triple for which the determination is performed (t); (b) the previously determined bindings for the subject part ($\mathbf{X}_s^i$); (c) the previously determined bindings for the domains of the predicate part ($\mathbf{X}_{pD}^i$); (d) the previously determined bindings for the object part ($\mathbf{X}_o^i$); and (e) the mappings ($\mathbf{M}$). Note that with the term previously determined bindings we refer to the bindings determined in the previous algorithm iteration.

$$
\begin{aligned}
&B_S\,(t, \mathbf{X}_s^i, \mathbf{X}_{pD}^i, \mathbf{X}_o^i, \mathbf{M}) = \\
&= \begin{cases}
\mathbf{X}_s^i \,\overline{\cap}\, \mathbf{X}_{pD}^i & \textbf{if Type 1} \\
\mathbf{X}_s^i \,\overline{\cap}\, \mathbf{X}_{pD}^i \lessdot \mathbf{X}_o^i & \textbf{if Type 2} \\
\begin{cases} \mathbf{X}_{S1} & \textbf{if } \mathbf{X}_s^i = \ominus \textbf{ and } \mathbf{X}_{pD}^i = \ominus \quad \{1\} \\ \mathbf{X}_s^i \,\overline{\cap}\, \mathbf{X}_{pD}^i & \textbf{else} \end{cases} & \textbf{if Type 3} \\
\begin{cases} \mathbf{X}_{S1} & \textbf{if } \mathbf{X}_s^i = \ominus \textbf{ and } \mathbf{X}_{pD}^i = \ominus \textbf{ and } \mathbf{X}_o^i = \ominus \quad \{2\} \\ \mathbf{X}_{pD}^i \,\overline{\cap}\, \mathbf{X}_s^i \lessdot \mathbf{X}_o^i & \textbf{else} \end{cases} & \textbf{if Type 4}
\end{cases}
\end{aligned}
\tag{3}
$$

In above rule, $\{1\}$ holds if the type of the triple pattern is $Type$ 3, in case that the subject XPath Set ($\mathbf{X}_s^i$) and the predicate XPath Set ($\mathbf{X}_p^i$) have not been determined (are equal to $\ominus$). Moreover, $\{2\}$ holds for triple patterns of $Type$ 4, in case that $\mathbf{X}_s^i$, $\mathbf{X}_p^i$ and the object XPath Set ($\mathbf{X}_o^i$) have not been determined.

In the above cases, we assign to the subject XPath Set ($\mathbf{X}_s^i$) the XPath Set union of the sets of all the mapped classes (exploiting $\mathbf{M}$). Thus, $\mathbf{X}_{S_1} = \mathbf{X}c_1 \,\overline{\bigcup}\, \mathbf{X}_{c_2} \,\overline{\bigcup} \cdots \overline{\bigcup}\, \mathbf{X}_{c_n}$, for each mapped class c_i, where $\mathbf{X}_{c_i}$ is the XPath Set corresponding to $c_i, \forall i \in \{1, \ldots, n\}$.

Similarly are defined the predicate (B_P) and the object (B_O) binding rules, which are available in [84].

5.7.3 XPath Set Relations for Triple Patterns

In several cases, XPath Sets that correspond to different SPARQL variables must be associated. For example, let the triple pattern $?x\ FirstName\_xs\_string\ ?y$; the variable x corresponds to Persons and Students and the variable y to their first name(s). The variable binding phase will result in two XPath Sets: one for all the Persons and Students corresponding to variable x (i.e., $\mathbf{X}_x = \{/Persons/Person, /Persons/Student\}$) and one for all the First Names corresponding to variable y (i.e., $\mathbf{X}y = \{/Persons/Person/FirstName$, $/Persons/Student/FirstName\}$). However, the association of persons and their names still has to be done. We introduce the extension relation which can hold among different XPath Sets and can be exploited in order to associate them.

Definition 15. (Extension Relation) An XPath Set **D** is said to be an extension of an XPath Set **A** if all the XPaths in **D** are descendants of the XPaths of **A**. This relation can be achieved if the XPath Set Concatenation ($\oplus$) operator is applied to the XPath Set **A** having as right operand an XPath Set **C**, and as result the XPath Set **D**, which will be an *extension* of **A** (i.e., $\mathbf{A} \oplus \mathbf{C} = \mathbf{D}$, **D** is an *extension* of **A**).

In the above example, the XPath Set $\mathbf{X}_y$ is an extension of the XPath Set $\mathbf{X}_x$. The XPath Set $\mathbf{X}_y$may result from the XPath Set $\mathbf{X}_x$ since: $\mathbf{X}_x \oplus \{/FirstName\} = \{/Persons/Person, /Persons/Student\} \oplus \{/FirstName\} = \{/Persons/Person/FirstName, /Persons/Student/FirstName\}$ which is equal to the XPath Set $\mathbf{X}_y$.

Based on the Subject–Predicate–Object Relation defined in (2), the extension relation holds for the XPath Sets and results from the Variable Binding Algorithm. It implies that the XPath Set bound to the object part corresponds to an extension of the XPath Set bound to one of the predicate and subject parts. Moreover, the XPaths bound to the predicate part correspond to an extension of the XPath Sets bound to the subject part. Thus, the extension relation is exploited by the translation process, using also the For and Let XQuery clauses, in order to associate different XQuery variables.

Note that the notion of extension is also used to describe relations between XQuery variables. If the extension relation holds for the XPaths used in the For/Let clauses that assign values to the variables, then the extension relation also holds between these variables. In particular, consider the following For or Let (For/Let) XQuery clauses: *For/Let* $\$v_1$ *in/:=* $expr_1$ and *For/Let* $\$v_2$ *in/:=* $expr_2$; if the XPath expressions occurring in $expr_2$ are extensions of the XPath expressions occurring in $expr_1$, then the XQuery variable $\$v_2$ is also said to be an *extension* of the $\$v_1$ XQuery variable. For example, consider the XQuery expressions: *Let* $\$x$:= */Persons/Person union /Persons/Student and For* $\$y$ *in* $\$x/FirstName$. In these XQuery expressions, the XQuery variable $\$y$ is said to be an *extension* of the XQuery variable $\$x$.

5.8 Graph Pattern Translation

In this section, we describe the *graph pattern translation* phase, which translates a graph pattern into semantically corresponding XQuery expressions. The XQuery and SPARQL basic notions are introduced in Section 5.8.1, an overview of the graph pattern translation is presented in Section 5.8.2, the basic graph pattern translation is described in Section 5.8.3 and we close with a discussion on the major challenges that we faced during the graph pattern translation in Section 5.8.4.

5.8.1 Preliminaries

In this section we provide an overview of the semantics of the SPARQL graph patterns (most of them defined in [304]), as well as some preliminary notions regarding the XQuery syntax representation.

Definition 16. (SPARQL Graph Pattern Solution) A SPARQL Graph Pattern solution $\omega : \mathbf{V} \longrightarrow (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is a partial function that assigns RDF terms of an RDF dataset to variables of a SPARQL graph pattern. The domain of ω, $dom(\omega)$, is the subset of **V** where ω is defined. The empty graph pattern solution $\omega_\varnothing$ is a graph pattern solution with an empty domain. The SPARQL graph pattern evaluation result is a set $\mathbf{\Omega}$ of graph pattern solutions ω.

Two Graph Pattern solutions ω_1 and ω_2 are *compatible* when for all $x \in dom(\omega_1) \bigcap dom(\omega_2)$ it holds that $\omega_1(x) = \omega_2(x)$. Furthermore, two graph pattern solutions with disjoint domains are always compatible, and the empty graph pattern solution $\omega_\varnothing$ is compatible with any other graph pattern solution.

Let $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$ be sets of Graph Pattern solutions. The *Join*, *Union*, *Difference* and *Left Outer Join* operations between $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$ are defined as follows: (a) $\mathbf{\Omega}_1 \bowtie \mathbf{\Omega}_2 = \{\omega_1 \bigcup \omega_2 \mid \omega_1 \in \mathbf{\Omega}_1, \omega_2 \in \mathbf{\Omega}_2$ are compatible graph pattern solutions$\}$, (b) $\mathbf{\Omega}_1 \bigcup \mathbf{\Omega}_2 = \{\omega \mid \omega_1 \in \mathbf{\Omega}_1$ or $\omega_2 \in \mathbf{\Omega}_2\}$, (c) $\mathbf{\Omega}_1 \ \mathbf{\Omega}_2 = \{\omega \in \mathbf{\Omega}_1 \mid$ for all $\omega' \in \mathbf{\Omega}_2$, ω and ω' are not compatible$\}$, (d) $\mathbf{\Omega}_1 \rtimes \mathbf{\Omega}_2 = (\mathbf{\Omega}_1 \bowtie \mathbf{\Omega}_2) \bigcup (\mathbf{\Omega}_1 \ \mathbf{\Omega}_2)$.

The semantics of the SPARQL graph pattern expressions are defined as a function $[[.]]_D$, which takes a graph pattern expression and an RDF dataset D and returns a set of graph pattern solutions.

Definition 17. (SPARQL Graph Pattern Evaluation) Let D be an RDF dataset over $(\mathbf{I} \bigcup \mathbf{B} \bigcup \mathbf{L})$, t a triple pattern, P, P_1, P_2 graph patterns and R a built-in condition. Given a graph pattern solution ω, we denote as $\omega \vDash R$ that ω satisfies R (the Filter operator semantics are described in detail in [311]). The evaluation of a graph pattern over D, denoted by $[[.]]_D$, is defined recursively as follows: (a) $[[t]]_D = \{\omega \mid dom(\omega) = var(t)$ and $\omega(t) \in D\}$, (b) $[[(P_1$ AND $P_2)]]_D = [[P_1]]_D \bowtie [[P_2]]_D$, (c) $[[(P_1$ OPT $P_2)]]_D = [[P_1]]_D \rtimes [[P_2]]_D$, (d) $[[(P_1$ UNION $P_2)]]_D = [[P_1]]_D \bigcup [[P_2]]_D$ and (e) $[[(P$ FILTER $R)]]_D = \{\omega \in [[P]]_D \mid \omega \vDash R\}$.

Finally, we introduce the SPARQL Return Variable notion, which is exploited throughout the SPARQL to XQuery translation, as well as some basic notions regarding the XQuery syntax.

Definition 18. (SPARQL Return Variable) A SPARQL return variable is a variable for which the SPARQL query would return some information. The *Return Variables* (RV) of a SPARQL query constitute the Return Variables set $\mathbf{RV} \subseteq \mathbf{V}$. In particular: (a) for Select and Describe SPARQL queries, the $\mathbf{RV}$ consists of the variables referred after the query form clause; in case of wildcard (*) use, $\mathbf{RV} = \mathbf{V}$; (b) for Ask SPARQL queries, $\mathbf{RV} = \varnothing$; (c) for Construct SPARQL queries, $\mathbf{RV}$ consists of the variables referred in the query graph template (i.e., the variables that belong to the graph template variable set $\mathbf{GTV}$), thus, $\mathbf{RV} = \mathbf{GTV}$.

Due to the fact that the term "predicate" is used in the SPARQL and XPath languages, in the rest of this chapter we will refer to the XPath predicate as *XPredicate*. Moreover, the XQuery variable $\$doc$ is defined to be initialized by the clauses: *let $doc := fn:doc("URI")*, or *let $doc := fn:collection("URI")*; where URI is the address of the XML document or document collection that contains the XML data over which the produced XQuery will be evaluated.

Finally, we define the abstract syntax representation of the XQuery For and Let clauses xC as follows: (a) *for $var in expr* ; and (b) *let $var := expr*, where $\$var$ is an XQuery variable named *var* and *expr* is a sequence of XPath expressions. As $xC.var$ we denote the name of the XQuery variable defined in xC, as $xC.expr$ we denote the XPath expressions of xC and as $xC.type$ we denote the type (For or Let) of the XQuery clause xC. Finally, as xE we denote a sequence of XQuery expressions.

5.8.2 Graph Pattern Translation Overview

The graph pattern concept is defined recursively. The basic graph pattern translation phase (Section 5.8.3) translates the basic components of a GP (i.e., BGPs) into XQuery expressions, which in several cases have to be associated in the context of a GP. That is, to apply the SPARQL operators (i.e., UNION, AND, OPT and FILTER) that may occur outside the BGPs. The *GP2XQuery* algorithm traverses the SPARQL evaluation tree resulting from the GP, so as to identify and handle the SPARQL operators.

Particularly, the SPARQL UNION operator corresponds to the union operation applied to the graph pattern solutions of its operand graphs. The implementation of the

UNION operator is straightforward in XQuery. The FILTER operator restricts the query solutions to the ones for which the filter expression is true. The translation of the FILTER operator in the context of BGPs is presented in Section 5.8.3.6. The same holds for the translation of the filters occurring outside the BGPs. The SPARQL AND and OPT operators correspond to the Join and Left Outer Join operators respectively, applied to the graph pattern solutions of their operand graphs. The semantics of the Join and Left Outer Join operators in SPARQL differ slightly from the relational algebra join semantics, in the case of *unbound*[9] variables[10]. In particular, the existence of an unbound variable in a SPARQL join operation does not produce an unbound result. In other words, the join in the SPARQL semantics, is defined as a non null-rejecting join. The semantics of the *compatible mappings* in the case of unbound variables have been discussed in [316, 311, 304].

Note however that SPARQL does not provide the *minus* operator at syntax level. The minus operator can be expressed as a combination of optional patterns and filter conditions which include the *bound* operator (like the *Negation as Failure* (NAS) in logic programming[11]). The semantics of the SPARQL minus operator have been extensively studied in [34].

The unbound variable semantics in conjunction with the OPT operator result in a "special" type of GPs. This type is well known as *non-well designed* GPs (Definition 13) with some of its properties being different from the rest of the GPs (i.e., the *well designed* ones). In particular, in the context of translating the AND and OPT operators, the possible evaluation strategies differ for the well designed and the non-well designed graph patterns (for more details see [304]). As a result, in order to provide an efficient translation for the AND and OPT operators, we must not handle all graph patterns in a uniform way. Below we outline the translation for both well-designed and non-well designed graph patterns in XQuery expressions.

5.8.2.1 Well Designed Graph Patterns

Every well-designed Union–Free Graph Pattern P_i contained in the normal form (1) is transformed in the form of (4) after the graph pattern normalization phase (Section 5.6.1):

$$(\cdots(t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1) \text{ OPT } O_2)\cdots) \text{ OPT } O_n), \tag{4}$$

where each t_i is a triple pattern, $n \geq 0$ and each O_j has the same form (4) [304].

We can observe from (4) that the AND operators are occurring only between triple patterns (expressed with "." in the SPARQL syntax) in the context of basic graph patterns (BGPs). As a consequence, in the case of well designed GPs, the AND operators are exclusively handled by the BGP2XQuery algorithm, as described in Section 5.8.3. In particular, the BGP2XQuery algorithm uses associated For/Let XQuery clauses to resemble nested loop joins. In addition, throughout the For/Let XQuery clauses creation, the BGP2XQuery algorithm exploits the extension relation (Definition 15) in order to use the already evaluated XQuery values, providing a more efficient join implementation.

Considering the well designed GP definition, as well as the form (4), we conclude that the following holds for the operands of an OPT operator: For the expressions of the form $P = (P_1 \text{ OPT } P_2)$ occurring in (4), every variable occurring both inside P and outside P, it occurs for sure in P_1. As a result, the variables occurring outside P have

[9] Similar to the *unknown/null* value in SQL.

[10] It is not clear why the W3C has adopted the specific semantics.

[11] Although the SPARQL language expresses the minus operator like *Negation as Failure* (NAS) [104], it does not make any assumption to interpret statements in an RDF graph using negation as failure or other *non-monotonic* [104] assumption (e.g., Closed World Assumption). Note that both SPARQL and RDF are based on the Open World Assumption (OWA).

always bound values, imposed from the P_1 evaluation. Note that the above property holds only for well designed GPs and not for the non-well designed ones. Exploiting this property, we can provide an efficient implementation of the OPT operators, which are going to use the already evaluated results (produced from the left operand evaluation) in the evaluation of the right operand. Consider for example the well designed graph pattern $P = (t_1 \text{ OPT } (t_2 \text{ OPT } t_3)$, where t_1, t_2 and t_3 triple patterns. The evaluation of P over a dataset D will be $[[t_1]]_D \bowtie (([[[t_1]]_D \bowtie [[t_2]]_D)) \bowtie (([[[t_1]]_D \bowtie [[t_2]]_D \bowtie [[t_3]]_D))$.

The GP2XQuery algorithm traverses the SPARQL execution tree in a depth-first manner, the BGP2XQuery algorithm translates the BGPs occurring in GP. In case of OPT operators, the XQuery expressions resulting from the translation of the right operands use the XQuery values already evaluated from the translation of the left operand, reducing the required computations.

5.8.2.2 Non-Well Designed Graph Patterns

The evaluation strategy outlined above can not be applied in the case of the non-well designed GPs. The unbound variables semantics and the "confused" use of variables in the OPT operators of the non-well designed GPs do not allow the use of the intermediate results during the graph pattern evaluation.

For example, consider the following non-well designed graph pattern $P = ((?x\ p_1\ ?y)$ OPT $(?x\ p_2\ ?z))$ OPT $(?w\ p_3\ ?z)$. The evaluation of the expression $((?x\ p_1\ ?y)$ OPT $(?x\ p_2\ ?z))$ will possibly return results with unbound values for the variable $?z$. In the evaluation strategy adopted for the well designed GPs, the results from the evaluation of $((?x\ p_1\ ?y)$ OPT $(?x\ p_2\ ?z))$ expression (intermediate results) and in particular the results from the variable $?z$, will be used to evaluate the OPT$(?w\ p_3\ ?z)$ expression. The unbound values that possibly occur for variable $?z$, will reject the evaluation of the OPT$(?w\ p_3\ ?z)$ expression. However, this rejection is not consistent with the unbound variable semantics. Due to that, an unbound $?z$ value resulting from the evaluation of expression $((?x\ p_1\ ?y)$ OPT $(?x\ p_2\ ?z))$, will not reject a bound value $?z$ resulting from the evaluation of expression OPT$(?w\ p_3\ ?z)$.

As a result, for the non-well designed GPs, we are forced to independently evaluate the BGPs, so that the AND and OPT operators will be applied over the results produced from the BGP evaluation. In the context of SPARQL to XQuery translation, the GP2XQuery algorithm traverses the SPARQL execution tree in a button-up fashion and the BGP are independently translated by the BGP2XQuery algorithm. Finally, the AND and OPT operators are applied using XQuery clauses among the XQuery expressions resulting from the BGP2XQuery translation, taking also into consideration the semantics of the compatible mappings for unbound variables.

5.8.3 Basic Graph Pattern Translation

This section describes the translation of basic graph pattern into XQuery expressions.

5.8.3.1 BGP2XQuery Algorithm Overview

We outline here the BGP2XQuery algorithm, which translates BGPs into XQuery expressions. The algorithm is not executed triple by triple. Instead, it processes the subjects, predicates, and objects of all the triples separately. For each SPARQL variable included in the BGP, the algorithm creates For or Let XQuery clauses, using the variable bindings, the input mappings, and the extension relation (Definition 15). In every case, the name of an XQuery variable is the same with that of the corresponding SPARQL variable, so the correspondences between the SPARQL and XQuery queries can be easily captured. Regarding

the literals included in the BGP, the algorithm translates them as XPath conditions using XPredicates. The translation of SPARQL Filters depends on the Filter expression. Most of the Filters are translated as XPath conditions expressed using XPredicates, however some "special" Filter expressions are translated as conditions expressed in XQuery Where clauses. Finally, the algorithm creates an XQuery Return clause that includes the Return Variables that were defined in the BGP. The translation of BGPs is described in detail in the following sections.

5.8.3.2 For or Let Clause?

A crucial issue in the XQuery expression construction is the enforcement of the appropriate solution sequence based on the SPARQL semantics. To achieve this, for a SPARQL variable v, we create a For or a Let clause according to the algorithm presented below (Algorithm 2). Intuitively, the algorithm chooses between the construction of For and Let clauses in order to produce the desired solution sequence. For example, consider a SPARQL query which returns persons and their first names. For a person A, that has two first name n_1 and n_2, the returned solution sequence will consist of two results A, n_1 and A, n_2.

Algorithm 2: For or Let XQuery Clause Selection (QF, $\mathbf{RV}$, v)

Input: SPARQL query form QF, *Return Variables* $\mathbf{RV}$, SPARQL variable v
Output: XQuery Clause Type

```
1.  if QF ≠ Ask
2.      if (v ∈ RV) or ( ∃K ∈ RV | K is extension of v )
3.          return Create a For XQuery Clause
4.      else
5.          return Create a Let XQuery Clause
6.      end if
7.  else
8.      return Create a Let XQuery Clause
9.  end if
```

For the Select, Construct and Describe query forms (*lines 1~6*) the algorithm will create for the variable v a For XQuery clause if v is included in the $\mathbf{RV}$ or if any return variable is an extension (Definition 15) of the variable v (*line 3*), otherwise it will create a Let XQuery clause (*line 5*). For Ask queries (*lines 7~9*) that do not return a solution sequence, and in order to make the generated XQueries more efficient, the algorithm will create only Let XQuery clauses (*line 8*), in order to check if a BGP can be matched over XML data.

5.8.3.3 Subject Translation

The *Subject Translation* algorithm (Algorithm 3) translates the subject part of all the triple patterns of a given BGP to XQuery expressions. It should be noted that, for the rest of the chapter the symbol N_X denotes the name of SPARQL variable X and the triple patterns are represented as $s\ p\ o$, where s is the subject, p the predicate and o the object part of the triple pattern.

Algorithm 3: Subject Translation (BGP, QF, $\mathbf{RV}$, $bindings$ **)**

Input: Basic Graph Pattern BGP, SPARQL query form QF,
SPARQL Return Variables $\mathbf{RV}$, Variable Bindings $bindings$
Output: For or Let XQuery Clause xC

```
1. for each triple in BGP
2.     if s ∈ V        // If subject is a variable
3.         xC.type ← For or Let XQuery Clause Selection ( QF, RV, s )
           //Create a For or Let XQuery Clause
4.         xC.var ← N_s        // Define an XQuery Variable with the name of SPARQL Variable s
5.         xC.expr ← $doc/x1 union $doc/x2 union … union $doc/xn , ∀ xi ∈ X s
           // Set expr equal to the XPath Set of the Subject prefixed with the $doc variable
           //X_s is the binding XPath Set for the variable s
6.     end if
7. end for
8. return xC
```

For each subject s that is a variable (*line 2*), the algorithm creates a For or Let XQuery clause xC, using the *For or Let XQuery Clause Selection* algorithm (*line 3*) to determine the type (i.e., For or Let) of the clause. The XQuery variable $xC.var$ defined in the XQuery clause being created has the same value with the name of the subject N_s (i.e., the SPARQL and the XQuery variables have the same name) (*line 4*). The XQuery expression $xC.expr$ is defined using the variable bindings of the subject X_s and the $\$doc$

variable (*line 5*). Finally, the algorithm returns the generated For or Let XQuery clause (*line 8*).

5.8.3.4 Predicate Translation

The *Predicate Translation* algorithm (Algorithm 4) translates the predicate part of all the triple patterns of a given BGP to XQuery expressions. For each predicate p that is a variable (*line 2*), the algorithm creates a For or Let XQuery clause xC, using the For or Let XQuery Clause Selection Algorithm (*line 3*) to determine the type (i.e., For or Let) of the clause. The XQuery variable $xC.var$ defined in the XQuery clause being created has the same value with the name of the predicate N_p (*line 4*). The XQuery expression $xC.expr$ is defined using: (a) the variable bindings of the predicate $\mathbf{X}_p$; (b) the variable bindings of the subject $\mathbf{X}_s$; (c) the XQuery variable $\$N_s$ that represents the subject of the triple; and (d) the extension relation. $xC.expr4$ associates the subject and predicate bindings (*line 5*). Finally, the algorithm returns the generated For or Let XQuery clause (*line 8*).

Algorithm 4: Predicate Translation (*BGP*, *QF*, **R V** , *bindings* **)**

Input: Basic Graph Pattern *BGP*, SPARQL query form *QF*,
SPARQL Return Variables **R V** , Variable Bindings *bindings*
Output: For or Let XQuery Clause *xC*

1. **for each** *triple* **in** *BGP*
2. **if** $p \in \mathbf{V}$ *// If predicate is a variable*
3. $xC.type \leftarrow$ **For or Let XQuery Clause Selection** (*QF*, *RV*, *p*) *//Create a For or Let XQuery Clause*
4. $xC.var \leftarrow N_p$ *// Define an XQuery Variable with the same name with the SPARQL Variable p*
5. $xC.expr \leftarrow \$N_s/x_1$ union $\$N_s/x_2$ union … union $\$N_s/x_n$, $\forall x_i \in \mathbf{X}_s \gg \mathbf{X}_p$ *// Set expr equal to the variable corresponding to the triple subject variable suffixed with XPaths that have resulted from the* $\mathbf{X}_s \gg \mathbf{X}_p$ *operation. The XPath Set.* $\mathbf{X}_p$ *is the binding XPath Set for the variable p and* $\mathbf{X}_S$ *is the binding XPath Set for the subject s*
6. **end if**
7. **end for**
8. **return** *xC*

Binding Assurance Condition. According to the SPARQL semantics, all the variables used in a BGP must be bound for all the solutions in the solution sequence. That is, RDF terms must be assigned to all the variables in all the solutions. In the BGP2XQuery translation, this is not always guaranteed when Let XQuery clauses are used to translate SPARQL variables. In these cases, we must check if a value has been bound to each variable (*lines* 7, 19, 26). In order to perform this check, we exploit the XQuery function *exists*( ), which allows checking the assignment of some value to a variable. A Binding Assurance

5.8.3.5 Object Translation

The *Object Translation* algorithm (Algorithm 5) translates the object part of all the triple patterns of a given BGP to XQuery expressions. For the objects o that are literals (*lines 2~12*), the algorithm creates XPredicates in order to translate them. If the predicate p of the triple is a variable, the XPredicate restriction is applied to the (For or Let) XQuery clause created during the translation of the predicate variable (*lines 3~5*). If the predicate is not a variable, the appropriate restrictions using XPredicates are applied to the (For or Let) XQuery clause, created during the translation of the subject s of the triple (*lines 9~12*).

For the objects o that are variables (*lines 13~29*), if the predicate p is also a variable (*lines 14~21*) the algorithm creates a Let XQuery clause (*lines 15~17*), in order to assign the predicate XQuery variable to the XQuery variable of the object. If the predicate is an IRI (*lines 21~28*), the algorithm creates a For or Let XQuery clause xC using the For or Let XQuery Clause Selection Algorithm (*line 22*) to determine the type (i.e., For or Let) of the clause. In this case, the algorithm uses the variable bindings of the subject $\mathbf{X}_s$, the mapping μ_p of the property defined in the predicate part and the extension relation (Definition 15) for triple patterns, in order to associate the subject, the predicate and the object bindings (*line 24*).

Algorithm 5: Object Translation (*BGP*, *QF*, **R V** , *bindings*, *mappings* **)**

Input: Basic Graph Pattern *BGP*, SPARQL query form *QF*, SPARQL Return Variables **R V** , Variable Bindings *bindings*, mappings between the ontology and the XML schema *mappings*
Output: For or Let XQuery Clause *xC*

1. **for each** *triple* **in** *BGP*
2. **if** $o \in \mathbf{I}$ *// If the object is a literal*
3. **if** $p \in \mathbf{V}$ *// If the predicate is a variable*
4. Create *XPredicate* over the *xC.expr* where *xC* is the For/Let clause created for the predicate *p*
5. *XPredicate* ← [.= "*o*"]
6. **if** Let XQuery Clause created for *p*
7. Create *"Bindings Assurance Condition"* for *p* *//see "Biding Assurance Condition" Section*
8. **end if**
9. **else** *// The predicate is not a variable – it is an IRI*
10. Create *XPredicate* $\forall x_i \in \mathbf{X}_s$ in *xC.expr*, where *xC* is the For/Let clause created for the subject *s*
11. *XPredicate* ← $[./y_1 = \text{"o"} \text{ or } ./y_2 = \text{"o"} \text{ or } \ldots \text{ or } ./y_n = \text{"o"}] \forall y_i \in \{x_i\} \gg \mu_p$
// $\mathbf{X}_S$ is the bindings XPath Set for the subject S and μ_P is the mappings XPath Set for the property p
12. **end if**
13. **else if** $o \in \mathbf{V}$ *// If the object is a variable*
14. **if** $p \in \mathbf{V}$ *// If the predicate is a variable*
15. *xC.type* ← Create a Let XQuery Clause
16. *xC.var* ← N_o *// Define an XQuery Variable with the name of the SPARQL Variable o*
17. *xC.expr* ← $\$ N_p$ *// Set expr equal to the predicate Variable*
18. **if** Let XQuery Clause created for *p*
19. Create *"Bindings Assurance Condition"* for *o* *//see "Biding Assurance Condition" Section*
20. **end if**
21. **else** *// The predicate is not a variable – it is an IRI*
22. *xC.type* ← **For or Let XQuery Clause Selection (** *QF*, **R V** , *o* **)** *//Create a For or Let XQuery Clause*
23. *xC.var* ← N_o *// Define an XQuery Variable with the name of the SPARQL Variable o*
24. *xC.expr* ← $\$ N_s / x_1$ union $\$ N_s / x_2$ union … union $\$ N_s / x_n \forall x_i \in \mathbf{X}_s \gg \mu_p$
// Set expr equal to the variable corresponding to the triple subject suffixed with some of the XPath of the Predicate XPath Set
// $\mathbf{X}_s$ is the bindings XPath Set for the subject s and μ_p is the mappings XPath Set for the property p.
25. **if** Let XQuery Clause created for *o*
26. Create *"Bindings Assurance Condition"* for *o* *//see "Biding Assurance Condition" Section*
27. **end if**
28. **end if**
29. **end if**
30. **end for**
31. **return** *xC*

Condition for a variable w corresponds to a definition of the form "$exists(\$w) = true$" in the XQuery Where clause.

5.8.3.6 Filter Translation

The *Filter Translation* algorithm (Algorithm 6) translates the SPARQL FILTERs that may be contained in a given BGP into XQuery expressions. A straightforward approach for handing SPARQL Filters would be to translate Filter expressions as conditions expressed in XQuery Where clauses. However, this approach would result in inefficient XQuery expressions, since the Filter conditions are evaluated at the final stage of the query processing.

Therefore, we attempt to provide an efficient Filter translation algorithm by applying the Filter restrictions earlier, when this is possible. The earlier the Filter conditions are applied the more efficient XQuery expressions are constructed. The conditions reduce the size of the evaluated data which are going to be used in the later stages of the query processing, similarly to the *predicate pushdown* technique which is used in the query optimization context.

Algorithm 6: Filter Translation (*BGP* **)**

Input: Basic Graph Pattern *BGP*
Output: Where XQuery Clause *xC* or Create *XPredicates* over XQuery clauses

1. **for each** *Filter* **in** *BGP*
2. Translate the SPARQL Operators of the *Filter* expression
3. **if** (*Filter* **is** *safe*)
4. Create *XPredicates* for the Filter expressions
5. **else**
6. *xC* ← Create an XQuery Where Clause Condition
7. **end if**
8. **end for**
9. **return** *xC*

The early evaluation of the Filter expressions in XQuery can be achieved using XPredicates. This way, the Filter conditions are applied when the XPath expressions are evaluated. However, not all the Filter expressions can be expressed as XPredicates conditions. There exist "special" cases, where the Filter expressions can not be evaluated in an earlier stage, because of the SPARQL variables that occur inside the Filter expression. In these cases the Filters are translated as conditions expressed in XQuery Where clauses. These "special" cases are known as *not safe* Filter expressions (Definition 19) and are discussed below.

Safe Filter. There are cases, where the evaluation of Filter expressions is not valid under the evaluation function semantics (Definition 17). These "special" cases are identified by the usage of the variables inside the Filter expression and the graph patterns.

Definition 19. (Safe Filter Expressions) A Filter expression R *is safe* if for Filter expressions of the form P FILTER R, it holds that, all the variables that occur in R also occur in P (i.e., $var(R) \subseteq var(P)$) [304].

In case of existence of Filter expressions which are not safe, the evaluation function of the SPARQL queries has to be modified in comparison to the standard SPARQL evaluation semantics. As an example, consider the following pattern: ($?x$ p_1 $?y$) OPT (($?x$ p_2 $?z$) FILTER ($?y$ =$?z$)). Based on the evaluation function, the expression ($?x$ p_2 $?z$) FILTER ($?y$ =$?z$) is evaluated first. However, the variable $?y$ inside the Filter expression does not exist in left side pattern (i.e., $?x$ p_2 $?z$), thus, this evaluation will produce an ambiguous result. Our translation method overcomes this issue by evaluating the Filter expression as an XQuery Where clause conditions, applied after the graph pattern (translation and) evaluation.

Filter Expressions Operators. The SPARQL query language provides several unary and binary operations which can be used inside the Filter expressions. Some of these operators (e.g., &&, ||, !, =, ≤, ≥, +, *regex*, *bound*) can be mapped directly to XQuery built-in functions and operators, whereas for other operators (e.g., *sameTerm*, *lang*) XQuery functions have to be implemented in order to simulate them. Finally, a few SPARQL operators can not be supported in the XQuery language. In particular, the *isBlank* SPARQL operator can not be implemented for the XML data model, since the blank node notion is not defined in XML. In addition, it is very complex to evaluate the isIRI, isLiteral and datatype SPARQL operators over XML data. The result of these operators is difficult and inefficient to be determined on-the-fly through the evaluation of the XQuery expressions over XML data. This can only be achieved via a complex and a large sequence of XQuery $if - if\ else$ conditions. The $if - if\ else$ conditions will exploit the mappings in order to evaluate the above operators, resulting in inefficient XQuery expressions. However, the results of these operators can be determined after the XQuery evaluation, by processing the return results and the mappings.

Filter Evaluation. The SPARQL query language supports *three-valued logic* (i.e., $True$, $False$ and $Error$) for Filter expression evaluation. Instead, the XQuery query language supports two-valued logic or Boolean logic (i.e., True and False). In order for our method to bridge this difference, based on the semantics presented at [314] and [311], the SPARQL Error value is mapped to the XQuery False value, while, the SPARQL Error value could be easily supported by our translation by exploiting XQuery $if - if\ else$ conditions throughout the Filter expression translation. These conditions would check for errors that have occurred during the evaluation of the XQuery Where clause conditions and would return the Error value. A common SPARQL error example occurs when unbound variables exist inside the Filter expression.

