



HAL
open science

Flexible querying of RDF databases: a contribution based on fuzzy logic

Olfa Slama

► **To cite this version:**

Olfa Slama. Flexible querying of RDF databases: a contribution based on fuzzy logic. Databases [cs.DB]. Université de Rennes, 2017. English. NNT : 2017REN1S089 . tel-01749470

HAL Id: tel-01749470

<https://theses.hal.science/tel-01749470v1>

Submitted on 29 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale MathSTIC

présentée par

Olfa SLAMA

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Système Aléatoires
École Nationale Supérieure des Sciences Appliquées et de
Technologie

**Flexible
Querying of
RDF Databases :
A Contribution
Based on
Fuzzy Logic**

**Thèse à soutenir à Lannion
le 22 Novembre 2017**

devant le jury composé de :

Salima BENBERNOU

Professeur, Université de Paris Descartes / *Rapporteur*

Anne LAURENT

Professeur, Université de Montpellier 2 / *Rapporteur*

David GROSS-AMBLARD

Professeur, Université de Rennes 1 / *Examineur*

Dominique LAURENT

Professeur, Université de Cergy-Pontoise / *Examineur*

Olivier PIVERT

Professeur, Université de Rennes 1 / *Directeur de thèse*

Virginie THION

Maître de conférences, Université de Rennes 1 /
Co-directrice de thèse

Keep your eyes on the stars, and your feet on the ground.

Theodore Roosevelt

Contents

Résumé en français	5
Introduction	13
1 Background notions	21
Introduction	21
1.1 The RDF Graph Data Model	21
1.2 SPARQL: Crisp Querying of RDF data	25
1.3 Fuzzy Set Theory	32
1.3.1 Definition	33
1.3.2 Characteristics of a Fuzzy Set	35
1.3.3 Operations on Fuzzy Sets	36
Conclusion	39
2 State of the art: Flexible Querying of RDF data	41
Introduction	41
2.1 Preference Queries on RDF Data	42
2.1.1 Quantitative Approaches	42
2.1.2 Qualitative Approaches: Skyline-based Approaches	49
2.2 Query Relaxation	53
2.3 Approximate Matching	57
Conclusion and Summary	58
3 FURQL: An extension of SPARQL with Fuzzy Navigational Capabilities	61
Introduction	61
3.1 Fuzzy RDF (F-RDF) Graph	62
3.2 FUZZY RDF Query Language (FURQL)	67
3.2.1 Syntax of FURQL	68
3.2.2 Semantics of FURQL	71
Conclusion	78

4	FURQL with Fuzzy Quantified Statements to Fuzzy RDF Databases	79
	Introduction	79
4.1	Refresher on Fuzzy Quantified Statements	80
4.1.1	Fuzzy Quantifiers	80
4.1.2	Interpretation of Fuzzy Quantified Statements	81
4.2	FURQL with Fuzzy Quantified Statements	86
4.2.1	Related Work: Quantified Statements in SPARQL	87
4.2.2	Fuzzy Quantified Statements in FURQL	87
	Conclusion	95
5	Implementation and Experimentations	97
	Introduction	97
5.1	Implementation of FURQL	98
5.1.1	Storage of Fuzzy RDF Graphs	98
5.1.2	Evaluation of FURQL Queries	99
5.2	Experimentations	102
5.2.1	Experimental Setup	103
5.2.2	Experiments for nonquantified FURQL Queries	103
5.2.3	Experiments for Quantified FURQL Queries	109
	Conclusion	112
6	Extensions to General Graph Databases	115
	Introduction	115
6.1	Background Notions	116
6.1.1	Graph Databases	116
6.1.2	Fuzzy Graphs	117
6.1.3	Fuzzy Graph Databases	120
6.1.4	The FUDGE Query Language	120
6.2	Related Work	122
6.3	Fuzzy Quantified Statements in FUDGE	124
6.3.1	Syntax of a Fuzzy Quantified Query	125
6.3.2	Evaluation of a Fuzzy Quantified Query	126
6.4	About Query Processing	131
6.5	Experimental Results	133
	Conclusion	137
	Conclusion	139
	List of Figures	143

CONTENTS	3
<hr/>	
List of Tables	145
Appendix A Sample of Queries	147
Bibliography	152

Résumé en français

La publication de données ouvertes (éventuellement liées) sur le web est un phénomène en pleine expansion. L'étude des modèles et langages permettant l'exploitation de ces données s'est donc grandement intensifiée ces dernières années.

Récemment, le modèle RDF (Resource Description Framework) s'est imposé comme le modèle de données standard, proposé par le W3C, pour représenter des données du web sémantique [W3C, 2014]. RDF est un cas particulier de graphe étiqueté orienté, dans lequel chaque arc étiqueté (représentant un prédicat) relie un sujet à un objet.

SPARQL [Prud'hommeaux and Seaborne, 2008] est le langage de requête standard recommandé par le W3C pour l'interrogation de données RDF. Il s'agit d'un langage fondé sur la mise en correspondance de patrons de graphe.

Les travaux que nous présentons visent à introduire plus de flexibilité dans le langage (SPARQL ici) en offrant la possibilité d'intégrer des préférences utilisateur aux requêtes.

Les motivations pour intégrer les préférences des utilisateurs dans les requêtes de base de données sont multiples. Tout d'abord, il semble souhaitable d'offrir à l'utilisateur la possibilité d'exprimer des requêtes dont la forme se rapproche, autant que possible, de la formulation de la requête en langage naturel. Ensuite, l'introduction de préférences utilisateur dans une requête permet d'obtenir un classement des réponses, par niveau décroissant de satisfaction, ce qui est très utile en cas d'obtention d'un grand nombre de réponses. Et enfin, là où une requête booléenne classique peut ne retourner aucune réponse, une version à préférence (qui peut être vue comme une version relaxée et donc moins restrictive), peut permettre de produire des réponses proches des objets idéals visés.

L'introduction de préférences utilisateur dans les requêtes a donné lieu à de nombreux travaux de recherche ces dernières décennies dans le contexte du modèle relationnel de bases de données [Bruno et al., 2002, Chomicki, 2002, Torlone and Ciaccia, 2002, Borzsony et al., 2001, Kiefling, 2002, Tahani, 1977, Bosc and Pivert, 1995, Pivert and Bosc, 2012]. La littérature sur les requêtes à préférences dans le contexte de bases de données RDF n'est pas aussi abondante puisque cette question n'a commencé à attirer l'attention que récemment. La plupart des approches

existantes sont des adaptations directes des propositions faites dans le contexte des bases de données relationnelles. En particulier, elles se limitent à l'expression de préférences sur les valeurs présentes dans les nœuds.

Dans un contexte de graphe RDF, la nécessité d'exprimer des conditions sur la structure des données, puis d'extraire les relations entre les ressources dans le graphe RDF, a motivé des travaux visant à étendre SPARQL et à le rendre plus expressif. Dans [Kochut and Janik, 2007, Anyanwu et al., 2007, Alkhateeb et al., 2009] et [Pérez et al., 2010], les auteurs étendent principalement SPARQL en permettant d'interroger RDF à l'aide de patrons de graphe en utilisant des expressions régulières. Mais dans ces approches, le graphe RDF et les conditions de recherche restent non-flous (booléens).

Le modèle RDF de base ne permet en effet de représenter nativement que des données de nature booléenne. Les concepts du monde réel à manipuler sont cependant souvent de nature graduelle. Il est donc nécessaire de disposer d'un langage plus flexible qui prenne en compte des graphes RDF dans lesquels les données sont intrinsèquement décrites de façon pondérée. Les poids peuvent représenter des notions graduelles telles qu'une intensité ou un coût. Par exemple, une personne peut être l'amie d'une autre avec un degré fonction de l'intensité de la relation d'amitié.

Afin de représenter ces informations, plusieurs auteurs ont proposé des extensions floues du modèle de données RDF. Cependant, les extensions floues de SPARQL qui peuvent être trouvées dans la littérature restent très limitées en termes d'expression de préférences.

Notre objectif dans cette thèse est de définir un langage de requête beaucoup plus expressif pour i) traiter des bases de données RDF floues et non floues et ii) exprimer des préférences complexes sur les valeurs des noeuds et sur la structure du graphe. Un exemple d'une telle requête est: trouver les acteurs *a* tels que la *plupart* des films *récents* où a joué l'acteur *a*, sont *bien* notés et ont été recommandés par un ami *proche* de *a*. Nos contributions principales sont décrites dans la suite.

Une extension floue de SPARQL avec des capacités de navigation floue

Notre objectif dans la première contribution est d'étendre le langage SPARQL de façon à lui permettre d'exprimer des préférences utilisateur pour exprimer des requêtes flexibles, portant sur des données RDF véhiculant ou non des notions graduelles.

Tout d'abord, nous proposons une extension de la notion de *patron de graphe*, fondée sur la théorie des ensembles flous, que l'on nomme *patron flou de graphe*. Cette extension repose sur celle de *patron de graphe SPARQL* introduite dans [Pérez et al., 2009] et [Arenas and Pérez, 2011]. Dans ces travaux, les auteurs définissent un *patron de graphe*

SPARQL dans un formalisme algébrique plus traditionnel que le formalisme introduit dans la norme officielle. Un *patron de graphe* est récursivement défini comme étant soit un graphe contenant des variables, soit un graphe complexe obtenu par l'application d'opérations sur *des patrons de graphe*.

Ensuite, on nous fondant sur cette notion de *patron flou de graphe*, nous proposons le langage FURQL qui est plus expressif que toutes les propositions existantes de la littérature, et qui permet:

1. d'interroger un **modèle de données RDF flou** dans lequel les triplets sont porteurs de notions graduelles (dont le modèle RDF non flou est un cas particulier), et
2. d'exprimer des **préférences floues** portant non seulement sur les données mais également sur la *structure* du graphe, que celui-ci soit flou ou non.

Modèle RDF flou Dans cette thèse, nous considérons un modèle de données, appelée F-RDF, qui synthétise les modèles RDF flous de la littérature ([Mazzieri and Dragoni, 2005, Udreă et al., 2006, Mazzieri and Dragoni, 2008, Lv et al., 2008, Straccia, 2009, Udreă et al., 2010, Zimmermann et al., 2012]), dont le principe commun consiste à ajouter un degré dans $[0, 1]$ à chaque triplet RDF, formalisé ou bien par l'encapsulation d'un degré flou dans chaque triplet ou bien par l'ajout au modèle d'une fonction associant un degré de satisfaction à chaque triplet (ces deux représentations sont sémantiquement équivalentes et présentent la même expressivité). Un degré attaché à un triplet $\langle s, p, o \rangle$ exprime à quel point l'objet o satisfait la propriété p sur le sujet s . Par exemple, le triplet flou $\langle \text{Beyonce}, \text{recommande}, \text{Euphoria} \rangle$ auquel est attaché le degré 0.8 indique que $\langle \text{Beyonce}, \text{recommande}, \text{Euphoria} \rangle$ est satisfait au niveau 0.8, ce qui peut être interprété comme *Beyonce recommande fortement Euphoria*.

Les degrés flous peuvent être donnés ou calculés, matérialisés ou non. Dans sa forme la plus simple, un degré peut correspondre au calcul d'une notion statistique reflétant l'intensité de la relation à laquelle le degré est attaché. Par exemple, l'intensité d'une relation d'amitié d'une personne p_1 vers une autre personne p_2 peut être calculée par la proportion d'amis communs par rapport au nombre total d'amis de p_1 .

Préférences floues Le langage FURQL est basé sur des *patrons flous de graphe* qui permettent d'exprimer des préférences floues sur les données d'un graphe flou F-RDF *via* des conditions floues (par exemple, l'année de publication d'un film est *récente*) et sur sa structure *via* des expressions régulières floues (par exemple, le chemin entre deux amis doit être *court*).

Syntaxiquement, le langage FURQL permet d'utiliser des *patrons flous de graphe* dans la clause **WHERE** et des conditions floues dans la clause **FILTER**. La syntaxe d'une ex-

pression floue de graphe est proche de celle de chemin, comme défini dans SPARQL 1.1 [Harris and Seaborne, 2013], permettant d'exhiber des nœuds reliés par des chemins exprimés sous forme d'une expression régulière. On permet ici l'expression d'une propriété floue portant sur les nœuds reliés. Une propriété d'un chemin concerne des notions classiques de la théorie des graphes flous [Rosenfeld, 2014] : la *distance* et la *force* de la connexion entre deux nœuds, où la *distance* entre deux noeuds est la longueur du plus court chemin entre ces deux noeuds et la *distance* d'un chemin est définie comme étant le poids de l'arc le plus faible du chemin.

Ce travail a été publié dans les actes de la 25ème Conférence internationale IEEE sur les systèmes flous (Fuzz-IEEE 16), Vancouver, Canada, 2016.

Requêtes quantifiées structurelles floues dans FURQL

La deuxième contribution traite de *requêtes quantifiées floues* adressées à une base de données RDF floue. Les *requêtes quantifiées floues* ont été étudiées de façon approfondie dans un contexte de bases de données relationnelles pour leur capacité à exprimer différents types de besoins d'information imprécis, voir notamment [Kacprzyk et al., 1989, Bosc et al., 1995], où elles servent à exprimer des conditions sur les valeurs des attributs des objets stockés.

Cependant, dans le cadre spécifique de RDF/SPARQL, les approches actuelles de la littérature traitant des *requêtes quantifiées* considèrent des quantificateurs non-flous uniquement [Bry et al., 2010, Fan et al., 2016] sur des données RDF non-floues.

Nous étudions une forme particulière de requête quantifiée floue structurelle et montrons comment elle peut être exprimée dans le langage FURQL défini précédemment. Plus précisément, nous considérons des *propositions quantifiées floues* du type “ QBX are A ” sur des bases de données RDF floues, où Q est le quantificateur qui est représenté par un ensemble flou et est soit relatif (par exemple, la plupart) soit absolu (par exemple, au moins trois), B est une condition floue, X est l'ensemble de noeuds dans le graphe RDF, et A désigne une condition floue. Un exemple d'une telle *proposition quantifiée floue* est : “la plupart des albums récents sont très bien notés”. Dans cet exemple, Q correspond au quantificateur flou relatif “la plupart”, B est la condition floue “être récent”, X correspond à l'ensemble des albums présents dans le graphe RDF et A correspond à la condition floue “être très bien noté”.

Conceptuellement, l'interprétation d'une telle *proposition quantifiée floue* dans une requête FURQL peut être basée sur l'une des approches de la littérature proposées dans [Zadeh, 1983, Yager, 1984, Yager, 1988]. Son évaluation comporte trois étapes:

1. la compilation de la requête quantifiée floue R en une requête non-floue R' ,
2. l'interprétation de la requête SPARQL R' ,
3. le calcul du résultat de R (qui est un ensemble flou) basé sur le résultat de R' .

Ce travail a été publié dans les actes de la 26ème Conférence internationale IEEE sur les systèmes flous (Fuzz-IEEE'17), Naples, Italie, 2017.

Mise en œuvre et expérimentation

Dans cette thèse, nous abordons également l'implantation du langage FURQL. Nous avons à cet effet considéré deux aspects:

1. le stockage de graphes flous (modèle de données étendu que nous considérons) et
2. l'évaluation de requêtes FURQL.

Le premier point peut être résolu par l'utilisation du mécanisme de *réification* qui permet d'attacher un degré flou à un triplet, solution proposée dans [Straccia, 2009].

Concernant l'évaluation de requêtes FURQL, nous avons développé une couche logicielle permettant la prise en compte de requêtes FURQL, que l'on associe à un moteur SPARQL standard. Cette couche logicielle, appelé SURF, est composée principalement des deux modules suivants:

- Dans une étape de prétraitement, un module de *compilateur de requête* FURQL produit
 - les fonctions dépendantes de la requête qui permettent de calculer les degrés de satisfaction pour chaque réponse retournée,
 - une requête SPARQL classique qui est ensuite envoyée au moteur de requête SPARQL pour récupérer les informations nécessaires pour calculer les degrés de satisfaction.

La compilation utilise le principe de dérivation introduit dans [Pivert and Bosc, 2012] dans un contexte de bases de données relationnelles qui consiste à “traduire” une requête floue en une requête non floue.

- Dans une étape de post-traitement, un module de *traitement des données floues* qui calcule le degré de satisfaction pour chaque réponse renvoyée, classe les réponses et les filtre qualitativement si une alpha-coupe a été spécifiée dans la requête floue initiale.

Une preuve de concept de l'approche proposée, le prototype SURF, est disponible et téléchargeable à l'adresse <https://www-shaman.irisa.fr/furql/>.

Pour évaluer les performances du prototype SURF que nous avons développé, nous avons effectué deux séries d'expériences sur différentes tailles de bases de données RDF floues. Les premières expériences visent à mesurer le coût supplémentaire induit par l'introduction du flou dans SPARQL, et les résultats obtenus montrent l'efficacité de notre proposition. Les deuxièmes expériences, qui concernent des *requêtes quantifiées floues*, montrent que le coût supplémentaire induit par la présence d'un quantificateur flou dans les requêtes reste très limité, même dans le cas de requêtes complexes.

Requêtes quantifiées structurelles floues dans FUDGE

A la fin de cette thèse, nous nous situons dans un cadre plus général: celui de bases de données graphe [Angles and Gutierrez, 2008]. Jusqu'à présent, une seule approche de la littérature, décrite dans [Castelltort and Laurent, 2014], considère des requêtes quantifiées floues dans un tel environnement, et seulement d'une manière assez limitée. Une limitation de cette approche tient au fait que seul le quantificateur est flou (alors qu'en général, dans une *proposition quantifiée floue* de la forme "*QBX are A*", les prédicats *A* et *B* peuvent également l'être).

Nous proposons quant à nous d'étudier *des requêtes quantifiées floues* impliquant des prédicats flous (en plus du quantificateur) sur des *bases de données graphe floues*. Nous considérons le même type de requête quantifiée floue structurelle que celui considéré dans FURQL mais dans un cadre plus général. Cette contribution est basée sur notre travail décrit dans [Pivert et al., 2016e], dans lequel nous avons montré comment il est possible d'intégrer ces *requêtes quantifiées floues* dans un langage nommé FUDGE, précédemment défini dans [Pivert et al., 2014a]. FUDGE est une extension floue de Cypher [Cypher, 2017] qui est un langage déclaratif pour l'interrogation des bases de données graphe classiques.

Une stratégie d'évaluation fondée sur un mécanisme de compilation qui dérive des requêtes classiques pour accéder aux données est également décrite. Elle s'appuie sur une surcouche logicielle au système Neo4j, baptisée SUGAR, dont une première version, décrite dans [Pivert et al., 2015, Pivert et al., 2016b], permet d'évaluer efficacement les requêtes FUDGE ne comportant pas de propositions quantifiées. A cet effet, nous avons mis à jour ce logiciel, qui est une couche logicielle qui implémente le langage FUDGE sur le SGBD Neo4j, pour lui permettre d'évaluer des requêtes FUDGE contenant des *conditions quantifiées floues*.

Comme preuve de concept de l'approche proposée, le prototype SUGAR est disponible et téléchargeable à l'adresse www-shaman.irisa.fr/fudge-prototype.

Afin de confirmer l'efficacité de l'approche proposée, nous avons effectué quelques expérimentations avec le prototype SUGAR en utilisant différentes tailles de bases de données graphe floues. Les résultats obtenus sont prometteurs et montrent que le coût du traitement de la

quantification floue dans une requête est très limité par rapport au coût de l'évaluation globale.

Conclusion & Perspectives

Cette thèse est la première proposant une extension floue du langage SPARQL visant à améliorer son expressivité et à permettre i) d'interroger des bases de données RDF floues et ii) d'exprimer des préférences complexes sur la valeur des données et sur la structure du graphe. Les résultats présentés dans ce manuscrit sont prometteurs et montrent que le coût supplémentaire dû à *l'introduction de conditions de recherche floues reste limité/acceptable*.

De nombreuses perspectives peuvent être envisagées. Une première perspective concerne l'extension des langages FURQL et FUDGE avec des préférences plus sophistiquées dont certaines font appel à des notions provenant du domaine de l'analyse des réseaux sociaux (centralité ou prestige d'un noeud) ou de la théorie des graphes (par exemple, clique, etc). Nous envisageons ensuite d'étudier d'autres types de requêtes quantifiées plus complexes, par exemple "trouver les auteurs ayant un article publié dans *la plupart* des revues de base de données *renommées*" (ou plus généralement, trouver les x tels que x est relié (par un chemin) à Q noeuds d'un type donné T satisfaisant la condition C). Les logiciels SURF et SUGAR peuvent également être améliorés afin de les rendre plus conviviaux, ce qui pose la question de l'élicitation de requêtes floues complexes. Il vaut également la peine d'étudier la manière dont notre cadre pourrait être appliqué à la gestion de **dimensions de qualité** des données (par exemple, précision, cohérence, etc.) qui sont en général d'une nature graduelle.

Introduction

THE relational model, introduced in 1970 by Edgar F. Codd [Codd, 1970], has been the most popular model for database management for many decades in academic, financial and commercial pursuits. In this framework, data can be stored and accessed thanks to a database management system like Oracle, Microsoft SQL Server, MySQL, etc.

However, in the recent decades, the traditional relational model faced new challenges, mainly related to the development of Internet. Data to be searched are more and more accessible on the Web (i.e., open environment) and never stop to increase in volume and complexity.

As a solution, an alternative model, called NoSQL (Not only Structured Query Language), came to existence and has attracted a lot of attention since 2007. It aims to process efficiently and store huge, distributed, and unstructured data such as documents, e-mail, multimedia and social media [Leavitt, 2010, Robinson et al., 2015].

Among NoSQL database systems, we may find the famous Google's BigTable [Chang et al., 2008], Facebook's Cassandra [Lakshman and Malik, 2009], Amazon's Dynamo [DeCandia et al., 2007], LinkedIn's Project Voldemort, Oracle's BerkeleyDB [Berkeley, 2010] and mostly Graph Databases Systems (e.g., Neo4j¹, Allegrograph², etc.), which are designed to store data in the form of a graph.

In the last decade, there has been increased attention in graphs to represent social networks, web site link structures, and others. Recently, database research has witnessed much interest in the W3C's Resource Description Framework (RDF) [W3C, 2014], which is a particular case of directed labeled graph, in which each labeled edge (called predicate) connects a subject to an object. It is considered to be the most appropriate knowledge representation language for representing, describing and storing information about resources available on the Web. This graph data model makes it possible to represent heterogeneous Web resources in a common and unified way, taking into consideration the semantic side of

¹<http://www.neo4j.org/>

²<http://franz.com/agraph/allegrograph/>

the information and the interconnectedness between entities. The SPARQL Protocol and RDF Query Language (SPARQL) [Prud'hommeaux and Seaborne, 2008] is the official W3C recommendation as an RDF query language. It plays the same role for the RDF data model as SQL does for the relational data model and provides basic functionalities (such as, union and optional queries, value filtering and ordering results, etc.) in order to query RDF data through graph patterns, i.e., RDF graphs containing variables data.

RDF data are usually composed of large heterogeneous data including various levels of quality e.g., over relevancy, trustworthiness, preciseness or timeliness of data (see [Zaveri et al., 2016]). It is then necessary to offer convenient query languages that improve the usability of such data. A solution is to integrate user preferences into queries, which allows users to use their own vocabulary in order to express their preferences and retrieve data in a more flexible way. This idea may be illustrated by an example of a real life scenario of movie online booking stated as follows: “I want to find a *recent* movie with a *high* rating”. In order to process such a query, fuzzy predicates, such as *recent* and *high* which model user preferences, have to be taken into account during database querying. These terms are vague and their satisfaction is a question of degree rather than an “all or nothing” notion.

Motivations for integrating user preferences into database queries are manifold [Hadjali et al., 2011]. First, it appears to be desirable to offer more expressive query languages that can be more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in case of large sets of items satisfying a query. Third, a classical query may also have an empty set of answers, while a relaxed (and thus less restrictive) version of the query might be matched by some items.

Introducing user preferences in queries has been a research topic for already quite a long time in the context of the relational database model. In the literature, one may find many flexible approaches suited to the relational data model: top-k queries [Bruno et al., 2002], the *winnnow* [Chomicki, 2002] and *Best* [Torlone and Ciaccia, 2002] operators, skyline queries [Borzsony et al., 2001], Preference SQL [Kießling, 2002], as well as approaches based on fuzzy set theory [Tahani, 1977, Bosc and Pivert, 1995, Pivert and Bosc, 2012]. The literature about preference SPARQL queries to RDF databases is not as abundant since this issue has started to attract attention only recently. Most of these approaches are straightforward adaptations of proposals made in the relational database context. In particular, they are limited to the expression of preferences over the *values* present in the nodes.

In an RDF graph context the need to query about the *structure of data* and then extract relationships between resources in the RDF graph, has motivated research aimed to extend

SPARQL and make it more expressive. In [Kochut and Janik, 2007, Anyanwu et al., 2007, Alkhateeb et al., 2009] and [Pérez et al., 2010], the authors mainly extend SPARQL by allowing to query crisp RDF through graph patterns using regular expressions but in these approaches, both the graph and the search conditions remain crisp (Boolean).

However, in the real world, many notions are not of a Boolean nature, but are rather gradual (as illustrated by the example above), so there is a need for a flexible SPARQL that takes into account RDF graphs where data is described by intrinsic weighted values, attached to edges or nodes. This weight may denote any gradual notion like a cost, a truth value, an intensity or a membership degree. For instance, in the real world, relationship between entities may be gradual (e.g., *close friend*, *highly recommends*, etc.) and an associated degree may express its intensity. A statement involving a gradual relationship is for instance “an artist recommends a movie with a degree 0.8” (roughly, this movie is highly recommended by this artist).

In order to represent such information, several authors proposed fuzzy extensions of the RDF data model. However, the fuzzy extensions of SPARQL that can be found in the literature appear rather limited in terms of expressiveness of preferences.

Our aim in this thesis is to define a much more expressive query language that i) deals with both crisp and fuzzy RDF graph databases and ii) supports the expression of complex preferences on the *values* of the nodes and on the *structure* of the graph. An example of such a query is “*most* of the *recent* movies that are recommended by an actor, are *highly* rated and have been featured by a *close* friend of this actor”.

Contributions

In this thesis, our main contributions are as follows.

1. We first propose a *fuzzy extension* of the SPARQL query language that improves its expressiveness and usability. This extension, called FURQL, allows (1) to query a fuzzy RDF data model involving fuzzy relationships between entities (e.g., *close* friends), and (2) to express fuzzy preferences on data (e.g., the release year of a movie is *recent*) and on the structure of the data graph (e.g., the path between two friends is required to be *short*). A prototype, called SURF, has been implemented and some experiments have been performed that show that introducing fuzziness in SPARQL does not come with a high price.
2. We then focus on the notion of *fuzzy quantified statements* for their ability to express different types of imprecise and flexible information needs in a (fuzzy) RDF database context. We show how a particular type of *fuzzy quantified structural query* can be

expressed in the FURQL language that we previously proposed and study its evaluation. SURF has been extended to efficiently process *fuzzy quantified queries*. It has been shown through some experimental results that introducing *fuzzy quantified statements* into a SPARQL query entails a very small increase of the overall processing time.

3. In the same way as we did with FURQL, we deal with *fuzzy quantified queries* in a more general (fuzzy) graph database context (RDF being just a special case). We study the same type of *fuzzy quantified structural query* and show how it can be expressed in an extension of the Neo4j Cypher query language, namely FUDGE, previously proposed in [Pivert et al., 2014a]. A processing strategy based on a compilation mechanism that derives regular (nonfuzzy) queries for accessing the relevant data is also described. Then, some experimental results are reported that show that the extra cost induced by the fuzzy quantified nature of the queries remains very limited.

Structure of the thesis

The remainder of the thesis is organized as follows:

- **Chapter 1** introduces background concepts and notations that are necessary to understand the rest of this thesis. We start with the RDF data model and SPARQL, which is the standard query language for RDF data, and briefly touch upon fuzzy set theory. Readers familiar with RDF, SPARQL and fuzzy set theory may want to skip this chapter.
- **Chapter 2** discusses the state-of-the-art research work related to this thesis. We give a classified overview of approaches from the literature that have been proposed to make SPARQL querying of RDF data more flexible. Then, we summarize the main features of these approaches and point out their limits.
- **Chapter 3** is devoted to the presentation of our first contribution which consists of a fuzzy extension of the SPARQL query language. First, we define the notion of a fuzzy RDF database. Second, we provide a formal syntax and semantics of FURQL, an extension of the SPARQL query language. To do so, we extend the concept of a SPARQL graph pattern defined over a crisp RDF data model, into the concept of a *fuzzy graph pattern* that allows: (1) to query a *fuzzy RDF data model*, and (2) to express *fuzzy preferences* on data (through fuzzy conditions) and on the *structure* of the data graph (through fuzzy regular expressions).
- **Chapter 4** is directly related to our second contribution that addresses the issue of integrating the notion of *fuzzy quantified statements* in the FURQL language introduced in Chapter 3 for querying fuzzy RDF databases. We first recall important notions about

fuzzy quantifiers, and present different approaches from the literature for interpreting *fuzzy quantified statements*. Then, we introduce the syntactic format for expressing a specific type of *fuzzy quantified structural query* in FURQL and we show how they can be evaluated in an efficient way.

- **Chapter 5** provides a detailed architectural implementation of the SURF prototype and reports experimental results related to approaches described in the previous chapters. These results are promising and show the feasibility of the presented approaches.
- **Chapter 6** concerns *fuzzy quantified queries* in a more general (fuzzy) graph database context. We start by recalling important notions about graph databases, fuzzy graph theory, fuzzy graph databases, and the FUDGE query language which is a fuzzy extension of the Neo4j Cypher query language. We then discuss related work about *fuzzy quantified statements* in a graph database context and point out their limits. In this chapter, we consider again a particular type of *fuzzy quantified structural query* addressed to a fuzzy graph database. We define the syntax and semantics of an extension of the query language Cypher that makes it possible to express and interpret such queries in the FUDGE language. A query processing strategy based on the derivation of nonquantified fuzzy queries is also proposed and some experiments are performed in order to study its performances.
- Finally, we conclude the thesis by summarizing our main contributions. Then, we discuss our upcoming perspectives for future research in order to improve and extend the proposed approach.

Publications & Softwares

Parts of this thesis have been published as i) regular papers at the IEEE International Conference on Fuzzy Systems [Pivert et al., 2016c] [Pivert et al., 2017], at the International Conference on Scalable Uncertainty Management [Pivert et al., 2016e], and the ACM Symposium on Applied Computing [Pivert et al., 2016g], ii) as posters and demos at the IEEE International Conference on Research Challenges in Information Science [Pivert et al., 2016a] [Pivert et al., 2016b].

[Pivert et al., 2016a] Pivert, O., Slama, O., and Thion, V. (2016a). A Fuzzy Extension of SPARQL for Querying Gradual RDF Data. In Proc. of the 10th IEEE International Conference on Research Challenges in Information Science (RCIS'16), poster session, Grenoble, France.

[Pivert et al., 2016b] Pivert, O., Slama, O., and Thion, V. (2016b). SUGAR: A Graph Database Fuzzy Querying System. In Proc. of the 10th IEEE International Conference on Research Challenges in Information Science (RCIS'16), demo paper, Grenoble, France.

[Pivert et al., 2016c] Pivert, O., Slama, O., and Thion, V. (2016c). An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data. In Proc. of the 25th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'16), Vancouver, Canada, pages 2409-2416.

[Pivert et al., 2016e] Pivert, O., Slama, O., and Thion, V. (2016e). Fuzzy Quantified Structural Queries to Fuzzy Graph Databases. In Proc. of the 10th International Conference on Scalable Uncertainty Management (SUM'16), Nice, France, pages 260-273.

[Pivert et al., 2016g] Pivert, O., Slama, O., and Thion, V. (2016g). SPARQL Extensions with Preferences: a Survey. In Proc. of the 31st ACM Symposium on Applied Computing (SAC'16), Pisa, Italy, pages 1015-1020.

[Pivert et al., 2017] Pivert, O., Slama, O., and Thion, V. (2017b). Fuzzy Quantified Queries to Fuzzy RDF Databases. In Proc. of the 26th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'17), Naples, Italy, 2017., In print.

Moreover, some works were published in French conferences:

[Pivert et al., 2016d] Pivert, O., Slama, O., and Thion, V. (2016d). FURQL: une extension floue du langage SPARQL. In Actes des Journées Bases de Données Avancées (BDA'16), Poitiers, France, 2016.

[Pivert et al., 2016f] Pivert, O., Slama, O., and Thion, V. (2016f). Requêtes quantifiées floues structurelles sur des bases de données graphe. In Actes des Rencontres Francophones sur la Logique Floue et ses Applications (LFA'16), La Rochelle, France, pages 9-16.

The SURF prototype and the SUGAR prototype are available and downloadable respectively on the following web sites:

- <https://www-shaman.irisa.fr/surf/>
- <https://www-shaman.irisa.fr/fudge-prototype/>

Chapter 1

Background notions

Contents

Introduction	21
1.1 The RDF Graph Data Model	21
1.2 SPARQL: Crisp Querying of RDF data	25
1.3 Fuzzy Set Theory	32
1.3.1 Definition	33
1.3.2 Characteristics of a Fuzzy Set	35
1.3.3 Operations on Fuzzy Sets	36
Conclusion	39

Introduction

I_N this chapter, we introduce some background notions that will be used throughout the thesis. Section 1.1 presents the RDF graph data model, section 1.2 presents the SPARQL language used for querying this model and section 1.3 presents fuzzy set theory.

1.1 The RDF Graph Data Model

Nowadays, the Resource Description Framework (RDF) [W3C, 2014], promoted by the **W3C** (**World Wide Web Consortium**), is considered to be the most appropriate model for representing, describing and storing linked and structured data available on the Web. RDF uses a set of resource names, a set of literals (e.g., a string, a number, etc.) and a set of blank nodes (i.e., unknown or anonymous resources) respectively denoted by \mathcal{U} , \mathcal{L} and \mathcal{B} in the following.

Let us consider an album as a resource of the Web. Characteristics may be attached to the album, like its title, its artist, its date or its tracks. In order to express such a characteristic,

the RDF data model uses a statement of the form of an RDF triple. Definition 1 provides a more formal definition.

Definition 1 (RDF triple). *Let \mathcal{U} be the set of URIs, \mathcal{B} the set of blank nodes, and \mathcal{L} the set of literals. An RDF triple $t := \langle s, p, o \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$ where the subject s denotes the resource being described, the predicate p denotes the property of the resource, and the object o denotes the property value. A triple t states that the subject s has a property p with a value o .*

Example 1 [RDF triple] For instance, the triple $\langle \text{Beyonce}, \text{creator}, \text{Lemonade} \rangle$ states that `Beyonce` has `Lemonade` as a `creator` property, which can be interpreted as `Beyonce` is a creator of `Lemonade`.

Definition 2 (RDF graph). *An RDF graph is a finite set of triples of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$. An RDF graph is said to be ground if it does not contain blank nodes.*

An RDF graph can be modeled by a directed labeled graph where for each triple $\langle s, p, o \rangle$, the subject s and the object o are nodes, and the predicate p corresponds to an edge from the subject node to the object one. RDF is then a graph-structural data model that makes it possible to exploit the basic notions of graph theory (such as, node, edge, path, neighborhood, connectivity, distance, in-degree, out-degree, etc.).

Example 2 [RDF graph] Let us consider an example of an RDF subgraph extracted from the MusicBrainz database ¹ which is an open music encyclopedia that collects music metadata. The resource `uri:lemonade` is an album, entitled *Lemonade*. It was released in *2016*, with genre *R&B* and rating *8.7*. It was created by the resource `uri:beyonce`, named *Beyonce*, being *38* years old and a rating of *7*. The resource `uri:sorry` and the resource `uri:holdup` entitled *hold up* are tracks of the latter resource. The resource `uri:beyonce` also created the resource `uri:B'Day`, entitled *B'Day*, that was released in *2006*. Figure 1.1 is a graphical representation of these data.

In Figure 1.1, we omit URI prefixes to avoid overcrowding the figure. A triple (s,p,o) is depicted as an edge $s \xrightarrow{p} o$, that is, s and o are represented as nodes and p is represented as an edge label. The nodes that represent resources are drawn as ellipses, those that represent literals are drawn as rectangles and the edges that represent named properties are drawn by an arrow. Each edge starts at the subject and points to the object of the triple. \diamond

¹<https://musicbrainz.org/>

even if they do not explicitly appear in it. They can be explicitly added to the graph. When all implicit triples are made explicit in the graph, then, the graph is said to be *saturated*. In this thesis, we only consider *saturated* RDF graph.

In fact, RDF data may be represented by different syntaxes such as, RDF/XML (eXtensible Markup Language)², N-Triples³, Notation 3 or N3⁴ and Turtle (Terse RDF Triple Language)⁵, etc.

Example 3 [RDF representations] Listing 1.1 is the RDF/XML representation corresponding to the resource “uri:Lemonade” from the RDF graph of Figure 1.1.

```

1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3    xmlns:mo="http://purl.org/ontology/mo/"
4    xmlns:dc="http://purl.org/dc/elements/1.1/">
5    <rdf:Description rdf:about="uri:Lemonade">
6      <dc:date> 2016 </dc:date>
7      <dc:title> Lemonade </dc:title>
8      <dc:rating> 8.7 </dc:rating>
9      <dc:genre> R & B </dc:genre>
10     <rdf:type> rdf:resource=mo:album </rdf:type>
11     <dc:track> rdf:resource="uri:sorry" </dc:track>
12     <dc:track> rdf:resource="uri:hold up" </dc:track>
13   </rdf:Description>
14 </rdf:RDF>

```

Listing 1.1: RDF/XML document

In this listing, line 1 indicates an XML declaration and line 2 says that the following XML document is about RDF. Lines 2-4 declare namespaces which indicate the URI that will be used later. Lines 5-14, as the tag is closed in line 14, present the description of a resource in which lines 6-12 describe characteristics of this resource.

Its corresponding N-Triples representation is given in Listing 1.2.

```

<uri:Lemonade> <http://purl.org/dc/elements/1.1/date> "2016" .
<uri:Lemonade> <http://purl.org/dc/elements/1.1/title> "Lemonade" .
<uri:Lemonade> <http://purl.org/dc/elements/1.1/rating> "8.7" .
<uri:Lemonade> <http://purl.org/dc/elements/1.1/genre> "R & B" .
<uri:Lemonade> <http://purl.org/dc/elements/1.1/track> <uri:sorry> .
<uri:Lemonade> <http://purl.org/dc/elements/1.1/track> <uri:hold up> .

```

²<http://www.w3.org/TR/rdf-syntax-grammar/>

³<http://www.w3.org/2001/sw/RDFCore/ntriples/>

⁴<http://www.w3.org/DesignIssues/Notation3>

⁵<http://www.w3.org/TR/turtle/>

```
<uri:Lemonade> <http://www.w3.org/1999/02/22-rdf-syntax-ns#/type> <mo:album> .
```

Listing 1.2: N-Triples file

A database which stores RDF graphs, containing statements of the form (subject-predicate-object), is called a *triple store* (or simply an *RDF database*). There have been a significant number of RDF databases over the last years mainly divided into two categories [Faye et al., 2012]:

- **Native RDF stores** implement their own database engine without reusing the storage and retrieval functionalities of other database management systems. Some examples of native RDF stores are AllegroGraph (commercial)⁶, Apache Jena TDB (open-source)⁷, etc.
- **Non-native RDF Stores** use the storage and retrieval functionalities provided by other database management systems. Among the non-native RDF stores, we find the Apache Jena SDB (open-source) using conventional relational databases⁸, etc.

1.2 SPARQL: Crisp Querying of RDF data

In order to efficiently query RDF data, the SQL-like language SPARQL [Prud'hommeaux and Seaborne, 2008] is promoted by the W3C as a standard query language. It is a declarative query language based on graph pattern matching, in the sense that the query processor searches for sets of triples in the data graph that satisfy a graph pattern expressed in the query.

A **Basic Graph Pattern (BGP)** is a basic building block of SPARQL, containing a set of triple patterns. A triple pattern is an RDF triple where variables may occur in the subject, predicate, or object position. Each variable is prefixed by the question mark symbol.

Example 4 [Basic Graph Pattern] The albums featuring the artist Beyonce, with their names are described by the following graph pattern.

```
?artist dc:creator ?album .
?artist dc:title "Beyonce" .
?album dc:title ?name .
```

Listing 1.3: A SPARQL Basic Graph Pattern

A graphical representation of this graph pattern is depicted in Figure 1.2.

According to the graph of Figure 1.1, two subgraphs that are isomorphic to this graph pattern may be found and they are given in Figure 1.3. ◊

⁶<http://www.franz.com/agraph/allegrograph/>

⁷<http://jena.apache.org/>

⁸<http://jena.apache.org/>

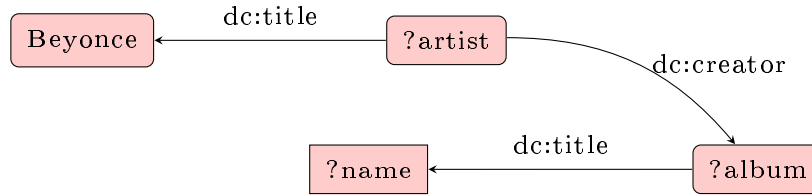


Figure 1.2: A graphical representation of the graph pattern from Listing 1.3

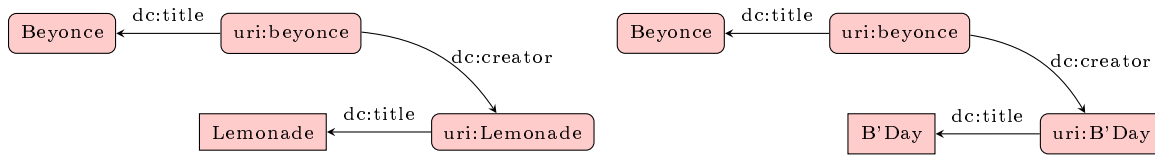


Figure 1.3: Possible subgraphs from Figure 1.1

A classical SPARQL query has the general form given in Listing 1.4, where the clause `PREFIX` is for abbreviating URIs (which will be omitted in the following examples), the clause `SELECT` is for specifying which variables should be returned, the clause `FROM` defines the datasets to be queried, and the clause `WHERE` contains the triple of the researched pattern.

```

PREFIX ... #PREFIX declarations
SELECT ... #Result
FROM ... #Dataset definition
WHERE ... #Pattern
ORDER BY ..., DISTINCT ..., LIMIT ..., OFFSET ..., PROJECTION ... #Modifiers
  
```

Listing 1.4: Skeleton of a SPARQL query

SPARQL also provides solution modifiers, which make it possible to modify the result set by applying classical operators like `ORDER BY` for ordering the result set in ascending (`ASC(.)` default ordering) or descending (`DESC(.)`) order, `DISTINCT` for removing duplicate answers, `LIMIT` to limit the number of answers to a fixed number (chosen by a user), `PROJECTION` to choose certain variables and eliminate others from the solutions, or `OFFSET` to define the position of the first returned answers.

Finally, the output of a `SELECT` SPARQL query is a set of mappings of variables which match the patterns in the `WHERE` clause.

Example 5 [SELECT SPARQL query] Listing 1.5 is a simple `SELECT` SPARQL query taken from the MusicBrainz database that aims to retrieve the names and the released dates of the albums featuring *Beyonce*, sorted in ascending order of their release date.

```

SELECT ?name ?date WHERE {
  ?artist dc:title "Beyonce" . ?artist dc:creator ?album .
  ?album dc:title ?name . ?album dc:date ?date . }
ORDER BY ASC(?date)

```

Listing 1.5: Example of a SPARQL SELECT query

The result of this query evaluated on the RDF graph of Figure 1.1 comprises two albums as given in Table 1.1. \diamond

Table 1.1: Results of Listing 1.5

?name	?date
Lemonade	2016
B'Day	2006

More complex graph patterns exist in SPARQL including.

- **Optional graph pattern:** uses the clause `OPTIONAL` and allows for a partial matching of the query. The query tries to match a graph pattern and does not discard a candidate answer when some part of the optional patterns is not satisfied.

Example 6 [Optional graph pattern] An example of how SPARQL implements the optional matching is to find the names of all the albums featuring Beyonce and if possible, their genre.

```

SELECT ?title ?genre WHERE {
  { ?artist dc:creator ?album . ?album dc:title ?title .
    ?artist dc:title "Beyonce" . }
  OPTIONAL { ?album dc:genre ?genre . } }

```

Listing 1.6: SPARQL query with OPTIONAL graph pattern

The result of the evaluation of the query on the running example of Figure 1.1 page 23 returns two albums, given in Table 1.2. Here, the second answer does not have a *genre* and is kept in the final answer since the property *genre* is in the *optional* part of the query. \diamond

Table 1.2: Results of Listing 1.6

?title	?genre
Lemonade	R&B
B'Day	

- **Union graph pattern:** forms a disjunction of two graph patterns thanks to the use of the clause `UNION` and allows for alternatives. Solutions to both sides of the `UNION` are included in the results.

Example 7 [Union graph pattern] Find the albums that are made by James Brown as a singer or a producer. ◊

```
SELECT ?album WHERE {
  { ?artist dc:creator ?album .
    ?artist dc:title "James Brown" . }
  UNION { ?producer dc:producer ?album .
          ?producer dc:title "James Brown" . } }
```

Listing 1.7: SPARQL query with UNION graph pattern

- **Graph pattern:** using the clause `GRAPH`, allows portions of a query pattern to match one or more named graphs identified by an URI in the RDF dataset. Note that anything outside the `GRAPH` clause has to match the default graph.
- **Filter graph pattern:** using the clause `FILTER` followed by an expression to select answers according to some criteria. This expression may contain classical operators (e.g., =, +, *, -, /, <, >, ≥, ≤) and functions (e.g., *isURI(?x)*, *isLiteral(?x)*, *isBlank(?x)*, *regex(?x, "A.*")*).

Example 8 [Filter graph pattern] the query that aims to find the titles of the albums featuring James Brown which have been released after 2008 is given in Listing 1.8.

```
SELECT ?title WHERE {
  ?artist dc:creator ?album . ?artist dc:title "James Brown" .
  ?album dc:title ?title . ?album dc:date ?date .
  FILTER (?date >= "2008" )
}
```

Listing 1.8: SPARQL FILTER query (classical operator)

And the query that aims to find all of the albums whose title starts with “Happy” is given in Listing 1.9. ◊

```
SELECT ?album ?title WHERE {
  ?album rdf:type "mo:Album" . ?album dc:title ?title .
  FILTER regex(?title, "^Happy")
}
```

Listing 1.9: SPARQL FILTER query (function)

Different types of queries are available in SPARQL.

- **SELECT query:** is equivalent to an SQL SELECT, used to return a set of variables from the query pattern using the `SELECT` clause. For instance, all the aforementioned examples of SPARQL queries are of the SELECT form;

- **CONSTRUCT query:** returns a single RDF graph by creating new triples that satisfy a specific template from the query pattern.

Example 9 [CONSTRUCT query] Let us assume that, if a person X knows a person Y and if this latter (X) knows a person Z , so, we can say that the first person X knows the person Z or any person known by Y . Thus, we can create this relationship thanks to the following CONSTRUCT query. \diamond

```
CONSTRUCT { ?x foaf:knows ?z . }
WHERE {
  ?x foaf:knows ?y .
  ?y foaf:knows ?z .
}
```

Listing 1.10: An example of a CONSTRUCT query

- **ASK query:** is used to return a Boolean result: *true* if there exists at least one result that matches the query pattern and *false* otherwise.

Example 10 [ASK query] The following query illustrates the use of the ASK query: Is “Beyonce” the name of the resource `uri:beyonce` ?

```
ASK { uri:beyonce dc:title "Beyonce" . }
```

Listing 1.11: An example of an ASK query

This query returns *true* since the resource `uri:beyonce` is indeed the artist “Beyonce”. \diamond

- **DESCRIBE query:** is used to return a single RDF graph with information about the selected resources.

Example 11 [DESCRIBE query] An example of a DESCRIBE query is given in Listing 1.12.

```
DESCRIBE uri:beyonce
```

Listing 1.12: An example of a DESCRIBE query

This query returns information about the resource $\langle uri:beyonce \rangle$, such as, its name, its age, its rating, its type, etc. \diamond

Recently, SPARQL 1.1 [Harris and Seaborne, 2013] is a new version of SPARQL supporting new features, such as, property paths, update functionalities, subqueries, negation, value assignments, aggregates functions, etc.

- **Property paths:** they are known as regular expressions tackled in [Kochut and Janik, 2007, Anyanwu et al., 2007, Pérez et al., 2008, Alkhateeb et al., 2009, Pérez et al., 2010]. These works proposed more expressive languages and extended SPARQL by allowing path extraction queries (generally of unknown length) within RDF datasets. Property paths came with the same principle which is to allow for navigational querying over RDF graphs and are officially integrated in SPARQL 1.1. A property path is a possible path through a graph between two nodes. It can be of a variable length. A property path of length exactly 1 is a triple pattern.

Definition 3 (SPARQL property path expressions). *SPARQL property path expressions are recursively defined by:*

- an IRI⁹ is a property path expression that denotes a path of length one,
- if exp_1 and exp_2 are property path expressions, then, $exp_1|exp_2$ and exp_1/exp_2 are property path expressions,
- if exp is a property path expression, then, exp^* , exp^+ , $exp^?$ and \hat{exp} are property path expressions.

where $exp_1|exp_2$ denotes alternative expressions, exp_1/exp_2 denotes a concatenation of exp_1 and exp_2 , exp^* denotes a path that connects the subject and object of the path by zero or more matches of exp , exp^+ is a shortcut for exp^*/exp and denotes a path that connects the subject and object of the path by one or more matches of exp , $exp^?$ denotes a path that connects the subject and object of the path by zero or one matches of exp , \hat{exp} is an inverse path (from an object to the subject).

Example 12 [SPARQL property path query] Find the names of the artists that recommend albums made by friends or related friends of friends. \diamond

```

SELECT ?name WHERE {
  ?art1 dc:title ?name . ?art1 dc:recommends ?alb .
  ?art2 dc:creator ?alb . ?art1 dc:friend+ ?art2 .
}

```

Listing 1.13: SPARQL query with property path

- **Update functionalities:** In addition to querying and manipulating RDF data, SPARQL 1.1 Update [Gearon et al., 2012] offers the possibility to modify the graph by adding/deleting triples, loading/clearing/creating/dropping an RDF graph and many other facilities.

⁹Internationalized Resource Identifier

Example 13 [Update query] The following query aims to add some information about a new artist "Ed Sheeran" in the default graph. ◊

```
INSERT DATA {
  uri:EdSheeran dc:title "Ed Sheeran" . uri:EdSheeran dc:age "26" .
  uri:EdSheeran dc:rating "8" . }
```

Listing 1.14: An example of an Update query

- **Subqueries:** The principle is the same as subqueries in SQL: a query may use the output of other queries for achieving complex results.

Example 14 [Subqueries in SPARQL] Return a name (the one with the lowest sort order) for all the artists who are friend with beyonce and have a name. ◊

```
SELECT ?art ?minName WHERE {
  uri:beyonce uri:friend ?y .
  {
    SELECT ?art (MIN(?name) AS ?minName) WHERE {
      ?art uri:title ?name .
    } GROUP BY ?art
  } }
```

Listing 1.15: Query involving a subquery

- **Negation:** can be expressed in two ways. The first uses the `NOT EXISTS` clause and aims to filter out the results that do not match a given graph pattern. The second one uses the `MINUS` clause and aims to remove answers related to another pattern.

Example 15 [Negation queries] The following query aims to find the artists who have issued no albums in 2015.

```
SELECT ?name WHERE {
  ?artist dc:creator ?album .
  FILTER NOT EXISTS { ?artist dc:date "2015" . }
}
```

Listing 1.16: SPARQL query with negation (`NOT EXISTS`)

The query that aims to retrieve names of artists having no albums is depicted in Listing 1.17. ◊

```
SELECT ?name WHERE {
  ?artist dc:title ?name .
  MINUS { ?artist dc:creator ?album . }
}
```

Listing 1.17: SPARQL query with negation (`MINUS`)

- **Assignments:** The value of a complex expression can be added to a solution mapping by binding a new variable to the value of this expression. The variable can then be used in the query and also can be returned in the result. The assignment is of the form: `(expression AS ?var)`.

Example 16 [Query with assignment] The following query aims to return the albums released less than 6 years before 2017. ◊

```
SELECT ?name ?album (2017 - ?date AS ?Newdate) WHERE {
  ?album dc:title ?name . ?album dc:date ?date .
  FILTER ( ?Newdate < 6 ) }
```

Listing 1.18: SPARQL query with assignments

- **Agregate functions:** Aggregates apply expressions over groups of answers. In SPARQL 1.1, we can find aggregate operators like, `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`, `GROUP_CONCAT`, and `SAMPLE`. Grouping may be specified using the `GROUP BY` clause.

Example 17 [Query with aggregates] Counting albums for each artist. ◊

```
SELECT ?artist (COUNT(?album) AS ?number) WHERE {
  ?artist dc:creator ?album . }
GROUP BY ?artist
```

Listing 1.19: SPARQL query with aggregates

Web services that make it possible to retrieve RDF data through SPARQL queries are called SPARQL endpoints. Among public SPARQL endpoints, we can mention IMDB <http://data.linkedmdb.org/sparql>, DBpedia <http://dbpedia.org/sparql>, data.gov <http://semantic.data.gov/sparql>, DBLP Bibliography Database <http://www4.wiwiss.fu-berlin.de/dblp/sparql> and many others.

1.3 Fuzzy Set Theory

In the classical set theory, there are two possible situations for an element: to belong or to not belong to a subset.

In 1965, Lotfi Zadeh [Zadeh, 1965] proposed to extend classical set theory by introducing the concept of gradual membership in order to model classes whose borders are not clear-cut. A fuzzy set is associated with a membership function which takes its values in the range of real numbers $[0,1]$, that is to say that graduations are allowed and an element may belong more or less to a fuzzy subset.

The theory of fuzzy sets has advanced applications in artificial intelligence, computer science, decision theory, expert systems, robotics, etc. They also play an important role in expressing fuzzy user preferences queries to relational databases [Dubois and Prade, 1997, Pivert and Bosc, 2012].

In the following, we first give a formal definition and some characteristics of the notion a fuzzy set, and then the main operations over fuzzy sets are detailed.

1.3.1 Definition

Let X be a classical set of objects called the Universe and x be any element of X . If A is a classical subset of X , the membership degree of every element can take only extreme values 0 or 1. This corresponds to the classical definition of a *characteristic function*:

$$\mu_A(x) = \begin{cases} 1 & \text{iff } x \in A, \\ 0 & \text{otherwise.} \end{cases}$$

When A is a fuzzy subset of X [Zadeh, 1965] it is denoted by:

$$A = \{(x, \mu_A(x)), x \in X\} \text{ with } \mu_A : X \rightarrow [0,1],$$

where $\mu_A(x)$ is a degree of membership (simply denoted degree in the following) that quantifies the membership grade of x in A . The closer the value of $\mu_A(x)$ to 1, the more x belongs to A . Therefore, we can have the three situations:

$$\mu_A(x)=0, 0 < \mu_A(x) < 1, \mu_A(x)=1.$$

where $\mu_A(x)=0$ means that x does not belong to A at all, $0 < \mu_A(x) < 1$ if x belongs partially to A and $\mu_A(x)=1$ means that x belongs entirely to A .

In practice, the membership function of A is of a trapezoidal shape (see Figure 1.4) and is expressed by the quadruplet $(A - a, A, B, B + b)$.