5.8.3.7 Return Clause Construction

The *Construct Return Clause* algorithm (Algorithm 7) builds the XQuery Return Clause. For Ask SPARQL queries (*lines 1~2*), the algorithm creates an XQuery Return clause which, for efficiency reasons, includes only the literal "yes" (*line 2*). For the other query forms (i.e., Select, Construct, Describe) (*lines 3~6*), the algorithm creates an XQuery Return clause xC that includes all the return variables (**RV**) used in the BGP (*line 4*). The syntax of the return clause allows (using markup tags) the distinction of each solution in the solution sequence, as well as the distinction of the corresponding values for each variable. The structure of the return results allows the SPARQL operators AND and OPT to be applied over the results returned by different XQuery Return clauses. Finally the algorithm, for each variable included in the return clause, and based on the variable types (*varTypes*), uses the appropriate function to format the result form of the variable (Section 5.6.2.2) and returns the generated return XQuery clause (*line 7*).

Algorithm 7: Construct Return Clause (*BGP*, *QF*, **RV**, *varTypes* **)**

Input: Basic Graph Pattern *BGP*, SPARQL query form *QF*,
SPARQL Return Variables **RV**, Variable Types *varTypes*
Output: Return XQuery Clause *xC*

1. **if** *QF* = Ask
2. *xC* ← return("yes") *//Create an XQuery Return clause*
3. **else** *//The query form is not Ask*
//Create an XQuery Return clause
4. *xC* ← return(<Result>
<var1>...</var1> , <var2>...<var2>,…,<vari>...</vari></Result>)
∀ *vari* ∈ **RV** ∩ *var*(*BGP*)
// Each Return Variable included in the given BGP is inserted in the XQuery return clause
5. ∀ *vari* ∈ **RV** ∩ *var*(*BGP*) use the *varTypes* to
determine the result form of *vari*
6. **end if**
7. **return** *xC*

5.8.4 Discussion

The Graph Pattern Translation is the most complex phase of the SPARQL to XQuery translation process. The noteworthy issues that have arisen throughout this phase are outlined and discussed here.

Creating XQuery Clauses. Throughout the XQuery clause creation we had to overcome several difficulties, involving the accurate solution sequence cardinality, the association of different XQuery variables and the binding assurance.

Associating Different XQuery Variables. Throughout the creation of the For/Let XQuery clauses, the BGP2XQuery algorithm (Section 5.8.3) exploits the extension relation (Definition 15) in order to achieve the association of different XQuery variables. For example, consider the case where the XQuery variable $\$per$ that refers to Persons (corresponding to the XPath: $/Persons/Person$) should be associated with the XQuery variable $\$fn$, which refers to the First Names of the Persons (corresponding to the XPath: $/Persons/Person/FirstName$). This can be accomplished using For XQuery clauses and defining the XQuery variable $\$fn$ as an extension of the XQuery variable $\$per$, i.e., for $\$per$ in $/Persons/Person$ for $\$fn$ in $\$per/FirstName$.

Accurate Solution Sequence Cardinality. An interesting issue in the graph pattern translation is to ensure the generation of the appropriate solution sequence based on the SPARQL semantics. In our translation, this has been accomplished by the For or Let XQuery Clause Selection algorithm (Section 5.8.3.2) which determines the creation of the appropriate For or Let XQuery clauses.

Binding Assurance. In order to guarantee that all the variables defined in a basic graph pattern are bound in all the solutions, we have developed a binding condition assurance mechanism. The binding assurance mechanism exploits the XQuery function $exists()$ when it is required to guarantee the assignment of a value to the XQuery variables (Section 5.8.3.5).

Implementing SPARQL Operators. Several issues have arisen throughout the implementation of the SPARQL algebra operations (i.e., UNION, AND, OPT and FILTER) using XQuery expressions.

Well Designed Graph Patterns vs. Non-Well Designed Graph Patterns. The existence of non-well designed graph patterns, as well as the SPARQL operator semantics, which define the Join (i.e., AND) and Left Outer Join (i.e., OPT) operators as non null-rejecting forced us to handle the well designed in a different way from the non-well designed graph patterns. This way we have provided an efficient implementation for the former. In this implementation the intermediate results are used in the XQuery expression in order to reduce the computation cost (Section 5.8.3.1).

Efficient Join & Condition Implementation. The efficient implementation of some basic SPARQL query features using XQuery expressions is an interesting part of the translation. Consider as an example the translation of the joins occurring between the triple patterns (expressed with "." in the SPARQL syntax) in the Basic Graph patterns. In the context of BGPs, the joins are implemented efficiently by the BGP2XQuery algorithm (Section 5.8.3) that associates For/Let XQuery clauses that resemble a nested loop join. In addition, throughout the For/Let XQuery clause creation, the BGP2XQuery algorithm exploits the extension relation (Definition 15) in order to use the already evaluated XQuery values providing a more efficient join implementation. Another issue, is the translation of the literal parts of the triple patterns (Section 5.8.3.5), which are translated as conditions over the For/Let XQuery clauses using XPredicates. In this way, the conditions imposed by the existence of the literals are applied early in the XQuery evaluation plan, resulting in a more efficient XQuery evaluation.

Solution Structure. Remarkable issues are the need of the distinction of the different solutions in the solution sequence, as well as the distinction of the corresponding values for each variable in each solution. In this way, the SPARQL solution sequence modifiers and algebra operators can be applied on the results produced by the XQuery expressions. The above issues have been resolved by exploiting "special" markup tags (e.g., <Result>) throughout the creation of the XQuery Return clause (Section 5.8.3.7).

Handling SPARQL Filters. Several interesting issues resulted from the translation of the Filter expressions, including handling the safe and non-safe Filter expressions, mapping the SPARQL three-valued logic to the XQuery two-valued logic, translating the SPARQL operators used in the Filter expressions, etc.

Safe vs. non-Safe Filter Expressions. In order to provide efficient Filter translation, we try to evaluate the Filter Expressions in an early stage of the XQuery evaluation. This is achieved using XPredicates that apply the Filter conditions over the For/Let XQuery clauses. However, there are "special" cases, where the Filter Expressions can not be evaluated in an earlier stage, due to the SPARQL variables that occur in the Filter Expression. These cases are known as not safe Filter expressions (Definition 19). They occur because of the flexibility of the SPARQL syntax in expressing queries. In order to overcome this issue, our translation method evaluates the conditions defined in these Filter expressions at the end (using Where clauses), in order to guarantee that the variables occurring in the Filter expression have already been evaluated (Section 5.8.3.6).

Implementing Filter Expression Operators. Regarding the SPARQL operators included in Filter expressions, most of them can be directly mapped to XQuery built-in functions and operators (e.g., *regex*, &&, ||, +). However, for some "more complex" SPARQL operators (e.g., *sameTerm*, *lang*, etc.) we have developed native XQuery functions that simulate them. Finally, a few SPARQL operators (e.g., *isBlank*) can not be implemented in the

XQuery language, since they are not supported by the XML data model (Section 5.8.3.6).

Three-valued Logic vs. Two-valued Logic. The evaluation of Filter expressions in the SPARQL query language is based on three-valued logic (i.e., True, False and Error), while the XQuery query language supports Boolean logic (i.e., True and False). An issue is to handle and relate the three-valued logic using the XQuery Boolean logic. In our translation, for efficiency reasons, the SPARQL Error value has been mapped to the False XQuery value. However, it is possible, but inefficient, to support the Error value in the generated XQuery expressions (Section 5.8.3.6).

5.9 Solution Sequence Modifiers & Query Forms

5.9.1 Translating Solution Sequence Modifiers

This section describes the *Solution Sequence Modifier* translation phase, which translates the SPARQL Solution Sequence Modifiers (SSMs) into XQuery expressions. The SSMs that may be contained in a SPARQL query are translated using XQuery clauses and built-in functions. The SSMs supported by the current SPARQL specification are the Distinct, Reduced, Order By, Limit, and Offset solution sequence modifiers.

Table 5.8 summarizes the XQuery expressions and built-in functions that are used for the translation of the solution sequence modifiers. Let xE_{GP} be the XQuery expressions produced from the graph pattern (GP) translation. In Table 5.8, $\$Results$ is an XQuery variable, which is bound to the solution sequence produced by the XQuery expressions xE_{GP} (i.e., let $\$Results := (xE_{GP})$), n, m are positive integers and $?x$, $?y$ are SPARQL variables.

Table 5.8: Translation of the SPARQL Solutions Sequence Modifiers in XQuery expressions

Solution Sequence Modifier	XQuery Expressions
LIMIT n	**return**($Results[**position**( )<= n])
OFFSET n	**return**($Results[**position**( )> n])
LIMIT n && **OFFSET** m	**return**($Results[**position**( )> m **and position**( )<= $n + m$])
ORDER BY DESC(?x) **ASC**(?y)	**for** $res **in** $Results **order by** $res/x **descending empty least**, $res/y **empty least** **return** $res

Solution Sequence Modifier Priorities. If more than one solution modifiers are declared in the given SPARQL query, the order in which they are applied during the translation phase is the following (the order is compatible with the SPARQL query language semantics): (a) Order By, (b) Distinct / Reduced and (c) Offset / Limit.

5.9.2 Translating Query Forms

The *Query Form Translation* is the final phase of the SPARQL to XQuery translation. The current specification of the SPARQL query language supports four query forms: Select, Ask, Construct and Describe. According to the query form, the type of the returned results is different. In particular, after the translation of any solution sequence modifier, the generated XQuery is enhanced with the appropriate, for this query form, XQuery expressions in order to form the appropriate type of the results (e.g., an RDF graph, a result sequence, or a Boolean value).

Select Queries. The *Select* SPARQL queries return (all or a subset of) the variables bound in a query pattern match. To simulate this query form in XQuery, the results are

returned as sequences of XML elements created by the XQuery Return clauses (see the Build Return Clause Algorithm, Section 5.8.3.7). This sequence should be contained in a root element in order to be a valid XML document. Thus, we create a root element "*Results*" containing the result sets produced by the XQuery return clause.

Ask Queries. The *Ask* SPARQL queries return a Boolean value (*yes* or *no*), indicating whether a query pattern is matched in a dataset or not. The cardinality of the solution for Ask queries is one (i.e., the value yes/no should be returned once). Thus, we check for the existence of any result and we return "*yes*" if one or more results exist and "*no*" otherwise.

Construct Queries. The *Construct* SPARQL queries return an RDF graph structured according to the *graph template* of the query. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting the variables in the graph template, and combining the triples into a single RDF graph using the union operation.

In order to implement the semantics for the unbound variables, for each triple pattern of the graph template that contains variables we check if any one of these variables is unbound. In that case, the triple is not returned. Moreover, in order to enforce the semantics of the Blank node naming conventions in the RDF graph, we exploited an XQuery *positional variable* (defined using "*at*" term in XQuery syntax).

Describe Queries. The *Describe* SPARQL queries return an RDF graph which provides a "description" of the matching resources. The "description" semantics are not determined by the current SPARQL specification, however they are determined by the SPARQL query engines (note that several SPARQL engines do not support Describe queries). As a result, we provide an "approximate" support for this query form, by evaluating the Describe SPARQL query against the source ontology and then translating it as a Select query. The overall result is a combination of the RDF graph returned by the SPARQL Describe query and the result sequence returned by the translated XQuery.

$$
Q_X = \begin{cases}
\textbf{let } \$Results \textbf{ := (}xE_Q\textbf{)} & \textbf{if } QF = Select \\
\textbf{return (} \langle Results \rangle \$Results \langle /Results \rangle \textbf{)} & \\
 & \\
\textbf{let } \$Results \textbf{ := (}xE_Q\textbf{)} & \textbf{if } QF = Ask \\
\textbf{return (if (empty (}\$Results\textbf{)) then } \text{“no”} \textbf{ else } \text{“yes”} \textbf{)} & \\
 & \\
\textbf{let } \$Results \textbf{ := (}xE_Q\textbf{)} & \textbf{if } QF = Construct \\
\textbf{for } \$res \textbf{ at } \$iter \textbf{ in } \$Results & \\
\textbf{return (if (exists(} \$res/x \textbf{)) then} & \\
\quad \textbf{concat (concat (} \text{“\_:a”} \textbf{ ,} \$iter \textbf{),} \text{“ iri:property ”} \textbf{ , string(} \$res/x \textbf{) ,} \text{“.”} \textbf{)} & \\
\quad \textbf{else ( )} & \\
\quad \textbf{if (exists(} \$res/p \textbf{) and exists(} \$res/y \textbf{)) then} & \\
\quad \textbf{concat (concat(} \text{“\_:a”} \textbf{ ,} \$iter \textbf{), string(} \$res/p \textbf{) , string(} \$res/y \textbf{) ,} \text{“.”} \textbf{)} & \\
\quad \textbf{else ( ))} &
\end{cases} \tag{5}
$$

5.9.2.1 Translation Overview

The translation of the query forms described above is outlined in (5), where Q_X is the set of the XQuery expressions resulted after the translation of SPARQL query form. Let the

SPARQL query $Q_S = \langle QF, GP, SSM \rangle$, where QF is the query form, GP is the query graph pattern and SSM the solution sequence modifiers. Let xE_Q be the XQuery expressions produced from the graph pattern (GP) and solution sequence modifier (SSM) translation. For the Construct query form (last case in (5)), we consider the graph template: "_:a iri:property ?x. _:a ?p ?y.", which consists of two triple patterns and containing the blank node "_:a".

5.10 XQuery Rewriting/Optimization

It was pointed out in Section 5.5.3 that among the objectives of the proposed SPARQL to XQuery translation method was the generation of simple XQuery expressions, so that their correspondence with the SPARQL queries could be easily understood. This has led to the generation of some inefficient XQuery expressions, but it was expected that the XQuery optimizer would optimize those queries to achieve better execution performance. However, we have attempted to use the query optimizer of two XQuery engines with no improvement to be achieved for any of the queries. This observation led us to develop some XQuery rewriting rules and integrate them in our Framework. The performance evaluation studies (Section 5.12) show that they are useful in improving the XQuery performance.

In this section, we introduce a small number of simple rewriting rules aiming to provide more efficient XQuery expressions. These rules are applied to the XQueries generated from the SPARQL to XQuery translation. Since the XQuery performance is beyond the scope of this work, we present here a limited number of simple rules and in the Section 5.12.3 we examine their effect on the XQuery evaluation performance.

5.10.1 Rewriting Rules

The proposed rewriting rules aim at providing more efficient XQuery expressions that benefit from the knowledge of how the XQuery expressions are generated from the SPARQL to XQuery translation, as well as from the XML Schema semantics. The rules exploit the aforementioned, in order to remove redundant XQuery clauses and variables, unnest nested For XQuery clauses and minimize the loops executed by the For XQuery clause.

Note that the rewriting rules that we currently include in the Framework, focus on the optimization of XQuery expressions generated from the translation of Basic Graph Patterns. Optimization of the XQuery expressions generated from the translation of solution sequence modifies, algebra operators, etc. are not considered in this work.

The rewriting rules are applied sequentially on the generated XQuery queries. Firstly, *Rule* 1 is applied, then *Rule* 2 is applied on the resulting XQuery query, and finally *Rule* 3 is applied.

Rewriting Rule 1 [Changing For Clauses to Let]: Let xC be a For XQuery clause, A be the set of the XML elements and/or attributes corresponding to the XPath expressions contained in $xC.expr$. If each of the XML elements/attributes contained in A may appear at most once, then xC is changed from a For XQuery clause to a Let XQuery clause. Formally, the Changing For Clauses to Let rule is expressed as:

$$\forall a \in A : a.cardinality \in [0, 1] \Rightarrow xC.type \leftarrow Let$$

The element and attribute cardinality can be specified in the XML Schema using the *minOccurs* (minimum number of occurrences) and *maxOccurs* (maximum number of occurrences) XML Schema declarations. Thus, in our case we examine if the values of the minOccurs and the maxOccurs declarations are between zero and one. In this case, the element/attribute can not have multiple values, so a For clause which is going to perform an iteration over the element/attribute value is meaningless.

The *Changing For Clauses to Let* rule is applied on the For XQuery clauses, from the top to the bottom (or the inverse). Intuitively, this rule exploits the schema information in order to convert the For in Let clauses in cases where multiple values can not exist. The objective of this rule is to avoid the unnecessary checks for possible multiple values performed by the For clauses, in cases where Let clauses can also be used. The use of this rule results in more Let clauses that may be removed later, when Rule 2 is applied.

Rewriting Rule 2 [Reducing Let Clauses]: Let xCl_1 be a Let XQuery clause. If the variable $xCl_1.var$ of the Let clause, is extension (Definition 15) of another XQuery variable $xCl_2.var$, where xCl_2 is a For or Let XQuery Clause. Then the Let clause is removed and $xCl_1.var$ is replaced everywhere with $xCl_1.epr$. In addition, if a Biding Assurance Condition has been defined for $xCl_1.var$ i.e., an exists ($xCl_1.var$) statement in the Where XQuery clause (Section 5.8.3.5). Then, the exists function is removed and replaced by a condition over $xCl_2.expr$. The condition is defined using XPredicates and the XPaths of $xCl_1.expr$.

The *Reducing Let Clauses* rule is applied iteratively to the Let XQuery clauses, from the bottom to the top. Intuitively, this rule removes the unnecessary Let clauses that have been produced from triple pattern translation and can be pruned. The objective of the rule is to eliminate the unnecessary XQuery clauses and variables. In addition, in the case of Biding Assurance Condition existence, a *predicate pushdown* is performed. In particular, the exists condition placed in the Where XQuery clause is evaluated in an earlier query processing stage since it is applied to the XPaths using XPath predicates.

Let $xp \in \mathbf{XP}$ be an XPath expression and *expr* be a sequence of the form "xp_1 union xp_2 union $\cdots$ union xp_n"[12]. Moreover, $\$v$ is an XQuery variable and $funcX()$ is a user-defined or built-in XQuery function.

Formally, the *Reducing Let Clauses* rule is described as:

Initial XQuery Expressions

for/let $v1 **in/:=** expr1
let $v2 **:=** $v1/xp1
for/let $v3 **in/:=** $v2/xp2
…
where (**exists**($v2) … funcX($v2) …)
return (… $v2 …)

⇒

Rewritten XQuery Expressions

for/let $v1 **in/:=** expr1[./xp1]
…
for/let $v3 **in/:=** $v1/xp1/xp2
…
where (… funcX($v1/xp1) …)
return (… $v1/xp1 …)

Rewriting Rule 3 [Unnesting For Clauses]: Let xCl_1 be a For XQuery clause. If the variable of the For clause $xCl_1.var$ is not a Return Variable ($xCl_1.var \notin \mathbf{RV}$) and only one XQuery variable $xCl_2.var$ is extension (Definition 15) of $xCl_1.var$. Then, the For clause is removed and $xCl_1.var$ is replaced by the $xCl_1.expr$.

The *Unnesting For Clauses* rule is applied iteratively on the For XQuery clauses, from the top to the bottom. Intuitively, this rule unnests nested For clauses that can be expressed as a single For clause. The objective of the rule is to reduce the nested For clauses, in this way, also some XQuery clauses and variables are removed.

Let $xp \in \mathbf{XP}$ be an XPath expression and expr be a sequence of the form "xp_1 union xp_2 union $\cdots$ union xp_n". Moreover, $\$v$ is an XQuery variable and $funcX()$ is a user-defined or built-in XQuery function.

[12] At this point, it should be reminded that the disjunctions that may occur in *epxr* have been expressed as predicates inside the XPath expressions *xp*.

Formally, the *Unnesting For Clauses* rule is described as:

Initial XQuery Expressions

for v₁ **in** expr₁
…
for v₂ **in** v₁/xp₁
…
where (… funcX(v₁) …)
return (…)

⇒

Rewritten XQuery Expressions

for v₂ **in** expr₁/xp₁
…
where (… funcX(expr₁) …)
return (…)

5.11 Towards Supporting SPARQL Update Operations

In this section, we briefly describe an extension of the SPARQL2XQuery Framework in the context of supporting the SPARQL 1.1 update operations [334]. We present how similar methods and algorithms used in the SPARQL2XQuery Framework can be adopted for the update operation translation. For instance, graph pattern and triple pattern translation are also used in the update operation translation.

In order to support SPARQL update operations, we have studied the extension of the SPARQL to XQuery translation using the recently introduced XQuery Update Facility [322]. SPARQL 1.1 supports two main categories of update operations: a) Graph update, which includes operations regarding graph additions and removals; and b) Graph management, which contains "storage-level" operations, e.g., CREATE, DROP, MOVE, COPY, etc. Our work focuses on the graph update operations, since the storage-level operations are out of scope of the SPARQL2XQuery Framework working scenario (i.e., interoperability/integration scenario).

Considering SPARQL update queries, in the RDB–RDF interoperability scenario, D2R/Update [158] (a D2R extension) and OntoAccess [207] enable SPARQL update queries over relational databases. Regarding the XML–RDB–RDF interoperability scenario, the work presented in [31] extends the XSPARQL language [86] in order to support update queries.

5.11.1 Translating SPARQL Update Queries to XQuery

Table 5.9 presents the SPARQL update operations and summarizes their translation in XQuery. In particular, there are three main categories of SPARQL update operations a) *Delete Data*; b) *Insert Data*; and c) *Delete/Insert*. For each update operation, a simplified SPARQL syntax template is presented, as well as the corresponding XQuery expressions. In SPARQL context, we assume the following sets, let tr be an RDF triple set, tp a triple pattern set, trp a set of triples and/or triple patterns, and gp a graph pattern. Additionally, in XQuery, we denote as xE_W, xE_I and xE_D the sets of XQuery expressions (i.e., FLOWR expressions) that have resulted from the translation of the graph pattern included in the Where, Insert and Delete SPARQL clauses, respectively. Let xE be a set of XQuery expressions, $xE(\$v_1, \$v_2, \ldots \$v_n)$ denote that xE are using (as input) the values assigned to XQuery variables $\$v_1, \$_v2, \ldots \$v_n$. Finally, xn denotes an XML fragment, i.e., a set of XML nodes, and xp denotes an XPath expression.

Delete Data. The *Delete Data* SPARQL operation removes a set of triples from RDF graphs. This SPARQL operation can be translated in XQuery using the *Delete Nodes* XQuery operation. Specifically, using the predefined mappings, the set of triples tr defined in the SPARQL Delete Data clause is transformed (using a similar approach such as the BGP2XQuery algorithm, Section 5.8.3) in a set of XPath expressions **XP**. For each $xp_i \in$ **XP** an XQuery Delete Nodes operation is defined.

Table 5.9: Translation of the SPARQL Update Operations in XQuery

SPARQL		Translated XQuery Expressions
SPARQL Update Operation	**Syntax Template** [1]	

DELETE DATA	**Delete data**{
	tr
	}
		delete nodes collection("http://dataset...")/xp_1
		...
		delete nodes collection("http://dataset...")/xp_n

INSERT DATA	**Insert data**{
	tr
	}
		let n1 := xn_1
		…
		let nn := xn_n
		let $data1 := ($nk, $nm,…) // k, m,… ∈[1,n]
		…
		let $datap := ($nj, $nv,…) // j, y,… ∈[1,n]
		let $insert_location1 **:= collection**("http://xmldataset...")/xp_1
		…
		let $insert_locationp **:= collection**("http://xmldataset...")/xp_p
		return(
		insert nodes $data1 **into** $insert_location1 ,
		…
		insert nodes $datap **into** $insert_locationp
		)

DELETE / INSERT	(a) **Delete**{
	trp
	}**Where**{
	gp
	}
	
	(b) **Insert**{
	trp
	}**Where**{
	gp
	}
	
	(c) **Delete**{
	trp
	}**Insert**{
	trp
	}**Where**{
	gp
	}
		(a)
		let $where_gp := xE_W
		let $delete_gp:= xE_D($where_gp)
		return delete nodes $delete_gp
		
		(b) **let** $where_gp := xE_W
		let $insert_location1 := xp_1
		for $it1 **in** $insert_location1
		xE_I($where_gp, $it1)
		return insert nodes into $it1
		…
		let $where_gp := xE_W
		let $insert_ location n := xp_n
		for $itn **in** $insert_locationn
		xE_I($where_gp, $itn)
		return insert nodes into $itn
		
		(c)
		Translate Delete Where same as (a),
		then translate Insert Where same as (b)

[1] For simplicity, the WITH, GRAPH and USING clauses are omitted.

Insert Data. The *Insert Data* SPARQL operation, adds a set of new triples in RDF graphs. This SPARQL operation can be translated in XQuery using the *Insert Nodes* XQuery operation. In the Insert Data translation, the set of triples tr defined in SPARQL are transformed into XML node sets xn_i, using the predefined mappings. In particular, a set of Let XQuery clauses is used to build the XML nodes and define the appropriate node nesting and grouping. Then, the location of the XML node insertion can be easily determined considering the triples and the mappings. Finally, the constructed nodes are inserted in their location of insertion using the XQuery Insert nodes clause.

Insert / Delete. The *Delete/Insert* SPARQL operations are used to remove and/or add a set of triples from/to RDF graphs, using the bindings that resulted from the evaluation of the graph pattern defined in the Where clause. According to the SPARQL 1.1 semantics, the Where clause is the first one that is evaluated. Then, the Delete/Insert clause is applied over the produced results. Especially, in case, that both Delete and Insert operations exist, the deletion is performed before the insertion, and the Where clause is evaluated once. The Delete and the Insert SPARQL operations can be translated to XQuery using the *Delete Nodes* and *Insert Nodes* operations, respectively. In brief, initially the graph pattern used in the Where clause is translated to XQuery expressions xE_W (similarly as in the GP2XQuery, Section 5.8). Then, the graph pattern used in the Delete/Insert clause is translated to XQuery expressions xE_D/xE_I (as it is also in the BGP2XQuery, Section 5.8.3) using also the bindings that resulted from the evaluation of xE_W.

5.12 Experimental Analysis

In this section we present the results of the experimental evaluation that we have conducted on the SPARQL2XQuery Framework using both synthetic and real datasets. The objective was to evaluate the efficiency of: (a) schema transformation; (b) mapping generation; (c)

query translation; and (d) query evaluation. We have used several query sets attempting to cover almost all the SPARQL syntax variations, features and special cases.

The SPARQL2XQuery Framework has been implemented using Java related technologies (Java 2SE and Jena) on top of the open source, native XML database. The experimental evaluation was performed on an Intel Xeon processor at 2.00 Ghz, with 16 GB RAM, running Linux and Java 1.6. We have used two native XML Databases (and their XQuery engines) denoted as "XML Store Y" and "XML Store Z". In addition, we have used a memory-based XQuery engine denoted as "Memory-based XQuery Engine". For RDF store, we have used the Jena TDB 0.10.1 storage component and the Jena ARQ 2.10.1 SPARQL engine. Finally, for the evaluation of the XS2OWL component we used two XSLT processors, a freeware XSLT processor denoted as "Freeware XSLT Processor", and the XSLT processor that is integrated in a commercial tool, denoted as "Commercial XSLT Tool ". Note that, in all experiments, the default configurations for all the software have been used.

The rest of this section is structured as follows. We discuss the performance of the schema transformation and mapping generation processes in Section 5.12.1, we examine the efficiency of the translation process in Section 5.12.2, we present the query evaluation efficiency in Section 5.12.3 and we provide an evaluation overview in Section 5.12.4.

5.12.1 Schema Transformation & Mapping Generation Performance

In order to evaluate the SPARQL2XQuery Framework, we have used several international standards from different domains (e.g., Digital Libraries, Cultural Heritage, Multimedia) that have been expressed in XML Schema. The Persons XML Schema that we have defined in Section 5.3.2 has also been used. These XML Schemas have been used in order to evaluate the schema transformation and mapping generation processes. The basic characteristics (e.g., number of elements, attributes, etc.) of the XML Schemas used in the evaluation can be found in [84].

In what follows, we present the results of the experiment we conducted in order to study the schema transformation and mapping generation performance. Both the schema transformation and the mapping generation processes are off-line processes and are performed once for every XML Schema in the context of first scenario (*Querying XML data based on automatically generated ontologies*). Although these processes are off-line and are performed once for every XML Schema, we can observe from this experiment that we can characterize them as lightweight processes that take negligible time even for very large XML Schemas (e.g., schemas with 4000 XML Schema constructs).

In this experiment we have used several international standards that have been expressed in XML Schema. For each of these XML Schemas, we have used the XS2OWL component in order to automatically transform the XML Schema in OWL ontologies, measuring the time required for this transformation (*Schema Transformation Time*). Then, using the generated Schema ontology and the XML Schema, we measure the time required for the Mapping Generator component of the SPARQL2XQuery Framework to automatically discover and generate the mappings (*Mapping Generation Time*).

Table 5.10 presents the Schema Transformation Time and the Mapping Generation Time for each XML Schema. Notice that the schema transformation time is presented for both *Freeware XSLT Processor* and *Commercial XSLT Tool*. The schema transformation time mainly depends on: (a) The number of the XML Schema constructs, since this number corresponds to the number of the transformations performed; and (b) The size of the XML Schema file, since it should be parsed. Similarly, the mapping generation time basically depends on: (a) The number of the XML Schema constructs, since this number equals to the number of the generated mappings; and (b) The size of the XML Schema and ontology files, since these files should be parsed.

Table 5.10: Schema Transformation & Mapping Generation Time (msec)

XML Schema	Schema Transformation Time		Mapping Generation Time
	Freeware XSLT Processor	Commercial XSLT Tool	
Persons (Section 5.3.2)	2.4	17.9	8.7
DBLP ⋆	62.5	22.5	360.9
METS [6]	58.2	270.5	388.9
Text MD [14]	7.7	45.1	14.5
MPEG-7 [9]	730.7	3500.6	1954.2
SCORM 12 [12]	132.7	415.2	421.1
MARC 21 [4]	6.3	51.4	12.5
MODS [7]	191.3	594.8	482.3
TEI [15]	840	980.1	2208.4
TEI Lite [15]	418	932.6	1288.3
EAD [2]	402.7	3305.7	1052
VRA Core 4 [16]	47.3	290	304.3
VRA Core 4 Strict [16]	3.3	122.1	10
MIX [10]	200	601.3	495.5
MADS [5]	50.1	393.4	345.6

⋆ Note that in our experiments, the DTD that originally describes the DBLP dataset has been expressed in XML Schema syntax.

We can observe from Table 5.10 that, for both the XSLT processors, the TEI and MPEG-7 require the maximum transformation time (840.0 and 730.7 msec respectively) due to their large number of XML Schema constructs (4279 and 2567 constructs respectively, see [84]). On the other hand, due to the small number of XML Schema constructs, the Persons (13 constructs, [84]) and VRA Core 4 Strict (19 constructs, [84]) require the minimum transformation time (2.4 and 3.3 msec respectively). Finally, as at is expected, the XML Schema file size slightly affects the transformation time. For example, despite the large size (345.3Kb, [84]) of the SCORM 21 XML Schema file, the transformation time is not analogously high (132.7 msec) due to its small number of XML Schema constructs (i.e., small number of transformations).

In addition, we observe that the TEI and MPEG-7 require the maximum mapping generation time (2208.4 and 1954.2 msec respectively) due to their large number of XML Schema constructs (i.e., number of mappings discovered and generated). On the other hand, the Persons and VRA Core 4 Strict require the minimum mapping generation time (8.7 and 10.0 msec respectively).

5.12.2 Translation Efficiency

In this section we present the experimental results related to the efficiency of the SPARQL to XQuery translation process. To evaluate the efficiency of the translation process, we measured the translation time required by the SPARQL2XQuery Framework to translate a SPARQL query to an XQuery query. Below, we present three experiments. In the first experiment (Section 5.12.2.1), we have generated several SPARQL queries by modifying their graph pattern size and type. In the second experiment (Section 5.12.2.1), we have varied in the previously generated queries the number of mappings between the ontology and the XML Schema. Finally, in the third experiment we have employed three SPARQL query sets attempting to cover all the SPARQL grammar variations (Section 5.12.2.2).

5.12.2.1 Translation Time for different Graph Patterns & Mappings

Here, we examine the efficiency of the query translation process. The translation time mainly depends on two factors:

(a) The number of the SPARQL variables included in the Graph Pattern, since the SPARQL variable number determines: (i) the number of the XQuery clauses generated throughout the translation; (ii) the number of the required Variable Binding phases; and (iii) the number of the required Variable Type Determination phases.

(b) The complexity of the variable binding determination process. In particular, the complexity of the variable binding determination depends on: (i) the number of the XPath Set operations; (ii) the type of the XPath Set operations; and (iii) the size of the operands (i.e., the size of the XPath Sets).

In the first experiment, we have generated several SPARQL queries by modifying the size and the type of their graph patterns. For the SPARQL query generation, we assumed that the queries are expressed on an ontology that has been mapped to an XML Schema. We also assume that the ontology has the properties P_i with $1 \leq i \leq 30$ (i.e., P_1, P_2,..., P_{30}). In the second experiment, for each of the generated SPARQL queries we have varied the number of the predefined mappings (i.e., the XPath Set sizes) between the ontology and the XML Schema.

Note that the queries generated for these experiments are Select SPARQL queries, containing one return variable and their Where clause is a Graph Pattern consisting of sequences of conjunctive triple patterns (i.e., Basic Graph Pattern).

5.12.2.1.1 Varying the Graph Pattern Type & Size

In this experiment, we have obtained several (different) SPARQL queries by modifying the type and the size of their graph pattern. To this end, we have varied (a) the number; and (b) the type of the triple patterns included in the graph pattern. The number of triple patterns determines the number of SPARQL variables and, as a consequence, the number of the generated XQuery clauses. The triple pattern type determines: (a) the number of the SPARQL variables; (b) the number of the XPath Set operations; and (c) the type of the XPath Set operations.