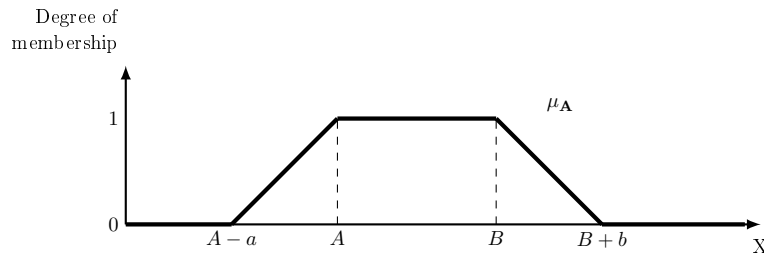


Figure 1.4: Trapezoidal membership function

Example 18 Let us consider the example of the predicate *tall* described in Table 1.3. Tall can be defined by a Boolean condition ($\text{height} \geq 180$). It corresponds to the crisp set (non fuzzy set) of Figure 1.5 and the result is in the third column of Table 1.3.

Name	height(cm)	Memberships	
		Crisp	Fuzzy
Chris	210	1	1
Marc	200	1	1
John	190	1	1
Tom	180	1	0.66
David	170	0	0.33
Tom	160	0	0
David	150	0	0

Table 1.3: Tall men

However, it seems more natural to define the predicate "tall" as a fuzzy set (cf., Figure 1.6). The membership degrees associated with some individuals are shown in the fourth column of Table 1.3. \diamond

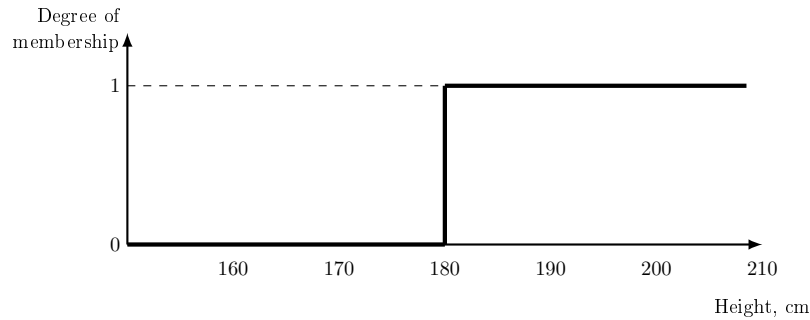


Figure 1.5: Graphical representation of the predicate *tall* (crisp set)

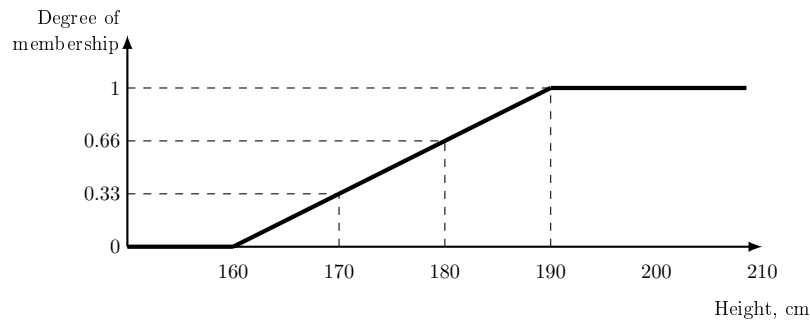


Figure 1.6: Graphical representation of the predicate *tall* (fuzzy set)

A more convenient notation, when X is a finite set $\{x_1, \dots, x_n\}$, is:

$$A = \{\mu_A(x_1)/x_1, \dots, \mu_A(x_n)/x_n\},$$

It is worth mentioning that in practice the elements for which the degree equals 0 are omitted.

Remark 1. *A fuzzy subset of X is called normal if there exists at least one element $x \in X$ such as $\mu_A(x) = 1$. Otherwise it is called subnormal.*

1.3.2 Characteristics of a Fuzzy Set

Several notions can be used to describe a fuzzy set. Among them we can cite.

1.3.2.1 Support, height and core

The support of a fuzzy subset A in the universal set X , denoted by $supp(A)$, is a crisp set that contains all the elements of X that have a strictly positive degree in A (i.e., which belong *somewhat* to A). More formally:

$$supp(A) = \{x \mid x \in X, \mu_A(x) > 0\}.$$

The core of a fuzzy subset A , denoted by $core(A)$, is the crisp subset of X containing all the elements with a degree equal to 1 (i.e. that completely belong to A with degree equal to 1). More formally:

$$core(A) = \{x \mid x \in X, \mu_A(x) = 1\}.$$

Remark 2. *Note that in the case of a crisp set, the support and the height collapse, since if x is somewhat in A it belongs (totally) to A .*

Example 19 Let us consider two fuzzy subsets A and B of the set X , with $X = \{x_1, x_2, x_3, x_4, x_5\}$, $A = \{1/x_1, 0.3/x_2, 0.2/x_3, 0.8/x_4, 0/x_5\}$ and $B = \{0.6/x_1, 0.9/x_2, 0.1/x_3, 0.3/x_4, 0.2/x_5\}$. \diamond

The supports of the two subsets A and B are:

$$supp(A) = \{x_1, x_2, x_3, x_4\}, \quad supp(B) = \{x_1, x_2, x_3, x_4, x_5\}.$$

The core of these two subsets is as follows:

$$core(A) = \{x_1\}, \quad core(B) = \emptyset.$$

The height of a fuzzy subset A of X denoted by $hgt(A)$ is the largest degree attained by any element of X that belongs to A . More formally:

$$hgt(A) = \sup_{x \in X} \mu_A(x).$$

A is said to be normalized iff $\exists x \in X, \mu_A(x) = 1$ which means that $hgt(A) = 1$.

1.3.2.2 α -cut

The ordinary set of such elements $x \in X$ having a membership degree larger or equal to a threshold $\alpha \in]0, 1]$ is the α -cut (A_α) of the fuzzy subset A defined as:

$$A_\alpha = \{x \mid x \in X, \mu_A(x) \geq \alpha\}.$$

Example 20 Let us consider $X = \{x_1, x_2, x_3\}$ and a fuzzy subset $A = \{0.3/x_1 + 0.5/x_2 + 1/x_3\}$, the α -cuts of this subset are as follows :

$$A_{0.5} = \{x_2, x_3\}, A_{0.1} = \{x_1, x_2, x_3\}, A_1 = \{x_3\}$$

The membership function of a fuzzy subset A can be expressed in terms of characteristic functions of its α -cuts according to the following formula:

$$\mu_A(x) = \sup_{\alpha \in]0, 1]} \min(\alpha, \mu_{A_\alpha}(x)) ,$$

where

$$\mu_{A_\alpha}(x) = \begin{cases} 1 & \text{iff } x \in A_\alpha, \\ 0 & \text{otherwise.} \end{cases}$$

The strict (or strong) α -cut of A , denoted by $A_{\bar{\alpha}}$, contains all the elements in X that have a membership value in A strictly greater than α :

$$A_{\bar{\alpha}} = \{x \mid x \in X, \mu_A(x) > \alpha\}.$$

The following properties hold:

- $A_{\bar{0}} = \text{supp}(A)$,
- $A_{\bar{1}} = \text{core}(A)$,
- $\alpha_1 > \alpha_2 \Rightarrow A_{\alpha_1} \subseteq A_{\alpha_2}$.

It can easily be checked that:

$$(A \cup B)_\alpha = A_\alpha \cup B_\alpha \text{ and } (A \cap B)_\alpha = A_\alpha \cap B_\alpha.$$

1.3.3 Operations on Fuzzy Sets

Classical operations on crisp sets have been extended to fuzzy sets. These extensions are equivalent to classical operations of set theory when dealing with membership functions belonging to values 0 or 1. The most commonly used operations are presented hereafter and the interesting reader may refer to [Dubois, 1980].

1.3.3.1 Complementation

The complement of a fuzzy set A , denoted by \bar{A} , is defined as:

$$\forall x \in X, \mu_{\bar{A}}(x) = 1 - \mu_A(x).$$

Example 21 Let us consider the fuzzy subset $A = \{1/x_1 + 0.3/x_2 + 0.2/x_3 + 0.8/x_4 + 0/x_5\}$.

Its complement is $\bar{A} = \{0/x_1 + 0.7/x_2 + 0.8/x_3 + 0.2/x_4 + 1/x_5\}$. \diamond

This operation is involutive, i.e., $\bar{\bar{A}} = A$ ($\mu_{\bar{\bar{A}}}(x) = \mu_A(x)$).

1.3.3.2 Inclusion

Let us consider two fuzzy sets A and B defined on X . If for any element x of X , x belongs less to A than B or has the same membership, then A is said to be included in B ($A \subseteq B$). Formally $A \subseteq B$ if and only if:

$$\forall x \in X, \mu_A(x) \leq \mu_B(x).$$

When the inequality is strict, the inclusion is said to be strict and is denoted by $A \subset B$. Obviously, $A=B$ iff $A \subseteq B$ and $B \subseteq A$.

1.3.3.3 Intersection and union of fuzzy sets

The intersection of two fuzzy subsets A and B in the universe of discourse X , denoted by $A \cap B$, is a fuzzy set given by:

$$\mu_{A \cap B}(x) = \top(\mu_A(x), \mu_B(x)),$$

where \top is a triangular *norm* (abbreviated *t-norm*) and usually we take the *minimum*.

The union of two fuzzy subsets A and B in the universe X (denoted by $A \cup B$) is a fuzzy subset given by:

$$\mu_{A \cup B}(x) = \perp(\mu_A(x), \mu_B(x)),$$

where \perp is a triangular *co-norm* (abbreviated *t-conorm*) and usually we take the *maximum*.

The *t-norms* and *t-conorms* operators follow the properties showed in Table 1.4. The set of *t-norms* (resp. *t-conorms*) has an upper (resp. lower) element which is the minimum (resp. maximum) operator.

Example 22 Let us come back to Example 19. The intersection of the two fuzzy subsets, taking $\top = \min$, is as follows:

$$A \cap B = \{0.6/x_1, 0.3/x_2, 0.1/x_3, 0.3/x_4, 0/x_5\}.$$

The union of the two fuzzy subsets, taking $\perp = \max$, is as follows:

$$A \cup B = \{1/x_1, 0.9/x_2, 0.2/x_3, 0.8/x_4, 0.2/x_5\}.$$

Table 1.4: Properties of t -norm and t -conorm operators

Property	T -norm	T -conorm
Identity	$1 \wedge x = x$	$0 \vee x = x$
Commutativity	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
Associativity	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
Monotonicity	if $v \leq w$ and $x \leq y$ then $v \wedge x \leq w \wedge y$	$v \vee x \leq w \vee y$

Remark 3. A t -norm is associated with a t -conorm (e.g., min/max) and they satisfy De Morgan's Laws.

Later, compensatory operators, such as *the averaging operators*, have appeared useful for aggregating fuzzy sets, especially in the context of decision making [Zimmermann, 2011]. *Averaging operators* for intersection (resp., union) are considered to be more optimistic (resp., pessimistic) than t -norms (resp., t -conorms).

Let us also mention many other operators that may be used for expressing different kinds of trade-offs, such as the weighted conjunction and disjunction [Dubois and Prade, 1986], fuzzy quantifiers [Fodor and Yager, 2000] (that are going to be explained in Chapter 4), or the non-commutative connectives described in [Bosc and Pivert, 2012].

1.3.3.4 Difference between fuzzy sets

The difference between two fuzzy sets A and B is defined as:

$$\forall x \in X, \mu_{A-B}(X) = \top(\mu_A(x), \mu_{\bar{B}}(x)) = \top(\mu_A(x), 1 - \mu_B(x)),$$

which leads to:

- $\mu_{A-B}(x) = \min(\mu_A(x), 1 - \mu_B(x))$ with $\top(x, y) = \min(x, y)$,
- $\mu_{A-B}(x) = \max(\mu_A(x) - \mu_B(x), 0)$ if $\top(x, y) = \max(x + y - 1, 0)$ is chosen.

Example 23 Consider the following fuzzy sets $A = \{1/a, 0.3/b, 0.7/c, 0.2/e\}$ and $B = \{0.3/a, 1/c, 1/d, 0.6/e\}$.

Using the minimum for the conjunction, one obtains $\{0.7/a, 0.3/b, 0.2/e\}$ for the difference $A - B$, while $A - B = \{0.7/a, 0.3/b\}$ with the other choice. \diamond

1.3.3.5 Cartesian product of fuzzy sets

The Cartesian product of the two fuzzy sets A and B , defined as:

$$\mu_{A \times B}(xy) = \top(\mu_A(x), \mu_B(y)),$$

where \top is a triangular *norm*.

Example 24 Let A and B be the fuzzy sets: $A = \{1/a, 0.3/b, 0.7/c, 0.2/e\}$,
 $B = \{1/y, 0.6/z\}$.

Their Cartesian product $A \times B$ (if the minimum is taken for \top) is:

$\{1/(a,y), 0.6/(a,z), 0.3/(b,y), 0.3/(b,z), 0.7/(c,y), 0.6/(c,z), 0.2/(e,y), 0.2/(e,z)\}.$ ◊

Conclusion

In this chapter, we provided some background notions concerning the RDF Graph data model, the SPARQL query language and fuzzy set theory. These notions will play a central role in the following since in this thesis we intend to propose a fuzzy extension of the SPARQL query language addressed to both crisp and fuzzy RDF graphs.

Chapter 2

State of the art: Flexible Querying of RDF data

Contents

Introduction	41
2.1 Preference Queries on RDF Data	42
2.1.1 Quantitative Approaches	42
2.1.2 Qualitative Approaches: Skyline-based Approaches	49
2.2 Query Relaxation	53
2.3 Approximate Matching	57
Conclusion and Summary	58

Introduction

I_N the last years, with the rapid growth in size and complexity of RDF graphs, querying RDF data in a flexible, expressive and intelligent way has become a challenging problem. In the following, we present the contributions from the literature that make SPARQL querying of RDF data more flexible. Three categories of approaches may be associated with the following objectives: i) introducing user preferences into queries (which is directly related to this thesis), ii) relaxing user queries and iii) computing an approximate matching of two RDF graphs. These approaches are discussed further in the following sections.

A part of this chapter related to introducing user preferences inside SPARQL queries was published in the form of a survey in the proceedings of the 31st ACM Symposium on Applied Computing (SAC'16).

2.1 Preference Queries on RDF Data

Introducing user preferences into queries has been a research topic for already quite a long time in the context of the relational database model. Motivations for integrating preferences are manifold [Hadjali et al., 2011]. First, it has appeared to be desirable to offer more expressive query languages that can be more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in case of large sets of items satisfying a query. Third, a classical query may also have an empty set of answers, while a relaxed (and thus less restrictive) version of the query might be matched by some items.

The literature about preference queries to RDF databases is not as abundant as in the relational context since this issue has started to attract attention only recently. In this section, we present an overview of approaches that have been proposed to extend SPARQL by integrating user preferences in queries, followed by a classification of these approaches into two categories according to their qualitative or quantitative nature. We first present quantitative approaches (Subsection 2.1.1), then qualitative ones (Subsection 2.1.2).

2.1.1 Quantitative Approaches

The quantitative approaches share the following principle: each involved preference is defined via an atomic scoring function allowing a score (aka., satisfaction degree) to be associated with each answer, making it possible to get a total ordering of the answers (i.e., tuple t_1 is preferred to tuple t_2 if the score of t_1 is higher than the score of t_2).

Among the works which belong to the quantitative approaches we may find those that are based on fuzzy set theory [Zadeh, 1965] and aim to a flexible extension of the query language (SPARQL) [Cheng et al., 2010, Wang et al., 2012, Ma et al., 2016]. We can find, also, those based on top- k querying of RDF data that aim to extend the SPARQL language with top- k queries [Bozzon et al., 2011, Bozzon et al., 2012, Magliacane et al., 2012, Wang et al., 2015].

2.1.1.1 Fuzzy set-based approach

The standard version of the SPARQL query language supports only a few classical ways of retrieval, all based on Boolean logic. In order to meet user needs more effectively, [Cheng et al., 2010] proposes a syntactical fuzzy extension of SPARQL, called f-SPARQL (fuzzy SPARQL), which supports the expression of fuzzy conditions including (possibly compound) fuzzy terms, e.g., *recent* or *young*, and fuzzy operators, e.g., *close to* or *at least*, interpreted in a gradual manner.

Most fuzzy terms are assumed to be represented by a trapezoidal membership function (see for instance a possible representation of *recent* in Figure 2.1).

Figure 2.1: Membership function of *recent*

Membership functions of the fuzzy predicates *at least Y*, *at most Y* and *close to Y* are proposed in [Cheng et al., 2010]. For instance, the membership function of the fuzzy number “at least Y” on the universe of discourse is shown in Figure 2.2 and is defined by the following equation:

$$\mu_{\text{atleast}Y}(x) = \begin{cases} 0, & \text{if } u \leq w; \\ \frac{u-w}{Y-w}, & \text{if } w < u < Y; \\ 1, & \text{if } u \geq Y. \end{cases} \quad (2.1)$$

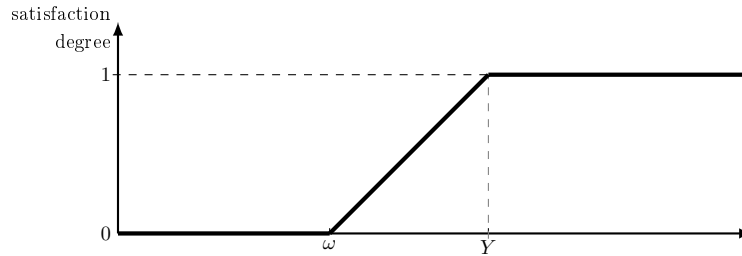


Figure 2.2: Membership function of the fuzzy number “at least Y”

The f-SPARQL extension of SPARQL concerns the **FILTER** clause whose syntax becomes

$$\text{FILTER } [(?X \theta \text{ FT}) \mid (?X \hat{\theta} Y)] [\text{WITH } \alpha],$$

where FT denotes a fuzzy term, θ denotes a classical operator (e.g., $>$, $<$, $=$, \geq , \leq , \neq), $\hat{\theta}$ denotes a fuzzy operator (such as *close to (around)*, *at least*, and *at most*), and Y is a string, an integer or an other types allowed in RDF. The optional parameter [WITH α] specifies the smallest acceptable membership degree in the interval $[0, 1]$. Each f-SPARQL query is prefixed by #FQ#.

Example 25 The fuzzy query “retrieve the name of the *recent* albums with Beyonce” is formulated by the *f-SPARQL* listing 2.1.

```
#FQ#
SELECT ?name WHERE {
  ?artist dc:title "Beyonce". ?artist dc:creator ?album .
  ?album dc:title ?name. ?album dc:date ?date.
  FILTER (?date = recent).}
```

Listing 2.1: An f-SPARQL query

This query aims to retrieve from a music database the albums by Beyonce that have been recently released. If the MusicBrainz RDF database of Figure 1.1 on page 23 is queried, then the album entitled *Lemonade* belongs to the answer, with a satisfaction degree of 0.66, which corresponds to the degree of membership of value 2016 to the fuzzy term *recent* (see Figure 2.1). The other album from Figure 1.1, released in 2006, does not belong to the answer as it is not at all recent according to Figure 1.1. \diamond

Let us now assume that the database of the running example embeds a rating value for each album, through a property named *dc:rate* connecting an album (URI resource) to a rating value (a label). When a user wants to express preferences on several attributes (e.g., date, rating, ...), he/she may assign an importance to every partial preference. If no importance is specified, it is implicitly assumed that the partial degrees are aggregated by means of the triangular norm *minimum* that is commonly used in fuzzy logic to interpret the conjunction.

In [Cheng et al., 2010], the authors propose to use a weighted mean in order to combine the partial scores coming from different atomic preference criteria:

$$score(A) = \sum_{i=1}^n \mu(A_i) \times w(F_i) \quad (2.2)$$

where $F = (F_1, \dots, F_n)$ is the set of FILTER conditions, A_i is the property concerned by F_i in the candidate answer A , $\mu(A_i)$ denotes the membership degree of the answer for F_i , and $w(F_i)$ denotes the weight assigned to F_i , assuming that $\sum_{i=1}^n w(F_i) = 1$.

Example 26 Consider the query “retrieve the name of the *recent* (importance 0.2) albums featuring Beyonce with a *high* rating (importance 0.8)”. It is expressed in f-SPARQL by Listing 2.2.

```
#FQ#
SELECT ?name WHERE {
  ?artist dc:title "Beyonce". ?artist dc:creator ?album.
  ?album dc:title ?name. ?album dc:date ?date. ?album dc:rating ?rating.
  FILTER (?date = recent) WITH 0.2.
  (?rating = high) WITH 0.8.}
```

Listing 2.2: f-SPARQL query with weights

It is also possible to apply a threshold α_i to an atomic fuzzy condition F_i (this threshold is associated with the underlying attribute in the `SELECT` clause). Then, an answer is qualified only if its membership degree relatively to F_i is at least equal to α_i . Surprisingly, it does not seem that f-SPARQL makes it possible to specify a threshold on the global satisfaction degree.

As in SQLF introduced in [Bosc and Pivert, 1995], two types of queries exist in f-SPARQL depending on the type of calibration:

- a qualitative calibration in the case of flexible queries (`#FQ#`) (see Listing 2.2);
- a quantitative calibration in the case of top- k flexible queries (`#top- k FQ#` with k (see Listing 2.3), and then, only the top- k answers are returned.

The query type has to be declared before the `SELECT` clause: `#FQ#` (flexible query) in the first case, and `#top- k FQ#` with k (top- k flexible query) when a quantitative threshold is used.

Example 27 Let us consider again the query from Example 26 and assume that a user only wants to get the 10 best answers. The corresponding f-SPARQL query is:

```
#top- $k$  FQ# WITH 10
SELECT ?name WHERE {
  ?artist dc:tilte "Beyonce". ?artist dc:creator ?album.
  ?album dc:tilte ?name. ?album dc:date ?date. ?album dc:rating ?rating.
  FILTER (?date = recent) WITH 0.2.
  (?rating = high) WITH 0.8.}
```

Listing 2.3: Top- k flexible query

The authors of [Cheng et al., 2010] exhibit a set of translation rules to convert f-SPARQL queries into Boolean ones so as to be able to benefit from the existing implementations of standard SPARQL. The same principle was initially proposed in [Bosc and Pivert, 2000] in the context of relational databases (under the name *derivation principle*) to process SQLf (fuzzy) queries. It aims to derive a crisp (SQL) query from an SQLf query involving a (global) qualitative threshold α in order to return only answers with satisfaction degree greater or

equal to the α -cut. Different types of translation rules were used in [Cheng et al., 2010] depending on the the types of fuzzy terms (including simple atomic terms, e.g., *recent*, modified fuzzy terms, e.g., *very recent*, and compound fuzzy terms, e.g., *popular and very recent*) and fuzzy operators.

Some of the authors of [Cheng et al., 2010] proposed two variants of f-SPARQL. The First one, called fp-SPARQL (fuzzy and preference SPARQL) [Wang et al., 2012], involves an alternative way of (i) interpreting modified fuzzy terms (i.e., an atomic fuzzy term modified by an adverb such as *extremely*, *rather*, etc), and (ii) interpreting compound fuzzy conditions where atomic predicates are assigned a priority.

The second query language, called SPARQLf-p [Ma et al., 2016], makes it possible to express i) more complex conditions including fuzzy relations (e.g., physical health is a fuzzy relation between height and weight) besides fuzzy terms and fuzzy operators and, ii) multi-dimensional user preferences.

From another point of view, the authors of [Buche et al., 2008, Buche et al., 2009, Buche et al., 2013] defined a flexible querying system using fuzzy RDF annotations based on the notion of similarity and imprecision. This approach is beyond the scope of our work since it does not explicitly propose an extension of the SPARQL query language.

2.1.1.2 Top-k-based approach

Top- k -query approaches have been proposed for already many years in a relational database context (cf., the survey of [Ilyas et al., 2008]). They have been useful in several application areas such as system monitoring, information retrieval, multimedia databases, sensor networks, etc. Top- k queries [Bruno et al., 2002] are a popular class of queries that return only the k most relevant (best) tuples according to user's preferences. The attribute values of each tuple are associated with a value or score using a simple linear function. Top- k -queries can be viewed as a special case of fuzzy queries limited to conditions of the form:

$$attribute \simeq constant$$

The distance between an attribute value and the ideal value is computed by mean of a difference (absolute value), after a normalization step (which yields domain values between 0 and 1). The overall distance is calculated by aggregating the elementary distances using a function which can be the minimum, the sum, or the Euclidean distance. The steps in the computation are the following:

1. using k , and taking into account both the chosen aggregation function and statistics about the considered relation, a threshold α over the global distance is deduced,
2. a Boolean query computing the desired α -cut or a superset of this α -cut is determined,

3. this query is processed and the score attached to every element of the result is calculated,
4. if at least k tuples with a score greater than or equal to α have been obtained, the k best are returned to the user; otherwise the procedure is run again (from step 2) using a lower threshold α .

For efficiently processing Top- k -queries in the context of relational databases, several algorithms have been proposed (e.g., Threshold Algorithm (TA) and No Random Access Algorithm (NRA) [Fagin et al., 2003], the Best Position Algorithm [Akbarinia et al., 2007], LPTA [Das et al., 2006], LPTA+ [Xie et al., 2013] and IV-Index [Xie et al., 2013]).

In the Semantic Web community, top- k -queries have raised a growing interest in the last few years [Bozzon et al., 2012, Magliacane et al., 2012, Dividino et al., 2012, Wang et al., 2015] for alleviating information overload problems. A major challenge is to make the processing of such queries efficient in a SPARQL-like setting.

Classical top- k -SPARQL queries can be expressed in SPARQL 1.1 by solution modifiers, such as, `ORDER BY` and `LIMIT` clauses, that respectively *order* the result set, and *limit* the number of results.

Example 28 The top- k -SPARQL query of Listing 2.4 aims to find the best five offers of albums ordered by a function of user ratings and offer date where g_1 and g_2 are scoring functions.◊

```
SELECT ?album ?offer (g1(?rating) + g2(?date) AS ?score) WHERE {
  ?album rdf:type mo:Album. ?album dc:rating ?rating.
  ?album dc:date ?date. ?album dc:hasOffers ?offer. }
ORDER BY DESC(?score) LIMIT 5
```

Listing 2.4: Standard top- k -SPARQL-query

Naive query processing then relies on a *materialize-then-sort* procedure which entails an evaluation of all the candidate answers (i.e., those satisfying the condition in the `WHERE` clause), followed by a computation of the ranking function for each of them, even if only a small number (typically, $k = 5$) of answers is requested. As a consequence, this processing strategy produces poor performances especially in the case of a large number of answers matching the selected query. A smart processing should stop as soon as the top- k results are returned.

In this respect, recent works have proposed solutions to optimize the evaluation of these queries. For instance, the authors of [Bozzon et al., 2011, Bozzon et al., 2012, Magliacane et al., 2012] introduced a SPARQL-RANK algebra which is an extension of the

SPARQL algebra [Pérez et al., 2006] and an incremental rank-aware execution model for top- k -SPARQL queries. This algebra enables splitting the scoring function that may be interleaved with other binary operators. The general objective is to derive an optimized query execution plan and reduce as much as possible the evaluation to a restricted number of answers.

[Bozzon et al., 2011] first applied this algebra to the processing of top- k SPARQL queries addressed to virtual RDF datasets through query rewriting using the rank-aware relational algebra presented in [Li et al., 2005]. Then, [Bozzon et al., 2012] proposed a detailed version of the SPARQL-RANK algebra, which can be applied to both RDBMS and native RDF datasets. They introduced a *rank-aware operator* denoted by ρ for evaluating a ranking criterion and redefined unary and binary operators (such as, selection (σ), join (\bowtie), union (\cup), difference (\setminus) and left joint (\bowtie_{\leftarrow})) for processing the ranked set of mappings in this context. New algebraic equivalence laws involving this operator have also been proposed. Among these equivalence laws we may find, pushing ρ over binary operators, splitting the criteria of a scoring function into a set of rank operators and using commutativity of ρ with itself.

In [Magliacane et al., 2012], an incremental execution model for the SPARQL-RANK algebra is proposed and a rank-aware SPARQL query engine denoted by ARQ-RANK based on this algebra is implemented. This engine efficiently improves the performance of top- k queries. Later, in [Zahmatkesh et al., 2014], the authors presented top- k DBPSB, an extension of DBPSB (DBpedia SPARQL benchmark) that makes it possible to automatically generate top- k queries from the queries of DBPSB and its datasets.

According to [Wang et al., 2015], the SPARQL-RANK algebra proposed by [Bozzon et al., 2011, Bozzon et al., 2012, Magliacane et al., 2012] suffers from frequent unnecessary input and output in the rank-join operation and this is seen as a drawback in the case of a large dataset. To deal with this issue, they proposed in [Wang et al., 2015] a graph-exploration-based method for efficiently processing top- k queries in crisp RDF graphs. They introduced a novel tree index called an MS-tree. Based on this MS-tree, candidate entities are constructed (ranked and filtered) in an appropriate way and the process immediately stops as soon as possible (i.e., as soon as the top- k answers are generated). In case of complex scoring functions, a cost-model-based optimization method is used in order to improve the query processing performance.

An evaluation of the approach with both synthetic and real-world datasets using SPARQL-RANK as a competitor is presented in the paper. The experimental results confirm that the model proposed in [Wang et al., 2015] significantly outperforms SPARQL-RANK approach in case of large datasets to be cached in memory.

From an RDF data model view, in [Dividino et al., 2012] the authors introduce an approach for top- k querying RDF data annotated with provenance information. In this context, annotations may concern the origin, history, truthfulness, or validity of an RDF statement. An annotated RDF statement is considered as a tuple $S = \langle \alpha : \theta_1, \dots, \theta_y \rangle$ with α being an RDF statement and $\theta_1, \dots, \theta_y$ being its annotations over a fixed set $\Gamma = \{p_1, \dots, p_y\}$ of independent annotation dimensions.

Example 29 Let us consider the RDF statement about music concerts shown in Table 2.1. Each statement is annotated by a set of dimensions $\Gamma = \{Time, Source, Certainty\}$.

Id	Statement	Dimensions		
		Time	Source	Certainty
#1	TAL playsIn Le Grand Rex	03.02.17	www.legrandrex.com	0.9
#2	KUNGS playsIn L'OLYMPIA	15.01.17	www.fnacspectacles.com	0.7
#3	TAL hasRating 7	10.01.17	www.itunes.apple.com	0.5
#4	KUNGS hasRating 8	08.02.17	www.itunes.apple.com	0.5

Table 2.1: The set of annotated RDF statements

The statement #1 says that the artist “TAL” plays in the “Le Grand Rex” and this information has been published on “03.02.17”, has “0.9” as a certainty degree, and was picked up from the Web site “www.legrandrex.com”. \diamond

The presence of multiple independent annotation dimensions in the query can induce different rankings of answers. In this regard, [Dividino et al., 2012] discusses the problem of preference aggregation (or judgement aggregation) and proposes a framework to aggregate all the annotation dimensions into a single joint ranking ordering using different aggregation methods. Finally, the authors of [Dividino et al., 2012] perform top- k querying using these ranking methods in offline (i.e., available results) and online (i.e., the aggregation of streaming data) settings.

2.1.2 Qualitative Approaches: Skyline-based Approaches

In the relational database domain, qualitative approaches to preference queries have attracted a large interest, in particular skyline queries [Borzsony et al., 2001], which aim to filter an n -dimensional dataset S according to a set of user preference relations and return only the tuples of S that are not dominated in the sense of Pareto order. Note that these approaches only yield a partial order, contrary to the quantitative ones.

Let us consider two tuples $t = (u_1, \dots, u_n)$ and $t' = (u'_1, \dots, u'_n)$ from S (reduced to the attributes on which a preference is expressed). The tuple t dominates (in the sense of Pareto order) the tuple t' , denoted by $t \succ t'$, iff t is at least good as t' in all dimensions and strictly

better than t' in at least one dimension. This may be represented by:

$$\begin{aligned} t \succ t' \Leftrightarrow & \forall i \in \{1, \dots, n\}, t.u_i \succeq_i t'.u'_i \text{ and} \\ & \exists j \in \{1, \dots, n\} \text{ such that } t.u_j \succ_j t'.u'_j \end{aligned} \quad (2.3)$$

Example 30 Let us assume that a user is looking for an album to listen to, and prefers an album which is *recent* and *high rated*. For every preference: *recent* (resp. *high rated*), the higher the date (resp. rating) is, the more preferred the tuple is. Consider three albums A_1 (date 2015, rating 5.8), A_2 (date 2013, rating 4) and A_3 (date 2014, rating 8). Album A_1 is more recent and has a higher rating than A_2 . So, A_1 dominates A_2 . Nevertheless, A_1 does not dominate A_3 since A_1 is more recent than A_3 but has a worse rating than A_3 . Hence, the skyline result is $\{A_1, A_3\}$. \diamond

In the literature, few works [Siberski et al., 2006, Gueroussova et al., 2013] have dealt with the expression and evaluation of skyline queries in a SPARQL-like language.

In [Siberski et al., 2006], Siberski *et al.* extend SPARQL with a PREFERRING clause in order to support the expression of multidimensional user preferences. This extension is based on the principle underlying skyline queries, i.e., it aims to find the nondominated objects.

The main syntax of this extension is as follows:

```
SELECT ...
WHERE ... {
FILTER (A OR B) }
PREFERRING P AND P' ... AND P*
```

Listing 2.5: Extension of SPARQL using Skyline

Two types of preferences may be distinguished: *Boolean* preferences where the answers that meet the condition are favored over those which do not, and *scoring* preferences (introduced by the keywords HIGHEST or LOWEST, where the elements with a higher value are favored over those with a lower value and vice versa).

Example 31 Let us consider that a user has the following preferences: (P_1) prefer the artists rated “excellent” over the “very good” ones (Boolean preference), (P_2) prefer the artist’s concert taking place between 9pm and 1am (Boolean preference) and (P_3) prefer the artist’s concert taking place the latest (scoring preference) provided that they are taking place between 9pm and 1am.

In the absence of a skyline functionality, one would use the classical SPARQL query of Listing 2.6 that returns those artists satisfying the Boolean conditions, ordered according to the starting time of their concert.

```

1 SELECT ?artist ?concert WHERE {
2   ?artist dc:concert ?concert. ?concert dc:starts ?startingTime.
3   ?concert dc:ends ?endingTime. ?artist dc:rating ?rating .
4   FILTER (?rating = ft:very-good || ?rating = ft:excellent) }
5   ORDER BY
6     DESC(?startingTime >= 9pm && ?endingTime <= 1am)
7     DESC(?startingTime)

```

Listing 2.6: Query in SPARQL (ordered answer)

As we can see, a classical skyline query can be expressed in SPARQL with the clauses **FILTER**, **ORDER BY** and **DESC**. However, the classical skyline query of Listing 2.6 also returns dominated artists, but only at the bottom of the list of answers.

In the extended SPARQL version of [Siberski et al., 2006], lines 5 to 7 of Listing 2.6 are replaced by:

```

5 PREFERRING
6   ?rating = ft:excellent
7 AND
8   (?startingTime >= 9pm && ?endingTime <= 1am)
9 CASCADE HIGHEST(?startingTime)

```

Listing 2.7: Skyline extension of SPARQL [Siberski et al., 2006]

Lines 1 to 4 represent the graph patterns and hard constraints. Line 6 corresponds to preference P_1 , line 8 corresponds to P_2 , and line 9 corresponds to P_3 . The **CASCADE** clause in line 9 specifies that P_3 is evaluated if and only if two answers are equivalent with respect to P_2 . \diamond

The authors of [Siberski et al., 2006] gave the semantics and the implementation of the new constructs aimed to compute a skyline query with SPARQL and extended the SPARQL implementation ARQ in order to process these types of queries. Nevertheless, no optimization aspects are discussed in the paper.

The approach proposed in [Guerousova et al., 2013] is based on [Siberski et al., 2006] and i) introduces user preferences in the **FILTER** clause, ii) replaces the **CASCADE** clause by a **PRIOR TO** clause in the spirit of Preference SQL [Kießling et al., 2011], iii) introduces new comparators for specifying atomic preferences: **BETWEEN**, **AROUND**, **MORE THAN**, and **LESS THAN**. This extension of SPARQL called PrefSPARQL supports not only the expression of qualitative preferences (skyline) but also conditional ones (if-then-else). A PrefSPARQL query returns a set of partially ordered tuples according to the satisfaction of the preferences.

Example 32 In order to illustrate the form taken by skyline queries in PrefSPARQL, let us consider again the query from Example 2.7. Listing 2.8 expresses this in PrefSPARQL. \diamond

```
SELECT ?artist ?concert WHERE {
  ?artist dc:concert ?concert. ?concert dc:starts ?startingTime.
  ?concert dc:ends ?endingTime. ?artist dc:rating ?rating .
PREFERRING ( ?rating = ft:excellent AND
(?startingTime BETWEEN (9pm, 1am) AND ?endingTime BETWEEN (9pm, 1am)
PRIOR TO HIGHEST (?endingTime)))}
```

Listing 2.8: Skyline query in PrefSPARQL

Example 33 So as to illustrate conditional preferences, let us now assume that a user prefers a concert which takes place after 7:30pm on the weekdays and before 7pm during the weekends, formulated in Listing 2.9. \diamond

```
SELECT ?concert WHERE {
?concert dc:day ?D. ?concert dc:starts ?startingTime.
PREFERRING
(IF (?D = ‘Saturday’ || ?D = ‘Sunday’)
THEN ?startingTime < 7pm ELSE ?startingTime >= 7:30pm)}
```

Listing 2.9: Conditional preference in PrefSPARQL

The authors of [Guerousova et al., 2013] show that PrefSPARQL preference queries can be expressed in SPARQL 1.0 and SPARQL 1.1 using an OPTIONAL clause or features available in SPARQL 1.1 such as NOT EXISTS. Nevertheless, they do not deal with implementation issues and query processing/optimization aspects.

In [Rosati et al., 2015], the authors are interested also in qualitative preferences but the preferences are represented by means of a CP-net. A CP-net (network of conditional preferences) has been earlier suggested by [Boutilier et al., 2004] for modeling relational database preference queries. It is a powerful graphical representation of statements that express conditional *ceteris paribus* (everything else being equal) preferences.

Example 34 Let us consider the following *ceteris paribus* preferences on clothes: i) P_1 : black (b) jackets are preferred to white (w) jackets, ii) P_2 : black (b) pants are preferred to white (w) pants, iii) P_3 : if the jackets and the pants are of the same color, red (r) shirts are preferred to white (w) ones; otherwise, white shirts are preferred. These preferences are modeled by means of the CP-net depicted in Figure 2.3, where J , P and S are binary variables corresponding to the colors of the jacket, the pants and the shirt respectively. \diamond

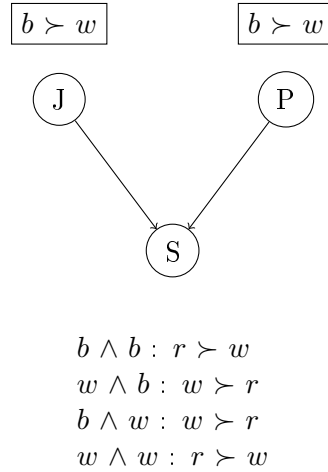


Figure 2.3: CP-net of Example 34

The authors of [Rosati et al., 2015] propose an RDF vocabulary to represent qualitative preference triples formulated under the *ceteris paribus* semantics.

Example 35 The RDF version of the CP-net representation of preference P_1 from Example 34 is as follows:

```

cp:jacket1 cp:color db:Black.
cp:jacket2 cp:color db:White.
cp:preference1 a cp:preference.
cp:preference1 cp:prefer cp:jacket1.
cp:preference1 cp:over cp:jacket2.

```

Listing 2.10: RDF version of the CP-net representation

Inspired by [Guerousova et al., 2013], the authors of [Rosati et al., 2015] present an algorithm to encode a CP-net into a standard SPARQL 1.1 query able to retrieve a ranked set of answers satisfying the user preferences. To the best of our knowledge, this work is the first attempt to translate the semantics of a CP-net into a SPARQL query.

Let us also mention that there exist some works (cf., [Chen et al., 2011]) that propose methods for the optimization of skyline queries in an RDF data context.

2.2 Query Relaxation

Nowadays, the size and the complexity of databases (including relational, semantic, etc.) increase over time at a sustained pace. In such circumstances, users when querying these databases do not have enough knowledge about their content and structure. So, they fail

sometimes to formulate meaningful queries to get the expected result or even to avoid empty responses.

In order to cope with these issues, some of the semantic Web systems include a query relaxation process for triple-pattern queries (i.e., addressed to data represented in the RDF format) sharing the same principle as the cooperative querying systems [Gaasterland et al., 1992] [Godfrey, 1997] [Chu et al., 1996] [Kleinberg, 1999] that operate on relational databases. These systems aim to automate the relaxation process of user queries when the selection criteria in the query do not make it possible to obtain answers that meet the user’s needs.

In a SPARQL/RDF setting, several works have been carried out [Hurtado et al., 2006, Hurtado et al., 2008, Huang et al., 2008, Poulouvassilis and Wood, 2010, Cali et al., 2014, Frosini et al., 2017] that propose a relaxation framework for RDF data through RDFS entailment using information provided by a given ontology (see Figure 2.4) and being characterized by RDFS inferences rules (see Table 2.2). These rules enable a generalization of the SPARQL query in order to release its conditions in case of an empty result.

Group A (<i>Subproperty</i>)	(1) $\frac{(a,sp,b)(b,sp,c)}{(a,sp,c)}$	(2) $\frac{(a,sp,b)(x,a,y)}{(x,b,y)}$
Group B (<i>Subclass</i>)	(3) $\frac{(a,sc,b)(b,sc,c)}{(a,sc,c)}$	(4) $\frac{(a,sc,b)(x,type,a)}{(x,type,b)}$
Group C (<i>Typing</i>)	(5) $\frac{(a,dom,c)(x,a,y)}{(x,type,c)}$	(6) $\frac{(a,range,d)(x,a,y)}{(y,type,d)}$

Table 2.2: RDFS Inferences Rules

Example 36 The rule (4) from Table 2.2 states that if a is a subclass of b and x is an instance of a , then, x is an instance of b .

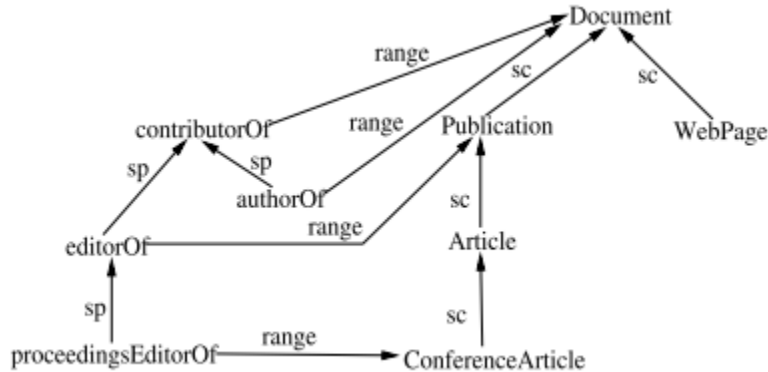


Figure 2.4: An RDFS Ontology

[Hurtado et al., 2006, Hurtado et al., 2008] is interested in the relaxation of a conjunctive fragment of queries over RDF data (e.g., See [Gutierrez et al., 2004, Haase et al., 2004]).

This type of queries has the following expression $H \leftarrow B$, where B is a graph pattern (i.e., a set of triples including URIs, literals, blanks nodes, and variables) and $H = \langle H_1, \dots, H_n \rangle$ is a list of variables. It firstly aims to find matchings from the graph pattern (i.e., the body of the query B) to the data and, secondly, applies these matchings to the head of the query (H) in order to get the final answers.

The authors propose to extend these conjunctive queries by introducing (one or several) **RELAX** clauses in the place of the **OPTIONAL** clauses. This extension is detailed in the following example.

Example 37 In order to avoid empty answers for some cases, a relaxation of some conditions using a specific ontology (see Figure 2.4) is needed. This ontology is represented in the form of an RDF graph based on an RDFS vocabulary that models documents along with properties that model different ways people contribute to them (e.g., as authors, editors, etc.).

Thanks to this ontology, the following query may be generalized and relaxed in the following way:

$$?Z, ?Y \leftarrow \{(?X, name, ?Z), \text{RELAX} \{(?X, proceedingsEditorOf, ?Y)\}\}.$$

The **RELAX** clause aims to return firstly editors of conference proceedings. Then, one can automatically rewrite the triple $(?X, proceedingsEditorOf, ?Y)$ into $(?X, editorOf, ?Y)$ or $(?X, contributorOf, ?Y)$ since *proceedingsEditorOf* is a *subproperty* of *editorOf* and *editorOf* is a *subproperty* of *contributorOf* according to the ontology and rules from Table 2.2. So, the relaxed query allows to obtain people who are editors of a publication or in a more general way contributors to a document. \diamond

The query relaxation strategy involves two types of relaxations:

- *simple type* without using an ontology, which includes dropping triple patterns using the **OPTIONAL** clause, replacing constants with variables in a triple pattern, etc.
- *more complex type* using an ontology and inference rules, which includes:
 - **Type relaxation** for example, following rule (4) from Table 2.2, the triple pattern $(?X, type, ConferenceArticle)$ can be relaxed into $(?X, type, Article)$ and then into $(?X, type, Publication)$ since we have $(ConferenceArticle, sc, Article) \in cl(O)$ and then $(Article, sc, Publication) \in cl(O)$;

- **Predicate relaxation** for example, using rule (2) from Table 2.2, the triple pattern $(?X, proceedingsEditorOf, ?Y)$ can be relaxed into $(?X, editorOf, ?Y)$ and then into $(?X, contributorOf, ?Y)$ since we have $(proceedingsEditorOf, sp, editorOf) \in cl(O)$ and then $(editorOf, sp, contributorOf) \in cl(O)$;
- **Predicate to domain relaxation** for example, using rule (5) from Table 2.2, the triple pattern (a, p, b) can be relaxed into the triple pattern $(a, type, c)$, since we have the triple pattern $(p, dom, c) \in cl(O)$.
- **Predicate to range relaxation** for example, using rule (6) from Table 2.2, the triple pattern $(JohnRobert, editorOf, ?Y)$ can be relaxed into $(?Y, type, Publication)$ since we have $(editorOf, range, Publication) \in cl(O)$.

For the purpose of incrementally computing the relaxed answers to the query, an algorithm is presented, which efficiently orders the answers according to how closely they meet the query conditions.

In, [Huang et al., 2008] the authors points out that the approaches proposed in [Hurtado et al., 2006, Hurtado et al., 2008] may still be insufficient. They propose a new similarity measure that requires computing the semantic similarity between the relaxed query and the original one. This measure makes it possible to reduce the number of answers as much as possible (or to the desired cardinality) and, then, ensure the quality of answers during the relaxation process.

More recently, [Reddy and Kumar, 2010] proposed an extension of the work [Huang et al., 2008] to the web of linked data, where they define an optimized query processing algorithm in which the relaxed queries are generated and answered on-the-fly during query execution (at run time). This work differs from the approach of [Huang et al., 2008], which is dedicated only to centralized RDF repositories and aims to generate multiple relaxed queries and execute them sequentially one by one.

Another related work is that by [Dolog et al., 2006, Dolog et al., 2009], where the authors present user centered process for automatically relaxing over-constrained RDF queries. This relaxation is carried out by rewriting rules for making patterns optional, replacing value, replacing patterns or predicate and deleting patterns or predicate. Background knowledge about the domain of interest and the preferences of the user are taken into account during the query relaxation to refine and guide this process.

From a different perspective, [Poulovassilis and Wood, 2010], introduce a framework wherein relaxations and approximations of regular path queries are combined in order to get a more flexible querying of RDF data when the user lacks knowledge of their structure.

[Frosini et al., 2017, Cali et al., 2014], rely on the work of [Poulovassilis and Wood, 2010] and propose a formal syntax and semantics of $SPARQL^{AR}$ which is an extension of the query language SPARQL 1.1 (i.e., SPARQL with property path queries) with query *approximation*

and query *relaxation* operators. A relaxation operator relies on RDF inference rules and follows the principle presented in [Hurtado et al., 2008] and the approximation operator aims to transform a regular expression pattern P into a new expression pattern P' using a set of edit operations (e.g., deletion, insertion and substitution).

In [Hogan et al., 2012], the authors base their relaxation framework on an industrial use-case from the European Aeronautic Defence and Space Company (EADS) that involve human observations which are presented in the form of natural language and may be imprecise or vague. They propose a conceptual framework to relax RDF queries relying on a matcher function (i.e., distance function) that assigns a relaxation score in $[0,1]$ to a pair of values.

Some contributions also address the problem of providing a guide for the user to relax his/her query. [Elbassuoni et al., 2011] propose a novel approach for query relaxation based on statistical language models (LMs) for structured RDF data in an automated way. This approach generates a set of relaxation candidates which can be derived from the RDF data and also from external sources like ontologies and textual documents.

From another angle, [Fokou et al., 2015, Fokou et al., 2017, Fokou et al., 2017] are inspired by some prior works in relational databases [Godfrey, 1997, Pivert and Smits, 2015] and recommendation systems [Jannach, 2009] and they deal with the problem of explaining the failure of RDF queries in order to help the user to relax his/her query. In [Fokou et al., 2014], the authors initially proposed an extension of SWDB (Semantic Web Database) query languages with new operators that allow to control the relaxation process. These operators describe the relaxation by specifying the part of the query to relax and the technique of relaxation to be used. Then, in [Fokou et al., 2015] [Fokou et al., 2017] the authors addressed the problem of computing the Minimal Failing Subqueries (MFS) and the Maximal Succeeding Subqueries (XSSs) (i.e., which return non-empty answers) that are used to find the parts of an RDF query that are responsible of the failure on the one hand, and the relaxed queries that are guaranteed to return a nonempty result on the other hand.

2.3 Approximate Matching

In the literature, the concept of graph isomorphism has been studied for a long time, cf., [Read and Corneil, 1977] [Fortin, 1996] and has as a principle to “determine if two given graphs are the same; if they are, find a matching (mapping) between them (i.e., which nodes from one graph correspond to which nodes in the other)”. Similarity measures based on graph matching are commonly used in this context. Essentially, queries are represented as a graph (called the *query graph*) and the aim is to find an appropriate matching between the *query graph* and the *resource graph*.

However, all of the existing classical graph isomorphism algorithms do not fit the semantic characteristics of RDF graphs (i.e., directed graphs with labeled edges and nodes) [Carroll, 2002]. Then, an efficient semantic similarity measure based on RDF graph is required. Therefore, few approaches have proposed new techniques dealing with approximate querying over RDF data [Zhu et al., 2002, De Virgilio et al., 2013, De Virgilio et al., 2015, Zheng et al., 2016].

[Zhu et al., 2002] introduces an approach for semantic search. The idea is to match RDF graphs in order to verify whether each *candidate resource RDF graph* matches the *query RDF graph*. The *resource RDF graph* is built up from a specific domain Web information and the *query RDF graph* corresponds to a user query. To do this, a new semantic similarity measure between two RDF graphs, based on an ontology has been defined. This measure takes into account the similarities between edges and also nodes.

The approach proposed in [Zhu et al., 2002] only takes the similarity of nodes and edges into account in an RDF graph but ignores the structure formed by the nodes and the edges.

To deal with this issue, [De Virgilio et al., 2013, De Virgilio et al., 2015] propose an approach dealing with approximate query answering in the context of large RDF data sets. This approach aims to measure the similarity between a portion of a (large) graph representing an RDF dataset and a sub-graph representing a query by applying substitutions and transformations to the paths of the latter. This operation is based on a scoring function that simulates the relevance of answers by taking into account two aspects: i) *quality* that measures how much the paths retrieved align with the paths in the query and ii) *conformity* that measures how much the combination of paths retrieved is similar to the combination of the paths in the query.

A more recent work is [Zheng et al., 2016], where the authors focus on the problem of Semantic SPARQL Similarity Search over RDF knowledge graphs. They propose a metric, *semantic graph edit distance* in order to measure the similarity between RDF graphs. This metric consider the graph structural, concept-level and semantic similarities in a uniform manner.

Conclusion

In this chapter, we reviewed several approaches from the literature that aim to query RDF data in a more expressive and flexible way, either by introducing fuzzy user preferences, relaxing some preferences or applying approximate matching. We present a summary of these approaches in Table 2.3.

Approach	SPARQL extension	Preference queries	Type	Navigational capabilities	Weighted RDF graph
[Cheng et al., 2010]	yes	yes	Fuzzy-set	no	no
[Wang et al., 2012]	yes	yes	Fuzzy-set	no	no
[Ma et al., 2016]	yes	yes	Fuzzy-set	no	no
[Buche et al., 2008]	no	yes	Fuzzy-set	no	Fuzzy annotations
[Cedeño and Candan, 2011]	yes	no	Top-k		Weighted edges
[Bozzon et al., 2012]	yes	yes	Top-k	no	no
[Magliacane et al., 2012]					
[Dividino et al., 2012]	no	yes	Top-k	no	Provenance information
[Wang et al., 2015]	yes	yes	Top-k	no	no
[Siberski et al., 2006]	yes	yes	Skyline	no	no
[Gueroussova et al., 2013]	yes	yes	Skyline	no	no
[Chen et al., 2011]	yes	yes	Skyline	no	no
[Rosati et al., 2015]	yes	yes	CP-net	no	no
[Hurtado et al., 2008]	yes	yes	Relaxation	no	no
[Huang et al., 2008]	yes	yes	Relaxation	no	no
[Reddy and Kumar, 2010]	yes	yes	Relaxation	no	no
[Poulovassilis and Wood, 2010]	yes	yes	Relaxation	Boolean	no
[Cali et al., 2014] [Frosini et al., 2017]	yes	yes	Relaxation	Boolean	no
[Dolog et al., 2006] [Dolog et al., 2009]	no	yes	Relaxation	no	no
[Fokou et al., 2015] [Fokou et al., 2017]	yes	yes	Relaxation	no	no
[Zhu et al., 2002]	no	yes	Approximate	no	no
[De Virgilio et al., 2013]	no	yes	Approximate	Boolean	no
[De Virgilio et al., 2015]					
[Zheng et al., 2016]	no	yes	Approximate	Boolean	no

Table 2.3: Main features of the preference query approaches

A first observation concerns the limited expressiveness of the approaches. Indeed, all of them are straightforward adaptations of proposals made in the relational database context: they make it possible to express preferences on the *values* of the nodes, but not on the **structure of the RDF graph** (structural preferences may concern the *strength* of a path, the *centrality* of nodes, etc). Some of the relaxation approaches (e.g., [Poulovassilis and Wood, 2010], [Calì et al., 2014] and [Frosini et al., 2017]), and approximation approaches (e.g., [De Virgilio et al., 2013], [De Virgilio et al., 2015] and [Zheng et al., 2016]) have considered this issue but only in a crisp way.

A second important remark is that all of the approaches presented above only deal with crisp RDF data. However, we believe that there is a real need for a flexible SPARQL that takes into account RDF graphs where data is described by intrinsic weighted values, attached to edges or nodes. This weight may denote any gradual notion like a cost, a truth value, an intensity or a membership degree.

The RDF data model should thus be enriched in order to represent gradual information, and new query languages should be defined. A first step in this direction is the approach proposed in [Cedeño and Candan, 2011] where the authors propose an extension of the RDF model embedding weighted edges and an extension of SPARQL to support this feature, allowing new path predicates to express nodes reachability and the ability to express ranked queries. This approach takes the weights into account in order to rank the answers, but does not propose any means to express preferences in user queries.

To the best of our knowledge, none of the existing approaches aims to define a general purpose flexible version of SPARQL to weighted RDF databases, which is the first contribution of this thesis.