Table 5.11: Translation characteristics over the number of Tripple Patterns (n)

Graph Pattern Type	Characteristics w.r.t. Number of Triple Patterns (n)		
	SPARQL Variables	Generated XQuery Clauses	XPath Set Operations
GP_1	$2n$	$2n$ For/Let, 1 Where and 1 Return	$2n$ ($\overline{\cap}$) and n ($\leq$)
GP_2	n	n For/Let, 1 Where and 1 Return	n ($\overline{\cap}$)
GP_3	$3n$	$3n$ For/Let, 1 Where and 1 Return	$2n$ ($\overline{\cap}$), n ($>$) and n ($\leq$)
GP_4	$2n$	$2n$ For/Let, 1 Where and 1 Return	$2n$ ($\overline{\cap}$) and n ($>$)

We have defined four types of graph patterns (GP_1, GP_2, GP_3 and GP_4) by modifying the types of the included triple patterns: (a) GP_1 =$?x_1$ P_1 $?y_1$. $?x_2$ P_2 $?y_2$ $\cdots$ $?x_n$ P_n $?y_n$ (b) GP_2 =$?x_1$ P_1 "abc". $?x_2$ P_2 "abc" $\cdots$ $?x_n$ P_n "abc" (c) GP_3 =$?x_1$ $?y_1$ $?z_1$. $?x_2$ $?y_2$ $?z_2$. $\cdots$ $?x_n$ $?y_n$ $?z_n$ and (d) GP_4 =$?x_1$ $?y_1$ "abc". $?x_2$ $?y_2$ "abc" $\cdots$ $?x_n$ $?y_n$ "abc", where n is the number of triple patterns. Table 5.11 presents the basic characteristics of the SPARQL to XQuery translation for the previous graph pattern types. The last column refers to the XPath Set operations occurring in the variable binding phase.

Table 5.12: Query Translation Time & SPARQL Parsing Time vs. Graph Pattern Type and Size

Query Translation Time [*SPARQL Parsing Time*]							
Graph Pattern Type	**Number of Triple Patterns (*n*)**						
	1	**3**	**7**	**10**	**15**	**20**	**30**
GP_1	2.09 [*0.14*]	2.13 [*0.15*]	2.17 [*0.17*]	2.20 [*0.66*]	2.37 [*0.69*]	2.91 [*0.70*]	3.93 [*0.72*]
GP_2	2.07 [*0.38*]	2.07 [*0.37*]	2.11 [*0.38*]	2.13 [*0.39*]	2.29 [*0.42*]	2.79 [*0.46*]	3.81 [*0.65*]
GP_3	3.22 [*0.22*]	3.26 [*0.24*]	3.28 [*0.29*]	3.39 [*0.32*]	3.74 [*0.37*]	3.89 [*0.41*]	4.25 [*0.46*]
GP_4	3.21 [*0.21*]	3.26 [*0.24*]	3.29 [*0.28*]	3.35 [*0.30*]	3.64 [*0.31*]	3.76 [*0.34*]	4.04 [*0.40*]
Average	2.65 [*0.24*]	2.68 [*0.25*]	2.71 [*0.28*]	2.76 [*0.42*]	3.01 [*0.45*]	3.34 [*0.48*]	4.01 [*0.56*]

For each of the above graph pattern types (GP_1, GP_2, GP_3 and GP_4), we have constructed graph patterns containing n triple patterns with $n = 1, 3, 7, 10, 15, 20, 30$. Finally, for each ontology property P_i, we have assumed the mapping $P_i \equiv \{/a/b/i\}$. The translation time required by the SPARQL2XQuery Framework for the SPARQL to XQuery translation of each query is presented in Table 5.12. The SPARQL parsing time is also presented in Table 5.12. As SPARQL Parsing Time we refer to the time required by an SPARQL query engine to parse the SPARQL query and build the query object. Note that, in order to translate the SPARQL queries, a parsing phase using a SPARQL engine is required. As a result, in Table 5.12 the translation time a [b] means that the total translation time is a msec and it includes the SPARQL parsing time, which is b msec.

We can observe from Table 5.12 that the GP_2 type has achieved the lowest translation time. This is due to the fact that GP_2 contains only one variable. On the other hand, the GP_3 type has taken the maximum translation time, since the GP_3 contains the maximum number of variables in the triples and as a result, a large number of variable binding and determination of variable type phases are required.

In particular, the single variable included in the triple patterns of the GP_2 graph patterns has resulted in the generation of a small number of XQuery clauses. For a query with n triple patterns, $n + 2$ XQuery clauses (n For/Let, one Where and one Return) have been generated (Table 5.11). Accordingly, the three variables included in the triple patterns of the GP_3 graph patterns have resulted in a large number (i.e., $3n+2$) of XQuery clauses.

Finally, the triple patterns of the GP_1 and GP_4 graph patterns include two variables. Throughout the translation, $2n+2$ XQuery clauses have been generated. Despite the same number of generated XQuery clauses, variable binding and variable type determination phases, the GP_1 type has achieved lower translation time than the GP_4 type. This is explained as follows: The variable binding determination process for the GP_4 graph patterns is of higher complexity than that of GP_1, since the predefined mappings for the P_i properties in the GP_1 graph patterns reduce the number of possible bindings, and, therefore, the complexity of determining the variable bindings decreases (see Section 5.7 for details).

5.12.2.1.2 Varying the Number of Mappings

In this experiment we have used the SPARQL queries generated in the previous experiment. In addition, we have varied here the number of the predefined mappings between the ontology and the XML Schema; In particular, we have modified the number of the mappings (i.e., the size of XPath Sets) for all the ontology properties P_i, and, therefore we have modified the complexity of the variable binding phase.

In this experiment, for each property P_i we have assumed the mapping $P_i \equiv \{/a/b/c_i\}$, which contains one XPath expression. We have then modified the number of the XPath expressions that correspond to each P_i mapping. Hence, for each property P_i, we have the

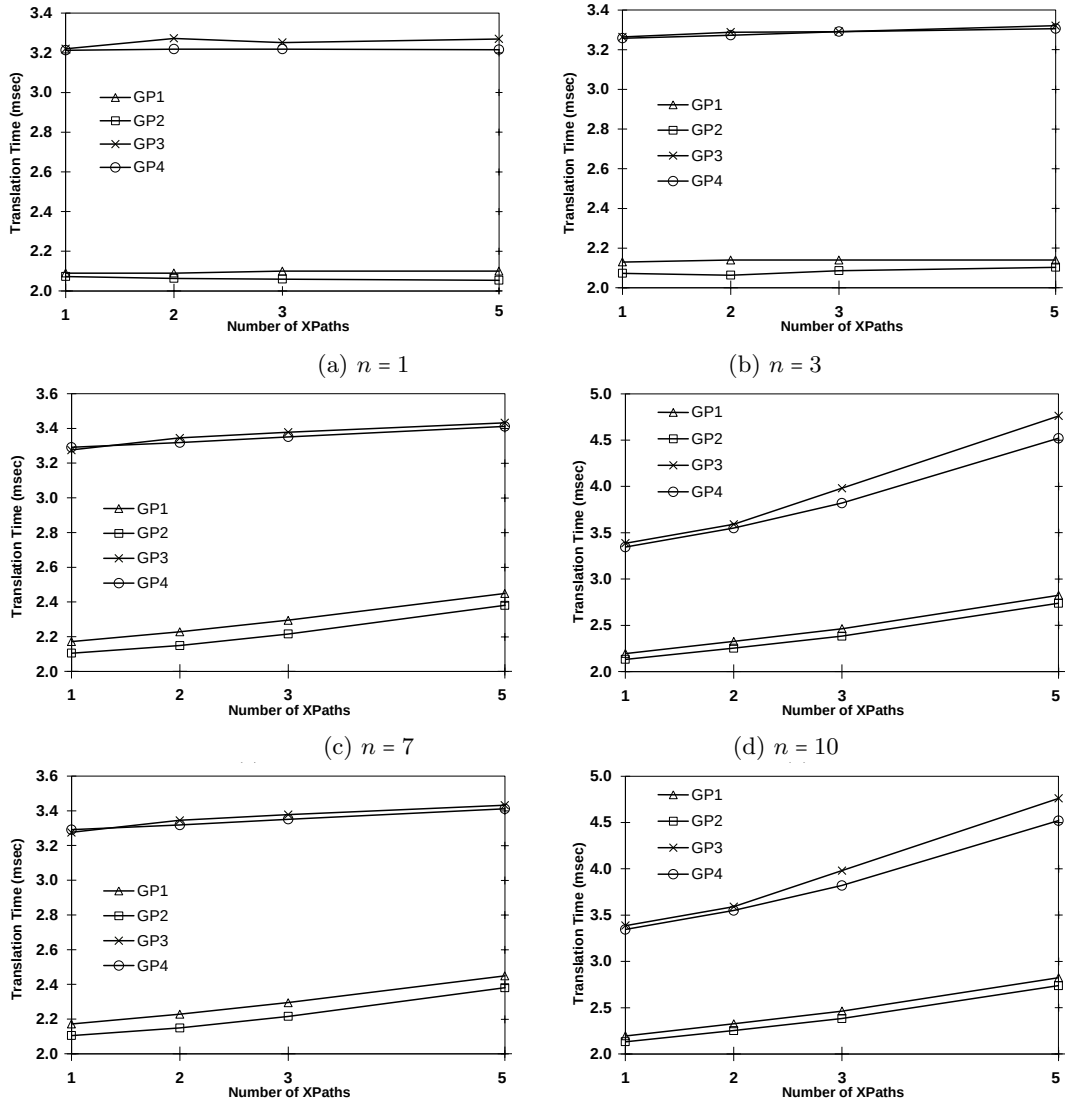


Figure 5.8: Query Translation Time vs. Number of mappings ($n = 1, 2, 3, 7, 10, 20$ and 30 triple patterns)

mapping $P_i \equiv \{/a/b/i_1, /a/b/i_2, \ldots, /a/b/i_k\}$, with $k = 1, 2, 3, 5$ being the number of XPath expressions for each P_i mapping.

Figure 5.8 presents the query translation time with a varying number of XPath expressions per mapping. Each diagram of Figure 5.8 corresponds to a specific number (n) of triple patterns and depicts all the graph pattern types (GP_1, GP_2, GP_3 and GP_4), while varying the number of XPath expressions from 1 to 5.

As it is expected, the number of XPath expressions per mapping had no effect in the query translation time for a small number of triple patterns (Figures 5.8a & 5.8b), because of the very low translation time required and the small number of operations involved. In particular, for $n = 1$ and $n = 3$ triple patterns the translation time for all the graph pattern types remains stable as the number of XPath expressions increases. For graph patterns containing $n = 7$ triple patterns (Figure 5.8c) we observe only a slight increase in the query translation time as the number of XPath expressions increases. For $n > 7$, as the number of XPath expressions increases, the query translation time grows linearly for all the graph pattern types. The former is explained as follows: Increasing the number of XPath expressions results in the (analogous) increase of the iterations for parsing and

processing the XPath Sets.

5.12.2.2 Translation Time for the Persons, DBLP & Berlin Query Sets

5.12.2.2.1 Query Sets

In this section we present the three query sets that have been exploited in our experiments. The first query set comprises the 12 SPARQL queries of the Berlin SPARQL Benchmark [91]. An overview of the SPARQL features that are used by the Berlin query set can be found in [84]. The second query set (Persons Queries) contains 15 SPARQL queries based on the Persons ontology (Table 6.5 and Table 6.6). The third query set (DBLP Queries) contains 5 SPARQL queries based on the DBLP ontology. The last two query sets have been used for evaluating our system in terms of: (a) the translation time; and (b) the query evaluation time.

The second and the third SPARQL query sets attempt to cover almost all the SPARQL grammar variations, features and special cases, with varying SPARQL query types (e.g., Select, Ask, etc.), graph patterns with different sizes and complexity. These queries use all the SPARQL algebra operators (e.g., Optional, Union, Filter, etc.), exploit combinations of the solution sequence modifiers (e.g., Limit, Offset, Order by, etc.) and contain several other features (e.g., Built-in functions, Schema triples, complex Filter conditions, etc.). In our attempt to cover almost all the possible SPARQL syntax variations and special cases, we have also considered the existing SPARQL Benchmarks (Berlin SPARQL Benchmark [91], SP2Bench [336], W3C SPARQL Implementation Coverage Report[13] and W3C DAWG Test cases[14]) throughout the query set specification.

All the SPARQL queries, the translated XQuery queries, as well as an analysis of their characteristics and features can be found in [84].

5.12.2.2.2 Results

In this experiment we have evaluated the efficiency of the translation process by exploiting several different SPARQL queries. We have utilized three SPARQL query sets, attempting to cover almost all the SPARQL syntax variations, features and special cases. For each query, we have measured the translation time required by the SPARQL2XQuery Framework to translate the SPARQL query in XQuery expressions. The query translation time and the SPARQL parsing time as well as the average parsing and translation time for each query set are presented in Table 5.13.

Table 5.13a presents the translation times for the 15 queries of the Persons query set. We can observe from [84] that the queries of the Persons query set have in average 4 triple patterns per query and 1–2 XPaths per mapping. In addition, some of these queries contain one or more solution sequence modifiers and Schema Triples. The translation of the solution sequence modifiers and the Schema Triples has made the translation time for the queries of the Persons query set slightly higher than the translation time of the queries of the previous experiment, since the later have neither solution sequence modifiers nor Schema Triples.

The translation time for the queries of the DBLP query set are presented in Table 5.13b. These queries have some characteristics similar to the ones of the Persons query set (i.e., in average 4 triple patterns per query and 1–2 XPaths per mapping). However, the DBLP queries are more complex in order to encapsulate most of the SPARQL features in five queries, thus resulting in slightly higher translation times compared to the ones of the Persons query set.

[13] www.w3.org/2001/sw/DataAccess/tests/implementations
[14] www.w3.org/2001/sw/DataAccess/tests/r2

Table 5.13: Query Translation & SPARQL Parsing Time (msec) for (a) Person, (b) DBLP and (c) Berlin Query Sets

(a) Person Query Set

Persons Query	Translation Time	SPARQL Parsing Time
Q_1	3.35	0.90
Q_2	3.35	0.80
Q_3	3.31	0.97
Q_4	3.32	0.74
Q_5	3.34	0.62
Q_6	3.30	0.50
Q_7	3.32	0.87
Q_8	6.23	0.49
Q_9	6.46	0.68
Q_{10}	3.26	0.34
Q_{11}	3.30	0.39
Q_{12}	3.29	0.39
Q_{13}	3.28	0.50
Q_{14}	3.26	0.32
Q_{15}	3.26	0.29
Avg.	**3.71**	**0.59**

(b) DBLP Query Set

DBLP Query	Translation Time	SPARQL Parsing Time
Q_1	5.73	1.2
Q_2	4.19	1.4
Q_3	7.70	1.2
Q_4	7.62	1.0
Q_5	3.89	0.6
Avg.	**5.83**	**1.1**

(c) Berlin Query Set

Berlin Query	Translation Time	SPARQL Parsing Time
Q_1	4.04	1.29
Q_2	13.82	0.90
Q_3	10.54	0.88
Q_4	7.26	0.82
Q_5	3.85	0.99
Q_6	3.61	0.50
Q_7	16.11	0.79
Q_8	19.02	0.71
Q_9	3.55	0.28
Q_{10}	3.70	0.51
Q_{11}	6.63	0.29
Q_{12}	3.72	0.48
Avg.	**7.99**	**0.70**

Finally, Table 5.13c presents the translation times for the Berlin query set. This query set is the most complex, with an average of 8 triple patterns per query, 1–4 XPaths per mapping, several solution sequence modifiers per query and several OPTIONAL and FILTER operators. The highest translation times occur in Queries 7 and 8 with 14 and 10 triple patterns respectively, 4 OPTIONAL operators, FILTERs and solution sequence modifiers.

5.12.3 Query Evaluation Efficiency

In this section we present the experimental results that refer to the efficiency of evaluating the XQuery expressions generated by the SPARQL2XQuery Framework. In Section 5.12.3.1 we outline the datasets, queries and metrics that are used in our evaluation scenario. In the first part of this experiment (Section 5.12.3.2) we have employed the synthetic Persons XML dataset and the Persons query set. In the second part (Section 5.12.3.3) we have utilized the real DBLP XML dataset and the corresponding query set.

5.12.3.1 Methodology

Datasets. In order to evaluate the SPARQL2XQuery Framework in term of query evaluation efficiency, we have used both real and synthetic datasets. The real dataset we have employed is the XML DBLP dataset. The characteristics of the DBLP dataset have been presented in [336]. The size of the DBLP dataset is 833Mb. First we have manually expressed the DTD that describes the DBLP dataset in XML Schema syntax. Then, the XML Schema has been transformed to an OWL ontology using the XS2OWL component. The DBLP XML Schema and the ontology generated by XS2OWL are available in [84].

Our synthetic dataset is structured according to the Persons XML Schema (Figure 6.2). The SPARQL queries expressed on it are based on the Persons OWL ontology generated for this XML Schema by the XS2OWL component (Table 6.5 and Table 6.6). For the generation of the synthetic XML dataset that follows the Persons XML Schema, we have implemented a data generator that takes as input a factor N, which is the number of the records to be generated. Finally, all the Persons XML datasets have been transformed in RDF format, in order to be able to perform a native evaluation of the SPARQL queries on them.

Table 5.14: Characteristics of the Persons XML datasets DT_1 to DT_{10} and the corresponding RDF datasets

XML Dataset Characteristics				Corresponding RDF Dataset Characteristics	
Dataset Name	***N***	**XML Nodes**	**Size** (Kb)	**Triples**	**Size** (Kb)
DT1	10^2	1450	20	$6{\cdot}10^2$	40
DT2	$5{\cdot}10^2$	7250	10^2	$3{\cdot}10^3$	$2{\cdot}10^2$
DT3	10^3	$145{\cdot}10^2$	$2{\cdot}10^2$	$6{\cdot}10^3$	$4{\cdot}10^2$
DT4	$5{\cdot}10^3$	$725{\cdot}10^2$	10^3	$3{\cdot}10^4$	$2{\cdot}10^3$
DT5	10^4	$145{\cdot}10^3$	$2{\cdot}10^3$	$6{\cdot}10^4$	$4{\cdot}10^3$
DT6	$5{\cdot}10^4$	$725{\cdot}10^3$	10^4	$3{\cdot}10^5$	$2{\cdot}10^4$
DT7	10^5	$145{\cdot}10^4$	$2{\cdot}10^4$	$6{\cdot}10^5$	$4{\cdot}10^4$
DT8	$5{\cdot}10^5$	$725{\cdot}10^4$	10^5	$3{\cdot}10^6$	$2{\cdot}10^5$
DT9	10^6	$145{\cdot}10^5$	$2{\cdot}10^5$	$6{\cdot}10^6$	$4{\cdot}10^5$
DT10	$5{\cdot}10^6$	$725{\cdot}10^5$	10^6	$3{\cdot}10^7$	$2{\cdot}10^6$

Table 5.14 summarizes the basic features of the Persons XML datasets, including the size in Kilobytes, the approximate number of XML nodes, etc. We have generated 10 datasets (DT_1 to DT_{10}), varying the N factor from 10^2 to $5 \cdot 10^6$. In addition, Table 5.14 presents the characteristics of the corresponding RDF datasets (i.e., number of triples and size in Kilobytes) that have been generated from the XML dataset transformation.

Queries. In our evaluation scenario, every SPARQL query Q_S of the Persons and DBLP query sets, has been automatically translated by the SPARQL2XQuery Framework to the XQuery query Q_{X_a}. Moreover, Q_S has been independently manually translated by an external expert to the XQuery Q_{X_m}. The Q_{X_m} queries have been expressed considering the XML Schema semantics and after applying techniques aiming to provide efficient XQuery queries. Finally, the rewriting rules defined in Section 5.10 have been applied on the automatically generated XQuery queries (Q_{X_a}), to obtain the automatically rewritten XQuery queries Q_{X_a-Rw}.

Evaluation Metrics. In order to study the efficiency of the XQuery queries generated by the SPARQL2XQuery Framework, we have measured and compared the query evaluation times for (a) the original SPARQL queries, natively executed using a SPARQL engine; (b) the automatically generated (Q_{X_a}) XQuery queries; (c) the automatically rewritten (Q_{X_a-Rw}) XQuery queries; and (d) the manually translated (Q_{X_m}) XQuery queries. Note that the XQuery evaluation times heavily rely on the underling XML data management system (e.g., storage, indexing, query engine, query optimizer, configuration, etc.).

5.12.3.2 Synthetic Dataset

In this experiment we study the efficiency of the XQuery queries generated by the SPARQL2XQuery Framework using synthetic datasets (Table 5.14). We have measured and compared the query evaluation times of the automatically generated, rewritten and manually translated XQuery queries.

In the rest of this section, we analyze the evaluation times for each query (Section 5.12.3.2.1), we vary the dataset size in order to examine the query evaluation efficiency over the dataset size (Section 5.12.3.2.2) and we compare the query evaluation time with the query translation time (Section 5.12.3.2.3).

Table 5.15: Query Evaluation Time over the Persons DT_8 dataset (XML Store Y)

Query Evaluation Time (sec)						
Query	**SPARQL** (Q_S)	**Manual** (Q_{Xm})	**Auto-Rw** ($Q_{Xa\text{-}Rw}$)	**Auto** (Q_{Xa})	**Auto-Rw vs. Auto**	**Auto-Rw vs. Manual**
Q_1	1.66	5.95	4.30	6.78	57.7 %	27.7 %
Q_2	1.69	5.96	4.28	6.76	57.8 %	28.1 %
Q_3	1.53	0.41	0.42	0.45	7.6 %	-1.0 %
Q_4	2.78	10.79	11.00	11.08	0.7 %	-1.9 %
Q_5	10.83	55.70	55.77	63.97	14.7 %	-0.1 %
Q_6	1.55	6.55	6.49	6.89	6.1 %	0.9 %
Q_7	1.36	0.91	0.92	0.93	1.2 %	-0.2 %
Q_8	6.03	12.93	13.09	13.11	0.2 %	-1.3 %
Q_9	5.34	3.21	3.22	5.76	79.1 %	-0.3 %
Q_{10}	0.00	6.63	5.74	6.91	20.4 %	13.4 %
Q_{11}	21.74	14.89	15.07	16.47	9.3 %	-1.2 %
Q_{12}	2.44	15.47	15.49	15.74	1.6 %	-0.1 %
Q_{13}	0.00	0.23	0.24	0.25	5.5 %	2.1 %
Q_{14}	1.37	3.69	3.61	3.80	5.2 %	2.2 %
Q_{15}	2.74	9.14	15.69	15.88	1.2 %	-71.7 %
Average	**4.07**	**10.17**	**10.36**	**11.65**	**12.5 %**	**-1.9 %**

5.12.3.2.1 Query Evaluation Time Analysis

We have used the synthetic Persons dataset DT_8 and the Persons query set. The DT_8 dataset comprises $5{\cdot}10^5$ records of persons and students (250K persons and 250K students), is of size 105Kb and has approximately $725 \cdot 10^4$ XML nodes.

Table 5.15 summarizes the results of the comparison of the execution of the SPARQL as well as the automatically generated, rewritten and manually translated XQuery queries. In particular, for each query, Table 5.15contains the evaluation times for (a) the SPARQL queries (denoted as SPARQL); (b) the manually translated XQuery queries (denoted as *Manual*); (c) the automatically rewritten XQuery queries (denoted as *Auto-Rw*); and (d) the automatically generated (without rewriting) XQuery queries (denoted as *Auto*). In addition, Table 5.15 presents the improvement of the rewritten compared to the automatically generated queries (denoted as *Auto-Rw vs. Auto*) as well as the comparison between the automatically rewritten and manually translated XQuery queries (denoted as *Auto-Rw vs. Manual*). The measuring unit for the evaluation time is second (sec).

Automatically Rewritten vs. Automatically Generated (Auto-Rw vs. Auto). We can observe from Table 5.15 that for almost all the queries the evaluation times for the rewritten queries presented a notable performance improvement compared to the automatically generated ones. The average reduction in the evaluation time for the rewritten queries was 12.5%, and the maximum 79.1%.

For five (Q_4, Q_7, Q_8, Q_{12} and Q_{15}) out of fifteen queries, the query evaluation time was almost the same for the rewritten and the automatically generated queries (with a time decrease between 0.2% and 1.6 %). For seven queries (Q_3, Q_5, Q_6, Q_{10}, Q_{11}, Q_{13} and Q_{14}), the rewritten queries have presented a slight improvement with an evaluation time decrease between 5.2% and 20.4% compared to the automatically generated ones. Finally, three queries (Q_1, Q_2 and Q_9) have presented a significant performance improvement with a time decrease between 57.7% and 79.1%. In more detail:

– For the queries Q_4, Q_8, Q_{13} and Q_{15}, the only difference between the rewritten the automatically generated queries, is that the rewritten have one Let XQuery clause less. In particular, in the rewritten queries the Let clause that is used to assign the XML data on which the query is evaluated (i.e., let $\$doc := collection(\,)$), has been

removed. The XML data declaration (i.e., *collection*( )) is directly used instead of the $*doc* XQuery variable. Hence, it is expected that the evaluation of these queries has not shown any significant efficiency improvement.

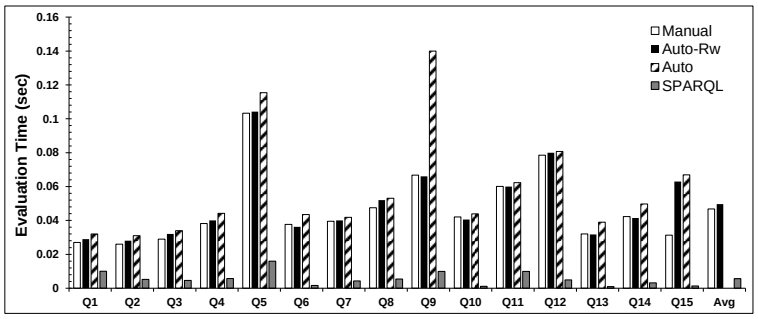
- For the queries Q_1, Q_2, Q_3, Q_6, Q_7 and Q_{10} the rewriting rule Rule 3 (*Unnesting For Clauses*) has been applied, which removes one For clause in each query, thus resulting in a less nested For clause. For the queries Q_3, Q_6 and Q_7, the improvement of the rewritten queries is not significant (1.2% to 7.6 %), since the outer loops (i.e., outer For clauses) are restricted with conditions (i.e., predicates over the XPath of the For clause) resulting into very few inner loops. For the queries Q_1 and Q_2, though, which have no restrictions in the For clauses, the improvement is significant (57%). Finally, we expected that the same should hold for the query Q_{10}; however, its improvement was not as significant as expected (20% improvement). This may happen, because this query returns only the first 100 of the results, thus, the query engine possibly selects an efficient execution plan (although the query optimizer has been turned-off).
- For the queries Q_{13} and Q_{14} the rewriting rule Rule 2 (*Reducing Let Clauses*) has been applied, which removes one Let clause from each query. As is expected, this rewriting resulted in a slight improvement of 5%.
- Finally, for the queries Q_5 and Q_9, the rewriting rule Rule 1 (*Changing For Clauses to Let*) has been initially applied. Rule 1 considers the exact cardinality of one in the SSN attribute and the Age element. As a result, two For clauses in each query are transformed to Let clauses. Then, the Rule 2 has been applied on the queries. From query Q_5, the two Let clauses that resulted from the application of Rule 1 on the For clause have been removed. In addition, from query Q_9 four Let clauses have been removed (two of them have resulted from the application of Rule 1 on the For clause). Compared to the initial queries, the query Q_5 has two For clauses less and the query Q_9 has two For and two Let clauses less. These rewritings have resulted in a considerable improvement of 14.7% and 79.1% for the queries Q_5 and Q_9 respectively.

Automatically Rewritten vs. Manually Translated (Auto-Rw vs. Manual). The evaluation times of the automatically rewritten queries are very close to the ones of the manually translated queries as shown in Table 5.15, with an average increase of 1.9%. For three (Q_1, Q_2 and Q_{10}) out of the fifteen queries, the rewritten queries have considerably outperformed the manually translated ones, with an evaluation time decrease between 13.4% and 28.1%. In addition, in other cases (Q_6, Q_9, Q_{13} and Q_{14}), the rewritten queries have shown a slight improvement (with an evaluation time decrease between 0.9% and 13.4%) compared to the manually translated ones. For the remaining queries, (with the exception of query Q_15), the evaluation time of the rewritten and the manually translated queries was almost the same. For query Q_{15} the manual translation has shown a significant evaluation time increase (71.7%). In more detail:

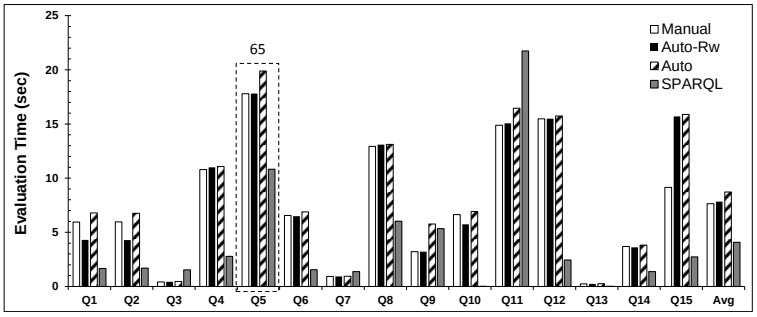
- For the queries Q_1, Q_2 and Q_{10}, the rewritten queries have one For clause less compared to the manually translated ones. The use of the rewriting rule Rule 3 has resulted to unnested For clauses in the rewritten queries. The resulting For clauses have presented an evaluation time improvement of 13.4% to 28.1% in the rewritten queries compared to the manually translated ones.
- For the remaining queries (except Q_{15}), the performance of the rewritten queries is almost the same with the manually translated ones. The only reason for delays in few rewritten queries is the use of several "special" markup tags (e.g., <Result>, <Results>) which are exploited to structure the query results. These markup tags

have resulted in a larger size of the results, hence a slight delay in evaluation time has been observed.

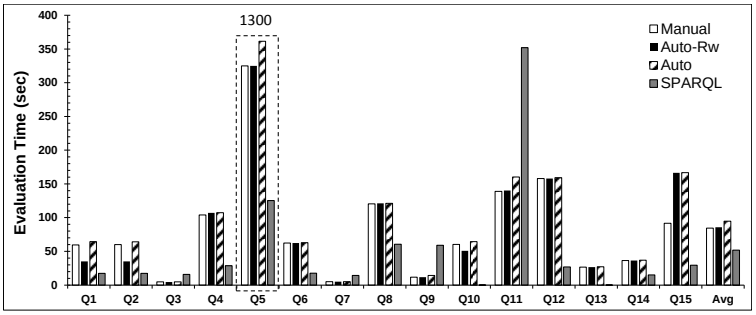
– Finally, for Q_{15}, the manually translated query takes into account the cardinality of the elements *FirstName* and *LastName*, which have been defined in the XML Schema to be more than one. In that case, there is no need to check if the *FirstName* and *LastName* XQuery variables were bound to some values during the construction of the RDF graph. This is done in the automatically generated queries (rewritten and not-rewritten) by using the *fn:exists*( ) XQuery built-in function. This query is the only case with a considerable difference in the evaluation time of the manually translated query compared to the rewritten one. However, it is obvious that simple rewriting rules similar to Rule 1 can be defined in order to exploit the XML Schema cardinality in several cases. For example, during the translation of Construct SPARQL queries, the cardinality value of more than one for elements or attributes can be considered by a rewriting rule, in order to avoid the unnecessary check if some values exist for these elements or attributes.



(a) DT_1 Persons dataset (XML Store Y)

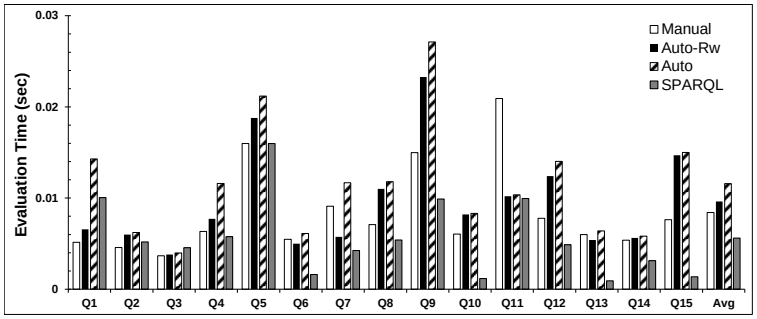


(b) DT_8 Persons dataset (XML Store Y)

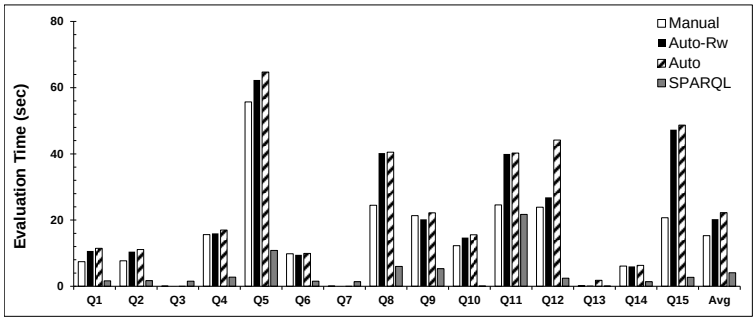


(c) DT_{10} Persons dataset (XML Store Y)

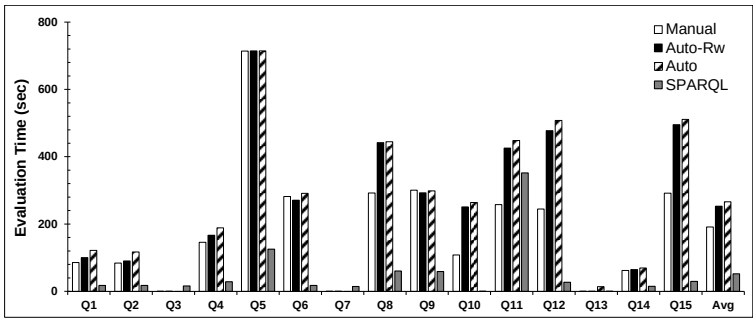
Figure 5.9: Query Evaluation Time over the Persons datasets DT_1, DT_8 and DT_{10} (XML Store Y)



(a) DT_1 Persons dataset (XML Store Z)



(b) DT_8 Persons dataset (XML Store Z)



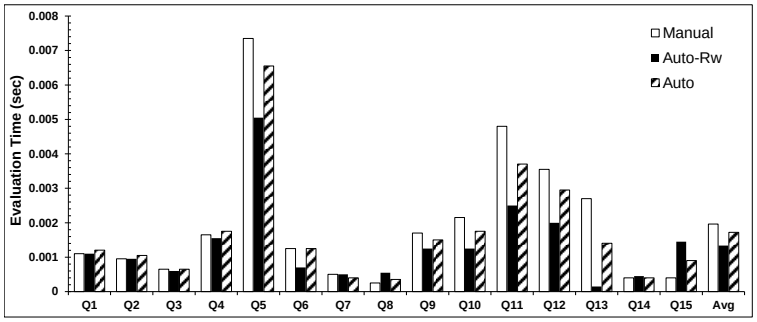
(c) DT_{10} Persons dataset (XML Store Z)

Figure 5.10: Query Evaluation Time over the Persons datasets DT_1, DT_8 and DT_{10} (XML Store Z)

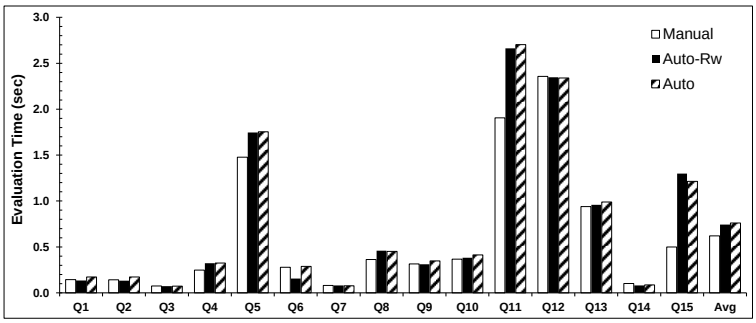
5.12.3.2.2 Varying the Size of the Dataset

In order to study the query evaluation efficiency over the dataset size, we have used the 10 synthetic Persons XML datasets. We first present an overview of the effect of the dataset size on the evaluation time. In the following figures, we present the results obtained using different XQuery engines. In particularly, Figure 5.9 corresponds to XML Store Y, Figure 5.10 corresponds to XML Store Z, and Figure 5.11 corresponds to Memory-based XQuery Engine. The figures show the query evaluation times for all the queries on three datasets (DT_1, DT_8 and DT_{10}). Each of the diagrams corresponds to one dataset. We can observe that in all cases the automatically rewritten queries outperform the automatically generated ones. In addition, the improvement of the automatically rewritten XQueries against the automatically generated XQueries does not show significant variations (is almost constant) over the dataset size.