Chapter 3

FURQL: An extension of SPARQL with Fuzzy Navigational Capabilities

Contents

Introduction	61
3.1 Fuzzy RDF (F-RDF) Graph	62
3.2 FUZZY RDF Query Language (FURQL)	67
3.2.1 Syntax of FURQL	68
3.2.2 Semantics of FURQL	71
Conclusion	78

Introduction

As we stated in the previous chapter, RDF is a graph-based standard data model for representing semantic web information, and SPARQL is a standard query language for querying RDF data. Because of the huge volume of linked open data published on the web, these standards have aroused a large interest in the last years.

In the literature, several types of approaches have been devoted to extending the SPARQL language among which: i) those that extend the research patterns with paths involving regular expressions, ii) those that consider fuzzy conditions. However, to the best of our knowledge, no approach cover both aspects at the same time.

In this chapter, we intend to tackle this issue and we propose the FURQL query language which is a fuzzy extension of SPARQL that improves its expressiveness and usability. This extension allows (1) to query both *crisp* and *fuzzy RDF data model*, and (2) to express *fuzzy preferences* on values present in the graph as well as on the *structure* of the data graph, which has not been proposed in any previous fuzzy extension of SPARQL.

This work has been published in the proceedings of the 25th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'16), Vancouver, Canada, 2016.

In the following, in Section 3.1, we first present the notion of the *fuzzy RDF data model* and then, in Section 3.2, we provide the syntax and the semantics of the FURQL query language.

3.1 Fuzzy RDF (F-RDF) Graph

The classical crisp RDF model is only capable of representing Boolean notions whereas real-world concepts are often of a vague or gradual nature. This is why several authors have proposed fuzzy extensions of the RDF model. Throughout the thesis, we consider the data model based on Definition 4 which synthesizes the existing fuzzy RDF models of literature ([Mazzieri and Dragoni, 2005], [Udrea et al., 2006], [Mazzieri and Dragoni, 2008], [Lv et al., 2008], [Straccia, 2009], [Udrea et al., 2010], [Zimmermann et al., 2012]), whose common principle consists in adding a fuzzy degree to edges, modeled either by a value embedded in each triple or by a function associating a satisfaction degree with each triple, expressing the extent to which the fuzzy concept attached to the edge is satisfied.

Example 38 [Fuzzy RDF triple] The corresponding fuzzy RDF triple “(`Beyonce`, `recommends`, `Euphoria`), 0.8)” states that $\langle \text{Beyonce}, \text{recommends}, \text{Euphoria} \rangle$ is satisfied to the degree 0.8, which could be interpreted as *Beyonce strongly recommends Euphoria*. \diamond

Definition 4 (Fuzzy RDF (F-RDF) graph). *A F-RDF graph is a tuple (\mathcal{T}, ζ) such that (i) \mathcal{T} is a finite set of triples of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$, (ii) ζ is a membership function on triples $\zeta : \mathcal{T} \rightarrow [0, 1]$.*

According to the classical semantics associated with fuzzy graphs, $\zeta(t)$ qualifies the intensity of the relationship involved in the statement t . Intuitively, ζ attaches fuzzy degrees to the edges of the graph. Having a value of 0 for ζ is equivalent to not belonging to the graph. Having a value of 1 for ζ is equivalent to fully satisfying the associated concept. In the graph G_{MB} of Figure 3.1, such edges appear as classical ones, i.e., with no degree attached.

The fuzzy degrees associated with edges are given or calculated. A simple case is when, each degree is based on a simple statistical notion, e.g., the intensity of friendship between two artists may be computed as the number of their common friends over the total number of friends with respect to each artist.

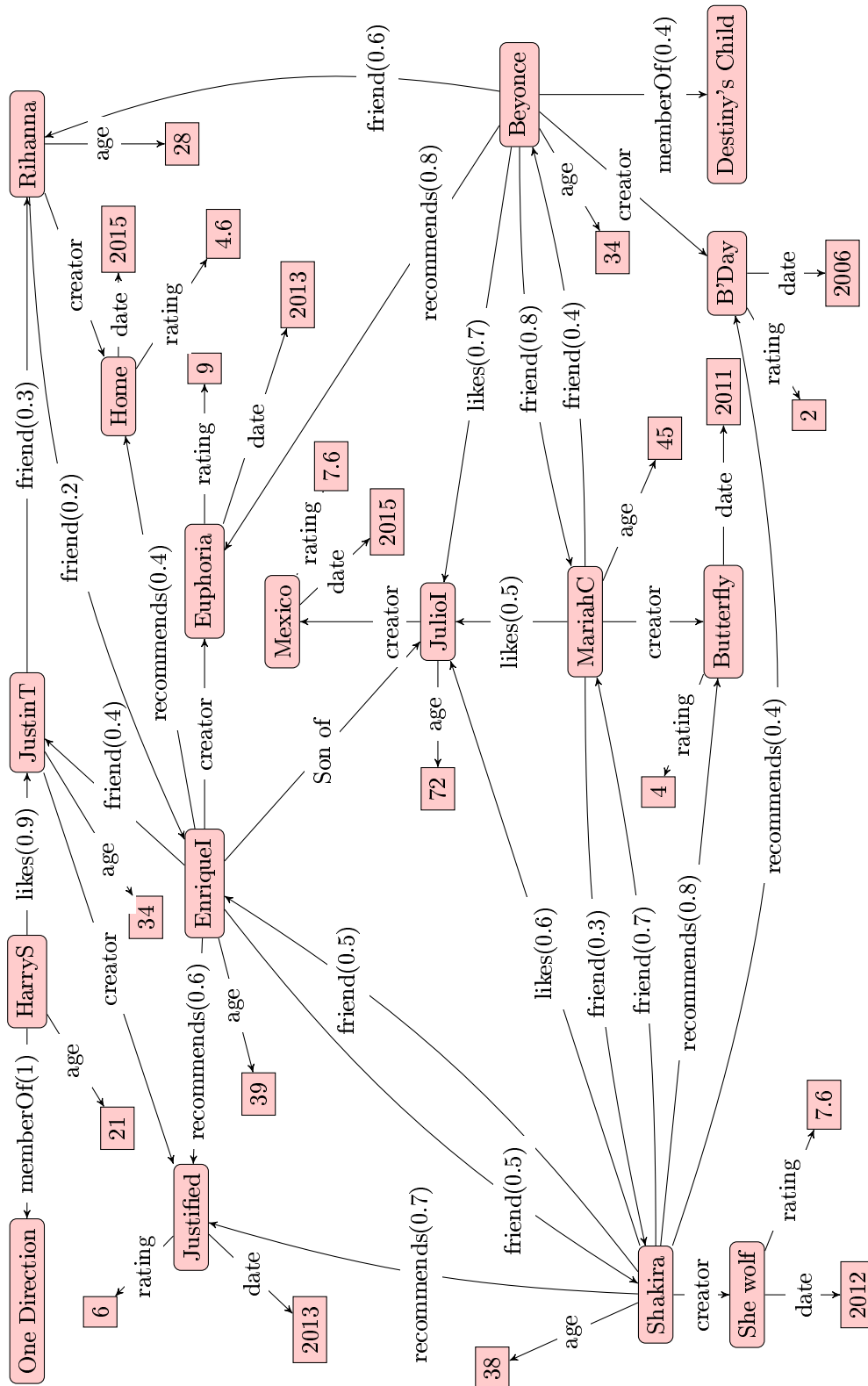


Figure 3.1: Fuzzy RDF graph G_{MB} inspired by MusicBrainz

Remark 4. A classical crisp RDF data graph is a special case of an F-RDF data graph where the co-domains of ζ are $\{0,1\}$. A fundamental implication is that the concepts and the flexible query language defined over an F-RDF graph in the following, remain relevant over a classical RDF graph.

Remark 5. In the same way as the RDF graph, an F-RDF graph is said to be ground if it contains no blank nodes. Such a graph may be ground at the beginning or made ground e.g. by a skolemization procedure. In the following, we only consider ground fuzzy RDF graphs.

Example 39 [Fuzzy RDF graph] Figure 3.1 is an example of a fuzzy RDF graph inspired by MusicBrainz¹. This graph, denoted by G_{MB} in the following, mainly contains artists and albums as nodes. For readability reasons, each URI node contains the value of its *name* instead of the URI itself. Literal values may be attached to an URI, like the age of an artist, the release date or the global rating of an album. The graph contains fuzzy relationships (e.g., *friend*, *likes*, *recommends*, *memberOf*) as well as crisp ones (e.g., *creator*, *date*, ...). We limit our example to some entities including artists and albums and omit URI prefixes to avoid overcrowding the figure.

In order to create this graph, we started from a MusicBrainz nonfuzzy subgraph for which every relationship between nodes was Boolean and, then, we made it fuzzy by adding satisfaction degrees denoting the intensity of some relationships. Here for instance,

- the degree associated with an edge of the form $Art1 - friend \rightarrow Art2$ is the proportion of common friends (i.e., Boolean relationship) between $Art1$ and $Art2$ over the total number of friends of $Art1$;
 - the degree associated with an edge of the form $Art - memberOf \rightarrow Group$ is the number of years the artist stayed in this group over the number of years this group has been existing;
 - the degree associated with an edge of the form $Art1 - likes \rightarrow Art2$ is the number of albums by $Art2$ that $Art1$ has liked over the total number of albums by $Art2$;
 - the degree associated with an edge of the form $Art - recommends \rightarrow Alb$ is the number of stars given by Art to Alb over the maximum number of stars.
- ◇

In the following, we rely on classical notions from fuzzy graph theory [Rosenfeld, 2014], which are the *path*, the *distance* and the *strength* (ST) of the connection between two nodes respectively given in Definitions 5, 6 and 7.

¹<https://musicbrainz.org/>

Definition 5 (Path between two nodes). *Let G be an F-RDF data graph.*

Classically, a path p in G corresponds to a possibly empty sequence of triples $(t_1, \dots, t_k, \dots, t_n)$ such that $\{t_i \mid 1 \leq i \leq n\} \subseteq G$ and for all $1 \leq k \leq n - 1$, the object of t_k is the subject of t_{k+1} .

Given two nodes x and y , $Paths(x, y)$ denotes the set of cycle-free paths² in G connecting x to y , i.e., the set of paths of the form $(t_1, \dots, t_k, \dots, t_n)$ such that x is the subject of t_1 and y is the object of t_n .

Example 40 [Path between two nodes] The (cycle free) paths between the nodes *Beyonce* and *Euphoria* from the fuzzy RDF graph G_{MB} of Figure 3.1 are shown in Figure 3.2.

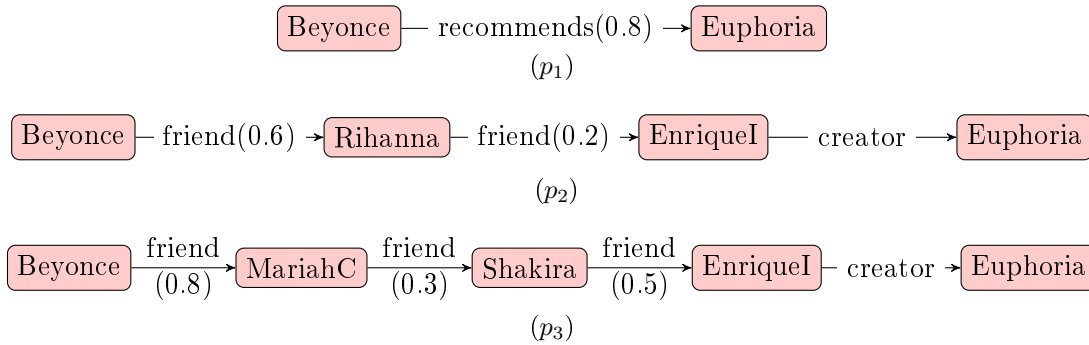


Figure 3.2: Cycle-free paths from G_{MB} connecting Beyonce to Euphoria

Definition 6 (Distance between two nodes). *The distance between two nodes x and y is defined by*

$$\text{distance}(x, y) = \min_{p \in Paths(x, y)} \text{length}(p) \quad (3.1)$$

where $\text{length}(p)$ is the length of a path p in a fuzzy graph [Rosenfeld, 2014], defined by

$$\text{length}(p) = \sum_{t \in p} \frac{1}{\zeta(t)}. \quad (3.2)$$

The distance between two nodes is the length of the shortest path between these two nodes.

Remark 6. *In a crisp RDF graph (when $\zeta(t) \in \{0, 1\}$), which is a special case of a fuzzy RDF graph, the distance between two nodes x and y given in Definition 6 is still valid and it expresses the number of edges between these nodes (which corresponds to the classical definition).*

²Considering paths containing a cycle would not change the result of the following expressions (3.1) and (3.3).

Definition 7 (Strength between two nodes). *The strength between two nodes x and y is defined by*

$$ST(x, y) = \max_{p \in Paths(x, y)} ST_path(p) \quad (3.3)$$

where $ST_path(p)$ is the strength of the path connecting x and y in a fuzzy graph [Rosenfeld, 2014], defined by

$$ST_path(p) = \min(\{\zeta(t) | t \in p\}) \quad (3.4)$$

The strength of a path is defined to be the weight of the weakest edge of the path.

Example 41 [Distance and strength between two nodes] Let us consider the cycle-free paths from G_{MB} connecting Beyonce to Euphoria, depicted in Figure 3.2, and let us compute the *distance* and the *strength* between the pair of nodes (*Beyonce*, *Euphoria*).

The *distance* between the pair of nodes (*Beyonce*, *Euphoria*) is calculated as follows

$$distance(Beyonce, Euphoria) = \min (length(p_1), length(p_2), length(p_3)),$$

with $length(p_1) = 1/\zeta(Beyonce, recommends, Euphoria) = 1/0.8 = 1.25$,

$$\begin{aligned} length(p_2) &= 1/\zeta(Beyonce, friend, Rihanna) + 1/\zeta(Rihanna, friend, EnriqueI) \\ &\quad + 1/\zeta(EnriqueI, creator, Euphoria) \\ &= 1/0.6 + 1/0.2 + 1 = 7.7, \text{ and} \end{aligned}$$

$$\begin{aligned} length(p_3) &= 1/\zeta(Beyonce, friend, MariahC) + 1/\zeta(MariahC, friend, Shakira) \\ &\quad + 1/\zeta(Shakira, friend, EnriqueI) + 1/\zeta(EnriqueI, creator, Euphoria) \\ &= 1/0.8 + 1/0.3 + 1/0.5 + 1 = 7.5. \end{aligned}$$

Finally, the length of the shortest path between the pair of nodes (*Beyonce*, *Euphoria*) is $distance(Beyonce, Euphoria) = 1.25$.

The *strength* between the pair of nodes (*Beyonce*, *Euphoria*) is calculated as

$$ST(Beyonce, Euphoria) = \max (ST_path(p_1), ST_path(p_2), ST_path(p_3)),$$

with $ST_path(p_1) = \zeta(Beyonce, recommends, Euphoria) = 0.8$,

$$\begin{aligned} ST_path(p_2) &= \min(\zeta(Beyonce, friend, Rihanna), \zeta(Rihanna, friend, EnriqueI), \\ &\quad \zeta(EnriqueI, creator, Euphoria)) \\ &= \min(0.6, 0.2, 1) = 0.2, \text{ and} \end{aligned}$$

$$\begin{aligned}
ST_path(p_3) &= \min(\zeta(Beyonce, friend, MariahC), \zeta(MariahC, friend, Shakira), \\
&\quad \zeta(Shakira, friend, EnriqueI), \zeta(EnriqueI, creator, Euphoria)) \\
&= \min(0.8, 0.3, 0.5, 1) = 0.3.
\end{aligned}$$

Thus, the *strength* between the pair of nodes $(Beyonce, Euphoria)$ is $ST(Beyonce, Euphoria) = 0.3$.

Here, the *distance* and the *strength* correspond to the same *path*, but it is of course not necessarily the case in general. \diamond

Let us also mention that except for introducing the degree of truth within an RDF triple in case of imprecise information, several other extensions of RDF were proposed in the literature in order to deal with:

- **time** ([Gutierrez et al., 2007], [Pugliese et al., 2008], [Tappolet and Bernstein, 2009]) to represent the validity periods of time of the information brought by the triple defined by an interval (containing the start and the end point of validity of this information),
- **trust** [Hartig, 2009], used in case of uncertainty about the trustworthiness of the RDF triples. It is represented by a *trust value* which is either unknown or a value in the interval $[-1, 1]$, where -1 encodes a full disbelief in the triple, 1 a total belief in the triple and 0 signifies the lack of belief as well as the lack of disbelief; and,
- **provenance** [Dividino et al., 2009]: may contain information attached to an RDF triple (such as, origins/source (Where is this information from?), authorship (Who provided the information?), time (When was this information provided?), and others).

Moreover, [Udrea et al., 2010] and [Zimmermann et al., 2012] provided a single theoretical framework to handle the aforementioned extensions along with an extension of the RDF query language to deal with such a framework.

3.2 FUZZY RDF Query Language (FURQL)

In this section, we introduce the FURQL query language, and we formally study its expressiveness. FURQL is based on the notion of *fuzzy graph pattern*, which is a fuzzy extension of the *SPARQL graph pattern* notion introduced in [Pérez et al., 2009] and [Arenas and Pérez, 2011] which present it in a more traditional algebraic formalism than the official syntax does [W3C, 2014]. In the following, we redefined the associated syntax and semantics in order to introduce fuzzy preferences expressed over the F-RDF data model of Definition 4.

3.2.1 Syntax of FURQL

FURQL (Fuzzy RDF Query Language) consists in extending SPARQL graph patterns into *fuzzy graph patterns*. Before formally introducing the syntax of FURQL, we first need to define the notion of a *fuzzy graph pattern*.

A *fuzzy graph pattern* allows to express fuzzy preferences on the entities of an F-RDF graph (through fuzzy conditions) and on the structure of the graph (through fuzzy regular expressions). It considers the following binary operators: AND (SPARQL `concatenation`), UNION (SPARQL `UNION`), OPT (SPARQL `OPTIONAL`) and FILTER (SPARQL `FILTER`). We fully parenthesize expressions making explicit the precedence and association of operators.

In the following, we assume the existence of an infinite set \mathcal{V} of variables such that $\mathcal{V} \cap (\mathcal{U} \cup \mathcal{L}) = \emptyset$. By convention, we prefix the elements of \mathcal{V} by a question mark symbol.

Let us first define the notion of a *fuzzy regular expression*.

Definition 8 (Fuzzy regular expression). *The set \mathcal{F} of fuzzy regular expression patterns, defined over the set \mathcal{U} of URIs, is recursively defined by:*

- ϵ is a fuzzy regular expression of \mathcal{F} ;
- $u \in \mathcal{U}$ and $'_'$ are fuzzy regular expressions of \mathcal{F} ;
- if $A \in \mathcal{F}$ and $B \in \mathcal{F}$ then $A|B, A.B, A^*, A^{cond}$ are fuzzy regular expressions of \mathcal{F} .

Above, ϵ denotes the empty pattern, the character $'_'$ denotes any element of \mathcal{U} , $A|B$ denotes alternative expressions, $A.B$ denotes the concatenation of expressions, A^* stands for the classical repetition of an expression (the Kleene closure), A^{cond} denotes paths satisfying the pattern A with a condition $cond$ where $cond$ is a Boolean combination of atomic formulas of the form: $sprop$ IS $Fterm$ where $sprop$ is a structural property of the path defined by the expression and $Fterm$ denotes a predefined or user-defined fuzzy term like *short* (see Figure 3.3). In the following, we limit the path structural properties to *ST* (see Definition 7) and *distance* (see Definition 6). Examples of conditions of this form are *distance* IS *short* and *ST* IS *strong*. We denote by A^+ the classical shortcut for $A.A^*$.

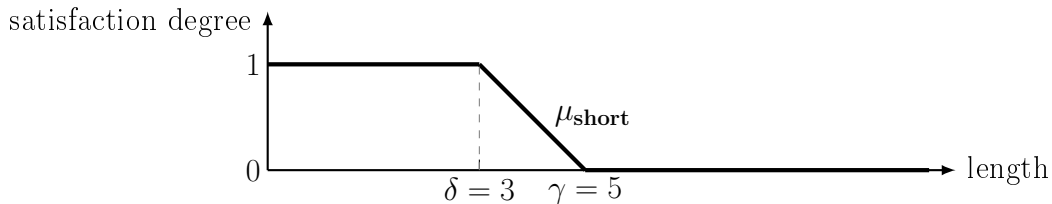


Figure 3.3: A possible representation of the fuzzy term *short*

Definition 9. [Fuzzy graph pattern] *Fuzzy graph patterns are recursively defined by:*

- A fuzzy triple from $(\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{F} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V})$ is a fuzzy graph pattern.
- If P_1 and P_2 are fuzzy graph patterns then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$ and $(P_1 \text{ OPT } P_2)$ are fuzzy graph patterns.
- If P is a fuzzy graph pattern and C is a fuzzy condition then $(P \text{ FILTER } C)$ is a fuzzy graph pattern. A fuzzy condition is a logical combination of fuzzy terms defined by:
 - if $\{?x, ?y\} \subseteq \mathcal{V}$ and $c \in (\mathcal{U} \cup \mathcal{L})$, then $\text{bound}(?x)$, $?x \theta c$ and $?x \theta ?y$ are fuzzy conditions, where θ is a fuzzy or crisp comparator,
 - if $?x \in \mathcal{V}$ and $F\text{term}$ is a fuzzy term then, $?x \text{ IS } F\text{term}$ is a fuzzy condition,
 - if C_1 and C_2 are fuzzy conditions then $(\neg C_1)$ and $(C_1 \odot C_2)$ (where \odot is a fuzzy connective) are fuzzy conditions. Fuzzy connectives include of course fuzzy conjunction \wedge (resp. disjunction \vee), usually interpreted by the triangular norm minimum (resp. maximum), but also many other operators that may be used for expressing different kinds of trade-offs, such as the weighted conjunction and disjunction [Dubois and Prade, 1986], mean operators, fuzzy quantifiers [Fodor and Yager, 2000], or the non-commutative connectives described in [Bosc and Pivert, 2012].

Given a pattern P (which can be a fuzzy triple pattern in particular), $\text{var}(P)$ denotes the set of variables occurring in P .

Example 42 [Fuzzy graph pattern] Let us consider $P_{\text{rec_low}}$ the fuzzy graph pattern defined by $(?Art1, (\text{friend}^+)^{\text{distance is short}}.\text{creator}, ?Alb) \text{ AND } (?Art1, \text{recommends}, ?Alb) \text{ AND } ((?Alb, \text{rating}, ?r) \text{ FILTER } (?r \text{ IS } \text{low}))$, of which Figure 6.3 is a graphical representation.

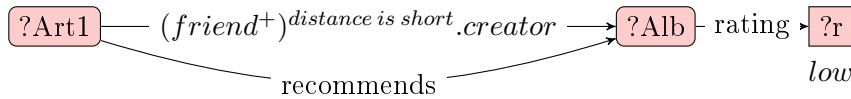


Figure 3.4: Graphical representation of the pattern $P_{\text{rec_low}}$

Intuitively, $P_{\text{rec_low}}$ retrieves the list of artists ($?Art1$) in G_{MB} who recommend a low rated album ($?Alb$) created by another artist who is a close friend of theirs ($?Art1$). \diamond

Syntactically, FURQL naturally extends SPARQL, by allowing the occurrence of *fuzzy graph patterns* (which may contain fuzzy regular expressions) in the `WHERE` clause and the occurrence of fuzzy conditions in the `FILTER` clause. A fuzzy regular expression is close to a property path, as defined in SPARQL 1.1 [Harris and Seaborne, 2013], but involves a fuzzy structural property (e.g. *distance* and *strength* over fuzzy graphs).

The general syntactic form of a FURQL query is given in Listing 3.1.

```

DEFINE ...
SELECT ?res WHERE {
  P [ FILTER C ] }
CUT  $\alpha$ 

```

Listing 3.1: Syntax of a FURQL query

A FURQL query is composed of:

1. a list of `DEFINE` clauses that makes it possible to define the fuzzy terms. If a fuzzy term `fterm` has a trapezoidal function defined by the quadruple $(A-a, A, B$ and $B+b)$ — meaning that its support is $[A-a, B+b]$ and its core $[A, B]$ —, then the clause has the form `DEFINE fterm AS (A-a,A,B,B+b)`. If `fterm` is a decreasing function, like the term *low* of Figure 3.5, then, the clause has the form `DEFINEDESC fterm AS (δ, γ)` (there is the corresponding `DEFINEASC` clause for increasing functions).
2. a `SELECT` clause that specifies which variables should be returned in the result set,
3. a `WHERE` clause of the form `WHERE P [FILTER C]` with `P [FILTER C]` is a fuzzy graph pattern,
4. an (optional) `CUT` clause, of the form `CUT α` which keep in the result set only the answers that have a satisfaction degree greater or equal to α .

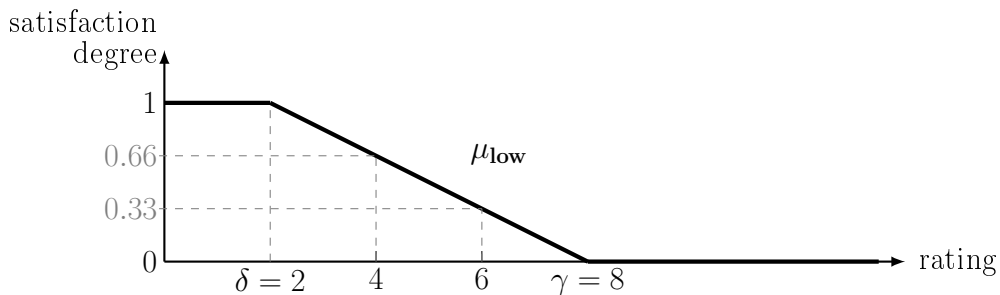


Figure 3.5: Representation of the fuzzy term *low* applied to a *rating* value

Example 43 [FURQL query] The FURQL query of Listing 3.2 retrieves artists that recommend low-rated albums by close friends (see Pattern P_{rec_low} of Example 42), and performs an alpha-cut on the answers (only those having a satisfaction degree greater or equal to 0.4 are kept). The `CUT` clause is of course optional (its default value is 0^+ , i.e., one keeps only the answers that have a nonzero degree).

```

1  DEFINEDESC low AS (2, 8)
2  DEFINEASC short AS (3, 5)
3  SELECT ?art1 WHERE {
4    { ?art1 (friend+ | distance IS short) ?art2 .
5      ?art2 creator ?alb .
6      ?alb rating ?r .
7      ?art1 recommends ?alb . }
8  FILTER (?r IS low)
9  } CUT 0.4

```

Listing 3.2: A FURQL query containing P_{rec_low}

In this example, the `DEFINEDESC` clause of line 1 defines the fuzzy term *low* of Figure 3.5, and the following clause defines the fuzzy term *short* of Figure 3.3. The pattern from lines 3 to 8 is the fuzzy pattern of Example 42. Line 9 specifies an α -cut of the *fuzzy pattern* with a satisfaction degree greater or equal to 0.4. \diamond

3.2.2 Semantics of FURQL

To define the semantics of FURQL, we need to define the semantics of fuzzy graph patterns. Intuitively, given an F-RDF data graph G , the semantics of a fuzzy graph pattern P defines a set of mappings, where each mapping (from $var(P)$ to URIs and literals of G) maps the pattern to an isomorphic subgraph of G . For introducing such a concept, the notion of satisfaction of a fuzzy regular expression must first be defined.

Definition 10 (Fuzzy regular expression matching of a path). *Let $G = (\mathcal{T}, \zeta)$ be an F-RDF graph and exp be a fuzzy regular expression. Let $p = (\langle s_1, p_1, o_1 \rangle, \dots, \langle s_n, p_n, o_n \rangle) \subseteq G$ be a path of G . The statement “ p satisfies exp with a satisfaction degree of $sat_{exp}(p)$ ” is defined as follows, according to the form of exp (in the following, f , f_1 and f_2 are fuzzy regular expressions):*

- *exp is of the form ϵ .*
If p is empty then $sat_{exp}(p) = 1$ else $sat_{exp}(p) = 0$.
- *exp is of the form $u \in \mathcal{U}$ (resp. “_”).*
If p_1 is u (resp. any $u \in \mathcal{U}$) then $sat_{exp}(p) = \zeta(\langle s_1, p_1, o_1 \rangle)$ else 0.

- *exp* is of the form $f_1.f_2$.
Let P be the set of all pairs of paths (p_1, p_2) s.t. p is of the form p_1p_2 . One has $sat_{exp}(p) = \max_P(\min(sat_{f_1}(p_1), sat_{f_2}(p_2)))$.
- *exp* is of the form $f_1 \cup f_2$.
One has $sat_{exp}(p) = \max(sat_{f_1}(p), sat_{f_2}(p))$.
- *exp* is of the form f^* .
If p is the empty path then $\mu_{exp}(p) = 1$. Otherwise, we denote by P the set of all tuples of paths (p_1, \dots, p_n) ($n > 0$) s.t. p is of the form $p_1 \dots p_n$. One has $sat_{exp}(p) = \max_P(\min_{i \in [1..n]}(sat_{exp}(p_i)))$.
- *exp* is of the form f^{Cond} where *Cond* is a fuzzy condition.
 $sat_{exp}(p) = \min(sat_f(p), \mu_{Cond}(p))$ where $\mu_{Cond}(p)$ denotes the degree of satisfaction of *cond* by p .

Again, not satisfying is equivalent to getting a degree of 0.

Definition 11 (Satisfaction of a fuzzy regular expression by a pair of nodes). Let $G = (\mathcal{T}, \zeta)$ be an *F-RDF* graph and *exp* be a fuzzy regular expression. Let (x, y) be a pair of nodes of G . The statement “the pair (x, y) satisfies *exp* with a satisfaction degree of $sat_{exp}(x, y)$ ” is defined by

$$sat_{exp}(x, y) = \max_{p \in Paths(x, y)} sat_{exp}(p).$$

Note that only cycle-free paths need to be considered in order to compute the satisfaction degree.

Example 44 [Satisfaction of a fuzzy regular expression] Let us consider the following fuzzy regular expressions, for which we give their satisfaction degree according to G_{MB} in Figure 3.1. Note that the paths represented in Figure 3.6 are some cycle-free paths among many others from the graph G_{MB} .

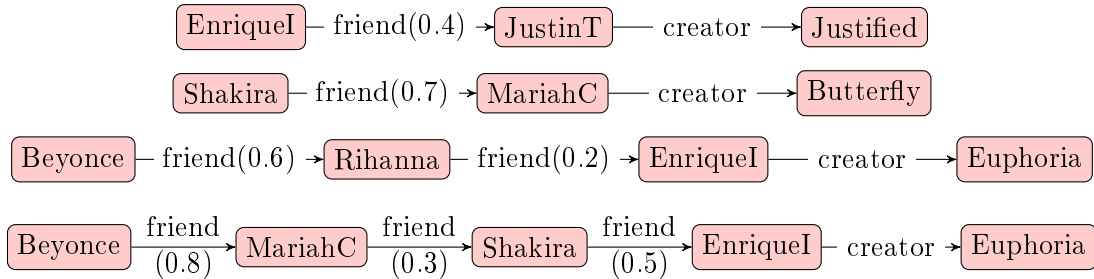


Figure 3.6: Some paths from G_{MB} .

Expression $f_1 = (\text{friend}^+).\text{creator}$ is a fuzzy regular expression. A pair of nodes (x, y) satisfies f_1 if x has a “friend-linked” artist (an artist connected to x with a path made of *friend* edges), that created the album y . All of the pairs of nodes $(\text{EnriqueI}, \text{Justified})$, $(\text{Shakira}, \text{Butterfly})$, $(\text{Beyonce}, \text{Euphoria})$, $(\text{Rihanna}, \text{Euphoria})$, $(\text{MariahC}, \text{Euphoria})$ and $(\text{Shakira}, \text{Euphoria})$, illustrated in Figure 3.6, satisfy f_1 with the following satisfaction degrees:

$$\begin{aligned} \text{sat}_{f_1}(\text{EnriqueI}, \text{Justified}) &= \min(\zeta(\text{EnriqueI}, \text{friend}, \text{JustinT}), \zeta(\text{JustinT}, \text{creator}, \text{Justified})) \\ &= \min(0.4, 1) = 0.4, \end{aligned}$$

$$\begin{aligned} \text{sat}_{f_1}(\text{Shakira}, \text{Butterfly}) &= \min(\zeta(\text{Shakira}, \text{friend}, \text{MariahC}), \zeta(\text{MariahC}, \text{creator}, \text{Butterfly})) \\ &= \min(0.7, 1) = 0.7, \end{aligned}$$

$$\begin{aligned} \text{sat}_{f_1}(\text{Beyonce}, \text{Euphoria}) &= \max(\min(\zeta(\text{Beyonce}, \text{friend}, \text{Rihanna}), \zeta(\text{Rihanna}, \text{friend}, \text{EnriqueI}), \\ &\quad \zeta(\text{EnriqueI}, \text{creator}, \text{Euphoria})), \min(\zeta(\text{Beyonce}, \text{friend}, \text{MariahC}), \\ &\quad \zeta(\text{MariahC}, \text{friend}, \text{Shakira}), \zeta(\text{Shakira}, \text{friend}, \text{EnriqueI}), \\ &\quad \zeta(\text{EnriqueI}, \text{creator}, \text{Euphoria}))) \\ &= \max(\min(0.6, 0.2, 1), \min(0.8, 0.3, 0.5, 1)) = 0.3, \end{aligned}$$

$$\begin{aligned} \text{sat}_{f_1}(\text{Rihanna}, \text{Euphoria}) &= \min(\zeta(\text{Rihanna}, \text{friend}, \text{EnriqueI}), \zeta(\text{EnriqueI}, \text{creator}, \text{Euphoria})) \\ &= \min(0.2, 1) = 0.2, \end{aligned}$$

$$\begin{aligned} \text{sat}_{f_1}(\text{MariahC}, \text{Euphoria}) &= \min(\zeta(\text{MariahC}, \text{friend}, \text{Shakira}), \zeta(\text{Shakira}, \text{friend}, \text{EnriqueI}), \\ &\quad \zeta(\text{EnriqueI}, \text{creator}, \text{Euphoria})) \\ &= \min(0.3, 0.5, 1) = 0.3, \text{ and} \end{aligned}$$

$$\begin{aligned} \text{sat}_{f_1}(\text{Shakira}, \text{Euphoria}) &= \min(\zeta(\text{Shakira}, \text{friend}, \text{EnriqueI}), \zeta(\text{EnriqueI}, \text{creator}, \text{Euphoria})) \\ &= \min(0.5, 1) = 0.5. \end{aligned}$$

Expression $f_2 = (\text{friend}^+)^{\text{distance is short}}.\text{creator}$ is a fuzzy regular expression. A pair of nodes (x, z) satisfies f_2 if x has a “close” friend artist y that created an album z , “close” meaning that x is connected to y by a *short* path made of *friend* edges (the term *short* is defined in Figure 3.3 on page 68). It is worth noticing that expression f_1 is a sub-expression of expression f_2 , so we are going to make use of the satisfaction degree of f_1 , denoted by sat_{f_1} , in order to calculate the satisfaction degree of f_2 , denoted by sat_{f_2} .

According to the paths depicted in Figure 3.6:

- the *length* of pair $(EnriqueI, Justified) = 1/0.4 + 1 = 3.5$, $\mu_{short}(3.5) = 0.75$ and $sat_{f_1}(EnriqueI, Justified) = 0.4$, then, $sat_{f_2}(EnriqueI, Justified) = \min(0.75, 0.4) = 0.4$,
- the *length* of pair $(Shakira, Butterfly) = 1/0.7 + 1 = 2.4$, $\mu_{short}(2.4) = 1$ and $sat_{f_1}(Shakira, Butterfly) = 0.7$, then, $sat_{f_2}(Shakira, Butterfly) = \min(1, 0.7) = 0.7$,
- the *length* of pair $(Beyonce, Euphoria) = 1/0.6 + 1/0.2 + 1 = 7.7$, $\mu_{short}(7.7) = 0$ and $sat_{f_1}(Beyonce, Euphoria) = 0.3$, then, $sat_{f_2}(Beyonce, Euphoria) = \min(0, 0.3) = 0$,
- the *length* of pair $(Rihanna, Euphoria) = 1/0.2 + 1 = 6$, $\mu_{short}(6) = 0$ and $sat_{f_1}(Rihanna, Euphoria) = 0.2$, then, $sat_{f_2}(Rihanna, Euphoria) = \min(0, 0.2) = 0$,
- the *length* of pair $(MariahC, Euphoria) = 1/0.3 + 1/0.5 + 1 = 6.33$, $\mu_{short}(6.33) = 0$ and $sat_{f_1}(MariahC, Euphoria) = 0.3$, then, $sat_{f_2}(MariahC, Euphoria) = \min(0, 0.3) = 0$, and
- the *length* of pair $(Shakira, Euphoria) = 1/0.5 + 1 = 3$, $\mu_{short}(3) = 1$ and $sat_{f_1}(Shakira, Euphoria) = 0.5$, then, $sat_{f_2}(Shakira, Euphoria) = \min(1, 0.5) = 0.5$.

Then, the pairs of nodes $(EnriqueI, Justified)$, $(Shakira, Butterfly)$ and $(Shakira, Euphoria)$ are the only ones that match the fuzzy regular expression f_2 and their satisfaction degrees are $sat_{f_2}(EnriqueI, Justified) = 0.4$, $sat_{f_2}(Shakira, Butterfly) = 0.7$ and $sat_{f_2}(Shakira, Euphoria) = 0.5$ respectively.

Expression $f_3 = (friend+)^{ST>0.65}.creator$ is a fuzzy regular expression. A pair of nodes (x, y) satisfies f_3 if x has a friend artist (an artist connected to x with a path made of *friend* edges which has a strength higher than 0.65), who created the album y . It is worth noticing that expression f_1 is a sub-expression of expression f_3 , so we are going to make use of the satisfaction degree of f_1 (denoted by sat_{f_1}) in order to calculate the satisfaction degree of f_3 (sat_{f_3}). The pair of nodes $(Shakira, Butterfly)$, shown in Figure 3.6, is the only one that matches the fuzzy regular expression f_3 with a non zero satisfaction degree: $sat_{f_3}(Shakira, Butterfly) = 0.7$, where the *strength* between the pair of nodes $(Shakira, Butterfly) = \min(0.7, 1) = 0.7$ and $sat_{f_1}(Shakira, Butterfly) = 0.7$, then, $sat_{f_3}(Shakira, Butterfly) = \min(0.7, 0.7) = 0.7$. \diamond

Let us now come to the definition of a mapping. A mapping is a pair (m, d) where $m : \mathcal{V} \rightarrow (\mathcal{U} \times \mathcal{L})$ and $d \in [0, 1]$. Intuitively, m maps the variables of a fuzzy graph pattern into a subgraph (“answer”) of the F-RDF data graph and d denotes the satisfaction degree associated with the mapping (the more satisfactory the subgraph, the higher the satisfaction degree). The expression $m(t)$, where t is a triple pattern, denotes the triple obtained by

replacing each variable x of t by $m(x)$. The domain of a mapping m denoted by $dom(m)$ is the subset of \mathcal{V} for which m is defined. Two mappings m_1 and m_2 are compatible iff for all $?v \in dom(m_1) \cap dom(m_2)$, one has $m_1(?v) = m_2(?v)$. Intuitively, m_1 and m_2 are compatible if m_1 can be extended with m_2 to obtain a new mapping $m_1 \oplus m_2$ and *vice versa*.

Let \mathcal{M}_1 and \mathcal{M}_2 be two fuzzy sets of mappings. We define the join, union, difference and left outer-join of \mathcal{M}_1 with \mathcal{M}_2 as:

Join

$$\begin{aligned} \mathcal{M}_1 \bowtie \mathcal{M}_2 = & \{(m_1 \oplus m_2, \min(d_1, d_2)) \mid (m_1, d_1) \in \mathcal{M}_1 \\ & \text{and } (m_2, d_2) \in \mathcal{M}_2 \text{ and } m_1, m_2 \text{ are compatible}\}. \end{aligned}$$

The operation $\mathcal{M}_1 \bowtie \mathcal{M}_2$ denotes the set of new mappings that result from extending mappings in \mathcal{M}_1 with their compatible mappings in \mathcal{M}_2 .

Union

$$\begin{aligned} \mathcal{M}_1 \cup \mathcal{M}_2 = & \{(m, d) \mid (m, d) \in \mathcal{M}_1 \text{ and } m \notin \text{support}(\mathcal{M}_2)\} \cup \\ & \{(m, d) \mid (m, d) \in \mathcal{M}_2 \text{ and } m \notin \text{support}(\mathcal{M}_1)\} \cup \\ & \{(m, \max(d_1, d_2)) \mid (m, d_1) \in \mathcal{M}_1 \text{ and } (m, d_2) \in \mathcal{M}_2\} \end{aligned}$$

Here, \cup corresponds to the classical set-theoretic union and *support* denotes the support of a fuzzy set of mappings and corresponds to the set of all elements of the universe of discourse whose their grade of membership is greater than zero.

Difference

$$\begin{aligned} \mathcal{M}_1 \setminus \mathcal{M}_2 = & \{(m_1, d_1) \mid (m_1, d_1) \in \mathcal{M}_1 \\ & \text{and } \forall (m_2, d_2) \in \mathcal{M}_2, m_1 \text{ and } m_2 \text{ are not compatible}\}. \end{aligned}$$

$\mathcal{M}_1 \setminus \mathcal{M}_2$ returns the set of mappings in \mathcal{M}_1 that cannot be extended with any mapping in \mathcal{M}_2 .

Leftouterjoin

$$\mathcal{M}_1 \bowtie\!\!\!\bowtie \mathcal{M}_2 = (\mathcal{M}_1 \bowtie \mathcal{M}_2) \cup (\mathcal{M}_1 \setminus \mathcal{M}_2).$$

A mapping m is in $\mathcal{M}_1 \bowtie\!\!\!\bowtie \mathcal{M}_2$ if it is the extension of a mapping of \mathcal{M}_1 with a compatible mapping of \mathcal{M}_2 , or if it is in \mathcal{M}_1 and cannot be extended with any mapping of \mathcal{M}_2 .

Definition 12 (Mapping satisfying a fuzzy condition). *Let m be a mapping and C be a fuzzy condition. Then m satisfies the fuzzy condition C with a satisfaction degree defined as follows, according to the form of C :*

- C is of the form $\text{bound}(?x)$: if $?x \in \text{dom}(m)$ then m satisfies the condition C with a degree of 1, else 0.
- C is of the form $?x \theta c$ (where θ is a (possibly fuzzy) comparator and c is a constant): if $?x \in \text{dom}(m)$ then m satisfies the condition C with a degree of $\mu_\theta(m(?x), c)$, else 0.
- C is of the form $?x \theta ?y$: if $?x \in \text{dom}(m)$ and $?y \in \text{dom}(m)$, then m satisfies the condition C with a degree of $\mu_\theta(m(?x), m(?y))$, else 0.
- C is of the form $?x \text{ IS } F\text{term}$: if $?x \in \text{dom}(m)$ then m satisfies the condition C to the degree $\mu_{F\text{term}}(m(?x))$ (which can be 0).
- C is of the form $\neg C_1$ or $C_1 \odot C_2$ where \odot is a fuzzy connective: we use the usual interpretation of the fuzzy operator involved (complement to 1 for the negation, minimum for the conjunction, maximum for the disjunction, etc [Fodor and Yager, 2000]).

Definition 13 (Evaluation (interpretation) of a fuzzy graph pattern). *The evaluation of a fuzzy graph pattern P over an F-RDF graph, denoted by $\llbracket P \rrbracket_G$ is recursively defined by:*

- if P is of the form of a (crisp) triple graph pattern $t \in (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \times \mathcal{L} \times \mathcal{V})$ then $\llbracket P \rrbracket_G = \{(m, 1) \mid \text{dom}(m) = \text{var}(t) \text{ and } m(t) \in G\}$,
- if P is of the form of a fuzzy triple graph pattern $t \in (\mathcal{U} \cup \mathcal{V}) \times \mathcal{F} \times (\mathcal{U} \times \mathcal{L} \times \mathcal{V})$ denoted by $\langle ?x, \text{exp}, ?y \rangle$ (where variables occur as subject and object) then $\llbracket P \rrbracket_G = \{(m, d) \mid \text{dom}(m) = \{?x, ?y\} \text{ and } (m(?x), m(?y)) \text{ satisfies exp with a satisfaction degree } d = \text{sat}_{\text{exp}}(x, y)\}$. The case where the subject (resp. the object) of t is a constant of \mathcal{U} (resp. $\mathcal{U} \cup \mathcal{L}$) is trivially induced from this definition.
- if P is of the form $(P_1 \text{ AND } P_2)$ then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \bowtie \llbracket P_2 \rrbracket_G$,
- if P is of the form $(P_1 \text{ OPT } P_2)$ then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \bowtie \llbracket P_2 \rrbracket_G$,
- if P is of the form $(P_1 \text{ UNION } P_2)$ then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$,
- if P is of the form $(P_1 \text{ FILTER } C)$ then $\llbracket P \rrbracket_G = \{(m, d) \mid m \in \llbracket P_1 \rrbracket_G \text{ and } m \text{ satisfies } C \text{ to the degree of } d\}$.

Intuitively, expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ FILTER } C)$ refer to conjunction graph patterns, union graph patterns, optional graph patterns, and filter graph patterns respectively. Optional graph patterns allow for a partial match of the query (i.e., the query tries to match a graph pattern and does not omit a solution when some part of the optional pattern is not satisfied).

Remark 7. Note that a crisp graph pattern is a special case of a fuzzy graph pattern where no fuzzy term or condition occurs (and thus, according to the previous definition, an answer necessarily has a satisfaction degree of 1).

Example 45 [Evaluation of a fuzzy graph pattern] Let us recall the fuzzy graph pattern P_{rec_low} from Example 42 defined by $(?Art1, (friend^+)^{distance\ is\ short}.creator, ?Alb) AND (?Art1, recommends, ?Alb) AND ((?Alb, rating, ?r) FILTER (?r\ is\ low))$, for which Figure 6.3 is a graphical representation.

Figure 3.8 gives the set of subgraphs of G_{MB} satisfying the pattern P_{rec_low} . The matching value of $Art1$ is either *Shakira* or *EnriqueI* who match the pattern P_{rec_low} (i.e they are the only artists that have liked a low rated album created by another artist among their close friends). Note that $(friend^+)^{distance\ is\ short}.creator$ is the fuzzy regular expression f_2 of Example 44 with $sat_{f_2}(EnriqueI, Justified) = 0.4$, $sat_{f_2}(Shakira, Butterfly) = 0.7$ and $sat_{f_2}(Shakira, Euphoria) = 0.5$ and we consider $\mu_{low_rating}(4) = 0.66$, $\mu_{low_rating}(6) = 0.33$ and $\mu_{low_rating}(9) = 0$ defined in Figure 3.5 on page 70.

Then, the evaluation of the pattern P_{rec_low} over the RDF graph G_{MB} includes two mappings with their respective satisfaction degrees:

$$\begin{aligned}
 sat_{f_2^{low}}(Shakira, Butterfly) &= \min(sat_{f_2}(Shakira, Butterfly), \\
 &\quad \zeta(Shakira, recommend, Butterfly), \\
 &\quad \zeta(Butterfly, rating, 4), \mu_{low_rating}(4)) \\
 &= \min(0.7, 0.8, 1, 0.66) = 0.66 \text{ and} \\
 sat_{f_2^{low}}(EnriqueI, Justified) &= \min(sat_{f_2}(EnriqueI, Justified), \\
 &\quad \zeta(EnriqueI, recommend, Justified), \\
 &\quad \zeta(Justified, rating, 6), \mu_{low_rating}(6)) \\
 &= \min(0.4, 0.6, 1, 0.33) = 0.33.
 \end{aligned}$$

It can be represented as follows:

$$\begin{aligned}
 \llbracket P_{rec_low} \rrbracket_{G_{MB}} &= \{ \\
 &(\{?Art1 \rightarrow EnriqueI, ?Alb \rightarrow Justified, ?r \rightarrow 6\}, 0.33), \\
 &(\{?Art1 \rightarrow Shakira, ?Alb \rightarrow Butterfly, ?r \rightarrow 4\}, 0.66)\}.
 \end{aligned}$$

Note that the mapping $\{?art1 \rightarrow Shakira, ?alb \rightarrow Euphoria, ?r \rightarrow 9\}$ is excluded from the result of the evaluation of the pattern P_{rec_low} since $\mu_{low_rating}(9) = 0$.

Finally, the result of the query of Example 43 (Listing 3.2 on page 71) over G_{MB} is the singleton $\{\text{Shakira}\}$ which is $m(?art1)$ in the mapping $\{?art1 \rightarrow \text{Shakira}, ?alb \rightarrow \text{Butterfly}, ?r \rightarrow 4\}$, i.e., the only mapping of $\llbracket P_{rec_low} \rrbracket_{G_{MB}}$ having a satisfaction degree greater or equal to 0.4. \diamond

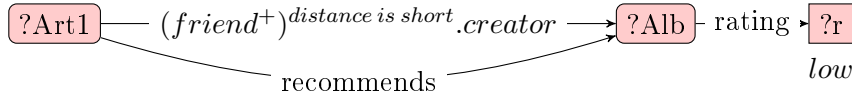


Figure 3.7: Graphical representation of pattern P_{rec_low}

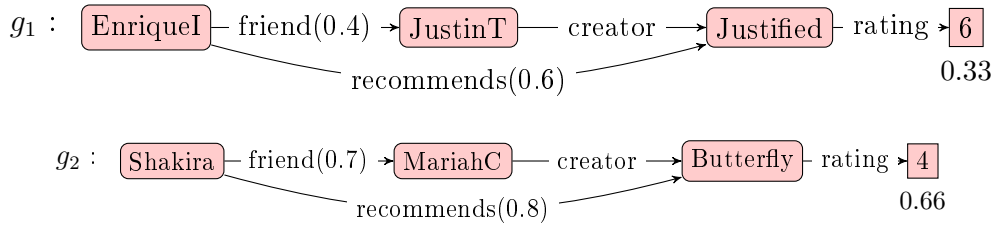


Figure 3.8: Subgraphs satisfying P_{rec_low}

Conclusion

In this chapter, we have introduced a new query language named FURQL which is a fuzzy extension of SPARQL that goes beyond the previous proposals in terms of expressiveness inasmuch as it makes it possible i) to deal with *crisp* and *fuzzy RDF data*, and ii) to express *fuzzy structural* conditions beside more classical fuzzy conditions on the values of the nodes present in the graph.

We first presented the notion of a fuzzy RDF graph that makes it possible to model relationships between entities and then, we formalized a formal syntax and semantics of FURQL based on the notion of *fuzzy graph pattern*, which extends Boolean graph patterns introduced by several authors in a crisp querying context. Associated implementation issues and experiments will be presented in Chapter 5. In the following chapter, we propose to extend the FURQL query language to be able to express more sophisticated fuzzy conditions, namely *fuzzy quantified statements*.

Chapter 4

FURQL with Fuzzy Quantified Statements to Fuzzy RDF Databases

Contents

Introduction	79
4.1 Refresher on Fuzzy Quantified Statements	80
4.1.1 Fuzzy Quantifiers	80
4.1.2 Interpretation of Fuzzy Quantified Statements	81
4.2 FURQL with Fuzzy Quantified Statements	86
4.2.1 Related Work: Quantified Statements in SPARQL	87
4.2.2 Fuzzy Quantified Statements in FURQL	87
Conclusion	95

Introduction

FUZZY quantified queries have been long recognized for their ability to express different types of imprecise and flexible information needs in a relational database context. However, in the specific RDF/SPARQL setting, the current approaches from the literature that deal with quantified queries consider crisp quantifiers only [Bry et al., 2010, Fan et al., 2016] over crisp RDF data.

In the present chapter, we integrate *fuzzy quantified statements* in FURQL queries addressed to a fuzzy RDF database. We show how these statements can be defined and implemented in FURQL, which is a fuzzy extension of the SPARQL query language that we previously presented in Chapter 3. This work has been published in the proceedings of the 26th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'17), Naples, Italy, 2017.

In the following, in Section 4.1 we first present a refresher on *fuzzy quantified statements* in a relational database context, then, in Section 4.2 we introduce the syntactic format for expressing *fuzzy quantified statements* in the FURQL language and we describe their interpretation using different approaches from the literature.

4.1 Refresher on Fuzzy Quantified Statements

In this section, we recall important notions about fuzzy quantifiers, then, we present three approaches that have been proposed in the literature for interpreting *fuzzy quantified statements*.

4.1.1 Fuzzy Quantifiers

Fuzzy logic extends the notion of quantifier from Boolean logic (e.g., \exists and \forall) and makes it possible to model quantifiers from the natural language such as most of, at least half, few, around a dozen, etc.

In [Zadeh, 1983], the author distinguishes between absolute and relative fuzzy quantifiers. Absolute quantifiers refer to a number while relative ones refer to a proportion. Quantifiers may also be increasing, as “at least half”, or decreasing, as “at most three”.

An absolute quantifier \mathcal{Q} is represented by a function $\mu_{\mathcal{Q}}$ from an integer range to $[0, 1]$ whereas a relative quantifier is a mapping $\mu_{\mathcal{Q}}$ from $[0, 1]$ to $[0, 1]$. In both cases, the value $\mu_{\mathcal{Q}}(j)$ is defined as the truth value of the statement “ $\mathcal{Q} X$ are A ” when exactly j elements from X fully satisfy A (whereas it is assumed that A is fully unsatisfied for the other elements).

According to [Yager, 1988], fuzzy quantifiers can be increasing (proportional) which means that if the criteria are all entirely satisfied, then the statement “ $\mathcal{Q} X$ are A ” is entirely true, and if the criteria are all entirely unsatisfied, then the statement “ $\mathcal{Q} X$ are A ” is entirely false. Moreover, the transition between those two extremes is continuous and monotonous. Therefore, when \mathcal{Q} is increasing (e.g., “most”, “at least a half”), function $\mu_{\mathcal{Q}}$ is increasing. Similarly, decreasing quantifiers (e.g., “at most two”, “at most a half”) are defined by decreasing functions.

The characteristics of monotonous fuzzy quantifiers are given in Table 4.1.

Increasing quantifier	Decreasing quantifier
$\mu_{\mathcal{Q}}(0) = 0$	$\mu_{\mathcal{Q}}(0) = 1$
$\exists k$ such that $\mu_{\mathcal{Q}}(k) = 1$	$\exists k$ such that $\mu_{\mathcal{Q}}(k) = 0$
$\forall a, b$, if $a > b$ then $\mu_{\mathcal{Q}}(a) \geq \mu_{\mathcal{Q}}(b)$	$\forall a, b$, if $a > b$ then $\mu_{\mathcal{Q}}(a) \leq \mu_{\mathcal{Q}}(b)$

Table 4.1: Characteristics of monotonous fuzzy quantifiers

Figure 4.1 gives two examples of monotonous decreasing and increasing fuzzy quantifiers respectively.

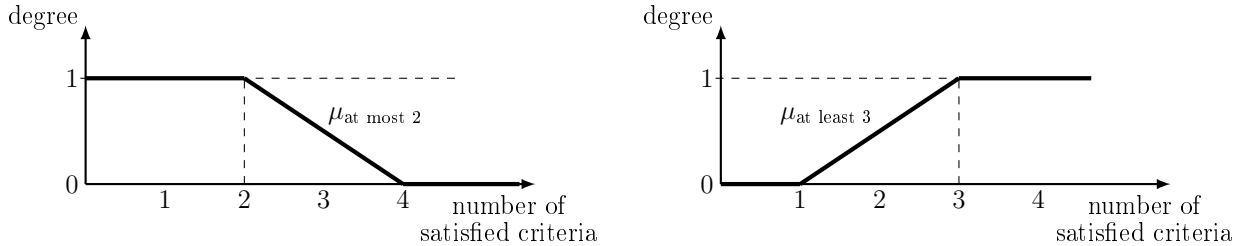


Figure 4.1: Quantifiers “at most 2” (left) and “at least 3” (right)

Calculating the truth degree of the statement “ $\mathcal{Q} X$ are A ” raises the problem of determining the cardinality of the set of elements from X which satisfy A . If A is a Boolean predicate, this cardinality is a precise integer (k), and then, the truth value of “ $\mathcal{Q} X$ are A ” is $\mu_{\mathcal{Q}}(k)$. If A is a fuzzy predicate, this cardinality cannot be established precisely and then, computing the quantification corresponds to establishing the value of function $\mu_{\mathcal{Q}}$ for an imprecise argument.