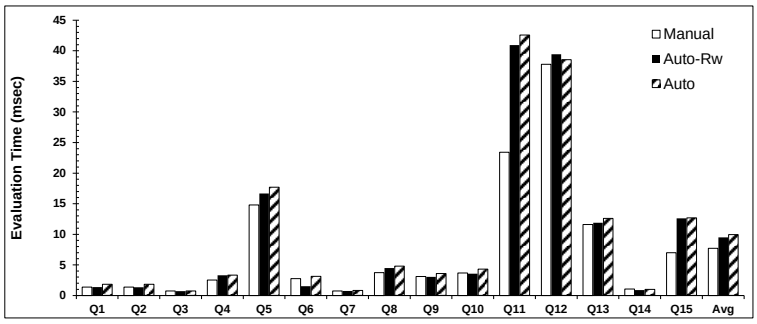
Figure 5.12 provides a thorough look at the query evaluation time over the dataset size. Particularly, Figure 5.12 presents the evaluation times (in logarithmic scale) for (a) the manually translated; (b) the automatically generated; and (c) the automatically rewritten



(a) DT_1 Persons dataset (Memory-based XQuery Engine)



(b) DT_8 Persons dataset (Memory-based XQuery Engine)



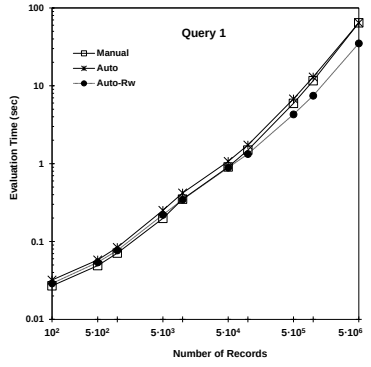
(c) DT_{10} Persons dataset (Memory-based XQuery Engine)

Figure 5.11: Query Evaluation Time over the Persons datasets DT_1, DT_8 and DT_{10} (Memory-based XQuery Engine)

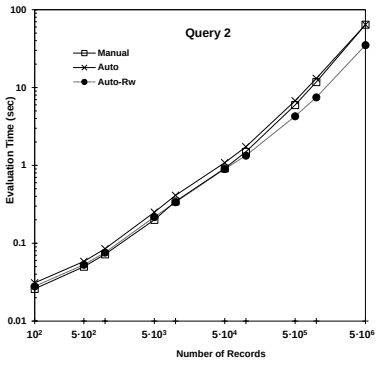
XQuery queries over the 10 datasets. Each of the first 15 diagrams corresponds to one query (e.g., Figure 5.12a corresponds to Q_1, Figure 5.12b corresponds to Q_2, etc.) and the last diagram (Figure 5.12q) corresponds to the average evaluation times for all the queries (Queries $1 \sim 15$).

We observe that the evaluation times for both the manually and automatically rewritten queries have almost similar performance over the dataset size. As the dataset size increases, the evaluation times increase in a sublinear manner for the specific query set. For some of the queries, the increase is less sharp than for others (e.g., Queries 3, 7, 9); this is due to the high selectivity (i.e., small result set) of these queries. However, for all the queries the increase is sharper for datasets larger than 10^5 records. Finally, with the exception of the queries 7, 10, 11 and 12 where the evaluation times are almost equal from the smallest dataset to the largest, as the dataset size increases, the difference between the evaluation times decreases, with most of the queries having almost equal evaluation times for the larger datasets (DT_7 to DT_{10}).

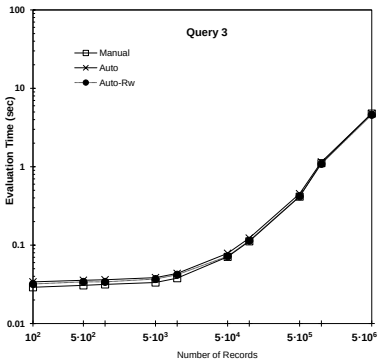
The average evaluation times (Figure 5.12q) increase very fast with a sharper increase for datasets larger than 105 records. In addition, as the dataset size increases, the differ-



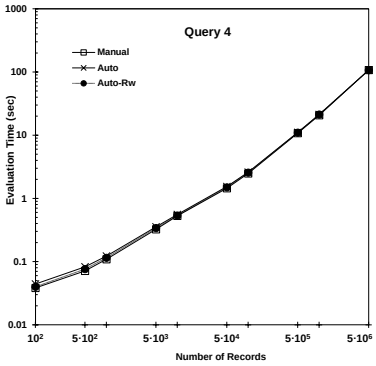
(a) Query 1



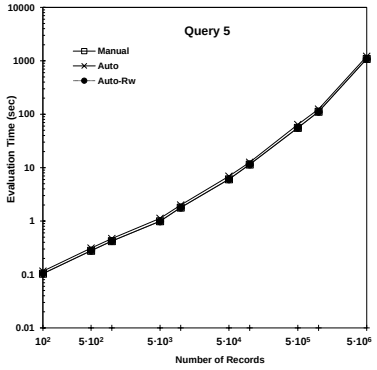
(b) Query 2



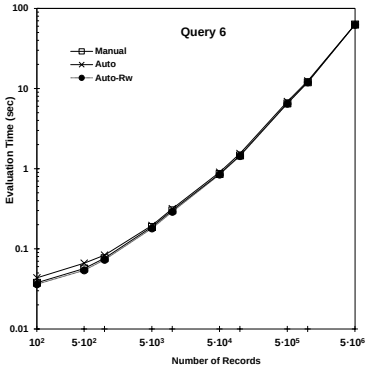
(c) Query 3



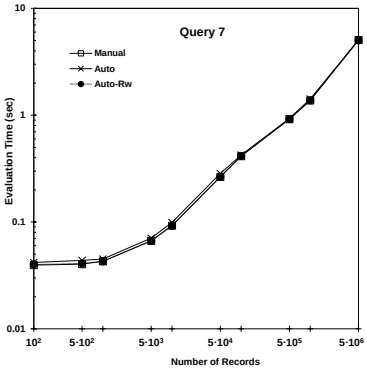
(d) Query 4



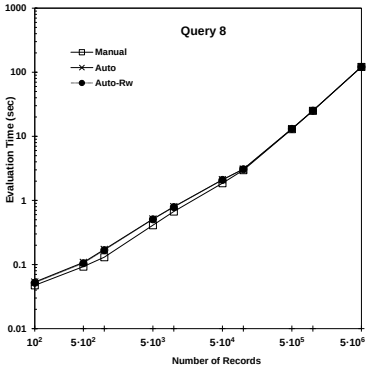
(e) Query 5



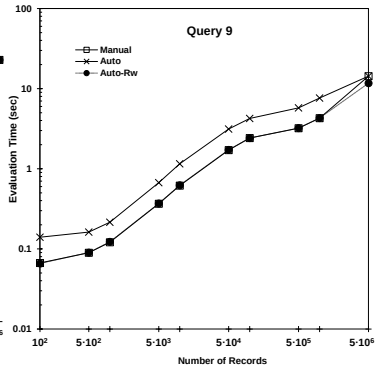
(f) Query 6



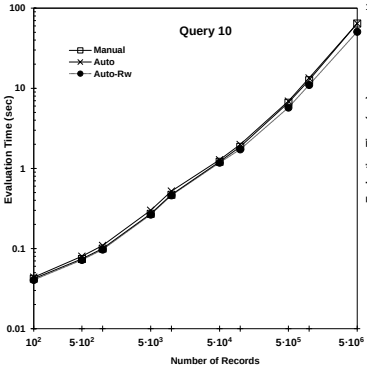
(g) Query 7



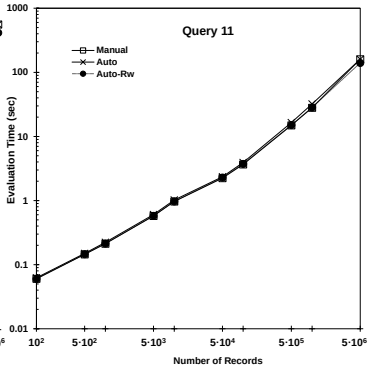
(h) Query 8



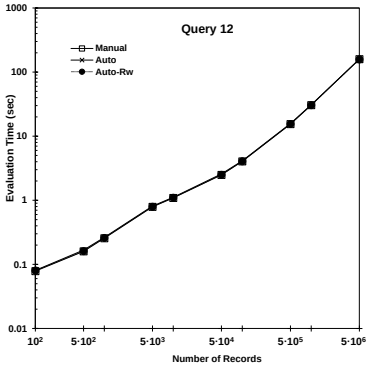
(i) Query 9



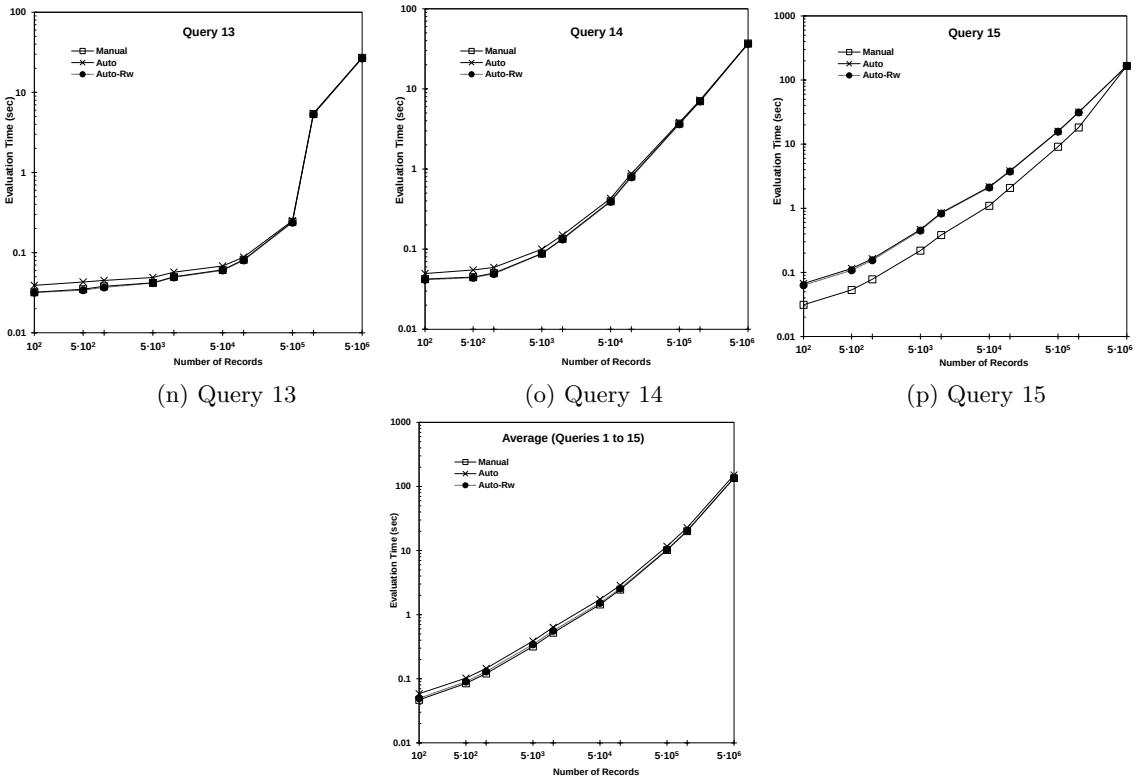
(k) Query 10



(l) Query 11



(m) Query 12



(n) Query 13 (o) Query 14 (p) Query 15

(q) Average (Queries 1~15)

Figure 5.12: Query Evaluation Time vs. Dataset size (Persons Datasets DT_1 ~ DT_{10}) (XML Store Y)

ence between the evaluation times decreases.

In more detail, for the smallest dataset (10^2 records), the average evaluation time for the automatically generated and rewritten queries has a 6.1% overhead compared to that of the manually translated ones. In addition, the rewritten queries have shown an evaluation time decrease of 18% compared to the automatically generated ones.

Regarding the DT_7 dataset (10^5 records), the automatically rewritten queries have a 4.1% overhead compared to the manually translated ones. In addition, the rewritten queries have shown an evaluation time decrease of 11.8% compared to the automatically generated ones.

Finally, for the largest dataset ($5{\cdot}10^6$ records), the automatically rewritten queries have a 1.0% overhead compared to the manually translated ones. In addition, the rewritten queries have shown an evaluation time decrease of 10.8% compared to the automatically generated ones.

The results show that even without extensive optimization, a noticeable performance improvement can be achieved. The query evaluation time decreases in average by 13% compared to the not-rewritten ones, with a maximum decrease 83% in some cases. Even the automatically generated queries have reasonable performance and scale rather well for sizes up to $725 \cdot 10^5$ XML nodes.

5.12.3.2.3 Query Evaluation Time vs. Query Translation Time

We present here a comparison of the query evaluation time with the query translation time. We have compared the query translation time and the query evaluation time to the total time which is the sum of the two (the charts are available in [84]).

We observe that the query translation takes negligible time in comparison to the query evaluation time even for the smallest dataset (i.e., the lowest evaluation times). In particular, for the dataset DT_1, the lowest ratio of translation time to total time (equal to

2.8%) occurs in query Q_5, while the highest ratio of translation time to total time (equal to 10.5%) occurs in query Q_8. Finally, the average ratio of translation time to total time is equal to 5.9%. Regarding the dataset DT_8, the lowest ratio of the translation time to the total time (equal to 0.01%) occurs in query Q_5, while the highest ratio of the translation time to the total time (equal to 1.3%) occurs in query Q_{13}. Finally, the average ratio of the translation time to the total time is equal to 0.04%.

5.12.3.3 Real Dataset

In this experiment we have studied the efficiency of the automatically generated XQuery queries using a real dataset. We have utilized the real DBLP dataset, as well as the DBLP query set, including 5 queries (Section 5.12.2.2.1). In an analogous manner with the previous experiment, we have measured and compared the query evaluation times for the automatically generated, rewritten and manually translated XQuery queries. Table 5.16 summarizes the experimental results.

Table 5.16: Query Evaluation Time for the DBLP dataset (XML Store Y)

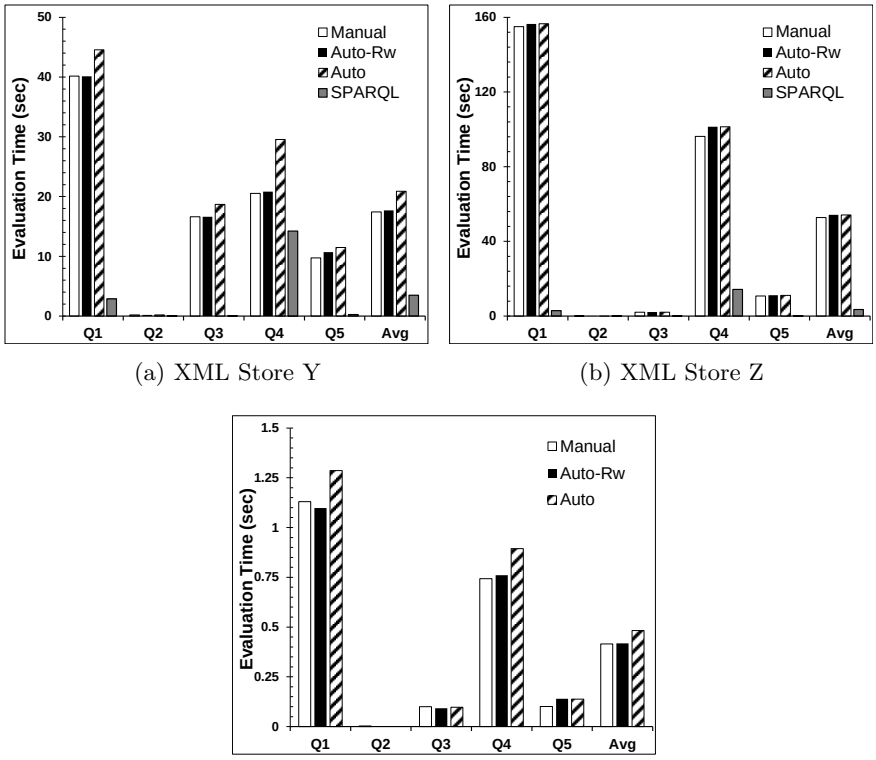
Query Evaluation Time (sec)						
Query	**SPARQL** (Q_S)	**Manual** (Q_{Xm})	**Auto-Rw** ($Q_{Xa\text{-}Rw}$)	**Auto** (Q_{Xa})	**Auto-Rw vs. Auto**	**Auto-Rw vs. Manual**
Q_1	2.88	40.14	40.12	44.56	10.0 %	0.1 %
Q_2	0.07	0.19	0.19	0.21	11.2 %	0.5 %
Q_3	0.06	16.61	16.63	18.72	11.2 %	-0.1 %
Q_4	14.24	20.52	20.82	29.57	29.6 %	-1.5 %
Q_5	0.26	9.73	10.69	11.51	7.1 %	-9.9 %
Average	**3.50**	**17.44**	**17.69**	**20.92**	**13.8 %**	**-2.2 %**

Automatically Rewritten vs. Automatically Generated (Auto-Rw vs. Auto) Queries. Table 5.16 shows that the evaluation times for the rewritten queries have presented a significant performance improvement compared to the automatically generated ones, with an average evaluation time decrease of 13.8%. In more detail:

- For the queries Q_1, Q_3 and Q_5, the rewriting rule Rule 1 (*Changing For Clauses to Let*) has been firstly applied. Rule 1 exploits the exact cardinality for the *Title* and *Year* elements. As a result, two For clauses for Q_1 and one For clause for Q_3 and Q_5 have been transformed to Let clauses. Afterwards, Rule 2 (*Reducing Let Clauses*) has been applied and has removed the Let clauses generated from Rule 1. Compared to the initial queries, the query Q_1 has two For clauses less and the queries Q_3 and Q_5 have one For clause less. The above rewritings have resulted in an improvement of 10.0%, 11.2% and 7.1% for the queries Q_1, Q_3 and Q_5, respectively.
- For query Q_2, the rewriting rule Rule 2 has been applied and has removed one Let clause, resulting in an improvement of 11.2%.
- Finally, for query Q_4 the rewriting rule Rule 3 (*Unnesting For Clauses*) has been applied and has removed two For clauses, resulting in an improvement of 29.6%.

Automatically Rewritten vs. Manually Translated (Auto-Rw vs. Manual) Queries. We can see from Table 5.16 that the evaluation times of the automatically rewritten queries are almost similar to the manually translated queries, with an average evaluation time increase of 2.2%. In more detail:

- For all the queries, with the exception of Q_5, the evaluation time for the rewritten queries is almost to the same with the manually translated ones. The only delay



(a) XML Store Y

(b) XML Store Z

(c) Memory-based XQuery Engine

Figure 5.13: Query Evaluation Time over the DBLP dataset (Using different XQuery Engines)

reason in the rewritten queries is the use of several "special" markup tags (e.g., <Result>, <Results>) which are exploited to structure the query results. These markup tags have resulted in a larger size of the results, hence a slight delay in evaluation time has been observed.

– For Q_5, the manually translated query has taken into account the cardinality of the elements Author and Title, which have been defined in XML Schema to be more than one. Thus, there was no need to check the existence of these values, as was done in the automatically generated query using the $fn{:}exists(\,)$ XQuery function.

Finally, we can observe from Table 5.16 that the query evaluation performance for the DBLP dataset is similar with that of the synthetic dataset of the same size.

In the following figure, we present the results obtained using different XQuery engines. In particularly, Figure 5.13a corresponds to XML Store Y, Figure 5.13b corresponds to XML Store Z, and Figure 5.13c corresponds to Memory-based XQuery Engine. The figures show the query evaluation times for all the queries over the DBLP dataset.

5.12.4 Result Overview

Schema Transformation and Mapping Generation. Although both the schema transformation and mapping generation processes are off-line processes, we wanted to have an indication of their performance. To this end, we have used several international XML Schema standards and have measured the time required for schema transformation and for mapping discovery and generation. We observed that both processes took negligible time even for very large XML Schemas.

Translation Efficiency. In order to demonstrate the efficiency of the SPARQL to XQuery translation process we measured the translation time required by the

SPARQL2XQuery Framework. In the first experiment, we generated several SPARQL queries by modifying their graph pattern size and type. In the second experiment, for the queries generated in the first experiment, we modified the number of the predefined mappings. Finally, in the third experiment, we have used three SPARQL query sets attempting to cover almost all the SPARQL grammar variations. The query sets used are the Berlin SPARQL Benchmark query set, a query set over the DBLP schema and a set over the Persons schema.

Query Evaluation Efficiency. Regarding the efficiency of the generated XQuery expressions, we have defined a small set of simple rewriting rules aiming to provide more efficient XQuery expressions. We have applied these rules on the automatically generated XQuery expressions. Then, we have compared the evaluation time of the automatically generated, rewritten and manually translated XQuery expressions. In the first set of experiments, a synthetic dataset and a set of 15 queries have been used. We have modified the dataset size and we have measured the query evaluation time for the automatically generated, rewritten and manually translated XQuery queries. In the second set of experiments, the real DBLP dataset has been utilized for demonstrating the query evaluation efficiency.

The results are similar for both the real and synthetic datasets. In particular, for the largest synthetic dataset ($5{\cdot}10^6$ records) the rewritten queries have presented an evaluation time decrease of 10.8% compared to the not-rewritten ones. In general, the rewriting rules have resulted in significant performance improvement, with an average evaluation time decrease of 13%, reaching 83% in some cases. Moreover, the average evaluation time for the automatically generated and rewritten queries has 1.0% overhead compared to the manually specified ones. Finally, the query evaluation times have been compared to the query translation times. The conclusion was that the query translation takes negligible time in comparison to the evaluation time, even for very small datasets.

5.13 Summary

The Web of Data (WoD) is an open environment comprised of hundreds of large interlinked, user contributed datasets. The WoD is founded on technologies and standards developed by the Semantic Web (SW) community (e.g., OWL, RDF/S, SPARQL) for Web information representation and management. On the other hand, in the current Web infrastructure the XML/XML Schema are the dominant standards for information exchange, and for the representation of semi-structured information. In addition, many international standards (e.g., Dublin Core, MPEG-7) have been expressed in XML Schema. The aforementioned have led to an increasing emphasis on XML data.

In the WoD users should not interact with different data models and languages for developing their applications or express-ing their queries. In addition, it is unrealistic to expect that all the legacy data (e.g., Relational, XML) will be converted to RDF data. Thus, it is crucial to provide interoperability mechanisms that allow the WoD users to transparently access exter-nal heterogeneous data sources from their own working environment. Finally, in the Linked Data era, offering SPARQL endpoints (i.e., SPARQL-based search services) over legacy data has become a major research challenge. However, despite the significant body of related work on relational data, to the best of our knowledge there is no work addressing neither the SPARQL to XQuery translation problem nor offering SPARQL endpoints over XML data. In the most recent research approaches, a combination of SW (SPARQL) and XML (XQuery, XPath and XSLT) technologies is exploited in order to transform XML data to RDF and vice versa.

In this work we have proposed the SPARQL2XQuery Framework, which bridges the heterogeneity gap and creates an interoperable environment between the SW and XML

worlds. The SPARQL2XQuery Framework comprises the key component for several WoD applications, allowing the establishment of SPARQL endpoints over XML data, as well as a fundamental component of ontology-based integration frameworks involving XML sources.

The SPARQL2XQuery Framework allows arbitrary SPARQL queries posed over ontologies to be automatically translated to XQuery expressions which are evaluated over XML data with respect to a set of predefined mappings. To this end, our Framework allows both manual and automatic mapping specification between ontologies and XML Schemas. Finally, the query results are returned either in RDF or in SPARQL Query Result XML Format. Thus, the WoD users are no longer required to interact with more than one models or query languages.

In more detail, we have introduced a mapping model for the expression of OWL–RDF/S to XML Schema mappings, as well as a method for SPARQL to XQuery translation both provided by the SPARQL2XQuery Framework. To the best of our knowledge, this is the first work addressing these issues. Moreover, we have presented the XS2OWL component, which allows transforming XML Schemas to OWL ontologies, exploiting the latest versions of the standards (XML Schema 1.1. and OWL 2). As far as we know, this is the first work that fully captures the XML Schema semantics and supports the XML Schema 1.1 constructs. The XS2OWL component has been integrated in the SPARQL2XQuery framework in order to provide automatic mapping generation and maintenance.

A thorough experimental evaluation of the SPARQL2XQuery framework has been conducted and presented, in order to demonstrate the efficiency of (a) schema transformation; (b) mapping generation; (c) query translation; and (d) query evaluation.

We have also discussed, in this chapter, the major technical and theoretical challenges we have faced throughout the development of the SPARQL2XQuery Framework. The major difficulties have arisen from the different data models and semantics adopted by the SW and XML worlds. In summary, we had to overcome several heterogeneity issues like Directed graphs vs. Tree structures, Three-valued logic vs. Two-valued logic, Graph patterns vs. Iterative procedures, etc. We have also discussed issues involved in the translation process that are related to the SPARQL semantics like Well Designed vs. Non-Well Designed Graph Patterns, Safe vs. non-Safe Filter Expressions, etc.

Chapter 6

Semantic Retrieval and Exploration

In this chapter we examine two problems. In the first one we study the problem of semantic information retrieval. For this problem, we proposed a framework that supports ontology-based document annotation and retrieval, in a fully collaborative environment. The framework provides an automatic annotation mechanisms that is based on a learning method that exploits user annotation history and textual information to automatically recommend annotations for new documents. Additionally, the framework provides search facilities beyond the traditional keyword-based search. A flexible combination of keyword-based and semantic-based search over documents is proposed in conjunction with advanced semantic-based search operations. The proposed methods are implemented in a fully functional tool and their effectiveness is experimentally validated.

Next, we study the problem of modeling, publishing and exploring evolving data, adopting the Linked Data principles. For this problem, we propose a change model based on RDF to capture versioned entities. Based on this model we convert legacy data from biological databases to diachronic Linked Data. Our Linked Data infrastructure can assist biologists to explore biological entities and their evolution, and provides a SPARQL endpoint for applications and services to query historical miRNA data and track changes, their causes and effects.

6.1 Semantic Information Retrieval

Document annotation and search have received tremendous attention by the Semantic Web [197] and the Digital Libraries [24] communities. Semantic annotation involves tagging documents with concepts (e.g., ontology classes) so that content becomes meaningful. Annotations help users to easily organize their documents. Also, they can help in providing better search facilities: users can search for information not only using keywords, but also using well-defined general concepts that describe the domain of their information need.

Although traditional Information Retrieval (IR) techniques are well-established, they are not effective when problems of concept ambiguity or synonymity appear. On the other hand, neither search based only on semantic information may be effective, since: (a) it does not take into account the actual document content, (b) semantic information may not be available for all documents and (c) semantic annotations may cover only a few parts of the document.

Hybrid solutions that combine keyword-based with semantic-based search deal with the above problems. Developing methodologies and tools that integrate document annotation and search is of high importance. For example, researchers need to be able to organize, categorize and search scientific material (e.g., papers) in an efficient and effective way.

Similarly, a press clipping department needs to track news documents, annotating specific important topics and searching for information.

This chapter describes GoNTogle, a framework for document annotation and retrieval, built on top of Semantic Web and IR technologies. GoNTogle provides both manual and automatic ontology-based annotations, supporting documents of several formats (e.g., doc, pdf, txt, rtf, odt, sxw). Annotation is based on standard Semantic Web technologies like, OWL and RDF/S. All annotations are stored in a centralized server, providing a collaborative environment. A learning method, exploiting textual information and user annotation history, is proposed to support the automatic annotation mechanism.

GoNTogle also provides three search types: (a) *Keyword-based*, (b) *Semantic-based* and c) *Hybrid*. Experimental evaluation validates the effectiveness of the proposed hybrid method, compared to the other two. Finally, several advanced ontology-based searching operations are provided, including the capability to expand or shrink the result list using ontology information, in order to retrieve higher quality results.

Regarding the design principles of our framework, they are based on the requirements set in previous works [23, 200, 380]. In contrast with the existing approaches, our aim was to design an easy-to-use document annotation and search framework that supports (a) viewing and annotating popular document types while maintaining their initial format, (b) offering a collaborative environment by sharing those annotations (c) supporting Semantic Web standards, (d) integrating textual information with semantics and (e) supporting a flexible combination of keyword-based and semantic-based search in conjunction with advanced ontology-base search operations.

Contributions. The main contributions of this work are summarized as follows.

1. We have designed and implemented an easy-to-use document annotation framework that supports the most widely used document formats, providing also advanced search facilities.
2. The framework is based on a server-based architecture, where document annotations are stored in a central repository, separately from the original document. This offers a collaborative environment where users can annotate and search documents.
3. We propose a learning method for automatic annotation of documents based on models trained from user annotation history and textual information, so that annotation suggestions are tailored to user behavior.
4. We introduce a hybrid search method that provides a flexible combination of traditional keyword-based and semantic-based search for effective document retrieval.
5. We present a user-based evaluation to demonstrate the effectiveness of the automatic annotation method. Moreover, we demonstrate a comparative evaluation to validate that the proposed hybrid search outperforms keyword-based and semantic-based search in terms of precision and recall.

6.1.1 Semantic Annotation

GoNTogle framework supports semantic, ontology-based annotations, for widely used document formats (e.g., doc, pdf, txt, rtf, odt, sxw). It allows annotating the whole document or parts of it. GoNTogle framework supports both manual and automatic annotations. For automatic annotation we propose a learning method that exploits user annotation history and textual information to automatically suggest annotations for new incoming documents. GoNTogle provides a common ontology-based annotation model (Figure 6.1) for all supported document formats. Annotations are stored on a centralized ontology

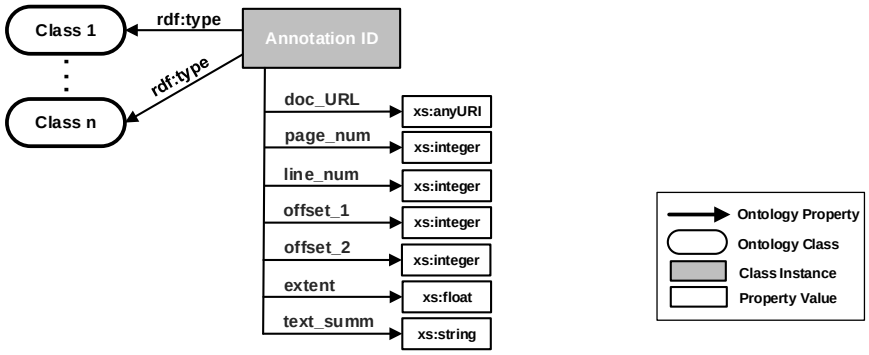


Figure 6.1: Ontology-based annotation model

server, separately from the original document. Annotations from different document formats are defined and stored in exactly the same way. Each annotation is stored as an ontology class instance, along with information about the annotated document. We define a set of ontology properties that are used to store the minimum essential information needed to provide a bidirectional connection between documents and ontologies. These properties contain information like: document URL, annotation offsets, page number, extent of annotation over the document, etc.

Figure 6.1 shows the ontology-based annotation model we developed in the context of the GoNTogle framework. Annotations are represented as class instances that can belong to one or more ontology classes. Using ontology properties, all the essential annotation information is attached to these instances. Property *doc_URL*, corresponds to the document's URL (including document's file name) of represented annotation. *page_num* and *line_num* properties, correspond to the number of the page and line respectively where the annotation begins. The property *offset_1* corresponds to a number that indicates the offset from the beginning of the document until the beginning of the annotation. As the same, property *offset_2* corresponds to the offset from the end of annotation until the end of the document. The property extent represents the extent of the annotation over the document. Finally, *text_summ* used for storing the summary of the annotated text (i.e., 1-3 tokens from the begin and the end) required for the GUI functionality.

6.1.1.1 Automatic Semantic Annotation

In this section, we present the learning method used for automatic document annotation. We propose a method based on *weighted kNN* classification [291] that exploits user annotation history and textual information to automatically suggest annotations for new documents. Next, we describe our approach in detail. The training data of our method include document annotations provided manually by the users. When a document is manually annotated, the annotation text is extracted and indexed using an inverted index. Along with the textual information, the index also stores information about the annotation classes for each annotated document (or part of document).