Fuzzy quantified queries have been thoroughly studied in a relational database context, see e.g. [Kacprzyk et al., 1989, Bosc et al., 1995] where they serve to express conditions about data *values*. The authors distinguished between two types of uses of fuzzy quantifiers:

- horizontal quantification (the quantifier is used as a connective for combining atomic conditions in a *where* clause; this use was originally suggested in [Kacprzyk et al., 1989]);
- vertical quantification (the quantifier appears in a *having* clause in order to express a condition on the cardinality of a fuzzy subset of a group, as in “find the departments where *most* of the employees are *well-paid*”). This is the type of use we make in our approach.

4.1.2 Interpretation of Fuzzy Quantified Statements

We now present different proposals from the literature for interpreting quantified statements of the type “ $\mathcal{Q} B X$ are A ” (which generalizes the case “ $\mathcal{Q} X$ are A ” by considering that the set to which the quantifier applies is itself fuzzy) where X is a (crisp) referential and A and B are fuzzy predicates.

4.1.2.1 Zadeh's interpretation

Let X be the usual (crisp) set $\{x_1, x_2, \dots, x_n\}$ and n the cardinality of X . Zadeh [Zadeh, 1983] defines the cardinality of the set of elements of X which satisfy A , denoted by $\Sigma count(A)$, as:

$$\Sigma count(A) = \sum_{i=1}^n \mu_A(x_i) \quad (4.1)$$

The truth degree of the statement “ $\mathcal{Q} X$ are A ” is then given by

$$\mu(\mathcal{Q} X \text{ are } A) = \begin{cases} \mu_{\mathcal{Q}}(\Sigma count(A)) & \text{(absolute),} \\ \mu_{\mathcal{Q}}\left(\frac{\Sigma count(A)}{n}\right) & \text{(relative)} \end{cases} \quad (4.2)$$

One may notice, however, that a large number of elements with a small degree $\mu_A(x)$ has a same effect as a small number of elements with a high degree $\mu_A(x)$, due to the definition of $\Sigma count$.

Example 46 Let us consider the following sets:

$$X_1 = \{0.9/x_1, 0.9/x_2, 0.9/x_3, 0.8/x_4, 0.8/x_5, 0.7/x_6, 0.6/x_7\},$$

$$X_2 = \{1/x_1, 1/x_2, 0.3/x_3, 0.2/x_4, 0.1/x_5, 0/x_6, 0/x_7\},$$

$$X_3 = \{1/x_1, 1/x_2, 1/x_3, 1/x_4, 1/x_5, 0.8/x_6, 0.3/x_7\}.$$

and the quantifier “at least five” represented in Figure 4.2. The $\Sigma count(A)$ values

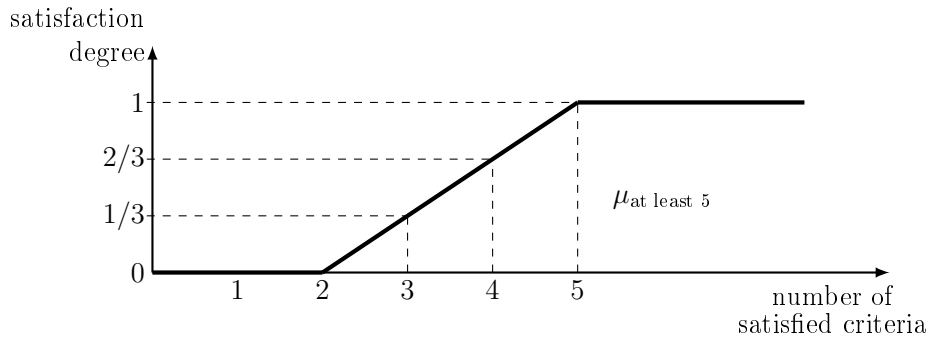


Figure 4.2: The fuzzy quantifier “at least five”

associated with the sets X_1 , X_2 , and X_3 are 5.6, 2.6, and 6.1 respectively. The associated values of the quantification are 1, 0.2, and 1 respectively. \diamond

As for quantified statements of the form “ $\mathcal{Q} B X$ are A ” (with \mathcal{Q} relative), their interpretation is as follows:

$$\mu(\mathcal{Q} B X \text{ are } A) = \mu_{\mathcal{Q}} \left(\frac{\Sigma \text{count}(A \cap B)}{\Sigma \text{count}(B)} \right) = \mu_{\mathcal{Q}} \left(\frac{\sum_{x \in X} \top(\mu_A(x), \mu_B(x))}{\sum_{x \in X} \mu_B(x)} \right) \quad (4.3)$$

where \top denotes a triangular norm (for instance the minimum).

Example 47 Let us evaluate the quantified statement “ $\mathcal{Q} B X$ are A ” where $B = \{0.6/x_1, 0.3/x_2, 1/x_3, 0.1/x_5\}$, $A = \{0.8/x_1, 0.4/x_2, 0.9/x_3, 1/x_4, 1/x_5\}$ and $\mathcal{Q}(x) = x^2$.

Then, $\mu(\mathcal{Q} B X \text{ are } A) = \mu_{\mathcal{Q}} \left(\frac{0.6+0.3+0.9+0+0.1}{0.6+0.3+1+0+0.1} \right) = \mu_{\mathcal{Q}} \left(\frac{1.9}{2} \right) = \mu_{\mathcal{Q}}(0.95) = 0.90$. \diamond

4.1.2.2 Yager’s Competitive Type Aggregation

The interpretation by decomposition described in [Yager, 1984] was originally limited to *increasing* quantifiers. It was later generalized to all kinds of fuzzy quantifiers in [Bosc et al., 1995], but hereafter, we consider the basic case where \mathcal{Q} is increasing.

The proposition “ $\mathcal{Q} X$ are A ” is true if an ordinary subset C of X satisfies the conditions c_1 and c_2 given hereafter:

c_1 : there are \mathcal{Q} elements in C ,

c_2 : each element x of C satisfies A .

The truth value of the proposition: “ $\mathcal{Q} X$ are A ” is then defined as:

$$\mu(\mathcal{Q} X \text{ are } A) = \sup_{C \subseteq X} \min(\mu_{c_1}(C), \mu_{c_2}(C)) \quad (4.4)$$

with

$$\mu_{c_1}(C) = \begin{cases} \mu_{\mathcal{Q}}(|C|) & \text{if } \mathcal{Q} \text{ is absolute,} \\ \mu_{\mathcal{Q}} \left(\frac{|C|}{n} \right) & \text{if } \mathcal{Q} \text{ is relative} \end{cases} \quad (4.5)$$

and

$$\mu_{c_2}(C) = \inf_{x \in C} \mu_A(x). \quad (4.6)$$

It has been shown in [Yager, 1984] that:

$$\mu(\mathcal{Q} X \text{ are } A) = \sup_{1 \leq i \leq n} \min(\mu_{\mathcal{Q}}(i), \mu_A(x_i)). \quad (4.7)$$

where the elements of X are ordered in such a way that $\mu_A(x_1) \geq \dots \geq \mu_A(x_n)$. Formula (4.7) corresponds to a Sugeno integral [Sugeno, 1974].

For quantified statements of the form “ $Q B X$ are A ”, the principle is similar. The statement is true if there exists a crisp subset C of X that satisfies the conditions c'_1 and c'_2 hereafter:

c'_1 : $Q B X$ are in C ,

c'_2 : each element x of C satisfies the implication

$$(x \text{ is } B) \Rightarrow (x \text{ is } A).$$

The truth value of the proposition: “ $Q B X$ are A ” is then defined as:

$$\mu(Q B X \text{ are } A) = \sup_{C \subseteq X} \min(\mu_{c'_1}(C), \mu_{c'_2}(C)) \quad (4.8)$$

with

$$\mu_{c'_1}(C) = \begin{cases} \mu_Q(\sum_{x \in C} \mu_B(x)) & \text{if } Q \text{ is absolute,} \\ \mu_Q\left(\frac{\sum_{x \in C} \mu_B(x)}{\sum_{x \in X} \mu_B(x)}\right) & \text{if } Q \text{ is relative} \end{cases} \quad (4.9)$$

and

$$\mu_{c'_2}(C) = \inf_{x \in C} \mu_B(x) \rightarrow \mu_A(x) \quad (4.10)$$

where \rightarrow is a fuzzy implication (see e.g. [Fodor and Yager, 2000]).

Notice that $\mu(Q B X \text{ are } A)$ is undefined when $\forall x \in X, \mu_B(x) = 0$ since this would result in a division by zero in Formula 4.9.

4.1.2.3 Interpretation based on the *OWA* operator

In [Yager, 1988], Yager considers the case of an increasing monotonous quantifier and proposes an ordered weighted averaging operator (*OWA*) to evaluate quantifications of the type “ $Q X$ are A ”. It is shown in [Bosc et al., 1995] i) how it can be extended in order to evaluate decreasing quantifications and ii) that this interpretation boils down to using a Choquet fuzzy integral.

The *OWA* operator is defined in [Yager, 1988] as:

$$OWA(x_1, \dots, x_n; w_1, \dots, w_n) = \sum_{i=1}^n w_i \times x_{k_i} \quad (4.11)$$

where x_{k_i} is the i^{th} largest value among the x_k 's and $\sum_{i=1}^n w_i = 1$.

Let n be the crisp cardinality of X . The truth value of the statement “ $\mathcal{Q} X$ are A ” is computed by an *OWA* of the n values $\mu_A(x_i)$. The weights w_i involved in the calculation of the *OWA* are given by

$$w_i = \begin{cases} \mu_{\mathcal{Q}}(i) - \mu_{\mathcal{Q}}(i-1) & \text{if } \mathcal{Q} \text{ is absolute,} \\ \mu_{\mathcal{Q}}\left(\frac{i}{n}\right) - \mu_{\mathcal{Q}}\left(\frac{i-1}{n}\right) & \text{if } \mathcal{Q} \text{ is relative.} \end{cases} \quad (4.12)$$

The aggregated value which is calculated is:

$$OWA(\mu_A(x_1), \mu_A(x_2), \dots, \mu_A(x_n); w_1, \dots, w_n) = \sum_{i=1}^n w_i \times c_i \quad (4.13)$$

where c_i is the i^{th} largest value among the $\mu_A(x_k)$'s.

Example 48 Let us consider the sets X_1 , X_2 , and X_3 , and the quantifier “at least five” from Example 46. We have:

$$w_1 = 0, w_2 = 0, w_3 = 1/3, w_4 = 1/3, w_5 = 1/3, w_6 = 0, w_7 = 0.$$

We evaluate the statement “at least five elements of X_1 are A ” and we get the degree 0.83 ($= 0.9 \times 1/3 + 0.8 \times 1/3 + 0.8 \times 1/3$). The same way, we get the degrees 0.2 for X_2 and 1 for X_3 . \diamond

This interpretation corresponds to using a Choquet integral [Choquet, 1954], see also [Murofushi and Sugeno, 1989, Grabisch et al., 1992].

As for statements of the form “ $\mathcal{Q} B X$ are A ”, Yager suggests to compute the truth degree of statements of the form “ $\mathcal{Q} B X$ are A ” by an *OWA* aggregation of the implication values

$$\mu_B(x) \rightarrow_{KD} \mu_A(x)$$

where \rightarrow_{KD} denotes Kleene-Dienes implication ($a \rightarrow_{KD} b = \max(1 - a, b)$).

Let $X = \{x_1, \dots, x_n\}$ be such that $\mu_B(x_1) \leq \mu_B(x_2) \leq \dots \leq \mu_B(x_n)$ and $\sum_{i=1}^n \mu_B(x_i) = d$. The weights of the *OWA* operator are defined by:

$$w_i = \mu_{\mathcal{Q}}(S_i) - \mu_{\mathcal{Q}}(S_{i-1}), \quad (4.14)$$

with

$$S_i = \sum_{j=1}^i \frac{\mu_B(x_j)}{d} \quad (4.15)$$

and

$$S_0 = 0. \quad (4.16)$$

The implication values are denoted by c_i and ordered decreasingly: $c_1 \geq c_2 \geq \dots \geq c_n$.
Finally:

$$\mu(\mathcal{Q} B X \text{ are } A) = \sum_{i=1}^n w_i \times c_i. \quad (4.17)$$

Example 49 Let us consider a quantified statement of the form “ $\mathcal{Q} B X$ are A ” from Example 47 and $\mathcal{Q}(x) = x^2$.

We first order the elements of X such that $\mu_B(x_{k_1}) \leq \dots \leq \mu_B(x_{k_n})$, $e_1 = 0, e_2 = 0.1, e_3 = 0.3, e_4 = 0.6, e_5 = 1$ and $d = 2$. Thus, we get $S_1 = 0, S_2 = 0.05, S_3 = 0.2, S_4 = 0.5, S_5 = 1$.

$$\mu_{\mathcal{Q}}(S_1) = 0, \mu_{\mathcal{Q}}(S_2) = 0.025, \mu_{\mathcal{Q}}(S_3) = 0.04, \mu_{\mathcal{Q}}(S_4) = 0.25, \mu_{\mathcal{Q}}(S_5) = 1.$$

Therefore, the weights of the OWA operator are:

$$\begin{aligned} w_1 &= \mu_{\mathcal{Q}}(S_1) - \mu_{\mathcal{Q}}(S_0) = 0, \\ w_2 &= \mu_{\mathcal{Q}}(S_2) - \mu_{\mathcal{Q}}(S_1) = 0.025, \\ w_3 &= \mu_{\mathcal{Q}}(S_3) - \mu_{\mathcal{Q}}(S_2) = 0.04 - 0.025 = 0.0375, \\ w_4 &= \mu_{\mathcal{Q}}(S_4) - \mu_{\mathcal{Q}}(S_3) = 0.25 - 0.04 = 0.21, \\ w_5 &= \mu_{\mathcal{Q}}(S_5) - \mu_{\mathcal{Q}}(S_4) = 1 - 0.25 = 0.75. \end{aligned}$$

For each x_i we calculate the implication value $c_i = \max((1 - \mu_B(x_i)), \mu_A(x_i))$ and these values are ordered decreasingly such that $c_1 \geq \dots \geq c_n$.

$$\begin{aligned} c_1 &= \max(0.4, 0.8) = 0.8, c_2 = \max(0.7, 0.4) = 0.7, \\ c_3 &= \max(0, 0.9) = 0.9, c_4 = \max(1, 1) = 1, \\ c_5 &= \max(0.9, 1) = 1. \end{aligned}$$

We reorder the implication values and we get $c'_1 = 1(c_4), c'_2 = 1(c_5), c'_3 = 0.9(c_3), c'_4 = 0.8(c_1), c'_5 = 0.7(c_2)$.

Finally, the satisfaction degree using the OWA aggregation is:

$$\mu = (1) * 0 + 0.025 * (1) + (0.375) * 0.9 + 0.21 * (0.8) + 0.75 * (0.7) = 0.73. \diamond$$

4.2 FURQL with Fuzzy Quantified Statements

In this section, we first present some recent proposals from the literature for incorporating quantified statements into SPARQL queries, and then, we propose to integrate *fuzzy quantified statements* in the FURQL language.

4.2.1 Related Work: Quantified Statements in SPARQL

In an RDF database context, quantified statements have only recently attracted the attention of database community. In [Bry et al., 2010], Bry et al. propose an extension of SPARQL (called SPARQLog) with first-order logic (FO) rules and existential and universal quantification over node variables. This query language makes it possible to express statements such as: “for each lecture there is a course that practices this lecture and is attended by all students attending the lecture”. This statement can be expressed in SPARQLog as follows:

```

ALL ?lec EX ?crs ALL ?stu
CONSTRUCT { ?crs uni:practices ?lec .
?stu uni:attends ?crs . }
WHERE { ?lec rdf:type uni:lecture .
?stu uni:attends ?lec . }

```

More recently, in [Fan et al., 2016], Fan et al. introduced quantified graph patterns, an extension of the classical SPARQL graph patterns using simple counting quantifiers on edges. Quantified graph patterns make it possible to express numeric and ratio aggregates, and negation besides existential and universal quantification. The authors also showed that quantified matching in the absence of negation does not significantly increase the cost of query processing.

However, to the best of our knowledge, there does not exist any work in the literature that deals with *fuzzy quantified statements* in the SPARQL query language, which is the main goal of the present chapter.

4.2.2 Fuzzy Quantified Statements in FURQL

In this subsection, we show how *fuzzy quantified statements* may be expressed in FURQL queries. We first propose a syntactic format for these queries, and then we show how they can be evaluated in an efficient way.

4.2.2.1 Syntax of a Fuzzy Quantified Query in FURQL

In the following, we consider *fuzzy quantified statements* of the type “ $\mathcal{Q} B X$ are A ” over fuzzy RDF graph databases, where the quantifier \mathcal{Q} is represented by a fuzzy set and denotes either a relative quantifier (e.g., *most*) or an absolute one (e.g., *at least three*), B is the fuzzy condition “to be connected to a node x ”, X is the set of nodes in the RDF graph, and A denotes a (possibly compound) fuzzy condition.

Example 50 [Fuzzy quantified statement] An example of a *fuzzy quantified statements* of the type “ $\mathcal{Q} B X$ are A ” is: “*most of the recent albums are highly rated*”. In this example, \mathcal{Q} corresponds to the relative quantifier *most*, B is the fuzzy condition to be *recent*, X corresponds to the set of albums present in the RDF graph, and A corresponds to the fuzzy conditions to be *highly* rated. \diamond

The general syntactic form of a *fuzzy quantified query* of the type “ $Q B X$ are A ” in the FURQL language is given in Listing 4.1.

```

DEFINE ...
SELECT ?res WHERE {
  B(?res,?x)
GROUP BY ?res
HAVING Q(?x) ARE ( A(?x) ) }

```

Listing 4.1: Syntax of a FURQL quantified query R

The **DEFINE** clause allows to define the fuzzy terms and the fuzzy quantifier (denoted here by Q). Fuzzy quantifiers are declared in the same way as fuzzy terms (see Subsection 3.2.1 of Chapter 3). The **SELECT** clause specifies which variables `?res` should be returned in the result set. The **GROUP BY** clause contains the variables (here `?res`) that should be partitioned. Expression $B(?res, ?x)$ (in the **WHERE** clause) denotes the *fuzzy graph pattern*, defined in the FURQL language (see Definition 9 on page 69), involving the variables `?res` and `?x` and expressing the (possibly fuzzy) conditions in B and expression $A(?x)$ (in the **HAVING** clause) denotes the *fuzzy graph pattern* involving the variable `?x` that appears in A .

Example 51 [Fuzzy Quantified Query in FURQL] The query, denoted by $R_{mostAlbums}$, that aims to retrieve every artist (`?art1`) such that *most* of the *recent* albums (`?alb`) that he/she recommends are *highly* rated and have been created by a *young* friend (`?art2`) of his/hers may be expressed in FURQL as follows:

```

1 DEFINERELATIVEASC most AS (0.3,0.8), DEFINEASC high AS (2,5)
2 DEFINEDESC young AS (25,40), DEFINEASC recent AS (2010,2015)
3 SELECT ?art1 WHERE {
4   ?art1 recommends ?alb . ?alb date ?date .
5   FILTER ( ?date IS recent ) }
6 GROUP BY ?art1
7 HAVING most(?alb) ARE
8   ( ?art1 friend ?art2 . ?art2 creator ?alb .
9     ?alb rating ?rating . ?art2 age ?age .
10  FILTER (?rating IS high && ?age IS young) )

```

Listing 4.2: Syntax of the FURQL quantified query $R_{mostAlbums}$

where the **DEFINE**RELATIVEASC clause defines the fuzzy relative increasing quantifier *most* of Figure 4.3.(c), the **DEFINE**ASC clauses define the (increasing) membership functions associated with the fuzzy terms *high* and *recent* of Figure 4.3.(a) and (b), and the **DEFIN**EDESC clause defines the (decreasing) membership function associated with the fuzzy term *young* of Figure 4.3.(d). In this query, `?art1` corresponds to `?res` of Listing 4.2, `?alb` corresponds to `?x` of Listing 4.2, lines 4 to 5 correspond to $B(?res, ?x)$ of Listing 4.2 and lines 8 to 10 correspond to $A(?x)$ of Listing 4.2. \diamond

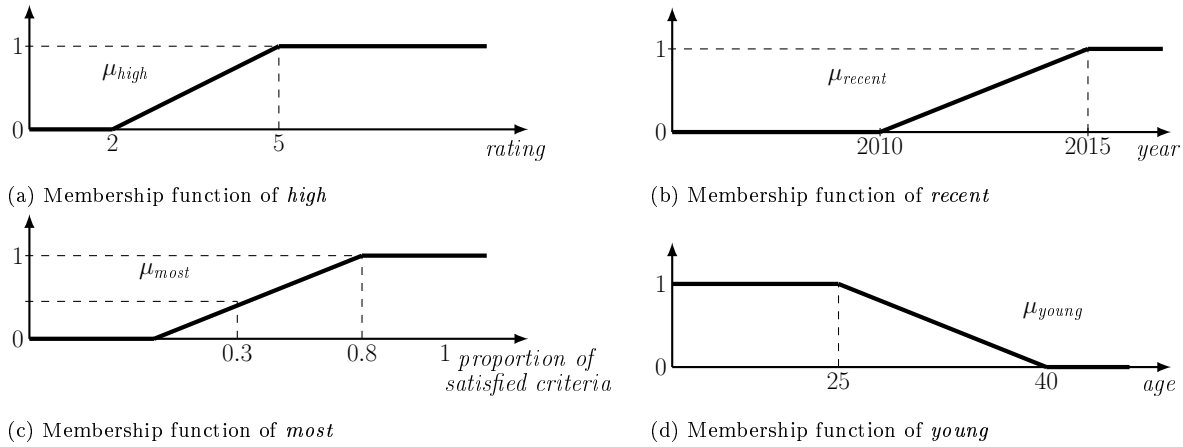


Figure 4.3: Membership functions of Example 51

Since the FURQL query language supports the expression of *fuzzy preferences* involving fuzzy structural properties (like for example, the *distance* and *strength* between two nodes over fuzzy graphs), *fuzzy quantified structural queries* can be expressed in the FURQL language and an example of such query is given hereafter.

Example 52 [Fuzzy Quantified Structural Query in FURQL] We now consider a slightly more complex version of the above example by adding a fuzzy structural condition on the strength of the authors’ recommendation: “retrieve every artist (?art1) such that *most* of the *recent* albums (?alb) that he/she *strongly* recommends are *highly* rated and have been created by a *young* friend (?art2) of his/hers”. The syntactic form of this query, denoted by $R_{mostAlbums_ST}$, is given in Listing 4.3.

```

1  DEFINE RELATIVEASC most AS (0.3,0.8) DEFINEASC recent AS (2010,2015)
2  DEFINEASC high AS (2,5) DEFINEDESC young AS (25,40)
3  DEFINEASC strong AS (0,1)
4  SELECT ?art1 WHERE {
5    ?art1 (recommends | ST IS strong) ?alb .
6    ?alb date ?date .
7    FILTER ( ?date IS recent ) }
8  GROUP BY ?art1
9  HAVING most(?alb) ARE
10     ( ?art1 friend ?art2 . ?art2 creator ?alb .
11       ?alb rating ?rating . ?art2 age ?age .
12       FILTER ( ?rating IS high && ?age IS young ) )

```

Listing 4.3: Syntax of the FURQL quantified query $R_{mostAlbums_ST}$

In this query, line 3 defines the fuzzy term *strong* of Figure 4.4 and line 5 corresponds to the fuzzy structural condition on the strength of the authors’ recommendation. \diamond

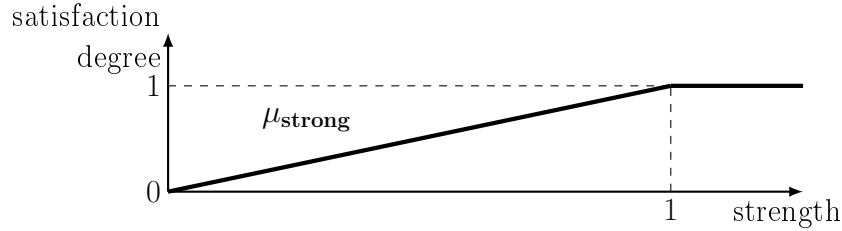


Figure 4.4: Membership function of the fuzzy term *strong*

4.2.2.2 Evaluation of a Fuzzy Quantified Query

The interpretation of a fuzzy quantified statement in a FURQL query can be based on one of the formulas (4.3), (4.8), or (4.17). Its evaluation involves three stages :

1. the compiling of the fuzzy quantified query R into a crisp query denoted by $R_{flatBoolean}$,
2. the interpretation of the crisp SPARQL query $R_{flatBoolean}$,
3. the calculation of the result of R (which is a fuzzy set) based on the result of $R_{flatBoolean}$.

Compiling

The compiling stage translates the fuzzy quantified query R into a crisp query denoted by $R_{flatBoolean}$. This compilation involves two translation steps.

First, R is transcribed into an intermediate query R_{flat} that allows to interpret the fuzzy quantified statement embedded in R . The query R_{flat} , whose general form¹ is given in Listing 4.4, is obtained by removing the `GROUP BY` and `HAVING` clauses from the initial query and adding the `OPTIONAL` clause for the A part. This query aims to retrieve the elements of the B part of the initial query, matching the variables `?res` and `?x`, and possibly the elements of the A part of the initial query, matching the variable `?x`, for which we will then need to calculate the final satisfaction degree.

```
SELECT ?res ?X IB IA WHERE {
  B(?res,?X)
  OPTIONAL { A(?X) } }
```

Listing 4.4: Derived query R_{flat} of $R_{mostAlbums}$

For each pair $(?res, ?x)$, we retrieve all the information needed for the calculation of μ_B and μ_A , i.e., the combination of fuzzy degrees associated with relationships and node attribute values involved in $B(?res, ?x)$ and in $A(?X)$, respectively denoted by I_B and I_A . Listing 4.5 of Example 53 below presents the derived query associated with the query $R_{mostAlbums}$.

¹Hereafter, the `DEFINE` clauses are omitted for the sake of simplicity.

The evaluation of R_{flat} is based on the *derivation principle* introduced by [Pivert and Bosc, 2012] in the context of relational databases: R_{flat} is in fact derived into another query denoted by $R_{flatBoolean}$. The derivation translates the fuzzy query into a crisp one by transforming its fuzzy conditions into Boolean ones that select the support of the fuzzy statements. For instance, following this principle, the fuzzy condition `?year IS recent` defined as `DEFINEASC recent AS (2013,2016)` becomes the crisp condition `?year > 2013` in order to remove the answers that necessary do not belong to the support of the answer. In the general case of a membership function having a trapezoidal form defined by a quadruple (a, b, c, d) , the derivation introduces two crisp conditions (`?var > a` and `?var < d`). Listing 4.6 of Example 53 below is an illustration of the derivation of the query R_{flat} .

Crisp interpretation

The previous compiling stage translates the fuzzy quantified query R embedding A fuzzy quantified statement and fuzzy conditions into a crisp query $R_{flatBoolean}$, whose interpretation is the classical Boolean one.

For the sake of simplicity, we consider in the following that the result of R_{flat} , denoted by $\llbracket r_{flat} \rrbracket$, is made of the quadruples $(?res_i, ?x_i, \mu_{Bi}, \mu_{Ai})$ matching the query.