To automatically annotate documents, the user first selects a document or a part of it. Then, given the set of training data, our method suggests a ranked list of ontology concepts (classes) to annotate the document (or its part). Algorithm 8 presents the pseudocode of our method. It takes as input the selected text st and the inverted index $\mathbf{I}$. Based on textual similarity $ts_{st,at}$ between st and each indexed annotated text at, the k most similar annotated texts are considered for further processing, and included in set S (*lines 1~3*). Then for each at in S, we retrieve the ontology classes used to annotate at. Each class cl is given a score Scr_{cl} that combines (a) the textual similarity (based on Lucene similarity

Algorithm 8. Annotation Suggestion Algorithm (st, **I**)

Input: st: selected text; **I**: index
Output: cl_i: suggested class; Scr_{cli_i}: suggested class score

```
1 foreach annotated text at in I do
2     calculate ts_{st,at}
3 insert the k most similar annotated texts in S
4 foreach at in S do
5     foreach class cl annotate at do
6         Scr_cl = Scr_cl + (w1 · ts_{st,at}) · (w2 · e_{cl,at})
7 return cl_i, Scr_{cl_i}
```

model[1]) score $ts_{st,at}$ between st and at and (b) a score $e_{cl,at}$ representing the extent to which each at in S is annotated with class cl (*line 6*). As $e_{cl,at}$ we define, the number of tokens of the cl annotations in at divided by the number of tokens in at.

$$e_{cl,at} = \frac{\text{number of tokens of } cl \text{ annotations over } at}{\text{number of tokens in } at}$$

The w_1 and w_2 weights are used to quantify the preference of textual similarity against semantic similarity (or vice versa). Finally, a ranked list of suggested annotation classes cl_i and their score Scr_{cl_i} is presented to the user (*line 7*). The user may choose one or more suggested classes to conclude the automatic annotation process.

Table 6.1: Basic Notation

Symbol	Description
q_{key}	Keyword query, consisting of search term $\{t_1, t_2, \ldots t_m\}$
$S_{key}(q_{key})$	Keyword-based search
RS_{key}	Keyword-based search result set
$Scr_{key}(q_{key}, d)$	Keyword-based similarity score
q_{sem}	Semantic query, consisting of search classes $\{cl_1, cl_2, \ldots cl_n\}$
$S_{sem}(q_{sem})$	Semantic-based search
RS_{sem}	Semantic-based search result set
$Scr_{sem}(q_{sem}, d)$	Semantic-based similarity score
$S_{hybr}(q_{sem}, q_{key})$	Hybrid search
RS_{hybr}	Hybrid search result set
$Scr_{hybr}(q_{sem}, q_{key}, d)$	Hybrid similarity score

6.1.2 Search

In this section, we present the search facilities proposed in the context of GoNTogle framework. We formally define the supported search types (Section 6.1.2.1) and we analyze the ontology-based advanced search operations (Section 6.1.2.2). Moreover, we introduce the hybrid search method, which combines keyword-based and semantic-based search. Table 6.1 outlines the basic notation used in the following paragraphs.

6.1.2.1 Search Types

We categorize the basic search facilities of our framework into three types: (a) *Keyword-based search*, (b) *Semantic-based search* and (c) *Hybrid search*.

[1]http://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/search/Similarity.html

6.1.2.1.1 Keyword-based search

This is the traditional search model. The user provides keywords and the system retrieves relevant documents based on textual similarity. We adopted the text similarity metric used in Lucene IR engine.

Keyword-based search is denoted as $S_{key}(q_{key})$, where $q_{key} = \{t_1, t_2, \dots t_m\}$ and t_i are the search terms with $m \geq 1$. Keyword-based search returns an ordered *result set* RS_{key} of tuples $\langle d, Scr_{key}(q_{key}, d)\rangle$, containing all the documents d matched with terms q_{key}. $Scr_{key}(q_{key}, d)$ is the *similarity score* of document d for the searching terms q_{key}. This score is based on document textual similarity with the searching terms.

6.1.2.1.2 Semantic-based search

This search facility allows the user to navigate through the classes of an ontology and focus their search on one or more of them.

Semantic-based search is denoted as $S_{sem}(q_{sem})$, where $q_{sem} = \{cl_1, cl_2, \dots cl_n\}$ and cl_i are the searching classes with $n \geq 1$. It return an ordered *result set* RS_{sem} of tuples $\langle d, Scr_{sem}(q_{sem}, d)\rangle$, containing all the documents d that have been annotated with one or more of the search classes q_{sem}. $Scr_{sem}(q_{sem}, d)$ is the *similarity score* of document d for the searching classes q_{sem}. This score is based on semantic similarity between the searching classes q_{sem} and document d. To define semantic similarity $ss_{cl_i,d}$ between a class cl_i and a document d, we consider the extent of the class annotations over the document: that is the number of tokens used to define the class annotations in d divided by the number of tokens in d.

The final similarity score is defined as follows:

$$Scr_{sem}(q_{sem}, d) = \sum_{i=1}^{n} \frac{ss_{cl_i,d}}{n},$$

$$ss_{cl_i,d} = \frac{\text{number of tokens of } cl_i \text{ annotations over } d}{\text{number of tokes in } d}$$

where n is the number of ontology classes used during the semantic-based search, and $ss_{cl_i,d}$ is a score representing the extent to which document d is annotated with class cl_i.

6.1.2.1.3 Hybrid search

The user may search for documents using keywords and ontology classes. She can, also, determine whether the results of her search will be the intersection or the union of the two searches.

Hybrid search is denoted as $S_{hybr}(q_{sem}, q_{key}) = S_{sem}(q_{sem})\ Op\ S_{key}(q_{key})$, where $q_{sem} = \{cl_1, cl_2, \dots cl_n\}$ and cl_i are the searching classes with $n \geq 1$, $q_{key} = \{t_1, t_2, \dots t_m\}$ and t_i are the searching terms with $m \geq 1$ and Op the Boolean operators OR or AND. Hybrid search returns an ordered *result set* RS_{hybr} of tuples $\langle d, Scr_{hybr}(q_{sem}, q_{key}, d)\rangle$, the contents and the order of the result set depend on Op value:

- $op = AND$ The result set contains all the documents d that have been annotated with one or more of the search classes q_{sem} and match with terms q_{key}.

$$RS_{hybr} = RS_{key} \bigcap_{\text{over } d} RS_{sem}$$

The final similarity score is defined as:

$$Scr_{hybr}(q_{sem}, q_{key}, d) = w_3 \cdot Scr_{sem}(q_{sem}, d) + w_4 \cdot Scr_{key}(q_{key}, d)$$

where $Scr_{sem}(q_{sem}, d)$ is the similarity score from semantic-based search, and $Scr_{key}(q_{key}, d)$ is the similarity score from keyword-based search. The w_3 and w_4 weights are used to quantify the relative importance of the semantic-based and keyword-based scores, when both keyword and semantic queries must be satisfied.

– $op = OR$ The result set contains all the documents d that have been annotated with one or more of the searching classes q_{sem} and all the documents d matched with terms q_{key}.

$$RS_{hybr} = RS_{key} \bigcup RS_{sem}$$

The final similarity score is defined as:

$$Scr_{hybr}(q_{sem}, q_{key}, d) = w_5 \cdot Scr_{sem}(q_{sem}, d) + w_6 \cdot Scr_{key}(q_{key}, d)$$

where $Scr_{sem}(q_{sem}, d)$ is the similarity score from semantic-based search, and $Scr_{key}(q_{key}, d)$ is the similarity score from keyword-based search. The w_5 and w_6 weights are used to quantify the relative importance of the semantic-based and keyword-based scores, when either keyword or semantic queries must be satisfied.

6.1.2.2 Advanced Search Operations

Here we present a set of advanced search operations that can be used after an initial search has been completed.

Find Related Documents. Starting from a result document d, the user may search for all documents that have been annotated with a class cl that also annotates d. For example, if a user had initially searched with class[2] H.2 [DATABASE MANAGEMENT] and selected one of the results that is also annotated with class H.2.5[Heterogeneous Databases], then "*Find Related Documents*" would return all documents annotated with both classes.

Find Similar Documents. This is a variation of the previous search facility. Starting from a result document d, the user may search for all documents that are already in the result list and have been annotated with a class cl that also annotates d. For example, if a user had initially searched with keyword "XML" AND class H.2 [DATABASE MANAGEMENT] and selected one of the results that is also annotated with class H.2.5 [Heterogeneous Databases], then "*Find Similar Documents*" would return all documents annotated with both classes and contained the keyword "XML".

Get Next Generation. The resulting list from a semantic-based (or hybrid) search can be confined by propagating the search on lower levels in the ontology (i.e., if class cl has been used, then search is propagated only in direct subclasses of cl). This is the case when the search topic is too general. For example, if a user had initially searched with H.2 [DATABASE MANAGEMENT], then "*Get Next Generation*" would return all documents annotated with at least one of its subclasses (H.2.5 [Heterogeneous Databases], H.2.3 [Languages], etc.).

Get Previous Generation. This offers the inverse functionality of the previous option. The resulting list from a semantic-based (or hybrid) search can be expanded by propagating the search on higher levels in the ontology (i.e., if class cl has been used, then search is propagated only in direct superclasses of cl). This is the case when a search topic is too

[2]We turned the ACM Computing Classification (www.acm.org/about/class) into an OWL ontology.

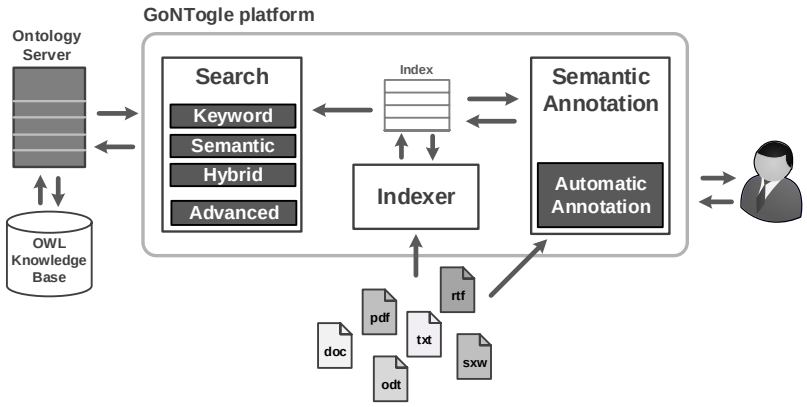


Figure 6.2: System architecture

narrow. For example, if a user had initially searched with H.2 [DATABASE MANAGEMENT], then "*Get Previous Generation*" would return all documents annotated with its superclass (H. [Information Systems]).

Proximity Search. This search option allows the user to search for documents that belong to all subclasses of a selected class, by applying a ranking model based on ontology hierarchy. That is, if class *cl* is the initial class, then search is propagated in all direct and indirect subclasses of *cl*. The resulting documents gathered from all levels of the ontology hierarchy are weighted properly (i.e., documents from the selected class *cl* get higher score than 1st level subclasses and even higher than 2nd level subclasses).

6.1.3 System Overview

6.1.3.1 System Architecture

Due to its centralized server-based annotation storage and management architecture, GoNTogle offers a collaborative user environment. Annotations are stored separately from the original document and may be shared by several user groups. GoNTogle's architecture is presented in Figure 6.2. The system is divided into 4 basic components:

- *Semantic Annotation Component* provides facilities regarding the semantic annotation of documents. It consists of 3 modules: (a) Document Viewer, (b) Ontology Viewer and (c) Annotation Editor.
- *Ontology Server Component* stores the semantic annotations of documents in the form of class instances. It consists of 2 modules: (a) an Ontology Manager and (b) an Ontology Knowledge Base.
- *Indexing Component* is responsible for indexing the documents using inverted indexes.
- *Search Component* allows users to search for documents using a flexible combination of textual (keyword-based search) and ontology (semantic-based search) information.

6.1.3.2 Semantic Annotation In-Use

Semantic Annotation Component offers 2 primary functionalities: (a) annotation of whole document and (b) annotation of parts of a document. Also, a user may choose between manual and automatic annotation.

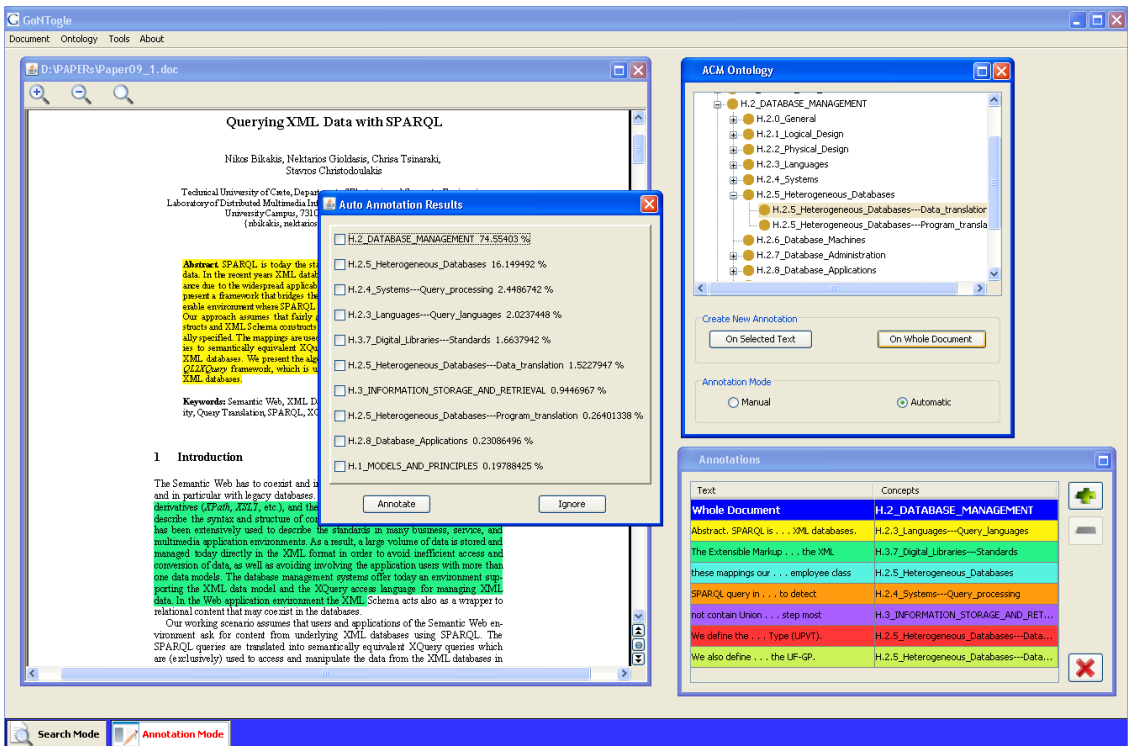


Figure 6.3: Semantic annotation example

Figure 6.3 shows the Semantic Annotation window of our application. The user may open a document in the Document Viewer, maintaining its original format. Moreover, she can load and view the hierarchy of an ontology through the Ontology Viewer. In the specific example, the loaded ontology corresponds to the ACM Computing Classification hierarchy. The user can, then, select one or more ontology classes and manually annotate the whole document or part of it. The annotation is stored as an ontology class instance in the Ontology Server, along with information about the annotated document. At the same time, an annotation instance is added in the Annotation Editor list. Each record of this list corresponds to an annotation stored in the Ontology Server. For example (Figure 6.3), the abstract of the document is annotated with class H.2.3 [Languages]: Query Languages, while the whole document is annotated with class H.2[DATABASE MANAGEMENT]. The user can manage those annotation instances, adding or removing ontology classes, or completely remove them. Also, when she selects an annotation from the list (regarding a part of a document), the document scrolls to the corresponding part, which is highlighted with the same color as the annotation instance.

6.1.3.3 Implementation

In what follows we provide technical information about the implementation of our system. All annotation and search facilities have been implemented in a Java prototype.

To develop our system, we used several open source tools and libraries. For indexing and keyword searching we used the Lucene search engine library. Lucene modules participate in several components of our system: (a) Document text indexing for search purposes (Indexing Component); (b) Document retrieval and scoring regarding textual similarity (Search Component); and (c) Indexing and querying documents for automatic annotation purposes (Semantic Annotation Component).

We used the ProtégéAPI[3] server and MySQL database for the Ontology Server Component, so that document annotations are stored as class instances. Through Protégé API, for each annotation, we store information that is required for processes such as retrieval of the specific annotation, ontology search scoring for a specific class-document pair, etc.

OpenOffice API[4] was essential in incorporating in our system a viewer that could maintain the exact format of .doc documents, which is a very common filetype. The same applies for Multivalent[5], a generalized document viewer that was integrated in our system so that PDF files could also maintain their format when being viewed and annotated.

6.1.4 Experimental Analysis

In this section, we present the experiments we performed in order to evaluate the effectiveness of our methods. In Section 6.1.4.1 we present the evaluation of the automatic annotation method. In Section 6.1.4.2, we compare our proposed hybrid search method with keyword-based and semantic-based search .

6.1.4.1 Automatic Annotation

In order to demonstrate the effectiveness of the proposed automatic annotation method, we perform a user-based evaluation. The effectiveness of our method is validated in terms of *Precision at position n* (P@n) and *Recall*.

6.1.4.1.1 Setting

We turned the ACM Computing Classification into an OWL ontology. The ontology produced is a 4-level structure with 1463 nodes. First, we performed an initial set of experiments in order to compare the simple kNN and the weighted kNN classification methods and also to identify the best value for the k factor. Best precision and recall values were observed for $k = 7$ using the weighted kNN algorithm.

Moreover, the weights used for the automatic annotation method (Section 6.1.1.1), w_1 and w_2 are calculated at 0.6 and 0.4 respectively after tuning. Intuitively, these values suggest that, in our problem setting, textual similarity is slightly more important than semantic similarity in case of automatic annotation.

6.1.4.1.2 Scenario

We asked from 15 users (PhD students and researchers in various areas of computer science) to participate in our experimental evaluation. Each user selected 2 areas of her research interests and for each area she collected 10 research papers that she was familiar with. In order to train our system, we asked from each user to annotate (parts or/and the whole of) 12 out of her 20 papers with at least one ACM class, using the GoNTogle framework.

After every user had performed the training task, we asked each of them to evaluate the automatic annotation suggestions provided by GoNTogle, for the remaining 8 papers of each user (test set). Note that, before reviewing the system suggestions, each user was asked which annotation classes she expected to be given by the system. The system presented a ranked list of annotation classes and each user was required to check the valid ones. Also, each user should point out valid classes that were not found between the system suggestions, as well as valid classes that, even they had not thought of, the system correctly suggested them.

[3]http://protege.stanford.edu
[4]http://api.openoffice.org
[5]http://multivalent.sourceforge.net

Table 6.2: The average Precision at position n (P@n) for each user

User	P@1	P@2	P@3	P@4	P@5
1	0.82	0.79	0.79	0.75	0.68
2	1.00	0.94	0.80	0.65	0.60
3	0.80	0.80	0.70	0.70	0.76
4	1.00	1.00	0.80	0.84	0.80
5	1.00	0.90	0.90	0.82	0.81
6	0.80	0.90	0.73	0.70	0.64
7	1.00	1.00	0.93	0.85	0.84
8	0.93	1.00	0.73	0.71	0.69
9	0.90	0.90	0.87	0.80	0.76
10	0.91	0.87	0.80	0.75	0.71
11	1.00	1.00	0.87	0.84	0.78
12	0.80	0.77	0.72	0.70	0.66
13	0.95	0.92	0.83	0.75	0.68
14	1.00	0.90	0.87	0.80	0.76
15	0.80	0.80	0.73	0.65	0.56
Avg	0.91	0.90	0.81	0.75	0.72

Table 6.3: The average Recall and the average UVCS for each user

User	Recall	UVCS
1	0.80	0.40
2	0.92	0.20
3	0.98	0.20
4	0.97	0.40
5	0.98	0.40
6	1.00	1.20
7	0.97	0.20
8	0.82	0.20
9	1.00	0.20
10	0.89	1.00
11	0.88	0.80
12	0.95	0.65
13	0.87	0.40
14	0.95	1.60
15	1.00	0
Avg	0.93	0.52

Based on the data collected, we calculated the *Precision at position n* (P@n) and *Recall* values for each user separately, as well as the mean average values for all users. Also, for correctly suggested annotation classes that the user had not initially thought of using them, we introduce the measure of *Unexpected Valid Class Suggestion* (UVCS), defined as follows:

$$\text{UVCS}= \#\textit{Correctly suggested and not initially though classes}$$

Finally, P@n and Recall are defined as follows:
$P@n = \dfrac{\#\text{relevant results in } \textit{top-n} \text{ suggestions}}{n}$ and $\textit{Recall} = \dfrac{\#\text{relevant results suggestions}}{\#\text{relevant results}}$,
where we count as relevant results, the ACM classes considered valid by the user.

6.1.4.1.3 Results

Table 6.2 presents, for each user, the average P@n values, for her 8 automatically annotated papers. In addition, the average Recall (regarding the *top*-5 results) and the average UVCS values are presented at Table 6.3.

Note that, due to our annotation scenario (annotating research papers with ACM classes), it is rational to regard only the *top*-5 results during the P@n computation. That is, because the majority of the research papers under consideration do not handle more than 5 ACM hierarchy topics.

As we can observe, our method achieves high values both for Precision and Recall metrics. Moreover high Recall values have been achieved, with an average Recall value equal to 0.93. We should note that the relatively low P@4 and P@5 are justified from the fact that, for a respectable amount of test documents, the users expected (and thus validated) no more than 1~3 classes, that were found in the *top*-3 positions of the system's ranked suggestion list. Finally, it is obvious from the UVCS metric, that the automatic annotation mechanism supports and guides users during the annotation process, by suggesting correct classes that users had not previously thought of.

Table 6.4: Keyword queries

ID	Keywords
$q_{key}1$	knowledge discovery and privacy
$q_{key}2$	stream mining
$q_{key}3$	RDF indexing
$q_{key}4$	spatial databases
$q_{key}5$	clustering
$q_{key}6$	spatial access
$q_{key}7$	query language
$q_{key}8$	data model
$q_{key}9$	XML interoperability
$q_{key}10$	information integration

Table 6.5: Semantic queries

ID	Classes
$q_{sem}1$	K.4.1 [Public Policy Issues]: Privacy
$q_{sem}2$	H.2.8 [Database Applications]: Data mining
$q_{sem}3$	H.3.1 [Content Analysis and Indexing]: Indexing methods
$q_{sem}4$	H.2.8 [Database Applications]: Spatial databases and GIS
$q_{sem}5$	H.3.3 [Information Search and Retrieval]: Clustering
$q_{sem}6$	H.2.2 [Physical Design]: Access Methods
$q_{sem}7$	H.2.3 [Languages]: Query languages
$q_{sem}8$	H.2.1 [Logical Design]: Data models
$q_{sem}9$	D.2.12 [Interoperability]
$q_{sem}10$	H.2.5 [Heterogeneous Databases]

6.1.4.2 Search

In this section, we present an evaluation comparing the effectiveness of the search types provided by our framework. The comparison is performed in terms of Precision at position n, Recall, F-measure and Precision-Recall curve. In all cases, the proposed hybrid search method delivers higher quality results than traditional keyword-based or semantic-based search methods.

6.1.4.2.1 Setting

The weights used for the hybrid search method (Section 6.1.2.1.3) are assigned the following values: $w_3 = 0.7$, $w_4 = 0.3$ and $w_5 = 0.6$, $w_6 = 0.4$ after tuning. Intuitively, these values suggest that, in our problem setting, semantic-based score is slightly more important than keyword-based score in hybrid search.

6.1.4.2.2 Scenario

Our corpus consists of the 300 manually and automatically annotated research papers from the previous experiment (Section 6.1.4.1). First, we collect all the keywords defined in these papers and we randomly choose 10 keywords to be used as queries. Note that, keywords queries may contain one or more tokens.

Also, we map the selected keyword queries to semantic queries, using the ontology

Table 6.6: The average Precision at position n (P@n), Recall and F-measure for all queries

	P@1	**P@2**	**P@3**	**P@4**	**P@5**	**P@6**	**P@7**	**P@8**	**P@9**	**P@10**	**Recall**	**F-measure**
q_{key}	0.73	0.73	0.70	0.70	0.60	0.53	0.49	0.49	0.46	0.45	0.55	0.50
q_{sem}	1.00	0.95	0.91	0.89	0.91	0.88	0.83	0.78	0.74	0.68	0.84	0.75
q_{hybrA}	1.00	1.00	1.00	1.00	0.98	-	-	-	-	-	0.66	0.79
q_{hybrO}	1.00	1.00	0.97	0.98	0.96	0.95	0.95	0.90	0.83	0.76	0.98	0.86

classes. That is, to construct semantic queries that correspond to the keyword ones, we select the ontology classes that are most similar to the keyword content. In this way, we are able to perform both keyword, and ontology search, as well as hybrid search, comparing the effectiveness of each approach.

Table 6.4 presents the 10 keyword queries (q_{key}) which are used for this experiment. Table 6.5, presents the corresponding semantic queries (q_{sem}) expressed using the classes from ACM ontology. Hybrid queries are expressed by the combination of a keyword query and its corresponded semantic query. For hybrid search we apply booth (OR, AND) Boolean operators. The hybrid queries applying AND and OR operators are denoted respectively as q_{hybrA} and q_{hybrO}.

For each query we measure the quality of retrieval method using the Precision at position n at position n, for $n \in [1, 10]$ and Recall. Based on these measures, we compare the various search types offered by our system: (a) Keyword-based search; (b) Semantic-based search; (c) Hybrid search using AND operator ($hybrA$); and (d) Hybrid search using OR operator ($hybrO$). Finally, for each search type, we compute the average *Precision at positions* 1 to 10, *Recall*, *F-measure* and *Precision-Recall curves* for all queries.

6.1.4.2.3 Results For All Queries

Table 6.6 presents the average P@n for $n \in [1, 10]$ and the average Recall and F-measure values for all queries. Note that, most queries in hybrid search using the AND operator, do not retrieve more than 5-6 documents (as we can see from Table 6.7). As a consequence, the precision, for this search type is calculated only at positions 1 to 5.

Precision. As we can observe from Table 6.6, the hybrid search (for both operators) outperforms the keyword-based and semantic-based search at every position, with hybrA achieving slightly higher values at positions 4 and 5. Moreover, we can see that keyword-based search radically decreases after position 4, where semantic-based and hybrid search start decreasing progressively after the 6th position.

Hybrid search compared to keyword-based search, achieves a maximum increase of 100% at position 7 and a minimum increase of 30.3% at position 2. Comparing hybrid with semantic-based search, hybrid, achieves a maximum increase of 17.2% at position 10 and a minimum increase of 0% at position 1.

Recall. As we can see, the $hybrO$ outperforms the keyword-based and semantic-based search, achieving recall value close to 1 (0.98). Moreover, $hybrA$ achieves slightly lower recall values than semantic-based search. This is due to the fact that $hybrA$ search is very restrictive. So, too few documents are returned for each query with negative influence on the recall values.

Comparing $hybrO$ with keyword-based search, $hybrO$, achieves an increase of 78.2%. Moreover, despite the low recall values of $hybrA$ method, in comparison with keyword-based search, it increases the recall value at 20%. In comparison with semantic-based search, $hybrO$ achieves a increase of 16.7%. Finally, $hybrA$ achieves lower recall values than semantic-based search, having a decrease of 21.4%.

F-measure. As we can see, the hybrid search outperforms the other methods in F-measure value. Comparing *hybrO* with keyword-based and semantic-based search, *hybrO* achieves an increase of 72% and 14.6% respectively. Moreover, comparing *hybrA* with keyword-based and semantic-based search, *hybrA* achieves an increase of 58% and 0.05% respectively.

Precision vs. Recall. Figure 6.4 shows the average precision-recall curve for all queries. As we can see, hybrid search has a very stable performance, achieving high precision (close to 1) even for recall values greater than 0.8. *hybrO* precision starts to decrease noticeably only after recall values are greater than 0.9. For recall values lower than 0.6, *hybrA* achieves precision values higher than *hybrO*. Semantic-based search precision, progressively decreases from the beginning while recall increases. Finally, keyword-based search precision values rapidly decrease for recall values greater than 0.4.

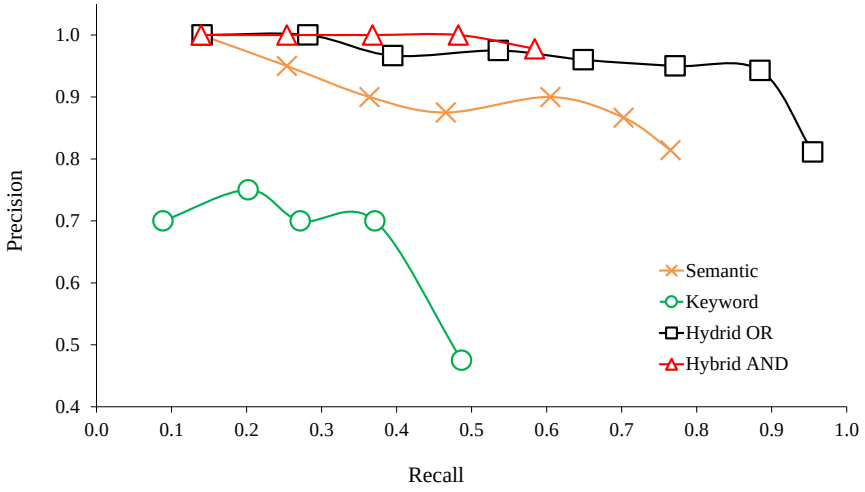


Figure 6.4: The average precision-recall curve for all queries

6.1.4.2.4 Results per Query

Table 6.7 presents for each query, the P@n for $n \in [1, 10]$ and Recall values. As we can see, in all queries, the hybrid search (for both boolean operators) outperforms the keyword-based and semantic-based search in precision values at every position. Moreover, regarding the recall measures, the *hybrO* search outperforms the other search methods in every query, with 9 out of 10 queries achieving recall values equal to 1.

As far as P@n is concerned, hybrid search achieves the highest precision values for all queries in every position. Hybrid search using AND and OR operators achieve similar precision values. However in many cases AND operator returns less than 10 documents. Semantic-based search achieves lower precision values (except *hybrA* for *Query* 6) than hybrid search, and higher values than keyword-based search (with 3 exceptions, *Queries* 4,5,6). Finally, keyword-based search achieves, in general, the lowest precision values.

As far as recall is concerned, hybrid search using OR operator achieves the highest recall values in all queries, with 9 out of 10 queries achieving recall values equal to 1. Semantic-based search achieves lower recall values than the former and higher or equal than rest methods, with two exceptions (*Queries* 6,8). Moreover, hybrid search using AND operator achieves lower or equal recall values than semantic-based search and higher than keyword-based search (with one exception, *Query* 2). Finally, keyword-based search achieves, in general, lowest recall values.

Table 6.7: The Precision at position n (P@n) and the Recall for each query

	Query 1				Query 2				Query 3				Query 4				Query 5			
	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}
P@1	1.00	1.00	1.00	1.00	0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P@2	1.00	1.00	1.00	1.00	0.50	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P@3	1.00	1.00	1.00	1.00	0.33	0.67	1.00	0.67	0.67	0.67	1.00	1.00	1.00	0.67	1.00	1.00	1.00	1.00	1.00	1.00
P@4	1.00	1.00	1.00	1.00	0.50	0.50	-	0.75	0.75	0.75	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
P@5	1.00	1.00	1.00	1.00	0.40	0.60	-	0.60	0.60	0.80	1.00	1.00	1.00	0.80	1.00	1.00	0.80	0.80	1.00	1.00
P@6	0.83	1.00	-	1.00	0.33	0.67	-	0.67	0.50	0.67	-	0.83	0.83	0.83	1.00	1.00	0.67	0.83	-	1.00
P@7	0.71	1.00	-	1.00	0.29	0.57	-	0.57	0.43	0.71	-	0.86	0.71	0.71	-	1.00	0.57	0.71	-	1.00
P@8	0.63	1.00	-	1.00	0.25	0.50	-	0.50	0.38	0.63	-	0.75	0.75	0.75	-	1.00	0.63	0.63	-	0.88
P@9	0.56	1.00	-	1.00	0.22	0.44	-	0.44	0.33	0.56	-	0.67	0.67	0.78	-	0.89	0.56	0.56	-	0.78
P@10	0.50	0.90	-	1.00	0.20	0.40	-	0.40	0.40	0.50	-	0.60	0.60	0.80	-	0.80	0.50	0.50	-	0.70
Recall	0.45	0.82	0.45	0.91	0.50	1.00	0.25	1.00	0.67	0.83	0.83	1.00	0.75	1.00	0.75	1.00	0.63	0.63	0.63	0.88

	Query 6				Query 7				Query 8				Query 9				Query 10			
	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}	q_{key}	q_{sem}	q_{hybrA}	q_{hybrO}
P@1	1.00	1.00	1.00	1.00	0	1.00	1.00	1.00	0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P@2	1.00	1.00	1.00	1.00	0	1.00	1.00	1.00	0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P@3	1.00	1.00	1.00	1.00	0.33	1.00	1.00	1.00	0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.67	1.00	1.00	1.00
P@4	1.00	1.00	1.00	1.00	0.25	1.00	1.00	1.00	0	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00	1.00
P@5	0.80	1.00	0.80	1.00	0.20	1.00	1.00	1.00	0	1.00	1.00	1.00	0.60	1.00	1.00	1.00	0.60	1.00	1.00	1.00
P@6	0.67	0.83	0.67	1.00	0.33	1.00	1.00	1.00	0	1.00	1.00	1.00	0.50	1.00	-	1.00	0.50	0.83	1.00	1.00
P@7	0.71	0.71	0.57	1.00	0.29	1.00	1.00	1.00	0.14	1.00	-	1.00	0.43	0.86	-	1.00	0.43	0.86	-	1.00
P@8	0.75	0.63	0.50	1.00	0.38	1.00	1.00	1.00	0.13	0.88	-	0.88	0.38	0.75	-	0.88	0.50	0.88	-	1.00
P@9	0.67	0.56	0.44	0.89	0.44	0.89	1.00	1.00	0.11	0.78	-	0.78	0.33	0.67	-	0.78	0.56	0.78	-	0.89
P@10	0.60	0.50	0.40	0.80	0.50	0.80	0.90	1.00	0.10	0.70	-	0.70	0.30	0.60	-	0.70	0.60	0.70	-	0.80
Recall	0.67	0.63	0.50	1.00	0.50	0.80	0.90	1.00	0.14	0.70	0.89	1.00	0.43	0.86	0.71	1.00	0.75	0.88	0.75	1.00

6.1.5 Related Work

A great number of approaches on semantic annotation have been proposed in the literature [317, 380]. Most of them are focused on annotating web resources such as HTML pages [241, 208, 127, 149, 13, 198, 384].

As far as plain text (or HTML) annotation is concerned, there are approaches that differ in the annotation and search facilities they offer. *GATE* [135] is a platform that offers an architecture, a framework and a graphical tool for language processing. Tools and resources are offered to perform textual annotation both manually and automatically using information extraction (IE) techniques.

KIM [240] provides an infrastructure for semantic annotation of documents (text or HTML), restricted, however, to its own ontology, called KIMO. The information extraction, document management and annotation part is based on GATE. The aim of the IE engine is the recognition of named entities with respect to the KIMO ontology. Compared to the above approaches, GoNTogle provides advanced searching facilities using a flexible combination of keyword-based and semantic-based search over documents. Also, it provides automatic annotation facilities based on models trained from user annotation history, so that annotation suggestions are tailored to user behavior.

AKTiveMedia [110] supports the annotation of text, images and HTML documents using both ontology-based and free-text annotations. For the automatic annotation task an underlying information extraction (IE) system has been integrated, learning from previous annotations and suggests annotations to the user. However, AKTiveMedia does not provide search facilities. Furthermore, the supported automatic annotation mechanism provides very low performance, when annotations are concern more than one tokens (due to the IE system). In addition, a serious limitation of the automatic annotation mechanism is that it takes into consideration only one class per annotation. In case of annotations with multiples classes, the rest of the classes are skipped.

The above tools support annotations on HTML or plain text. As far as popular document formats are concerned, *PDFTab* [163] is a Protégé plug-in for annotating PDF documents with OWL ontologies classes. Annotations are stored in the internal document representation, with the document structure remaining unchanged. Compared to GoNTogle, PDFTab has several limitations: it does not provide any search facilities or automatic annotation method. *SemanticWord* [361] is a MS Word plug-in which offers MS Word annotations with DAML+OIL ontologies. Compared to GoNTogle, SemanticWord integrates an information extraction system with no learning support to suggest annotations. Also, SemanticWord does not provide search facilities and does not support OWL and RDF/S ontologies.

Regarding the semantic search, in the recent years, numerous systems and approaches have been proposed in the literature [278]. An approach close to our, is introduced at [70], where a combination of keyword and semantic search over web sources is supported, on top of the AKTiveMedia framework [110]. A noticeable drawback of this approach is that the ranking of hybrid search, is relying only at keyword search where the semantic part is utilized only to exclude or include a result and not to rank it. Moreover, [70] does not support advanced search operations related to ontology semantics. Additionally, an interesting but less relative approach [184], analyzes the meaning of words and phrases, to define semantic relations between lexicalized concepts. In that case, syntactic search is extended with semantics, by converting words into concepts and exploiting the arisen semantics.

6.1.6 Summary

In this section we presented GoNTogle, a framework for document annotation and retrieval, built on top of Semantic Web and IR technologies. GoNTogle supports both manual and automatic document annotation using ontologies. A learning mechanism is implemented, providing automatic document annotation facilities based on textual information and user annotation history. In order to overcome the drawbacks of traditional keyword-based (like concept polysemy and synonymy) and semantic-based search (like partial or not existing annotations) we propose a hybrid search method. Hybrid search provides a flexible combination of keyword-based and semantic-based search. Moreover, several advanced ontology-based search operations are provided. Ontology information is exploited, to help the user expand or shrink the resulting list in order retrieve high quality results. A user-based evaluation is performed, in order to demonstrate the effectiveness of the automatic annotation method. Moreover, a comparative evaluation validates that, the proposed hybrid search, outperforms in all cases the keyword-based and semantic-based search in terms of precision and recall. Finally, all the proposed methods are implemented as a fully functional tool.

6.2 Publishing and Exploring Evolving Linked Data

The technology advances in scientific hardware (e.g., sensors, new-generation sequencers), together with the explosion of Web 2.0 technologies, have completely changed the way scientists create, disseminate and consume large volumes of information and new content. More and more scientific datasets break the walls of "private" management within their production cite, are published, and become available for potential data consumers, i.e., individual users, scientific communities, applications/services. Typical examples include experimental or observational data and scientific models from the life science domain, climate, earth, astronomy, etc.

Linked Data[6] (LD) is a compelling approach for the dissemination and re-use of scientific data, realizing the vision of the so-called Linked Science[7]. The LD paradigm involves practices to publish, share, and connect data on the Web, and offers a new way of data integration and interoperability. Briefly, LD is about using the Web to create links between data from different sources. The driving force to implement LD spaces is the RDF technology. The basic principles of the LD paradigm is (a) use the RDF data model to publish structured data on the Web, and (b) use RDF links to interlink data from different data sources. The aim of the LD technologies is to give rise to the Web of Data.

The Web of Data is impelled by the current trend towards an open Web. The open data movement is a significant and emerging force towards this direction. Open science data is open data related to observations and results of scientific activities, which are publicly available for anyone to analyze and reuse.

However, by just converting legacy scientific data as LD, we do not fully meet the requirements of data re-use. To ensure re-use and allow exploitation and validation of scientific results, several challenges related to scientific data dynamics should be tackled. Scientific data are evolving and diverse data. Users and services (a) should have access not only to up-to-date scientific LD bases but to any of the previous versions of those bases, and (b) should be able to track the changes among versions, as well as their cause and effects.

In this section, we present our work on publishing and exploring evolving life science data, and more specifically, genomic and experimental data related to microRNA

[6] linkeddata.org

[7] linkedscience.org

biomolecules (Section 6.2.1). We propose a change model based on RDF to capture versioned entities. Based on this model legacy data from two well-known microRNA databases are fused and exported as LD. The first database (Section 6.2.2) provides experimental data and observations, while the second one (Section 6.2.3) provides change and version information about microRNA entities. Our LD infrastructure can assist biologists to explore biological entities and their evolution by either using SPARQL queries or navigating among entity versions.

6.2.1 Background

Biologists used to consider proteins and DNA as movers and shakers in genomics, seeing RNA as nothing more than a messenger to carry information between the two. This has dramatically changed after the discovery, in early 2000s, of the key role played in gene expression by small RNA molecules, called *microRNAs* (miRNAs). miRNAs can completely silence proteins. They do so by binding themselves to complementary sequences on message RNA (mRNA) transcripts, called *targets*. The knowledge of *miRNA targets* (i.e., which mRNA transcripts are targeted by a miRNA) is important for therapeutic uses. For example, based on such knowledge, biologists can shut off genes by delivering artificial miRNA molecules into cells.

The first miRNA molecules were identified in 1993. Since then, there has been a dramatic increase in the number of miRNAs discovered and registered in *miRBase*[8], a searchable database of published miRNA sequences and annotation. However, there is a lack of high-throughput experimental methods for identifying miRNA targets. Thus, *computational methods* to predict targets have become increasingly important, and led to the experimental identification of many miRNA targets.

Our team in Athena R.C. and the DNA Intelligent Analysis (DIANA) group of "Alexander Fleming" B.S.R.C.[9] have developed a set of advanced Web applications to provide access to computationally predicted miRNA targets. Since its original launch, DIANA Web app has been one of the most widely used service for miRNA analysis. It includes the following two core services.

microT[10]. The service provides target prediction data for 1884 miRNAs and more than six million predicted target genes, organized in a relational database. Besides the target prediction experimental results, we provide miRNAs and genes functional analysis that goes beyond simple biological pathways, like, for example, relation of miRNAs to functional features, and diseases and medical descriptors. All retrieved miRNAs are associated to diseases, using textual information from PubMed[11], a well-known digital library for biomedical literature.

mirGen[12]. The service provides information about transcripts, and their transcription factors (TF) that correspond to miRNAs. A transcription factor is a protein that binds to specific DNA sequences, thereby controlling the flow of genetic information from DNA to mRNA. MirGen database stores information about 811 human genes, 1270 human miRNAs, 386 mouse genes and 1012 mouse miRNAs, organized in a relational database.

[8] www.mirbase.org
[9] www.fleming.gr
[10] diana.cslab.ece.ntua.gr/DianaTools/index.php?r=microtv4
[11] www.ncbi.nlm.nih.gov/pubmed
[12] diana.cslab.ece.ntua.gr/?sec=databases

Table 6.8: Part of miRNA database schema

Core tables	Column Description
Hairpins	id (`mima_id`), name, sequence, species, gene location info, etc.
Matures	id (`mimat`), name, sequence, species.
Transcripts	tid, id given from ensembl.org (`enstid`), species, DNA strand, gene location info, etc.
ProteinGenes	id given from ensembl.org (`ensgid`), name, description.
Keggs	id given from genome.jp (`kegg_id`), name.
Tissues	name, species.
Join Tables	**Column Description**
MatureHairpinConn	It relates matures and hairpins.
MicroT5Interactions	It contains all the experimentally verified gene-mature interactions (bindings).
ProteinGeneKeggConn	It relates genes to kegg pathways.
MatureTissueConn	It relates matures to tissues.

6.2.2 Data Schema

In this section, we present an overview of the miRNA database maintained by IMIS/Athena R.C. and the DIANA group of "A. Fleming" B.S.R.C., storing info about computationally predicted miRNA targets produced by the target prediction algorithm proposed by DIANA group[281].

To better understand the miRNA domain and the database schema design, we next clarify some issues. Since the term "miRNA" is nowadays used in a wide scope, it is common to distinguish between `hairpin miRNAs` and `mature miRNAs`, or just `hairpins` and `matures` from now on. The former signify the genomic location of the latter. A hairpin is actually processed into several matures. Matures can bind themselves to transcripts and prevent the creation of functional ribosomes (and, thus, prohibit protein construction). A transcript is a stretch of DNA transcribed into an RNA molecule (messenger RNA, ribosomal RNA, transfer RNA, etc).

The miRNA database has some core tables to store the key entities of the miRNA domain (hairpins, matures, transcripts and protein-encoding genes) and model their relationships (see Table 6.8 for a part of miRNA database schema).

There are also tables storing info about `Kegg pathways`[13] and `tissues`. Kegg pathways is a collection of manually drawn pathway maps, with textual descriptions, representing biologists' knowledge on molecular interaction and reaction networks.

6.2.3 A Model for Change and Version Management

The miRBase database is a searchable database of published miRNA sequences and annotation. The miRBase database maintains info for 18443 hairpins and 49670 matures. Each entry in miRBase represents a predicted hairpin miRNA with information on the location and sequence of the corresponding mature miRNA sequence. Hairpins, mature miRNAs and their relationship between them change in time. miRBase maintains a list of files that record successive versions along with the changes between them. A short description for each file follows.

- **miRNA.dat** It maintains info related to all known hairpins (like ID, name, related matures, related publications, sequence, etc.) at the time of each version. Every new version of miRNA.dat contributes to the previous one with all the newly discovered miRNAs, omitting the deleted ones. Example entries of miRNA.dat are shown in

[13] www.genome.jp/kegg/pathway.html

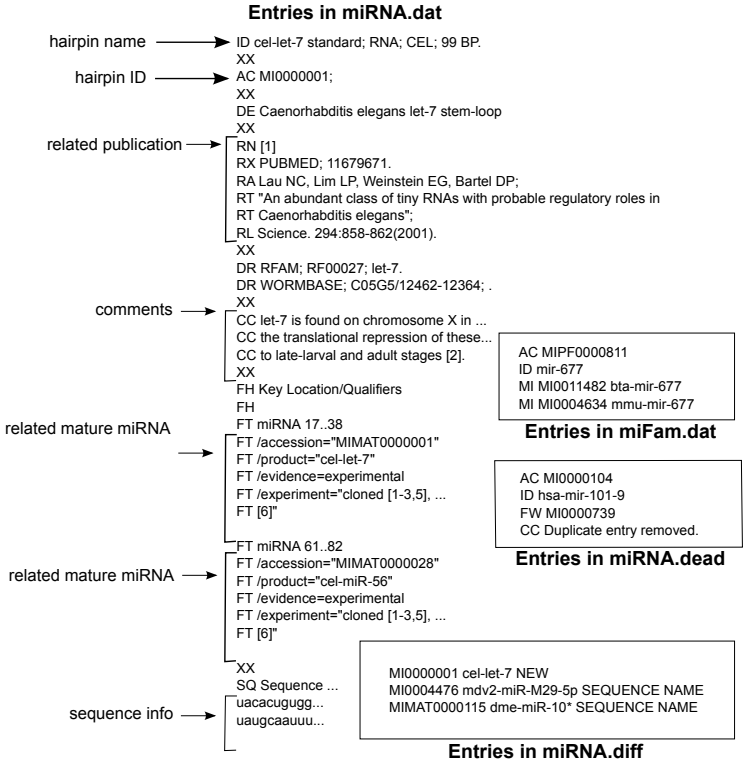


Figure 6.5: File examples of tracking miRNA changes

Figure 6.5, where info about the hairpin with name `cel-let-7` and id (i.e., key) `MI0000001` is presented.

– **miRNA.diff** It tracks change operations on hairpins and matures. Each version of miRNA.diff refers to a certain time period and tracks changes only for that period.

  Example entries of miRNA.diff are shown in Figure 6.5. For instance, `MI0000001 cel-let-7 NEW` means that the hairpin with ID MI0000001 and name cel-let-7 is created. Also, `MI0004476 mdv2-miR-M29-5p SEQUENCE NAME` means that the hairpin with ID MI0004476 has changed its name (to mdv2-miR-M29-5p) and its sequence. Note that to find the old name and the old sequence, we should refer to the older version of the miRNA.dat file, where hairpin names and info about sequences are available. Similarly, `MIMAT0000115 dme-miR-10* SEQUENCE NAME` means that the mature with ID MIMAT0000115 has changed its name (to dme-miR-10*) and its sequence. Note that IDs starting with "MIMA" refer to matures.

– **miRNA.dead** It keeps all deleted hairpins at the time of a version. It is maintained incrementally. Deletion means either getting rid of a hairpin (e.g., incorrectly characterized in previous versions) or replacing a hairpin with another one. For the latter case, links to existing hairpins are provided. Contrary to deleted hairpins, deleted mature miRNAs are not stored in miRNA.dead file.

  Example entries of miRNA.dead are shown in Figure 6.5. For instance, the hairpin with ID hsa-mir-101-9 and NAME MI0000104 has been deleted. The reason is that it was a duplicate entry (see the comment in CC field). There is a hairpin (MI0000739), though, that replaces the deleted one (see the FW field).

– **miFam.dat** It stores info about hairpin families at the time of a version. Hairpins that produce similar mature miRNAs belong to the same family. It is maintained incrementally. Example entries of miFam.dat are shown in Figure 6.5. For instance,

Table 6.9: Table HairpinsHistory (sample records)

mimaid	change	name	seq	first_appearance	last_appearance
..1364	NEW	dre-mir-10b	..xyz..	13	15
..1364	NAME	dre-mir-10b-1	..yzx...	16	17
..1364	SEQ	dre-mir-10b-1	..sdf..	18	19
..1364	SEQ	dre-mir-10b-1	..xxx..	20	32

Table 6.10: Table MaturesHistory (sample records)

mimat	change	name	seq	par. hairpin	first_appearance	last_appearance
..9477	NEW	bfl-miR-79	..yyx..	...	28	...
..9477	APH	bfl-miR-79	..xyz..	..021	28	29
..9477	NS	bfl-miR-9-3p	..xzy..	...	30	32

hairpins with IDs MI0011482 (NAME bta-mir-677) and MI0004634 (NAME mmu-mir-677) belong to the same family with is mir-677.

We have examined all files and recorded the following types of changes for hairpins: (1) NEW: a new hairpin is created; (2) NAME: a hairpin changes its name; (3) SEQUENCE (SEQ): a hairpin changes its sequence; (4) NAME/SEQUENCE (NS): a hairpin changes both its name and sequence at the same time; (5) FORWARD (FW): a hairpin is deleted, but miRBase give a link to another hairpin for replacement; and (6) DELETE (DEL): a hairpin is deleted (no replacement).

Similarly, we have identified the following type of changes for matures: (1) NEW: a new mature is created, (2) NAME: a mature changes its name, (3) SEQUENCE (SEQ): a mature changes its sequence; (4) NAME/SEQUENCE (NS): a mature changes both its name and sequence at the same time; (5) ADD PARENT HAIRPIN (APH): a new hairpin is added to the list of hairpins that produces a mature; (6) REMOVE PARENT HAIRPIN (RPH): a hairpin is removed from the list of hairpins that produces a mature; and (7) DELETE (DEL): a mature is deleted.

To manage change and version info, we maintain two history tables: HairpinsHistory and MaturesHistory. Tables 6.9 & 6.10 show how change and version info is maintained in history tables. For each hairpin change, HairpinsHistory keeps a record with, among others, the hairpin id, the type of change, the version number where the change occurred, and the version number where the next change occurs. The hairpin with id ..1364 is first created in version 13. In version 16, it changes name from `dre-mir-10b` to `dre-mir-10b-1`. No other change has occurred till version 18, where a change in its sequence has occurred. Another sequence change has occurred in version 20. Similarly, the mature with id ..9477 is first created in version 28, getting the name `bfl-miR-79`, and having the parent hairpin ..021. In version 30, it changes name (to `bfl-miR-9-3p`) and sequence.

6.2.4 Publishing Evolving miRNA Linked Data

6.2.4.1 Background

To publish miRNA and miRBase databases as LD, we adopted the "virtual RDF" approach: accessing a non-RDF database using an RDF view. Such an approach enables the access of non-RDF, legacy databases without having to replicate the whole database into RDF. The D2R server [88] is a popular tool that follows the "virtual RDF" approach for publishing the content of relational databases on the Semantic Web. Database content is mapped to RDF using the D2RQ declarative language that captures mappings between database schemas and RDFS/OWL schemas.

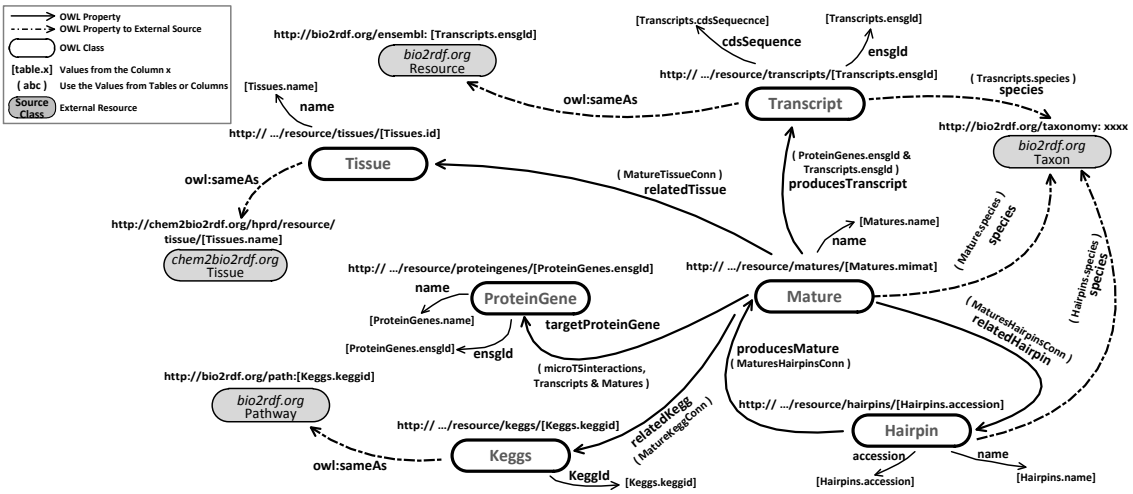


Figure 6.6: RDFS to database mappings: Up-to-date data

A D2RQ mapping specifies how RDF resources are identified and how RDF property values are generated from database content. Mappings in D2RQ are declared based on *ClassMaps* and *PropertyBridges*. A ClassMap maps a set of database records to an RDF class of resources. Resources are assigned URIs using URI patterns. The pattern `hairpins/@@diana_hairpins.mima_id@@`, for instance, produces a relative URI like `hairpins/MI0000005` by inserting a value from the column `mima_id` of table `hairpins` of miRNA database into the pattern. The D2R Server turns relative URIs into absolute URIs by expanding them with the server's base URI. If a database already contains URIs for identifying database content, then these external URIs can be used instead of pattern-generated URIs. The following ClassMap definition creates the class of hairpin resources, and assigns them URIs using their ids from the miRNA database:

```
map:Hairpins a d2rq:ClassMap;
    d2rq:dataStorage map:database;
    d2rq:uriPattern "hairpins/@@diana_hairpins.mima_id@@";
    d2rq:class diana:Hairpin;
    d2rq:classDefinitionLabel "Hairpin";
```

Each ClassMap has a set of PropertyBridges which specify how the properties of an RDF instance are created. Property values can be literals, URIs or blank nodes, and can be created directly from database values or by employing patterns. The following PropertyBridge definition creates the property `diana:name`. Values for that property are created from the `name` column of table `diana_haipins`:

```
map:diana_hairpins_name a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Hairpins;
    d2rq:property diana:name;
    d2rq:propertyDefinitionLabel "Hairpins name";
    d2rq:column "diana_hairpins.name";
```

Note that D2R provides flexible mappings of complex relational structures, allowing SQL statements directly in the mapping rules. The resulting record sets are grouped afterwards and the data is mapped to the created instances.

We used D2R as a full-fledge Linked Data server. The size of the LD base is around 100M triples.

6.2.4.2 Schema and Mappings

The miRNA LD schema has been designed around four core classes: `Hairpin`, `Mature`, `ProteinGene` and `Transcript` (defined as ClassMap entities in D2R - see previous sub-

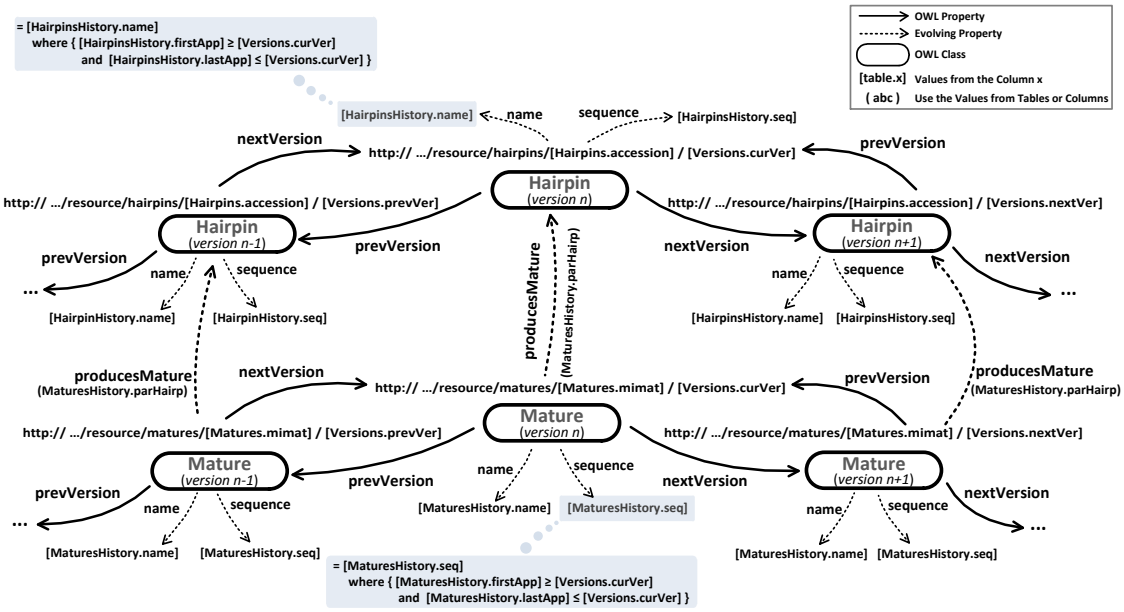


Figure 6.7: RDFS to database mappings: Historic data

section). Figure 6.6 shows an overview of the schema adopted and part of the mappings used. Consider, for example, the class `Mature`. Resources of that class are assigned URIs of the form `http://.../resource/matures/[Matures.mimat]`, where `Matures.mimat` gets values from column `mimat` of Table `Matures`. Some of the class properties are: `name`, `species`, `relatedKegg`, `targetsProteinGene` (defined as PropertyBridge entities in D2R - see Section 6.2.4.1).

Consider also the property `targetsProteinGene` that relates matures with genes (targets). Note that `ProteinGene` resources are assigned URIs of the form `http://.../resource/proteingenes/[ProteinGenes.ensgid]`, where `ProteinGenes.ensgid` gets values from column `ensgid` of Table `ProteinGenes`. For a given `Mature` URI, to calculate the URIs of related `ProteinGene` resources, the mapping definition should include the following join:

```
Matures.mimat=MicroT5Interactions.mimat AND
MicroT5Interactions.tid=Transcripts.tid AND
Transcripts.enstid=ProteinGenes.enstid.
```

Figure 6.7 shows an overview of the schema adopted and part of the mappings used to manage changes and versions (more details in Section 6.2.5.2). Finally, to link our LD to the LD cloud, we provide `owl:sameAs` links to appropriate biological LD infrastructures. See, for example, the BIO2RDF[14] data source that provide RDF descriptions for transcripts, tissues, keggs, and species.

6.2.5 Exploring Evolving miRNA Linked Data

One of the major research problems in LD publishing is how to deal with linked data that changes over time. While handling changes for information resources [90] is rather straightforward, handling changes for non-information resources is a challenging issue. Key requirements for dealing with changes in miRNA LD are the following:

– Biologists that care only about the current state of data should be able to browse or query the miRNA LD base easily to get up-to-date data. Also, up-to-date data should be easily retrieved using SPARQL.

[14] bio2rdf.org

– Biologists should be able to query historic miRNA data, and navigate through versions. Also, miRNA changes should be treated as first-class citizens so that one can form SPARQL queries that involve change resources, and trace those changes and their effects.

Next we present how we deal with the above requirements.

6.2.5.1 Up-to-date Data

Using the D2R browsing facilities, biologists can navigate through the miRNA LD base, exploring hairpin, mature, gene or transcript resources and their descriptions. All data provided refer to the current version of miRNA database. Also, any resource URI refers to the current version of that resource. This is ensured because all triples involving resources from `Hairpin`, `Mature`, `ProteinGene` and `Transcript` classes are populated from the core and join tables of Table 6.8 that are up-to-dated.

Using the D2R SPARQL end-point facilities, biologists can pose SPARQL queries to the miRNA LD. Whenever a resource URI is used in a query, it refers to the current version of that resource. To get up-to-date results, a property should be used to avoid the retrieval of out-of-date triples. For example, the following SPARQL query retrieves 10 hairpins, and their sequences, that are located in chromosome X from the current version of miRNA LD:

```
SELECT ?h ?s WHERE {
  ?h rdf:type diana:Hairpin.
  ?h diana:sequence ?s.
  ?h diana:chromosome "X".
  ?h diana:label "now". } LIMIT 10
```

6.2.5.2 Historic Data

Out-of-date resource descriptions are retrieved using the following pattern for URIs: `URI/{version number}`. For example, the URI: `http://.../resource/hairpins/MI0000044/8.0` retrieves the RDF description of hairpin `MI0000044` in version 8.0 of miRBase. To pose the previous SPARQL query on that version of miRBase, one should replace `?h diana:label "now".` with `?h diana:version "8.0".` Note that we also provide properties (`diana:nextVersion`, `diana:prevVersion`) to move to the next and the previous version of a resource description.

To be able to provide the property values and URIs which are valid at a certain version, we exploit the version information present in the history tables HairpinsHistory and MaturesHistory (see Tables 6.9 and 6.10). Figure 6.7 shows an overview of the schema adopted and part of the mappings used to manage changes and versions. For example, given a current version `curVer`, to retrieve the valid value for the `name` property of a hairpin, we should define a conditional mapping to focus the retrieval on values that remain unchanged for a time period that starts before `curVer` and ends after `curVer` (similarly for, e.g., mature names).

Each hairpin or mature resource description includes properties that capture the changes which those resources are affected by. For each change, we track its effect and its cause, where appropriate. Figure 6.8 shows the description of mature MIMAT0010008 at version 16[15]. The following SPARQL query retrieves 10 hairpins that where deleted or replaced in version in version 1.3. of miRBase, and the URIs of the change operations:

```
SELECT ?h ?d ?c WHERE {
  ?h rdf:type diana:Hairpin.
  {{?h diana:changeDelete ?d.} UNION {?h diana:changeForward ?c.}}
  ?h diana:version "1.3". } LIMIT 10
```

[15]URI: http://.../resource/matures/MIMAT0010008/16.0

Property	Value
diana:changeName	<http://62.217.113.118:8080/resource/mchange/1612>
diana:changeNew	<http://62.217.113.118:8080/resource/mchange/34396>
diana:changeParentHairpin	<http://62.217.113.118:8080/resource/mchange/MIMAT0010008_28>
diana:mimat	MIMAT0010008
diana:name	bfl-miR-129
diana:nextVersion	<http://62.217.113.118:8080/resource/matures/MIMAT0010008/17.0>
diana:prevVersion	<http://62.217.113.118:8080/resource/matures/MIMAT0010008/15.0>
is diana:producesMature of	<http://62.217.113.118:8080/resource/hairpins/MI0010519/16.0>
diana:sequence	CCUUUUUGUGGUUUGGGGCUUUU
diana:species	<http://bio2rdf.org/taxonomy:7739>
rdf:type	diana:Matures
diana:version	16.0

Property	Value
diana:inVersion	17.0
diana:newName	bfl-miR-129a
rdf:type	diana:matureNameChange

Generated by D2R Server

Figure 6.8: Resource description of mature MIMAT0010008 at version 16.0

We can also retrieve historical info about change occurrences. The following SPARQL query returns name or sequence changes that happened on hairpin MI0001364:

```
SELECT ?h ?c ?v WHERE {
  ?h rdf:type diana:Hairpin.
  ?h diana:accession "MI0001364".
  {{?h diana:changeName ?c. ?c diana:inVersion ?v.}
     UNION
  {?h diana:changeSequence ?c. ?c diana:inVersion ?v.}}}
```

6.2.6 Related Work

The Linked Data (LD) paradigm involves practices to publish, share, and connect data on the Web, and offers a new way of data integration and interoperability. Briefly, LD is about using the Web to create typed links between data from different sources. The driving force to implement Linked Data spaces is the RDF technology. The basic principles of the LD paradigm is (a) use the RDF data model to publish structured data on the Web, and (b) use RDF links to interlink data from different data sources. Linked Data technologies have given rise to the Web of Data: a Web of things in the world, described by data on the Web. The Web of Data extents current Web to a global data space connecting data from diverse domains. The Web of Data is impelled by the current trend towards an open Web. The open data movement is a significant and emerging force towards this direction. Open data is public data which are available to people without any restriction. LD serve a great cause, enabling transparency, accountability and good governance for public administrations.

In the context of LD, numerous approaches have been proposed to study the problems of evolution, versioning, and change detection. In [377], the term dataset dynamics is coined, essentially addressing content and interlinking changes in linked data sources. In [378], a comparative study on the approaches and tools for detecting, propagating and describing changes in LD resources and datasets is provided. This survey identifies the following interesting problems for LD dynamics: change detection at several granularity levels (i.e., at the dataset level, at the triple level, etc), common vocabulary for change description across multiple domains, appropriate communication and notification mechanisms for change propagation and finally automatic change (i.e., broken links) discovery. In [313], the authors deal with changes in the linkage between datasets and specifically with the problem of broken links. They propose, DSNotify, a framework able to assist human and machine actors fixing broken links. A similar approach is the Silk linking framework [389], which is used for discovering and maintaining data links between web data sources. It consists of a link discovery engine, a tool for evaluating the generated links and a protocol for maintaining data links between continuously changing data sources. Regarding versioning and temporal approaches to LD, in [144] the Memento framework is introduced as a resource versioning mechanism for LD. It is based on HTTP and handles different

versions of linked data by attaching time-specific attributes to HTTP requests. Finally, in [129] they propose linked timelines, a temporal representation and management for LD. This approach augments URIs with temporal attributes and employs temporal reasoning for resolving URIs validity.

Our approach is specially-tailored to the scientific domain of life science data, and more specifically to genomic and experimental data related to microRNA biomolecules. Several attempts have been recently made to provide scientific LD services. W3C has established the *Semantic Web Health Care and Life Sciences Interest Group* (HCLS)[16], aiming to exploit Semantic Web technologies for the management and the representation of biological, medicine and health care data. The HCLS group works on *Linking Open Drug Data* (LODD) project which provides linked RDF data exported from several data sources like *ClinicialTriasl.gov*, *DrugBank*, *DailyMed*, etc. Additionally, *Bio2RDF*[17] provides linked RDF data produced from over 30 biological data sources. Some earlier efforts include *YeastHub* [121], LinkHub [348], *BioDash* [296] and *BioGateway*[18]. Finally, *Chem2Bio2RDF* [117] integrates chemical and biological information. Also, several chemogenomics repositories have been transformed into RDF and linked to Bio2RDF and LODD RDF resources.

6.2.7 Summary

In this section we presented our work on publishing diachronic life science data. Particularly, legacy data from two well-known microRNA databases with experimental data and observations, as well as change and version information about microRNA entities, are fused, modeled and exported as Linked Data. Our Linked Data infrastructure can assist biologists to explore biological entities, navigate between versions, and also allow applications to query historical miRNA data and track changes via a SPARQL endpoint.

[16] www.w3.org/blog/hcls
[17] bio2rdf.org
[18] www.semantic-systems-biology.org/biogateway

Part IV

Conclusions

Chapter 7

Summary and Future Work

This thesis presented novel methods for managing and analysing large amounts of data. We focused on three complementary directions for enabling Big Data management and analysis. Initially, we proposed scalable methods for preference-aware data management and analysis. Then, we implemented techniques for efficient exploration and visualization over large sets of numeric, temporal and graph data. Finally, we proposed methods for semantic data integration, exploration and retrieval.

In the remainder of this chapter, we discus in more detail our contributions and we identify interesting aspects that we propose for future work.

7.1 Summary

Initially, we considered the personalization problem of finding and ranking objects that are preferable by a group of users based on their preferences. For this problem, we proposed an objective and fair interpretation based on Pareto-based aggregation. Based on this interpretation, we studied three related problems. The first is to find the set of objects that are unanimously considered ideal by the entire group. In the second problem, we relaxed the requirement for unanimity and only require a percentage of users to agree. Then, in the third problem, we devised an effective ranking scheme based on our aggregation framework. For the aforementioned problems we proposed index-based algorithms which employ a space partitioning index to hierarchically group objects. Regarding the ranking problem, we theoretically studied our ranking scheme and presented a number of theoretical properties satisfied by our approach.

Then, we studied some of the most well-known skyline algorithms. We adapted the algorithms based on a realistic I/O model that better captures performance in a real system. Furthermore, we studied the management of in-memory objects and we introduced various policies. In our experiments, we evaluated real disk-based implementations, rather than simulations. Our analysis, demonstrated that, in many cases and contrary to common belief, algorithms that pre-process the dataset are not faster. Finally, we evaluated the proposed policies, and reached the conclusion that in some settings these policies can reduce the number of dominance checks by more than 50%.

In the context of exploration, we examined the problem of on-the-fly efficient visual exploration over large sets of data. As a result, we proposed a framework that offers personalized multilevel exploration. Our framework is built on top of a lightweight tree-based structure that aggregates input objects into a hierarchical multilevel model. On top of this model, we defined different exploration scenarios, assuming various user exploration preferences. In order to enable efficient exploration over large datasets, our framework offers incremental hierarchy construction and prefetching based on user interaction. Finally, the proposed framework provides a method which dynamically and efficiently adapts an

existing hierarchy to a new, taking into account a set of user preferences.

Furthermore, we considered the problem of visualizing large graphs. For this problem we proposed a novel platform that introduces a new paradigm to interact with the visualized graph in a way that is similar to maps exploration. The platform bases its efficiency to a disk-based scheme for indexing and storing the graph. Finally, in order to enable very large graphs visualization, a partition-based approach is proposed.

Regarding semantic techniques, we considered the problem of integration between XML and semantic data sources. As a result, we proposed a framework which bridges the heterogeneity gap and creates an interoperable environment. In this context, we defined a mapping model, as well as query and schema transformation methods.

Next, we examined the problem of semantic information retrieval. For this purpose, we proposed a framework that supports ontology-based annotation and retrieval, in a fully collaborative environment. The framework provides an automatic annotation mechanisms that is based on a learning method that recommend annotations for new documents. Further, we introduced a model for flexible combination of textual-based and semantic-based retrieval in conjunction with advanced semantic-based operations.

Finally, we considered the problem of modeling, publishing and exploring evolving data, adopting the Linked Data paradigm. We proposed an RDF-based change model to capture versioned entities. Based on this model we converted legacy data from biological databases to evolving Linked Data. Moreover, we developed a Linked Data infrastructure that offers exploration and retrieval over evolving data.

7.2 Future Work

During the course of this dissertation, we have identified the following interesting aspects that we propose as future work.

- Recently, there has been a lot of work in partitioning-based skyline algorithms. Although these approaches significantly reduce the number of checks between objects, they do not consider the number I/Os that are possibly required. As a result, in cases where the skyline size exceeds the memory size, these algorithms perform a large number of I/Os which significantly affect the overall performance. A nice alternative would be to design a simple scan-based algorithm (i.e., BLN-like) that performs well in I/Os, enriched with a lightweight space partition scheme (e.g., grid), which can be exploited to reduce the object checks by considering both the notions of dominance and incomparability.

- In this thesis, we proposed a partition-based method for visualizing very large graphs. This method adopts an approach where the visualized partitions are combined and organized into a "global" partition. In this context, we presented a greedy algorithm that attempts to avoid node overlaps, as well as minimize the length of the edges connecting different partitions. Currently, we work on evaluating the effectiveness of the proposed algorithm, as well as on the development and the comparison of several alternative methods. A challenging extension would be to define a more flexible and complex setting for the partition organization problem. A potential solution to this problem would be to also examine the rotation of the partitions, and the reposition of nodes inside partitions. Finally, an interesting issue is to examine proving whether the partition organization problem is NP-hard.

- Another interesting problem would be the on-the-fly visualization of large graphs. In this setting, there is no preprocessing phase and the graph is stored in a file as raw data. Initially, the user selects a node to starts her exploration. This starting node

can be determined by several techniques, e.g., by providing the node's name, using keyword search, recommendation, etc. Each time the user visits a node, the nodes that are connected with this node within a predefined path length, are retrieved from the file and visualized. In the described problem, several challenging issues arise with respect to real-time exploration; for instance, how to efficiently find and retrieve from the large raw file, the parts of the graph that are required. Furthermore, prefetch and index parts of the graph that are likely to be accessed by user. Finally, in cases where a large number of edges and nodes are visualized, large amounts of memory are required from the user interface. Hence, in such cases there is a problem to select parts of the visualized graph to remove from the canvas.

- Regarding multilevel exploration, the framework proposed in this thesis considers objects to be organized and explored over one attribute. A challenging extension of this would be to provide methods and structures that support exploration over more than one dimension. For example, extend our framework to offer multilevel exploration (over two attributes) using scatter plots. Furthermore, an interesting problem would be the modification of our methods to consider issues related to efficient object management. For example, resigned our method in order to reduce the I/O cost over the raw input data; minimize the number of objects required to be accessed in each step by the incremental method, etc.

- In an exploration scenario, it is common that users are interesting in finding something interesting and useful without previously knowing what exactly they are searching for, until the time they identify it. In this case, users perform a sequence of operations (e.g., queries), in which the result of each operation determines the formulation of the next operation. In this setting, caching and prefetching the sets of data that are likely to be accessed by the user in the near future can significantly reduce the response time. Several works have recently studied the problem of caching and prefetching in an exploration scenario. An interesting direction would be the development of caching and prefetching techniques considering several exploration settings. These settings may be characterized by the supported interactive operations and/or the visualization type. For instance develop "operation-aware" caching and prefetching techniques for a specific operation, e.g., pan, drill-down, roll-up, zoom. In conjunction with "operation-aware" techniques, it would be interesting to developed "type-aware" techniques based on the visualization type, e.g., graph, line chart, scatter, histograms. Finally, an interesting topic would be the adaptation of caching and prefetching techniques from location-based and spatial-based query processing for the visual exploration context.

- In the context of multilevel exploration, an interesting direction would be to extend the presented framework in order to support more rendering policies and interaction operations. Particularly, it would be challenging to develop more flexible rendering policies; e.g., rendering all or a number of nodes of current level; rendering all nodes below (and including) the current level, etc. Regarding interaction operations, the currently supported operations (i.e., roll-up, drill-down) enable users to navigate over the hierarchy in a vertical fashion, by moving up and down to hierarchy levels. It would be useful to also support operations which will allow users to explore the hierarchy horizontally. For example, it would be valuable for the user to access different sets of sibling nodes without the need to change level on the hierarchy.

- The hierarchical aggregation framework presented in this thesis, organizes data based on binning methods. Particularly, in order to achieve efficient on-the-fly data pro-

cessing, simple binning methods (i.e., equal-width and equal-frequency) have been adopted. However, it is known that these methods are vulnerable to skewed data distributions and outliers. Hence, in order to effectively handle non-uniform data distributions, more sophisticated (e.g., supervised) discretization methods are required. On the other hand, in our setting, where on-the-fly data processing is required, the use of sophisticated discretization methods is not an option (due to their high computational complexity). Considering also the great importance of data reduction in the general problem of large data visualization, the development of both effectively and efficiently data reduction techniques would be an interesting topic.

– In the context of XML and Semantic Web interoperability, several interesting problems may be considered. First, an interesting topic would be the specification of sophisticated XQuery rewriting rules that exploit the XML Schema semantics, in order to improve the performance of the XQuery expressions resulted from the SPARQL translation. Another direction would be the extension of the presented methods in order to support the new SPARQL features introduced by SPARQL 1.1 (e.g., nested queries, aggregate functions). Finally, a useful extension would be the integration of our framework with other interoperability systems that handle different types of heterogeneous data sources.

In conclusion, we believe that there is a plethora of interesting and novel topics relevant to the issues studied in this dissertation. We hope that this thesis will be an instigation for further research in these areas.

Bibliography

[1] *Dublin Core Metadata Element Set.* Dublin Core Metadata Initiative. dublincore.org/documents/dces.

[2] *Encoded Archival Description (EAD).* Library of Congress. www.loc.gov/ead.

[3] *IEEE WG-12: IEEE Standard for Learning Object Metadata (LOM).* ltsc.ieee.org/wg12.

[4] *MARC 21 concise format for bibliographic metadata.* Library of Congress. www.loc.gov/marc/bibliographic/ecbdhome.html.

[5] *Metadata Authority Description Standard (MADS).* Library of Congress. www.loc.gov/standards/mads.

[6] *Metadata Encoding and Transmission Standard (METS).* Library of Congress. www.loc.gov/standards/mets.

[7] *Metadata Object Description Schema (MODS).* Library of Congress. www.loc.gov/standards/mods.

[8] *MPEG-21 Multimedia framework, ISO 21000-17:2003-2007.* Intl. Standardization Organization.

[9] *MPEG-7 Multimedia content description interface, ISO 15938-1-11:2002-2007.* Intl. Standardization Organization.

[10] *Niso Metadata for Images in XML (MIX).* Library of Congress. www.loc.gov/standards/mix.

[11] *RDB2RDF.* W3C Working Group. www.w3.org/2001/sw/rdb2rdf.

[12] *Sharable Content Reference Model (SCORM).* Advanced Distributed Learning Initiative (ADL). www.adlnet.gov/scorm/index.aspx.

[13] *SMORE: Create OWL Markup for HTML Web Pages.* http://www.mindswap.org/2005/SMORE.

[14] *Technical Metadata for Text (TextMD).* Library of Congress. www.loc.gov/standards/textMD/.

[15] *Text Encoding and Interchange (TEI).* TEI Consortium. www.tei-c.org.

[16] *Vra Core 4.0.* Visual Resources Association (VRA). www.vraweb.org/projects/vracore4/index.html.

[17] B. A. and et al., editors. *XML Path Language (XPath) 2.0.* W3C Rec., 2007. www.w3.org/TR/xpath20.

[18] M. A. and et al., editors. *XQuery 1.0 and XPath 2.0 Functions and Operators*. W3C Rec., 2010. `www.w3.org/TR/xpath-functions`.

[19] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Trans. Vis. Comput. Graph.*, 12(5), 2006.

[20] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(6), 2005.

[21] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[22] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9), 1988.

[23] M. Agosti, H. Albrechtsen, N. Ferro, I. Frommholz, P. Hansen, E. Panizzi, A. M. Pejtersen, and U. Thiel. DiLAS: a digital library annotation service. In *Intl. Workshop on Annotation for Collaboration - Methods, Tools and Practices (IWAC)*, 2005.

[24] M. Agosti and N. Ferro. A Formal Model of Annotations of Digital Content. *ACM Transactions on Information Systems (TOIS)*, 26(1), 2008.

[25] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 1989.

[26] R. Agrawal and E. L. Wimmers. A Framework for Expressing and Combining Preferences. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2000.

[27] V. Aguilera, S. Cluet, T. Milo, P. Veltri, and D. Vodislav. Views in a large-scale XML repository. *The Intl. Journal on Very Large Data Bases (VLDBJ)*, 11:3, 2002.

[28] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In *Extended Semantic Web Conference (ESWC)*, 2008.

[29] F. Alahmari, J. A. Thom, L. Magee, and W. Wong. Evaluating Semantic Browsers for Consuming Linked Data. In *Australasian Database Conference (ADC)*, 2012.

[30] H. Alani. TGVizTab: An Ontology Visualisation Extension for Protege. In *Workshop on Visualizing Information in Knowledge Engineering*, 2003.

[31] M. I. Ali, N. Lopes, O. Friel, and A. Mileo. Update semantics for interoperability among xml, RDF and RDB - A case study of semantic presence in cisco's unified presence systems. In *Web Technologies and Applications (APWeb)*, 2013.

[32] M. Alonen, T. Kauppinen, O. Suominen, and E. Hyvonen. Exploring the Linked University Data with Visualization Tools. In *Extended Semantic Web Conference (ESWC)*, 2013.

[33] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Querying XML Sources Using an Ontology-Based Mediator. In *Cooperative Information Systems (CoopIS)*, 2002.

[34] R. Angles and C. Gutierrez. The Expressive Power of SPARQL. In *Intl. Semantic Web Conference (ISWC)*, 2008.

[35] S. F. C. Araujo, D. Schwabe, and S. D. J. Barbosa. Experimenting with Explorator: a Direct Manipulation Generic RDF Browser and Querying Tool. In *Visual Interfaces to the Social and the Semantic Web*, 2009.

[36] D. Archambault, T. Munzner, and D. Auber. Grouse: Feature-Based, Steerable Graph Hierarchy Exploration. In *EuroVis07*, 2007.

[37] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Trans. Vis. Comput. Graph.*, 14(4), 2008.

[38] D. Archambault, T. Munzner, and D. Auber. Tugging Graphs Faster: Efficiently Modifying Path-Preserving Hierarchies for Browsing Paths. *IEEE Trans. Vis. Comput. Graph.*, 17(3), 2011.

[39] L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Intrigue: Personalized Recommendation of Tourist Attractions for Desktop and Hand Held Devices. *Applied Artificial Intelligence*, 17(8-9), 2003.

[40] M. Arenas and L. Libkin. XMLdata Exchange: Consistency and Query Answering. *Journal of ACM (JACM)*, 55:2, 2008.

[41] K. J. Arrow. *Social Choice and Individual Values*. Yale University Press, 2nd edition, 1963.

[42] J. A. Aslam and M. H. Montague. Models for Metasearch. In *Intl. ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2001.

[43] G. A. Atemezing and R. Troncy. Towards a linked-data based visualization wizard. In *Workshop on Consuming Linked Data*, 2014.

[44] D. Auber. Tulip - A Huge Graph Visualization Framework. In *Graph Drawing Software*. 2004.

[45] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *Intl. Semantic Web Conference (ISWC)*, 2007.

[46] S. Auer, J. Demter, M. Martin, and J. Lehmann. LODStats - An Extensible Framework for High-Performance Dataset Analytics. In *Knowledge Engineering and Knowledge Management*, 2012.

[47] S. Auer, R. Doehring, and S. Dietzold. LESS – Template-Based Syndication and Presentation of Linked Data. In *Extended Semantic Web Conference (ESWC)*, 2010.

[48] E.-A. Baatarjav, S. Phithakkitnukoon, and R. Dantu. Group Recommendation System for Facebook. In *OTM Workshops*, 2008.

[49] B. Bach, E. Pietriga, and I. Liccardi. Visualizing Populated Ontologies with OntoTrix. *Intl. J. Semantic Web Inf. Syst.*, 9(4), 2013.

[50] L. Baltrunas, T. Makcinskas, and F. Ricci. Group recommendations with rank aggregation and collaborative filtering. In *ACM conference on Recommender systems, RecSys*, 2010.

[51] R. Bamford, V. Borkar, M. Brantner, P. Fischer, D. Florescu, D. Graf, D. Kossmann, T. Kraska, D. Muresan, S. Nasoi, and M. Zacharioudakis. XQuery Reloaded. *VLDB Endowment*, 2:2, 2009.

[52] D. G. Bar and O. Glinansky. Family Stereotyping - A Model to Filter TV Programs for Multiple Viewers. In *Workshop on Personalization in Future TV*, 2002.

[53] I. Bartolini, P. Ciaccia, and M. Patella. Efficient Sort-based Skyline Evaluation. *ACM Transactions on Database Systems (TODS)*, 33(4), 2008.

[54] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *Conference on Weblogs and Social Media (ICWSM)*, 2009.

[55] L. Battle, R. Chang, and M. Stonebraker. Dynamic Prefetching of Data Tiles for Interactive Visualization. Technical Report, 2015.

[56] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *IEEE Conference on Big Data*, 2013.

[57] C. Becker and C. Bizer. Exploring the Geospatial Semantic Web with DBpedia Mobile. *J. Web Sem.*, 7(4), 2009.

[58] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 1990.

[59] B. B. Bederson. PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps. In *ACM symposium on User interface software and technology*, 2001.

[60] I. Bedini, G. Gardarin, and B. Nguyen. Deriving Ontologies from XML Schema. In *EDA*, 2008.

[61] I. Bedini, C. Matheus, P. Patel-Schneider, A. Boran, and B. Nguyen. Transforming XML Schema to OWL Using Patterns. In *ICSC*, 2011.

[62] F. Benedetti, L. Po, and S. Bergamaschi. A Visual Summary for Linked Open Data sources. In *Intl. Semantic Web Conference (ISWC)*, 2014.

[63] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls. In *ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[64] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *Journal of ACM (JACM)*, 25(4), 1978.

[65] K. Bereta, C. Nikolaou, M. Karpathiotakis, K. Kyzirakos, and M. Koubarakis. Sex-Tant: Visualizing Time-Evolving Linked Geospatial Data. In *Intl. Semantic Web Conference (ISWC)*, 2013.

[66] S. Berkovsky and J. Freyne. Group-based recipe recommendations: analysis of data aggregation strategies. In *ACM conference on Recommender systems, RecSys*, 2010.

[67] A. Bernd, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *Intl. Semantic Web Conference (ISWC)*, 2002.

[68] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Intl. Semantic Web User Interaction*, 2006.

[69] D. Berrueta, J. E. Labra, and I. Herman. XSLT+SPARQL: Scripting the Semantic Web with SPARQL embedded into XSLT stylesheets. In *Workshop on Scripting for the Semantic Web*, 2008.

[70] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *Extended Semantic Web Conference (ESWC)*, 2008.

[71] N. Bikakis, K. Benouaret, and D. Sacharidis. Reconciling Multiple Categorical Preferences with Double Pareto-Based Aggregation. In *Intl. Conference on Database Systems for Advanced Applications (DASFAA)*, 2014.

[72] N. Bikakis, K. Benouaret, and D. Sacharidis. Finding Desirable Objects under Group Categorical Preferences. *Knowledge and Information Systems Journal (KAIS)*, 2015.

[73] N. Bikakis, G. Giannopoulos, T. Dalamagas, and T. K. Sellis. Integrating Keywords and Semantics on Document Annotation and Search. In *Intl. Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2010.

[74] N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Querying XML data with SPARQL. In *Intl. Conference on Database and Expert Systems Applications (DEXA)*, 2009.

[75] N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Semantic Based Access over XML Data. In *World Summit on the Knowledge Society (WSKS)*, 2009.

[76] N. Bikakis, J. Liagouris, M. Kromida, G. Papastefanatos, and T. K. Sellis. Towards Scalable Visual Exploration of Very Large RDF Graphs. In *Extended Semantic Web Conference (ESWC)*, 2015.

[77] N. Bikakis, J. Liagouris, M. Krommyda, G. Papastefanatos, and T. K. Sellis. graphVizdb: A Scalable Platform for Interactive Large Graph Visualization. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2016.

[78] N. Bikakis, G. Papastefanatos, M. Skourla, and T. K. Sellis. A Hierarchical Framework for Efficient Multilevel Visual Exploration and Analysis. *Semantic Web Journal*, 2016.

[79] N. Bikakis, D. Sacharidis, and T. K. Sellis. A study on external memory scan-based skyline algorithms. In *Intl. Conference on Database and Expert Systems Applications (DEXA)*, 2014.

[80] N. Bikakis and T. Sellis. Exploration and Visualization in the Web of Big Linked Data: A Survey of the State of the Art. In *International Workshop on Linked Web Data Management (LWDM)*, 2016.

[81] N. Bikakis, M. Skourla, and G. Papastefanatos. rdf: SynopsViz - A Framework for Hierarchical Linked Data Visual Exploration and Analysis. In *Extended Semantic Web Conference (ESWC)*, 2014.

[82] N. Bikakis, C. Tsinaraki, N. Gioldasis, I. Stavrakantonakis, and S. Christodoulakis. The XML and Semantic Web Worlds: Technologies, Interoperability and Integration: A Survey of the State of the Art. In *Semantic Hyper/Multimedia Adaptation - Schemes and Applications, Springer*. 2013.

[83] N. Bikakis, C. Tsinaraki, I. Stavrakantonakis, and S. Christodoulakis. Supporting SPARQL Update Queries in RDF-XML Integration. In *Intl. Semantic Web Conference (ISWC)*, 2014.

[84] N. Bikakis, C. Tsinaraki, I. Stavrakantonakis, N. Gioldasis, and S. Christodoulakis. The SPARQL2XQuery Interoperability Framework. Technical report, 2012. `www.dblab.ntua.gr/~bikakis/SPARQL2XQueryTR2012.pdf`.

[85] N. Bikakis, C. Tsinaraki, I. Stavrakantonakis, N. Gioldasis, and S. Christodoulakis. The SPARQL2XQuery interoperability framework - Utilizing Schema Mapping, Schema Transformation and Query Translation to Integrate XML and the Semantic Web. *World Wide Web*, 18(2), 2015.

[86] S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres. Mapping between RDF and XML with XSPARQL. *J. Data Semantics*, 1:3, 2012.

[87] S. Bischof, N. Lopes, and A. Polleres. Improve Efficiency of Mapping Data between XML and RDF with XSPARQL. In *Intl. Conference on Web Reasoning and Rule Systems (RR)*, 2011.

[88] C. Bizer and R. Cyganiak. D2r server – publishing relational databases on the semantic web. In *Intl. Semantic Web Conference (ISWC)*, 2006.

[89] C. Bizer and R. Cyganiak. D2R Server - Publishing Relational Databases on the Semantic Web. In *Intl. Semantic Web Conference (ISWC)*, 2006.

[90] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Intl. J. Semantic Web Inf. Syst.*, 5(3), 2009.

[91] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Intl. Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems*, 2009.

[92] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowl.-Based Syst.*, 46, 2013.

[93] P. Bohannon, W. Fan, M. Flaster, and P. Narayan. Information Preserving XML Schema Embedding. In *Intl. Conference on Very Large Databases (VLDB)*, 2005.

[94] H. Bohring and S. Auer. Mapping XML to OWL Ontologies. In *Leipziger Informatik-Tage*, 2005.

[95] T. Boinski, A. Jaworska, R. Kleczkowski, and P. Kunowski. Ontology visualization. In *Conference on Information Technology*, 2010.

[96] A. Bonifati, E. Q. Chang, T. Ho, L. V. S. Lakshmanan, R. Pottinger, and Y. Chung. Schema Mapping and Query Translation in Heterogeneous P2P XML Databases. *The Intl. Journal on Very Large Data Bases (VLDBJ)*, 19:2, 2010.

[97] L. Boratto and S. Carta. State-of-the-Art in Group Recommendation and New Approaches for Automatic Identification of Groups. In *Information Retrieval and Mining in Distributed Environments*. 2011.

[98] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2001.

[99] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph.*, 17(12), 2011.

[100] D. Brickley and G. R. V., editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Rec., 2004. www.w3.org/TR/rdf-schema.

[101] M. Bruls, K. Huizing, and J. van Wijk. Squarified Treemaps. In *Joint Eurographics and IEEE TCVG Symposium on Visualization*, 1999.

[102] J. M. Brunetti, S. Auer, R. Garcia, J. Klimek, and M. Necasky. Formal Linked Data Visualization Model. In *Intl. Conference on Information Integration and Web-based Applications & Services, (IIWAS)*, 2013.

[103] J. M. Brunetti, R. Gil, and R. Garcia. Facets and Pivoting for Flexible and Usable Linked Data Exploration. In *Interacting with Linked Data Workshop*, 2012.

[104] B. C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[105] B. C. *Mapping Relational Data to RDF with Virtuoso's RDF Views*. OpenLink Software, 2007.

[106] D. V. Camarda, S. Mazzini, and A. Antonuccio. LodLive, exploring the web of data. In *Conference on Semantic Systems (I-SEMANTICS)*, 2012.

[107] A. E. Cano, A. Dadzie, and M. Hartmann. *Who's Who* - A Linked Data Visualisation Tool for Mobile Environments. In *Extended Semantic Web Conference (ESWC)*, 2011.

[108] I. Cantador and P. Castells. Group Recommender Systems: New Perspectives in the Social Web. In *Recommender Systems for the Social Web*. 2012.

[109] K. Chakrabarti, S. Chaudhuri, and S. Hwang. Automatic Categorization of Query Results. In *ACM Conference on Management of Data (SIGMOD)*, 2004.

[110] A. Chakravarthy, V. Lanfranchi, and F. Ciravegna. Cross-media document annotation and enrichment. In *Semantic Authoring and Annotation Workshop*, 2006.

[111] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified Computation of Skylines with Partially-Ordered Domains. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2005.

[112] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant Skylines in High Dimensional Space. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2006.

[113] S. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, 2008.

[114] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The Onion Technique: Indexing for Linear Optimization Queries. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2000.

[115] D. L. Chao, J. Balthrop, and S. Forrest. Adaptive radio: achieving consensus using negative preferences. In *ACM Conference on Supporting Group Work*, 2005.

[116] A. Chebotko, S. Lub, and F. Fotouhib. Semantics preserving SPARQL-to-SQL translation. *Data & Knowl. Eng. (DKE)*, 68:10, 2009.

[117] B. Chen, X. Dong, D. Jiao, H. Wang, Q. Zhu, Y. Ding, and D. J. Wild. Chem2bio2rdf: a semantic framework for linking and data mining chemogenomic and systems chemical biology data. *BMC Bioinformatics*, 11, 2010.

[118] H. Chen, Z. Wu, H. Wang, and Y. Mao. RDF/RDFS-based Relational Database Integration. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2006.

[119] L. Chen and X. Lian. Efficient Processing of Metric Skyline Queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 21(3), 2009.

[120] Z. Chen and T. Li. Addressing diverse user preferences in SQL-query-result navigation. In *ACM Conference on Management of Data (SIGMOD)*, 2007.

[121] K.-H. Cheung, K. Y. Yip, A. K. Smith, R. de Knikker, A. Masiar, and M. Gerstein. Yeasthub: a semantic web use case for integrating data in life sciences domain. In *ISMB (Supplement of Bioinformatics)*, 2005.

[122] J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems (TODS)*, 28(4), 2003.

[123] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2003.

[124] V. Christophides, G. Karvounarakis, I. Koffina, G. Kokkinidis, A. Magkanaraki, D. Plexousakis, G. Serfiotis, and V. Tannen. The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware. In *Workshop on the Semantic Web, Ontologies and Databases (SWDB)*, 2003.

[125] V. Christophides, G. Karvounarakis, A. Magkanaraki, D. Plexousakis, and V. Tannen. The ICS-FORTH Semantic Web Integration Middleware (SWIM). *IEEE Data Eng. Bull.*, 26(4), 2003.

[126] W. W. Chu and K. Chiang. Abstraction of High Level Concepts from Numerical Values in Databases. In *AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[127] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *World Wide Web Conference (WWW)*, 2004.

[128] O. Corby, L. Kefi-Khelif, H. Cherfi, F. Gandon, and K. Khelif. Querying the semantic web of data using SPARQL, RDF and XML. Technical report, INRIA, 2009.

[129] G. Correndo, M. Salvadores, I. Millard, and N. Shadbolt. Linked timelines: Temporal representation and management in linked data. In *Intl. Workshop on Consuming Linked Data (COLD)*, 2010.

[130] A. Crossen, J. Budzik, and K. J. Hammond. Flytrap: intelligent group music recommendation. In *Int. Conference on Intelligent User Interfaces*, 2002.

[131] C. Cruz and C. Nicolle. Ontology Enrichment and Automatic Population from XML Data. In *Intl. Workshop on Ontology-based Techniques*, 2008.

[132] I. Cruz, X. Huiyong, and F. Hsu. An Ontology-Based Framework for XML Semantic Integration. In *Intl. Database Engineering & Applications Symposium (IDEAS)*, 2004.

[133] I. Cruz, H. Xiao, and F. Hsu. Peer-to-peer semantic integration of XML and RDF data sources. In *Agents and Peer-to-Peer Computing Workshop (AP2PC)*, 2004.

[134] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-Based Edge Clustering for Graph Visualization. *IEEE Trans. Vis. Comput. Graph.*, 14(6), 2008.

[135] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Annual Meeting of the Association for Computational Linguistics (ALC)*, 2002.

[136] B. D., editor. *SPARQL Query Results XML Format*. W3C Rec., 2008. `www.w3.org/TR/rdf-sparql-XMLres`.

[137] C. D., editor. *Gleaning Resource Descriptions from Dialects of Languages*. W3C Rec., 2007. `www.w3.org/TR/grddl`.

[138] F. D., editor. *XML Schema Part 0: Primer*. W3C Rec., 2004. `www.w3.org/TR/xmlschema-0`.

[139] A. Dadzie, V. Lanfranchi, and D. Petrelli. Seeing is believing: Linking data with knowledge. *Information Visualization*, 8(3), 2009.

[140] A. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2), 2011.

[141] A. Dadzie, M. Rowe, and D. Petrelli. *Hide the Stack:* Toward Usable Linked Data. In *Extended Semantic Web Conference (ESWC)*, 2011.

[142] T. Dalamagas, N. Bikakis, G. Papastefanatos, Y. Stavrakas, and A. G. Hatzigeorgiou. Publishing life science data as linked open data: the case study of miRBase. In *Intl. Workshop on Open Data (WOD)*, 2012.

[143] S. Das, S. Sundara, and R. Cyganiak, editors. *R2RML: RDB to RDF Mapping Language*. W3C Rec., 2012. `www.w3.org/TR/r2rml`.

[144] H. V. de Sompel, R. Sanderson, M. L. Nelson, L. Balakireva, H. Shankar, and S. Ainsworth. An http-based versioning mechanism for linked data. In *Intl. Workshop on Linked Data on the Web, (LDOW)*, 2010.

[145] D. DeHaan, D. Toman, M. Consens, and T. Ozsu. A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2003.

[146] L. Deligiannidis, K. Kochut, and A. P. Sheth. RDF data exploration and visualization. In *Workshop on CyberInfrastructure: Information Management in eScience*, 2007.

[147] D. V. Deursen, C. Poppe, G. Martens, E. Mannens, and R. V. Walle. XML to RDF conversion: a generic approach. In *AXMEDIS*, 2008.

[148] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *Intl. Conference on Database Theory (ICDT)*, 2003.

[149] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. S. McCurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. A Case for Automated Large-Scale Semantic Annotation. *Journal of Web Semantics*, 1(1), 2003.

[150] J. Dokulil and J. Katreniakova. Using Clusters in RDF Visualization. In *Advances in Semantic Processing*, 2009.

[151] P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. Prefetching for Visual Data Exploration. In *Conference on Database Systems for Advanced Applications (DASFAA)*, 2003.

[152] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Intl. Conference on Machine Learning*, 1995.

[153] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schopf, H. Staffler, and S. Zugal. Translating XPath Queries into SPARQL Queries. In *OTM Workshops*, 2007.

[154] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schopf, H. Staffler, and S. Zugal. Bringing the XML and Semantic Web Worlds Closer: Transforming XML into RDF and Embedding XPath into SPARQL. In *Intl. Conference on Enterprise Information Systems (ICEIS)*, 2008.

[155] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schöpf, H. Staffler, and S. Zugal. Embedding XPATH Queries into SPARQL Queries. In *Intl. Conference on Enterprise Information Systems (ICEIS)*, 2008.

[156] M. Dudas, O. Zamazal, and V. Svatek. Roadmapping and Navigating in the Ontology Visualization Landscape. In *Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2014.

[157] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *World Wide Web Conference (WWW)*, 2001.

[158] V. Eisenberg and Y. Kanza. D2rq/update: updating relational data via virtual RDF. In *World Wide Web Conference (WWW)*, 2012.

[159] M. Elahi, M. Ge, F. Ricci, D. Massimo, and S. Berkovsky. Interactive Food Recommendation for Groups. In *ACM Conference on Recommender Systems, RecSys*, 2014.

[160] A. Eldawy, M. Mokbel, and C. Jonathan. HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2016.

[161] B. Elliott, E. Cheng, C. Thomas-Ogbuji, and Z. M. Ozsoyoglu. A Complete Translation from SPARQL into Efficient SQL. In *Intl. Database Engineering & Applications Symposium (IDEAS)*, 2009.

[162] N. Elmqvist and J. Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Trans. Vis. Comput. Graph.*, 16(3), 2010.

[163] H. Eriksson. An annotation tool for semantic documents. In *Extended Semantic Web Conference (ESWC)*, 2007.

[164] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked Open Data Statistics: Collection and Exploitation. In *Knowledge Engineering and the Semantic Web*, 2013.

[165] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea. Skeleton-Based Edge Bundling for Graph Visualization. *IEEE Trans. Vis. Comput. Graph.*, 17(12), 2011.

[166] Z. F. Converting SPARQL to SQL. Technical report, 2006. lists.w3.org/Archives/Public/public-rdf-dawg/2006OctDec/att-0058/sparql-to-sql.pdf.

[167] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci. (TCS)*, 2005.

[168] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top k Lists. *SIAM J. Discrete Math.*, 17(1), 2003.

[169] S. Falconer, C. Callendar, and M.-A. Storey. A Visualization Service for the Semantic Web. In *Knowledge Engineering and Management by the Masses*. 2010.

[170] M. Farah and D. Vanderpooten. An Outranking Approach for Rank Aggregation in Information Retrieval. In *Intl. ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2007.

[171] J. Farrell and H. Lausen, editors. *Semantic Annotations for WSDL and XML Schema*. W3C Rec., 2007. www.w3.org/TR/sawsdl.

[172] M. Ferdinand, C. Zirpins, and D. Trastour. Lifting XML Schema to OWL. In *Intl. Conference on Web Engineering (ICWE)*, 2004.

[173] D. Fisher, I. O. Popov, S. M. Drucker, and m. c. schraefel. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Conference on Human Factors in Computing Systems CHI*, 2012.

[174] E. A. Fox and J. A. Shaw. Combination of Multiple Searches. In *Text Retrieval Conference (TREC)*, 1993.

[175] B. Fu, N. F. Noy, and M.-A. Storey. Eye Tracking the User Experience - An Evaluation of Ontology Visualization Techniques. *Semantic Web Journal (to appear)*, 2015.

[176] Y. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *IEEE Visualization*, 1999.

[177] E. R. Gansner, Y. Hu, S. C. North, and C. E. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *IEEE Pacific Visualization Symposium (PacificVis)*, 2011.

[178] S. Gao, C. M. Sperberg-McQueen, and T. H. S., editors. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C Rec., 2012. www.w3.org/TR/xmlschema11-1.

[179] I. Garcia, L. Sebastia, and E. Onaindia. On the design of individual and group recommender systems for tourism. *Expert Syst. Appl.*, 38(6), 2011.

[180] R. Garcia and O. Celma. Semantic integration and retrieval of multimedia metadata. In *Intl. Workshop on Knowledge Markup and Semantic Annotation*, 2005.

[181] S. Garcia, J. Luengo, J. A. Saez, V. Lopez, and F. Herrera. A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Trans. Knowl. Data Eng.*, 25(4), 2013.

[182] M. Gartrell, X. Xing, Q. Lv, A. Beach, R. Han, S. Mishra, and K. Seada. Enhancing group recommendation by incorporating social relationship interactions. In *ACM Int. conference on Supporting group work, GROUP*, 2010.

[183] G. Giannopoulos, N. Bikakis, T. Dalamagas, and T. K. Sellis. GoNTogle: A Tool for Semantic Annotation and Search. In *Extended Semantic Web Conference (ESWC)*, 2010.

[184] F. Giunchiglia, U. Kharkevich, and I. Zaihrayeu. Concept search. In *Extended Semantic Web Conference (ESWC)*, 2009.

[185] B. Glimm and O. C., editors. *SPARQL 1.1 Entailment Regimes*. W3C Working Draft, 2012. `www.w3.org/TR/sparql11-entailment`.

[186] P. Godfrey, J. Gryz, and P. Lasek. Interactive Visualization of Large Data Sets, 2015. Technical Report, York University.

[187] P. Godfrey, J. Gryz, P. Lasek, and N. Razavi. Visualization through inductive aggregation. In *Conference on Extending Database Technology (EDBT)*, 2016.

[188] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *The Intl. Journal on Very Large Data Bases (VLDBJ)*, 16(1), 2007.

[189] A. Graves. Creation of Visualizations Based on Linked Data. In *Conference on Web Intelligence, Mining and Semantics*, 2013.

[190] S. Groppe, J. Groppe, V. Linnemann, D. Kukulenzm, N. Hoeller, and C. Reinke. Embedding SPARQL into XQuery/XSLT. In *ACM Intl. Symposium on Applied Computing (SAC)*, 2008.

[191] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *ACM Conference on Management of Data (SIGMOD)*, 1984.

[192] F. Haag, S. Lohmann, S. Negru, and T. Ertl. OntoViBe: An Ontology Visualization Benchmark. In *Workshop on Visualizations and User Interfaces for Knowledge Engineering and Linked Data Analytics*, 2014.

[193] P. Haase, M. Schmidt, and A. Schwarte. The Information Workbench as a Self-Service Platform for Linked Data Applications. In *Workshop on Consuming Linked Data (COLD)*, 2011.

[194] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza Data Management Infrastructure for Semantic Web Applications. In *World Wide Web Conference (WWW)*, 2003.

[195] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2003.

[196] J. Han and Y. Fu. Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. In *AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[197] S. Handschuh and S. Staab, editors. *Annotation for the Semantic Web*. IOS Press, 2003.

[198] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM: Semi-automatic CREAtion of Metadata. In *Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2002.

[199] A. Harth. VisiNav: A system for visual search and navigation on web data. *J. Web Sem.*, 8(4), 2010.

[200] B. Haslhofer, W. Jochum, R. King, C. Sadilek, and K. Schellner. The LEMO annotation framework: weaving multimedia annotations with the web. *Int. J. on Digital Libraries (JODL)*, 10(1), 2009.

[201] T. Hastrup, R. Cyganiak, and U. Bojars. Browsing Linked Data with Fenfire. In *World Wide Web Conference (WWW)*, 2008.

[202] J. Heer and S. Kandel. Interactive Analysis of Big Data. *ACM Crossroads*, 19(1), 2012.

[203] P. Heim, S. Lohmann, and T. Stegemann. Interactive Relationship Discovery via the Semantic Web. In *Extended Semantic Web Conference (ESWC)*, 2010.

[204] P. Heim, S. Lohmann, D. Tsendragchaa, and T. Ertl. SemLens: visual analysis of semantic data with scatter plots and semantic lenses. In *Conference on Semantic Systems (I-SEMANTICS)*, 2011.

[205] J. Helmich, J. Klimek, and M. Necasky. Visualizing RDF Data Cubes Using the Linked Data Visualization Model. In *Extended Semantic Web Conference (ESWC)*, 2014.

[206] N. Henry, J. Fekete, and M. J. McGuffin. NodeTrix: a Hybrid Visualization of Social Networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6), 2007.

[207] M. Hert, G. Reif, and H. C. Gall. Updating relational data via sparql/update. In *EDBT/ICDT Workshops*, 2010.

[208] A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the World Wide Web. In *World Wide Web Conference (WWW)*, 2005.

[209] D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Trans. Vis. Comput. Graph.*, 12(5), 2006.

[210] W. Hop, S. de Ridder, F. Frasincar, and F. Hogenboom. Using Hierarchical Edge Bundles to visualize complex ontologies in GLOW. In *ACM Symposium on Applied Computing (SAC)*, 2012.

[211] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2001.

[212] A. Hussain, K. Latif, A. Rextin, A. Hayat, and M. Alam. Scalable Visualization of Semantic Nets using Power-Law Graphs. *AMIS*, 8(1), 2014.

[213] D. Huynh, S. Mazzocchi, and D. R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *Intl. Semantic Web Conference (ISWC)*, 2005.

[214] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of Data Exploration Techniques. In *ACM Conference on Management of Data (SIGMOD)*, 2015.

[215] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4), 2008.

[216] J. Im, F. G. Villegas, and M. J. McGuffin. VisReduce: Fast and responsive incremental information visualization of large datasets. In *IEEE Conference on Big Data*, 2013.

[217] Z. Ives, A. Halevy, P. Mork, and I. Tatarinov. Piazza: mediation and integration infrastructure for Semantic Web data. *J. Web Sem.*, 1:2, 2004.

[218] A. Jameson. More than the sum of its members: challenges for group recommender systems. In *Working conference on Advanced visual interfaces*, 2004.

[219] A. Jameson and B. Smyth. Recommendation to Groups. In *The Adaptive Web*, 2007.

[220] P. Jayachandran, K. Tunga, N. Kamat, and A. Nandi. Combining User Interaction, Speculative Query Execution and Sampling in the DICE System. *Proc. of the VLDB Endowment (PVLDB)*, 7(13), 2014.

[221] J. F. R. Jr., H. Tong, J. Pan, A. J. M. Traina, C. T. Jr., and C. Faloutsos. Large Graph Analysis in the GMine System. *IEEE Trans. Knowl. Data Eng.*, 25(1), 2013.

[222] J. F. R. Jr., H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec. GMine: A System for Scalable, Interactive Graph Visualization and Mining. In *Conference on Very Large Databases (VLDB)*.

[223] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. Faster Visual Analytics through Pixel-Perfect Aggregation. *VLDB Endowment (PVLDB)*, 7(13), 2014.

[224] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. VDDA: automatic visualization-driven data aggregation in relational databases. *Journal on Very Large Data Bases (VLDBJ)*, 2015.

[225] E. Kalampokis, A. Nikolov, P. Haase, R. Cyganiak, A. Stasiewicz, A. Karamanou, M. Zotou, D. Zeginis, E. Tambouris, and K. A. Tarabanis. Exploiting Linked Data Cubes with OpenCube Toolkit. In *Intl. Semantic Web Conference (ISWC)*, 2014.

[226] A. Kalinin, U. Çetintemel, and S. B. Zdonik. Interactive Data Exploration Using Semantic Windows. In *ACM Conference on Management of Data (SIGMOD)*, 2014.

[227] A. Kalinin, U. Çetintemel, and S. B. Zdonik. Searchlight: Enabling Integrated Search and Exploration over Large Multidimensional Data. *Proc. of the VLDB Endowment (PVLDB)*, 8(10), 2015.

[228] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *IEEE Conference on Data Engineering (ICDE)*, 2014.

[229] B. Kampgen and A. Harth. OLAP4LD - A Framework for Building Analysis Applications Over Governmental Statistics. In *Extended Semantic Web Conference (ESWC)*, 2014.

[230] R. Kannan, M. Ishteva, and H. Park. Bounded matrix factorization for recommender system. *Knowl. Inf. Syst.*, 39(3), 2014.

[231] G. Kappel, E. Kapsammer, and W. Retschitzegger. Integrating XML and Relational Database Systems. *World Wide Web Journal (WWWJ)*, 7:4, 2004.

[232] G. Karypis and V. Kumar. Multilevel Graph Partitioning Schemes. In *Intl. Conference on Parallel Processing (ICPP)*, 1995.

[233] A. Kashyap, V. Hristidis, M. Petropoulos, and S. Tavoulari. Effective Navigation of Query Results Based on Concept Hierarchies. *IEEE Trans. Knowl. Data Eng.*, 23(4), 2011.

[234] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. G. Giannopoulou. Ontology visualization methods - a survey. *ACM Comput. Surv.*, 39(4), 2007.

[235] J. Kay and W. Niu. Adapting Information Delivery to Groups of People. In *Workshop on New Technologies for Personalized Information Access*, 2005.

[236] H. A. Khan, M. A. Sharaf, and A. Albarrak. DivIDE: efficient diversification for interactive data exploration. In *Conference on Scientific and Statistical Database Management (SSDBM)*, 2014.

[237] W. Kießling. Foundations of Preferences in Database Systems. In *Intl. Conference on Very Large Databases (VLDB)*, 2002.

[238] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *Proc. of the VLDB Endowment (PVLDB)*, 8(5), 2015.

[239] J. K. Kim, H. K. Kim, H. Y. Oh, and Y. U. Ryu. A group recommendation system for online communities. *Int. Journal of Information Management*, 30(3), 2010.

[240] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Journal of Web Semantics*, 2(1), 2004.

[241] N. Kiyavitskaya, N. Zeni, J. R. Cordy, L. Mich, and J. Mylopoulos. Cerno: Lightweight tool support for semantic annotation of textual documents. *Data & Knowl. Eng. (DKE)*, 68(12), 2009.

[242] J. Klimek, J. Helmich, and M. Necasky. Payola: Collaborative Linked Data Analysis and Visualization Framework. In *Extended Semantic Web Conference (ESWC)*, 2013.

[243] J. Klimek, J. Helmich, and M. Necasky. Use cases for linked data visualization model. In *Workshop on Linked Data on the Web, (LDOW)*, 2015.

[244] I. Koffina, G. Serfiotis, V. Christophides, and V. Tannen. Mediating RDF/S Queries to Relational and XML Sources. *Intl. J. Semantic Web Inf. Syst. (IJSWIS)*, 2(4), 2006.

[245] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Intl. Conference on Very Large Databases (VLDB)*, 2002.

[246] G. Koutrika and Y. E. Ioannidis. Personalization of Queries in Database Systems. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2004.

[247] S. Kriglstein and R. Motschnig-Pitrik. Knoocks: New Visualization Approach for Ontologies. In *Conference on Information Visualisation*, 2008.

[248] K. Krishnamoorthy, R. Kumar, and S. R. Dua. Converting SPARQL Queries to SQL Queries. *Microsoft Corporation, U.S. Patent 7818352*, 2010.

[249] R. Krishnamurthy, R. Kaushik, and J. Naughton. XML-SQL Query Translation Literature: The State of the Art and Open Problems. In *Intl. XML Database Symposium (Xsym)*, 2003.

[250] S. Krivov, R. Williams, and F. Villa. GrOWL: A tool for visualization and editing of OWL ontologies. *J. Web Sem.*, 5(2), 2007.

[251] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of ACM (JACM)*, 22(4), 1975.

[252] M. Lacroix and P. Lavency. Preferences: Putting More Knowledge into Queries. In *Intl. Conference on Very Large Databases (VLDB)*, 1987.

[253] A. Lambert, R. Bourqui, and D. Auber. Winding Roads: Routing edges into bundles. *Comput. Graph. Forum*, 29(3), 2010.

[254] A. Langegger and W. Woss. RDFStats - An Extensible RDF Statistics Generator and Library. In *Database and Expert Systems Applications*, 2009.

[255] M. Lanzenberger, J. Sampson, and M. Rester. Visualization in Ontology Tools. In *Intl. Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, 2009.

[256] J. Lee and S.-w. Hwang. BSkyTree: scalable skyline computation using a balanced pivot selection. In *Intl. Conference on Extending Database Technology (EDBT)*, 2010.

[257] J. Lee, G. won You, S. won Hwang, J. Selke, and W.-T. Balke. Interactive skyline queries. *Inf. Sci.*, 211, 2012.

[258] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *Intl. Conference on Very Large Databases (VLDB)*, 2007.

[259] P. Lehti and P. Fankhauser. XML data integration with OWL: Experiences and challenges. In *Intl. Symposium on Applications and the Internet*, 2004.

[260] A. d. Leon, F. Wisniewki, B. Villazon-Terrazas, and O. Corcho. Map4rdf- Faceted Browser for Geospatial Datasets. In *Using Open Data: policy modeling, citizen empowerment, data journalism*, 2012.

[261] C. Li, G. Baciu, and Y. Wang. ModulGraph: Modularity-based Visualization of Massive Graphs. In *Visualization in High Performance Computing*, 2015.

[262] T. Liebig and O. Noppens. OntoTrack: A semantic approach for ontology authoring. *J. Web Sem.*, 3(2-3), 2005.

[263] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2007.

[264] Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. P. Chau. Demonstrating Interactive Multi-resolution Large Graph Exploration. In *IEEE Conference on Data Mining Workshops*, 2013.

[265] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Trans. Vis. Comput. Graph.*, 19(12), 2013.

[266] B. Liu and C.-Y. Chan. ZINC: Efficient Indexing for Skyline Computation. *VLDB Endowment*, 4(3), 2010.

[267] C. Liu, M. Vincent, and J. Liu. Constraint Preserving Transformation from Relational Schema to XML Schema. *World Wide Web Journal (WWWJ)*, 9:1, 2006.

[268] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time Visual Querying of Big Data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[269] C. Lofi and W.-T. Balke. On Skyline Queries and How to Choose from Pareto Sets. In *Advanced Query Processing (1)*. 2013.

[270] S. Lohmann, S. Negru, F. Haag, and T. Ertl. VOWL 2: User-Oriented Visualization of Ontologies. In *Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2014.

[271] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing Ontologies with VOWL. *Semantic Web Journal*, 2015.

[272] H. Lu, C. S. Jensen, and Z. Zhang. Flexible and Efficient Resolution of Skyline Query Size Constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(7), 2011.

[273] K. M. Interpreting XML via an RDF schema. In *Intl. Workshop on Electronicy Business Hubs*, 2002.

[274] L. M. Data Integration: A Theoretical Perspective. In *Symposium on Principles of Database Systems (PODS)*, 2002.

[275] L. Ma, C. Wang, J. Lu, F. Cao, Y. Pan, and Y. Yu. Effective and Efficient Semantic Web Data Management over DB2. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2008.

[276] K. Makris, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL-RW: Transparent Query Access over Mapped RDF Data Sources. In *Intl. Conference on Extending Database Technology (EDBT)*, 2012.

[277] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources. In *Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2010.

[278] C. Mangold. A survey and classification of semantic search approaches. *Int. J. Metadata Semantics and Ontology*, 2(1), 2007.

[279] F. Manola and M. E., editors. *RDF Primer*. W3C Rec., 2004. `www.w3.org/TR/rdf-primer`.

[280] S. Mansmann and M. H. Scholl. Exploring OLAP aggregates with hierarchical visualization techniques. In *ACM Symposium on Applied Computing (SAC)*, 2007.

[281] M. Maragkakis, M. Reczko, V. A. Simossis, P. Alexiou, G. L. Papadopoulos, T. Dalamagas, G. Giannopoulos, G. Goumas, E. Koukis, K. Kourtis, T. Vergoulis, N. Koziris, T. Sellis, P. Tsanakas, and A. G. Hatzigeorgiou. Diana-microt web server: elucidating microrna functions through target prediction. *Nucleic Acids Research*, 37(suppl 2), 2009.

[282] N. Marie and F. L. Gandon. Survey of Linked Data Based Exploration Systems. In *Workshop on Intelligent Exploration of Semantic Data (IESD)*, 2014.

[283] J. Masthoff. Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *User Model. User-Adapt. Interact.*, 14(1), 2004.

[284] J. Masthoff. Group Recommender Systems: Combining Individual Models. In *Recommender Systems Handbook*. 2011.

[285] J. F. McCarthy. Pocket Restaurant Finder: A situated recommender systems for groups. In *Workshop on Mobile Ad-Hoc Communication*, 2002.

[286] J. F. McCarthy and T. D. Anagnost. MusicFX: An Arbiter of Group Preferences for Computer Aupported Collaborative Workouts. In *ACM Conference on Computer Supported Cooperative Work*, 1998.

[287] K. McCarthy, L. McGinty, and B. Smyth. Case-Based Group Recommendation: Compromising for Success. In *Int. Conference on Case-Based Reasoning, ICCBR*, 2007.

[288] K. McCarthy, M. Salamó, L. Coyle, L. McGinty, B. Smyth, and P. Nixon. CATS: A Synchronous Approach to Collaborative Group Recommendation. In *Florida Artificial Intelligence Research Society Conference*, 2006.

[289] D. L. McGuinness and van Harmelen F., editors. *OWL Web Ontology Language: Overview*. W3C Rec., 2004. `www.w3.org/TR/owl-features`.

[290] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *Intl. Conference on Very Large Databases (VLDB)*, 2000.

[291] T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[292] M. H. Montague and J. A. Aslam. Condorcet Fusion for Improved Retrieval. In *Intl. Conference on Information and Knowledge Management*, 2002.

[293] M. D. Morse, J. M. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *Intl. Conference on Very Large Databases (VLDB)*, 2007.

[294] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the Data Enthusiast: Challenges for Next-Generation Data-Analysis Systems. *VLDB Endowment (PVLDB)*, 7(6), 2014.

[295] E. Motta, P. Mulholland, S. Peroni, M. d'Aquin, J. M. Gomez-Perez, V. Mendez, and F. Zablith. A Novel Approach to Visualizing and Navigating Ontologies. In *Intl. Semantic Web Conference (ISWC)*, 2011.

[296] E. K. Neumann and D. Quan. Biodash: A semantic web dashboard for drug development. In *Pacific Symposium on Biocomputing*, 2006.

[297] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast Group Recommendations by Applying User Clustering. In *Intl. Conference on Conceptual Modeling (ER)*, 2012.

[298] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl. PolyLens: A recommender system for groups of user. In *European Conference on Computer Supported Cooperative Work, ECSCW*, 2001.

[299] D. P, P. M. Deshpande, D. Majumdar, and R. Krishnapuram. Efficient skyline retrieval with arbitrary similarity measures. In *Intl. Conference on Extending Database Technology (EDBT)*, 2009.

[300] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1), 2005.

[301] M.-H. Park, H.-S. Park, and S.-B. Cho. Restaurant Recommendation for Group of People in Mobile Environments Using Probabilistic Multi-criteria Decision Making. In *Asia Pacific Conference on Computer Human Interaction*, 2008.

[302] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-Aware Sampling for Very Large Databases. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2016.

[303] H. Paulheim. Generating Possible Interpretations for Statistics from Linked Open Data. In *Extended Semantic Web Conference (ESWC)*, 2012.

[304] J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34:3, 2009.

[305] I. Petrou, M. Meimaris, and G. Papastefanatos. Towards a methodology for publishing Linked Open Statistical Data. *eJournal of eDemocracy & Open Government*, 6(1), 2014.

[306] D. Phan, L. Xiao, R. B. Yeh, P. Hanrahan, and T. Winograd. Flow Map Layout. In *IEEE Symposium on Information Visualization (InfoVis)*, 2005.

[307] F. Picalausa and S. Vansummeren. What are real SPARQL queries like? In *Intl. Workshop on Semantic Web Information Management*, 2011.

[308] E. Pietriga. IsaViz: a Visual Environment for Browsing and Authoring RDF Models. In *World Wide Web Conference (WWW)*, 2002.

[309] A. Piliponyte, F. Ricci, and J. Koschwitz. Sequential Music Recommendations for Groups by Balancing User Satisfaction. In *User Modeling, Adaptation, and Personalization*, 2013.

[310] S. Pizzutilo, B. De Carolis, G. Cozzolongo, and F. Ambruoso. Group Modeling in a Public Space: Methods, Techniques, Experiences. In *Int. Conference on Applied Informatics and Communications*, 2005.

[311] A. Polleres. From SPARQL to Rules (and back). In *World Wide Web Conference (WWW)*, 2007.

[312] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *Intl. Conference on Very Large Databases (VLDB)*, 2000.

[313] N. Popitsch and B. Haslhofer. Dsnotify: handling broken links in the web of data. In *World Wide Web Conference (WWW)*, 2010.

[314] E. Prud'hommeaux and S. A., editors. *SPARQL Query Language for RDF*. W3C Rec., 2008. `www.w3.org/TR/rdf-sparql-query`.

[315] D. A. Quan and R. Karger. How to make a semantic web browser. In *World Wide Web Conference (WWW)*, 2004.

[316] C. R. A relational algebra for SPARQL. Technical report, Hewlett-Packard Laboratories, 2005. `www.hpl.hp.com/techreports/2005/HPL-2005-170.html`.

[317] L. Reeve and H. Han. Survey of semantic annotation platforms. ACM Symposium on Applied Computing (SAC), 2005.

[318] G. Reif, M. Jazayeri, and H. Gall. Towards semantic Web Engineering: WEESA-mapping XML schema to ontologies. In *Workshop on Application Design (WWW2004)*, 2004.

[319] W. H. Riker. *Liberalism Against Populism.* Waveland Press Inc, 1988.

[320] P. Ristoski, C. Bizer, and H. Paulheim. Mining the Web of Linked Data with RapidMiner. In *Intl. Semantic Web Conference (ISWC)*, 2014.

[321] P. Ristoski and H. Paulheim. Visual Analysis of Statistical Data on Maps using Linked Open Data. In *Extended Semantic Web Conference (ESWC)*, 2015.

[322] J. Robie, D. Chamberlin, and et al., editors. *XQuery Update Facility 1.0" W3C Rec.* 2011. `www.w3.org/TR/xquery-update-10`.

[323] T. Rodrigues, P. Rosa, and J. Cardoso. Mapping XML to Exiting OWL ontologies. In *Intl. Conference WWW/Internet*, 2006.

[324] T. Rodrigues, P. Rosa, and J. Cardoso. Moving from syntactic to semantic organizations using JXML2OWL. *Computers in Industry*, 59:8, 2008.

[325] M. Rodriguez-Muro, J. Hardi, and D. Calvanese. Quest: Effcient SPARQL-to-SQL for RDF and OWL. In *Intl. Semantic Web Conference (ISWC)*, 2012.

[326] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Space efficiency in group recommendation. *VLDB J.*, 19(6), 2010.

[327] L. Rutledge, J. van Ossenbruggen, and L. Hardman. Making RDF presentable: integrated global and local semantic Web browsing. In *World Wide Web Conference (WWW)*, 2005.

[328] B. S. Gloze: XML to RDF and back again. In *Jena User Conference*, 2006.

[329] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically Sorted Skylines for Partially Ordered Domains. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2009.

[330] S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau, S. Auer, J. Sequeda, and A. Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. Technical report, Technical Report. RDB2RDF W3C Working Group, 2009.

[331] P. E. R. Salas, F. M. D. Mota, K. K. Breitman, M. A. Casanova, M. Martin, and S. Auer. Publishing Statistical Data on the Web. *Intl. J. Semantic Computing*, 6(4), 2012.

[332] A. D. Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized Multi-pass Streaming Skyline Algorithms. *VLDB Endowment*, 2(1), 2009.

[333] C. Sayers. Node-centric RDF Graph Visualization, 2004. Technical Report HP Laboratories.

[334] S. Schenk, P. Gearon, and P. A., editors. *SPARQL 1.1 Update.* W3C Rec., 2013. `www.w3.org/TR/sparql11-update`.

[335] K. Schlegel, T. Weißgerber, F. Stegmaier, C. Seifert, M. Granitzer, and H. Kosch. Balloon Synopsis: A Modern Node-Centric RDF Viewer and Browser for the Web. In *Extended Semantic Web Conference (ESWC)*, 2014.

[336] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL Performance Benchmark. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2009.

[337] J. Sequeda, R. Depena, and D. Miranker. Ultrawrap: Using SQL Views for RDB2RDF. In *Intl. Semantic Web Conference (ISWC)*, 2009.

[338] H. Shang and M. Kitsuregawa. Skyline Operator on Anti-correlated Distributions. *VLDB Endowment*, 6(9), 2013.

[339] C. Shen and Y. Chen. A dynamic-programming algorithm for hierarchical discretization of continuous attributes. *European Journal of Operational Research*, 184(2), 2008.

[340] C. Sheng and Y. Tao. On finding skylines in external memory. In *Symposium on Principles of Database Systems*, 2011.

[341] C. Sheng and Y. Tao. Worst-Case I/O-Efficient Skyline Algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4), 2012.

[342] B. Shneiderman. Tree Visualization with Tree-Maps: 2-d Space-Filling Approach. *ACM Trans. Graph.*, 11(1), 1992.

[343] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *IEEE Symposium on Visual Languages*, 1996.

[344] B. Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *ACM Conference on Management of Data (SIGMOD)*, 2008.

[345] B. Shneiderman and M. Wattenberg. Ordered Treemap Layouts. In *IEEE Symposium on Information Visualization (INFOVIS)*, 2001.

[346] J. Siméon and C. D., editors. *XQuery 1.0: an XML Query Language*. W3C Rec., 2007. `www.w3.org/TR/xquery`.

[347] M. G. Skjæveland. Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets. In *Extended Semantic Web Conference (ESWC)*, 2012.

[348] A. K. Smith, K.-H. Cheung, K. Y. Yip, M. H. Schultz, and M. Gerstein. Linkhub: a semantic web system that facilitates cross-database queries and information retrieval in proteomics. *BMC Bioinformatics*, 8(S-3), 2007.

[349] D. E. Spanos, P. Stavrou, and N. Mitrou. Bringing relational databases into the Semantic Web: A survey. *Semantic Web Journal*, 3:2, 2012.

[350] D. W. Sprague, F. Wu, and M. Tory. Music selection using the PartyVote democratic jukebox. In *Working Conference on Advanced Visual Interfaces*, 2008.

[351] C. Stadler, J. Lehmann, K. Hoffner, and S. Auer. LinkedGeoData: A core for a web of spatial open data. *Semantic Web*, 3(4), 2012.

[352] C. Stadler, M. Martin, and S. Auer. Exploring the web of spatial data with facete. In *World Wide Web Conference (WWW)*, 2014.

[353] I. Stavrakantonakis, C. Tsinaraki, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL2XQuery 2.0: Supporting Semantic-based queries over XML data. In *Intl. Workshop on Semantic Media Adaptation and Personalization (SMAP)*, 2010.

[354] I. Stavrakantonakis, C. Tsinaraki, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL2XQuery 2.0: Supporting Semantic-based Queries over XML Data. In *Workshop on Semantic Media Adaptation and Personalization (SMAP)*, 2010.

[355] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Transactions on Database Systems (TODS)*, 36(3), 2011.

[356] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using Polaris. In *ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2002.

[357] M. D. Storey, N. F. Noy, M. A. Musen, C. Best, R. W. Fergerson, and N. A. Ernst. Jambalaya: an interactive environment for exploring ontologies. In *IUI*, 2002.

[358] M. Stuhr, D. Roman, and D. Norheim. LODWheel - JavaScript-based Visualization of RDF Data. In *Workshop on Consuming Linked Data*, 2011.

[359] S. Sundara, M. Atre, V. Kolovski, S. Das, Z. Wu, E. I. Chong, and J. Srinivasan. Visualizing large-scale RDF data using Subsets, Summaries, and Sampling in Oracle. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2010.

[360] B. T. and et al., editors. *Extensible Markup Language (XML) 1.1*. W3C Rec., 2006. `www.w3.org/TR/xml11`.

[361] M. Tallis. SemanticWord processing for content authors. In *Knowledge Markup and Semantic Annotation Workshop*, 2003.

[362] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *Intl. Conference on Very Large Databases (VLDB)*, 2001.

[363] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-Based Representative Skyline. In *IEEE Intl. Conference on Data Engineering (ICDE)*, 2009.

[364] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2004.

[365] F. Tauheed, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. SCOUT: Prefetching for Latent Feature Following Queries. *Proc. of the VLDB Endowment (PVLDB)*, 5(11), 2012.

[366] A. D. Taylor. *Social choice and the mathematics of manipulation*. Cambridge University Press, 2005.

[367] K. Techapichetvanich and A. Datta. Interactive Visualization for OLAP. In *Computational Science and Its Applications (ICCSA)*, 2005.

[368] K. Thellmann, M. Galkin, F. Orlandi, and S. Auer. Linkdaviz - automatic binding of linked data to visualizations. In *Intl. Semantic Web Conference (ISWC)*, 2015.

[369] P. Thiran, F. Estievenart, J. L. Hainaut, and G. J. Houben. A Generic Framework for Extracting XML Data from Legacy Databases. *J. Web Eng. (JWE)*, 4:3, 2005.

[370] P. T. T. Thuy, Y. K. Lee, and S. Lee. DTD2OWL: automatic transforming XML documents into OWL ontology. In *Interaction Sciences Conference*, 2009.

[371] P. T. T. Thuy, Y. K. Lee, S. Lee, and B. S. Jeong. Transforming Valid XML Documents into RDF via RDF Schema. In *Intl. Conference on Next Generation Web Services Practices*, 2007.

[372] P. T. T. Thuy, Y. K. Lee, S. Lee, and B. S. Jeong. Exploiting XML Schema for Interpreting XML Documents as RDF. In *Intl. Conference on Services Computing*, 2008.

[373] C. Tominski, J. Abello, and H. Schumann. CGV - An interactive graph visualization system. *Computers & Graphics*, 33(6), 2009.

[374] G. Tschinkel, E. E. Veas, B. Mutlu, and V. Sabol. Using Semantics for Interactive Visual Analysis of Linked Open Data. In *Intl. Semantic Web Conference (ISWC)*, 2014.

[375] C. Tsinaraki and S. Christodoulakis. Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools. In *Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2007.

[376] C. Tsinaraki and S. Christodoulakis. Support for Interoperability between OWL based and XML Schema based Applications. In *DELOS Conference II*, 2007.

[377] J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, and S. Decker. Towards dataset dynamics: Change frequency of linked open data sources. In *Intl. Workshop on Linked Data on the Web, (LDOW)*, 2010.

[378] J. Umbrich, B. Villazo'n-Terrazas, and M. Hausenblas. Dataset dynamics compendium: A comparative study. In *Intl. Workshop on Consuming Linked Data (COLD)*, 2010.

[379] J. Unbehauen, C. Stadler, and S. Auer. *Accessing Relational Data on the Web with SparqlMap.* 2012.

[380] V. S. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Journal of Web Semantics*, 4, 2006.

[381] F. Valsecchi, M. Abrate, C. Bacciu, M. Tesconi, and A. Marchetti. Dbpedia atlas: Mapping the uncharted lands of linked data. In *Workshop on Linked Data on the Web, LDOW*, 2015.

[382] F. Valsecchi and M. Ronchetti. Spacetime: a two dimensions search and visualisation engine based on linked data. In *Conference on Advances in Semantic Processing (SEMAPRO)*, 2014.

[383] J. J. van Wijk and H. van de Wetering. Cushion Treemaps: Visualization of Hierarchical Information. In *IEEE Symposium on Information Visualization (INFOVIS)*, 1999.

[384] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-automatic and Automatic Support for Semantic Markup. In *Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2002.

[385] M. Vartak, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: Automatically Generating Query Visualizations. *VLDB Endowment (PVLDB)*, 7(13), 2014.

[386] E. Vildjiounaite, V. Kyllönen, T. Hannula, and P. Alahuhta. Unobtrusive dynamic modelling of TV programme preferences in a Finnish household. *Multimedia Syst.*, 15(3), 2009.

[387] M. Voigt, S. Pietschmann, L. Grammel, and K. Meißner. Context-aware recommendation of visualization components. In *Conference on Information, Process, and Knowledge Management (eKNOW)*, 2012.

[388] M. Voigt, S. Pietschmann, and K. Meißner. A Semantics-Based, End-User-Centered Information Visualization Process for Semantic Web Data. In *Semantic Models for Adaptive Interactive Systems*. 2013.

[389] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *Intl. Semantic Web Conference (ISWC)*, 2009.

[390] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huebner. Ontology-based integration of information - A survey of existing approaches. In *Workshop on Ontologies and Information Sharing*, 2001.

[391] T. D. Wang and B. Parsia. CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In *Intl. Semantic Web Conference (ISWC)*, 2006.

[392] M. O. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *IEEE Visualization*, 1994.

[393] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. Technical report, 2013.

[394] R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu. Efficient skyline querying with variable user preferences on nominal attributes. *VLDB Endowment*, 1(1), 2008.

[395] E. Wu, L. Battle, and S. R. Madden. The Case for Data Visualization Management Systems. *VLDB Endowment (PVLDB)*, 7(10), 2014.

[396] H. Xiao and I. Cruz. RDF-based metadata management in peer-to- peer systems. In *IST MMGPS Workshop*, 2004.

[397] H. Xiao and I. Cruz. Integrating and Exchanging XML Data Using Ontologies. *Journal on Data Semantics VI*, 2006.

[398] M. L. Yiu and N. Mamoulis. Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. In *Intl. Conference on Very Large Databases (VLDB)*, 2007.

[399] C. Yu and L. Popa. Constraint-based XML Query Rewriting for Data Integration. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2004.

[400] H. Yu, C. Hsieh, S. Si, and I. S. Dhillon. Parallel matrix factorization for recommender systems. *Knowl. Inf. Syst.*, 41(3), 2014.

[401] Z. Yu, X. Zhou, Y. Hao, and J. Gu. TV Program Recommendation for Multiple Viewers Based on user Profile Merging. *User Model. User-Adapt. Interact.*, 16(1), 2006.

[402] A. Zaveri, A. M. Anisa Rula, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment methodologies for linked open data. *Semantic Web Journal (to appear)*, 2015.

[403] K. Zhang, H. Wang, D. T. Tran, and Y. Yu. ZoomRDF: semantic fisheye zooming on RDF data. In *World Wide Web Conference (WWW)*, 2010.

[404] S. Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2009.

[405] S. Zhang, N. Mamoulis, B. Kao, and D. W.-L. Cheung. Efficient Skyline Evaluation over Partially Ordered Domains. *VLDB Endowment*, 3(1), 2010.

[406] Y. Zhiwen, Z. Xingshe, and Z. Daqing. An adaptive in-vehicle multimedia recommender for group users. In *IEEE Vehicular Technology Conference*, 2005.

[407] M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt. Interactive Level-of-Detail Rendering of Large Graphs. *IEEE Trans. Vis. Comput. Graph.*, 18(12), 2012.

[408] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *ACM Conference on Management of Data (SIGMOD)*, 2014.