Final result calculation

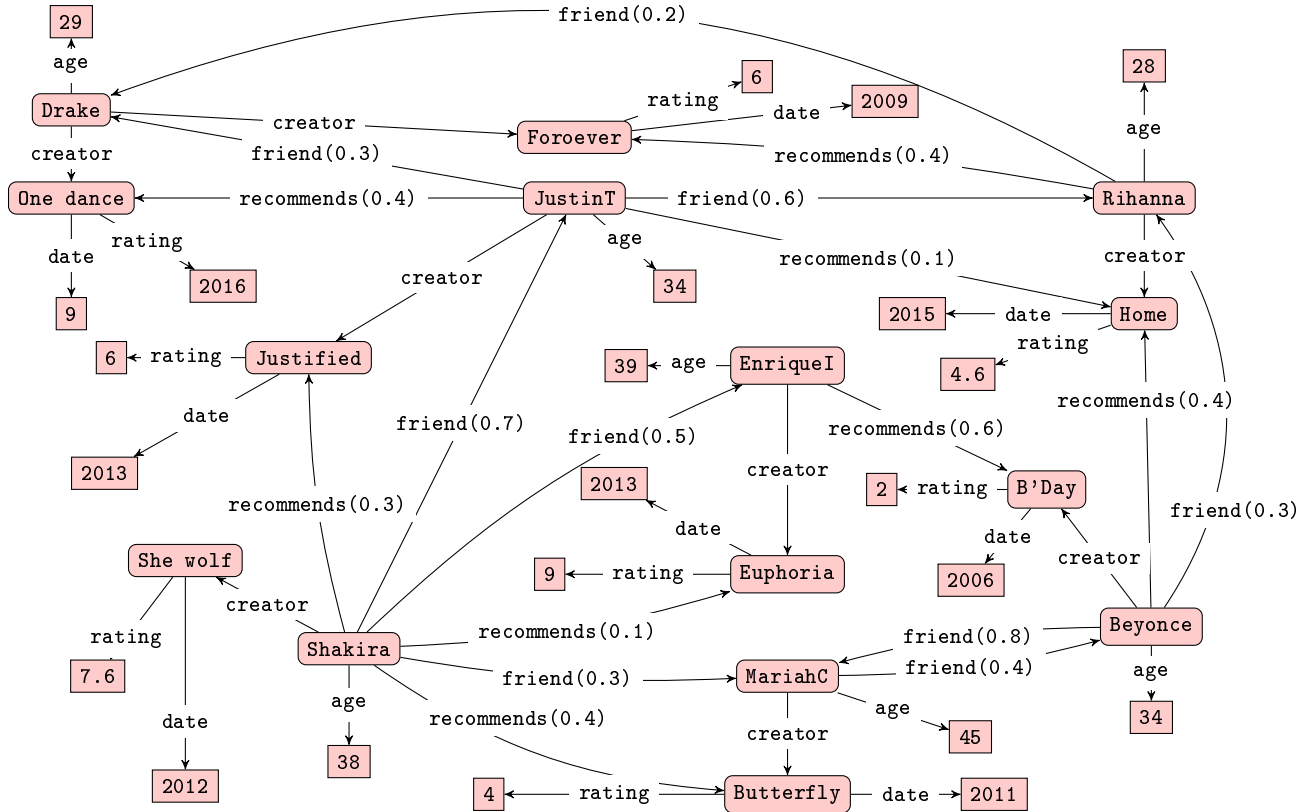
The last stage of the evaluation calculates the satisfaction degrees μ_B and μ_A according to I_B and I_A . If the optional part does not match a given answer, then $\mu_A = 0$. The answers of the initial fuzzy quantified query R (involving the fuzzy quantifier \mathcal{Q}) are answers of the query R_{flat} derived from R , and the final satisfaction degree associated with each element e can be calculated according to the three different interpretations mentioned earlier in Subsection 4.1.2. Hereafter, we illustrate this using [Zadeh, 1983] and [Yager, 1988]’s approaches (which are the most commonly used for interpreting *fuzzy quantified statements*).

- Following Zadeh’s Sigma-count-based approach (cf. Subsection 4.1.2.1) we have:

$$\mu(e) = \mu_{\mathcal{Q}} \left(\frac{\sum_{\{(?res_i, ?x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket R_{flat} \rrbracket \mid ?res_i = e\}} \min(\mu_{Ai}, \mu_{Bi})}{\sum_{\{(?res_i, ?x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket R_{flat} \rrbracket \mid ?res_i = e\}} \mu_{Bi}} \right) \quad (4.18)$$

In the case of a fuzzy absolute quantified query, the final satisfaction degree associated with each element e is simply

$$\mu(e) = \mu_{\mathcal{Q}} \left(\sum_{\{(?res_i, ?x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket R_{flat} \rrbracket \mid ?res_i = e\}} \mu_{Ai} \right).$$

Figure 4.5: Fuzzy RDF graph G_{MB} inspired by MusicBrainz

Example 53 [Evaluation of a Fuzzy Quantified Query] Let us consider the fuzzy quantified query $R_{mostAlbums}$ of Listing 4.2. We evaluate this query according to the fuzzy RDF data graph G_{MB} of Figure 4.5. In order to interpret $R_{mostAlbums}$, we first derive the following query R_{flat} from $R_{mostAlbums}$, that retrieves “the artists (?art1) who recommended at least one recent album (corresponds to $B(?art1, ?alb)$ in lines 2 and 3), possibly (OPTIONAL) highly rated and created by a young friend (corresponds to $A(?alb)$ in lines 5 to 7)”.

```

1 SELECT ?art1 ?alb  $\mu_B$   $\mu_A$  WHERE {
2   ?art1 recommends ?alb . ?alb date ?date .
3   FILTER (?date IS recent)
4   OPTIONAL {
5     ?art1 friend ?art2 . ?art2 creator ?alb .
6     ?alb rating ?rating . ?art2 age ?age .
7     FILTER (?rating IS high && ?age IS young) } }
```

Listing 4.5: Query R_{flat} derived from $R_{mostAlbums}$

Then, we evaluate the SPARQL query $R_{flatBoolean}$ given in Listing 4.6, derived from the FURQL nonquantified query R_{flat} of Listing 4.5.

```

1 SELECT ?art1 ?alb  $\mu_B$   $\mu_A$  WHERE {
2   ?art1 recommends ?alb . ?alb date ?date .
3   FILTER ( ?date > 2010.0 )
4   OPTIONAL {
5     ?art1 friend ?art2 . ?art2 creator ?alb .
6     ?alb rating ?rating . ?art2 age ?age .
7     FILTER ( ?rating > 2.0 && ?age < 40.0 ) } }
```

Listing 4.6: Query $R_{flatBoolean}$ derived from R_{flat}

This query returns a list of artist ($?art1$) with their recommended albums ($?alb$), satisfying the conditions of query R_{flat} , along with their respective satisfaction degrees

$$\mu_B = \min(\mu_{recent}(?alb), \zeta(?art1, recommends, ?alb)) \text{ and}$$

$$\mu_A = \min(\mu_{high}(?rating), \mu_{young}(?age), \zeta(?art1, friend, ?art2)).$$

where μ_p denotes the membership degree of the predicate p and $\zeta(t)$ denotes the membership value associated with the triple t (cf., Definition 4 on page 62).

For the sake of readability, the query of Listing 4.6 is a simplified version of the real derived query (cf. Listing A.1 in Appendix A).

According to the fuzzy RDF data graph G_{MB} of Figure 4.5, R_{flat} concerns three artists {JustinT, Shakira, Beyonce}. EnriqueI, Drake, Mariah and Rihanna do not belong to the result set of R_{flat} because EnriqueI, Drake and Mariah have not recommended any album made by any of their friends and Rihanna did not recommend any somewhat recent album.

Then, the set of answers of the query R_{flat} , denoted by $\llbracket R_{flat} \rrbracket$, is as follows:

$$\llbracket R_{flat} \rrbracket = \{$$

$$\begin{aligned}
& (?art1 \rightarrow \text{JustinT}, ?alb \rightarrow \text{One dance}, \mu_B \rightarrow 0.4, \mu_A \rightarrow 0.3), \\
& (?art1 \rightarrow \text{JustinT}, ?alb \rightarrow \text{Home}, \mu_B \rightarrow 0.1, \mu_A \rightarrow 0.6), \\
& (?art1 \rightarrow \text{Shakira}, ?alb \rightarrow \text{Euphoria}, \mu_B \rightarrow 0.1, \mu_A \rightarrow 0.07), \\
& (?art1 \rightarrow \text{Shakira}, ?alb \rightarrow \text{Butterfly}, \mu_B \rightarrow 0.2, \mu_A \rightarrow 0), \\
& (?art1 \rightarrow \text{Shakira}, ?alb \rightarrow \text{Justified}, \mu_B \rightarrow 0.3, \mu_A \rightarrow 0.4), \\
& (?art1 \rightarrow \text{Beyonce}, ?alb \rightarrow \text{Home}, \mu_B \rightarrow 0.4, \mu_A \rightarrow 0.3) \}.
\end{aligned}$$

Finally, assuming for the sake of simplicity that $\mu_{most}(x) = x$, the final result of the query $R_{mostAlbums}$ evaluated on G_{MB} using Formula 4.18 is:

$$\begin{aligned} \llbracket R_{mostAlbums} \rrbracket = \{ & \\ (\{?art1 \rightarrow JustinT\}, 0.80), & \\ (\{?art1 \rightarrow Beyonce\}, 0.75), & \\ (\{?art1 \rightarrow Shakira\}, 0.62)\}. & \diamond \end{aligned}$$

- Using Yager's OWA-based approach, for each element e returned by R_{flat} we calculate

$$\mu(e) = \sum_{\{(?res_i, ?x_i, \mu_{B_i}, \mu_{A_i}) \in \llbracket R_{flat} \rrbracket \mid ?res_i = e\}} w_i \times c_i. \quad (4.19)$$

Let us consider condition $B = \{\mu_{B_1}/x_1, \dots, \mu_{B_n}/x_n\}$ such that $\mu_{B_1} \leq \dots \leq \mu_{B_n}$, condition $A = \{\mu_{A_1}/x_1, \dots, \mu_{A_n}/x_n\}$ and $d = \sum_{i=1}^n \mu_{B_i}$.

The weights of the OWA operator are defined by

$$w_i = \mu_{\mathcal{Q}}(S_{x_i}) - \mu_{\mathcal{Q}}(S_{x_{i-1}})$$

with

$$S_{x_i} = \sum_{j=1}^i \frac{\mu_{B_j}}{d}$$

The implication values are denoted by $c_{x_i} = \max(1 - \mu_{B_i}, \mu_{A_i})$ and ordered decreasingly such that $c_1 \geq \dots \geq c_n$.

Example 54 In order to calculate $\mu(\text{Shakira})$ from R_{flat} , let us consider B (resp. A) the set of satisfaction degrees corresponding to condition B (resp. A) of element *Shakira* as follows $B = \{0.1/\text{Euphoria}, 0.2/\text{Butterfly}, 0.3/\text{Justified}\}$ and $A = \{0.07/\text{Euphoria}, 0/\text{Butterfly}, 0.4/\text{Justified}\}$. We have $d = 0.6$ and:

$$\begin{aligned} S_{Euphoria} &= \frac{0.1}{0.6} = 0.17, S_{Butterfly} = \frac{0.1 + 0.2}{0.6} = 0.5, \text{ and} \\ S_{Justified} &= \frac{0.1 + 0.2 + 0.3}{0.6} = 1. \end{aligned}$$

Then, with $\mu_{most}(x) = x$, we get $\mu_{\mathcal{Q}}(S_{Euphoria}) = 0.17$, $\mu_{\mathcal{Q}}(S_{Butterfly}) = 0.5$ and $\mu_{\mathcal{Q}}(S_{Justified}) = 1$.

Therefore, the weights of the OWA operator are:

$$\begin{aligned} W_1 &= \mu_{\mathcal{Q}}(S_{Euphoria}) - \mu_{\mathcal{Q}}(S_0) = 0.17, \\ W_2 &= \mu_{\mathcal{Q}}(S_{Butterfly}) - \mu_{\mathcal{Q}}(S_{Euphoria}) = 0.33, \text{ and} \\ W_3 &= \mu_{\mathcal{Q}}(S_{Justified}) - \mu_{\mathcal{Q}}(S_{Butterfly}) = 0.5. \end{aligned}$$

The implication values are:

$$\begin{aligned} c_{Euphoria} &= \max(1 - 0.1, 0.07) = 0.9, \\ c_{Butterfly} &= \max(1 - 0.2, 0) = 0.8, \text{ and} \\ c_{Justified} &= \max(1 - 0.3, 0.36) = 0.7. \end{aligned}$$

Thus, $c_1 = 0.9$, $c_2 = 0.8$ and $c_3 = 0.7$. Finally, we get:

$$\mu(\text{Shakira}) = 0.17 \times 0.9 + 0.33 \times 0.8 + 0.5 \times 0.7 = 0.15 + 0.26 + 0.35 = 0.77.$$

Finally, assuming for the sake of simplicity that $\mu_{most}(x) = x$, the final result of the query $R_{mostAlbums}$ evaluated on G_{MB} using Formula 4.19 is:

$$\begin{aligned} \llbracket R_{mostAlbums} \rrbracket &= \{ \\ &(\{?art1 \rightarrow Shakira\}, 0.77), \\ &(\{?art1 \rightarrow JustinT\}, 0.66), \\ &(\{?art1 \rightarrow Beyonce\}, 0.6) \}. \diamond \end{aligned}$$

Conclusion

In this chapter, we have investigated the issue of integrating *fuzzy quantified structural queries* of the type “ QBX are A ” into the FURQL query language (a fuzzy extension of the SPARQL that we proposed in Chapter 3) aimed to query fuzzy RDF databases. We have defined the syntax and semantics of an extension of FURQL, that makes it possible to deal with such queries. A query processing strategy based on the derivation of nonquantified fuzzy queries has also been proposed using different interpretations from the literature previously discussed in Section 4.1. The following chapter discusses implementation issues and presents some experiments.

Chapter 5

Implementation and Experimentations

Contents

Introduction	97
5.1 Implementation of FURQL	98
5.1.1 Storage of Fuzzy RDF Graphs	98
5.1.2 Evaluation of FURQL Queries	99
5.2 Experimentations	102
5.2.1 Experimental Setup	103
5.2.2 Experiments for nonquantified FURQL Queries	103
5.2.3 Experiments for Quantified FURQL Queries	109
Conclusion	112

Introduction

CHAPTERS 3 and 4 contain the main contributions of the thesis which consist of the definition of the FURQL query language, which is a fuzzy extension of SPARQL with *fuzzy preferences* (including *fuzzy quantified statements*) addressed to fuzzy RDF databases as well as crisp ones.

In the present chapter, in Section 5.1, we describe a prototype implementation of FURQL built on top of a classical SPARQL engine and, then in Section 5.2, we present a performance evaluation of the prototype system using different sizes of fuzzy RDF databases. The main objective behind these experiments is to show that the **extra cost due to the introduction of fuzziness remains limited/acceptable**.

5.1 Implementation of FURQL

In this section, we discuss implementation issues related to the FURQL query language. Two aspects have to be considered: i) the storage of fuzzy RDF graphs (see Subsection 5.1.1), and ii) the evaluation of FURQL queries with and without *fuzzy quantified statements* (see Subsection 5.1.2).

5.1.1 Storage of Fuzzy RDF Graphs

In this thesis we deal with *fuzzy RDF graph*, for which we need to attach fuzzy degrees to some edges in the RDF graph.

The classical RDF model does not naturally support this, but fortunately, it provides a mechanism, called *reification*, for making assertions or descriptions about statements. The *reification* of a statement in RDF is a description of this statement represented by a set of classical RDF triples. The vocabulary for doing so consists of *rdf:Statement*, *rdf:subject*, *rdf:predicate*, and *rdf:object*.

In our case and in order to attach fuzzy degrees to RDF triples, we use this *reification mechanism*. Here, a fuzzy RDF triple can be represented as a set of RDF triples. Example 55 illustrates this principle.

Example 55 The fuzzy RDF triple “(*Shakira*, *friends*, *MariahC*), 0.7” states that (*Shakira*, *friends*, *MariahC*) is satisfied to the degree 0.7, which could be interpreted as *Shakira* is a *close* friend of *MariahC*.

The representation of this fuzzy RDF triple using *reification* is given in Listing 5.1.

```

1 _:bn rdf:type rdf:Statement.
2 _:bn rdf:subject Shakira.
3 _:bn rdf:predicate friend.
4 _:bn rdf:object MariahC.
5 _:bn uri:degree "0.7".

```

Listing 5.1: Reification of a fuzzy RDF triple

The statement of Line 1 says that the resource identified by the blank RDF node (i.e., a node without label) *_:bn* is of type *RDF statement*. The statement of Line 2 indicates that the subject of the statement refers to the resource identified by *Shakira*, in Line 3 the predicate of the statement refers to the resource identified by *friend*, and in Line 4 the object of the statement refers to the resource *MariahC*. The satisfaction degree 0.7 is given by the statement in Line 5.

A possible graphical representation of this *reification* is depicted in Figure 5.1. The nodes in dashed lines represent reified nodes with the properties *rdf:type*,

rdf:subject, *rdf:predicate*, *rdf:object* and *uri:degree* that model respectively the type, the subject, the predicate, the object and the degree of the new statement. \diamond

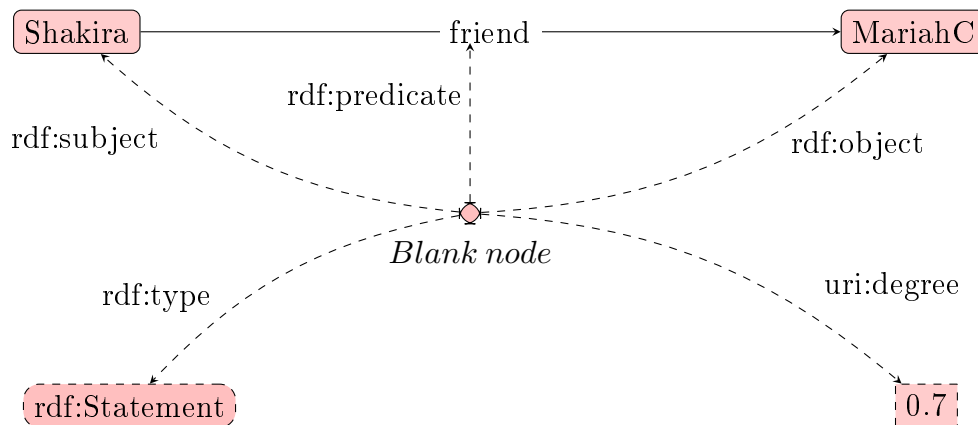


Figure 5.1: Reification of fuzzy triple of Example 55

In order to create a fuzzy RDF database, we start from a nonfuzzy RDF subgraph database for which every relationship between nodes is Boolean and then, we make it fuzzy by adding satisfaction degrees denoting the intensity of some relationships using the *reification* mechanism (as illustrated in Example 55).

5.1.2 Evaluation of FURQL Queries

Concerning the evaluation of FURQL queries, two architectures may be thought of:

- A first solution consists in implementing a specific query evaluation engine inside the data management system. Figure 5.2 is an illustration of this architecture. The advantage of this solution is that optimization techniques implemented directly in the query engine should make the system very efficient for query processing. An important downside is that the implementation effort is substantial, but the strongest objection for this solution is that the evaluation of a FURQL query in a distributed architecture would imply having available a FURQL query evaluator at each SPARQL endpoint, which is not realistic at the time being.
- An alternative more realistic architecture consists in adding a software add-on layer over a standard — and possibly distant — classical SPARQL engine (endpoint) which is the evaluation strategy that we adopted for processing FURQL queries.

This software, called SURF¹ (Sparql with fUZZy quantifieRs for rdF data), is implemented within the Jena Semantic Web Java Framework² for creating and manipulating

¹<https://www-shaman.irisa.fr/surf/>

²<https://jena.apache.org>

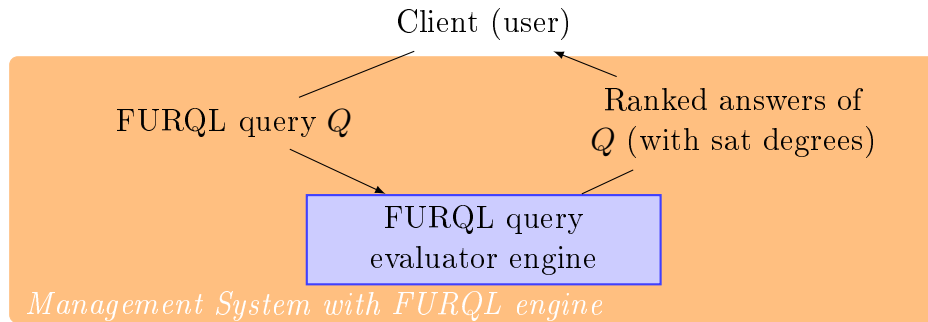


Figure 5.2: Implementation of a specific FURQL query evaluation engine

RDF graphs.

SURF evaluates FURQL queries that may contain *fuzzy quantified statements* whose syntax was presented in Chapter 3 and Chapter 4. It basically consists of two modules:

1. In a pre-processing step, the *Query compiler* module, produces
 - the query-dependent functions that allow to compute the satisfaction degrees for each returned answer,
 - a (crisp) SPARQL query which is then sent to the SPARQL query engine for retrieving the information needed to calculate the satisfaction degrees.

The compilation uses the *derivation principle* introduced in [Bosc and Pivert, 2000] in a relational database context that consists in translating a fuzzy query into a Boolean one.

2. In a post-processing step, the *Score calculator* module calculates the satisfaction degree for each returned answer, ranks the answers, and qualitatively filters them if an α -cut has been specified in the initial fuzzy query.

Figure 5.3 illustrates this architecture.

SURF makes it possible to query FURQL queries (including quantified ones) as well as regular SPARQL queries. The different evaluation scenarios are presented hereafter.

1. For a FURQL query (that does not involve any quantified statement), the principle is simple, we first evaluate the corresponding (crisp) SPARQL query returned by the *Query compiler* module (obtained using the derivation rules). For each tuple x from the result of the crisp SPARQL query, we calculate its satisfaction degree using the *Score calculator* module. Finally, a set of answers ranked in decreasing order of their satisfaction degree is returned.

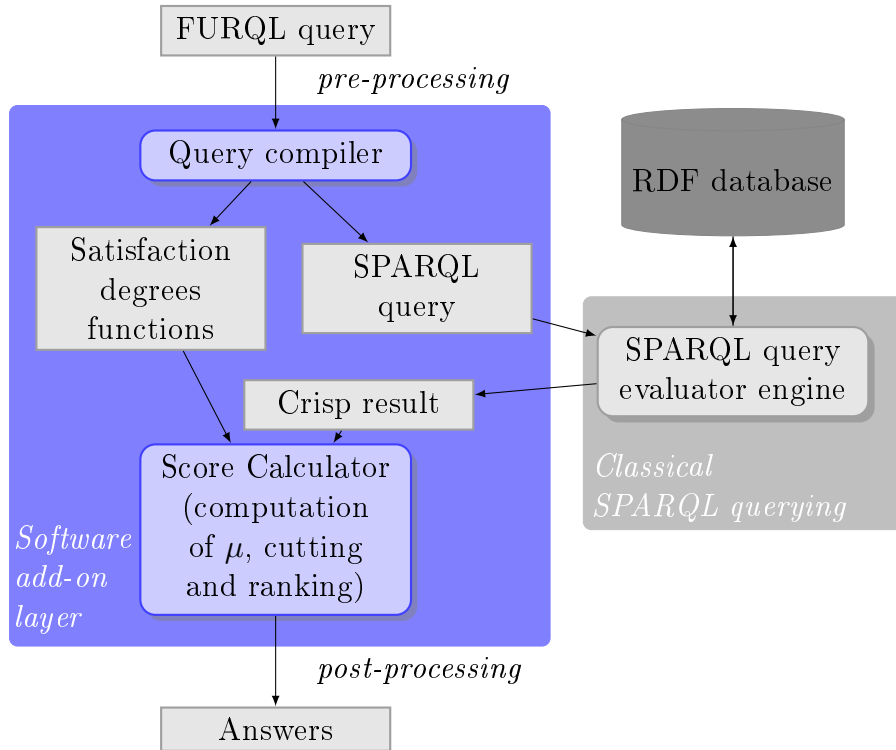


Figure 5.3: SURF software architecture

2. For quantified queries of the type “ $Q B X$ are A ”, the principle is to first evaluate the (crisp) SPARQL query (returned by the *Query compiler* module) derived from the original query. We first perform a `group by` of the elements from the result of the derived query and then for each tuple from the result set, we return the satisfaction degrees related to conditions A and B , denoted respectively by μ_A and μ_B . The final satisfaction degree μ can be calculated according to Formulas (4.3), (4.8) or (4.17) (presented in Chapter 4 Subsection 4.1.2) using the value of μ_B and μ_A .

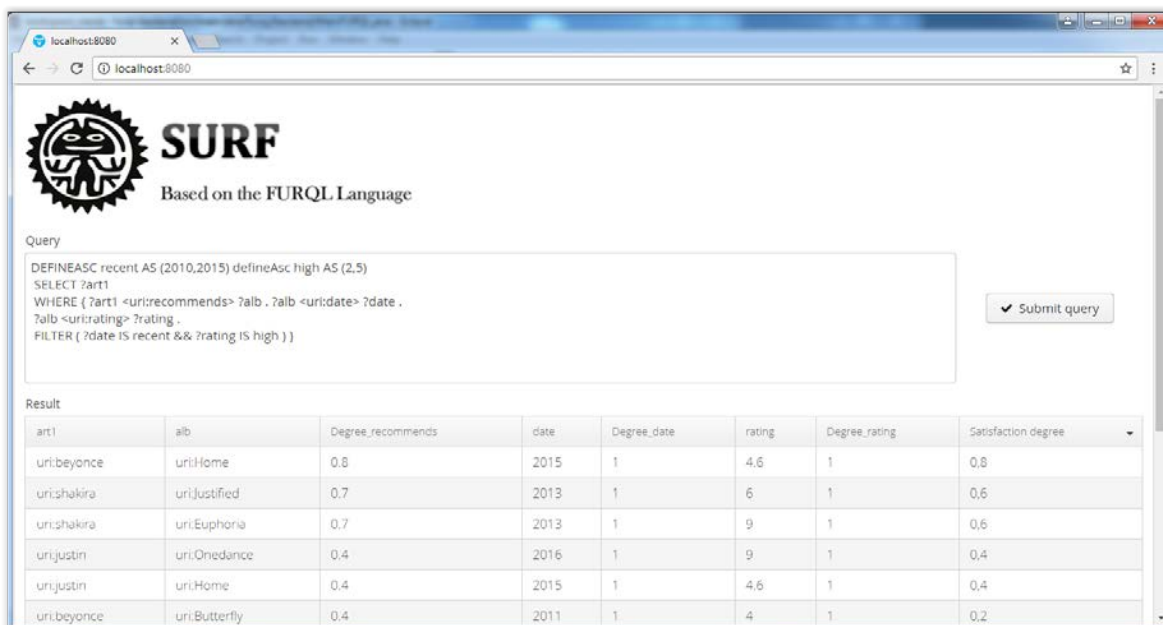
At the current time, Zadeh’s approach [Zadeh, 1983] and Yager’s OWA-based approach [Yager, 1988] have been implemented, and the choice of the interpretation to be used is made through the configuration tool of the system. Finally, we get a set of answers ranked in decreasing order of their satisfaction degree.

3. For a classical SPARQL query, we skip the *Query compiler* and *Score calculator* modules and the original query is transferred directly to the classical SPARQL engine. All the answers returned by the SPARQL engine are kept in the final resultset with a satisfaction degree equal to 1.

The SURF GUI was created using Vaadin³, a web framework for Java under NetBeans IDE 8.2. It is mainly composed of two frames:

- an input text area for entering and running a FURQL query, and
- a table for visualizing the results of a query.

Example 56 Figure 5.4 presents a screenshot of the SURF GUI, which contains the final result of the evaluation of a FURQL query.



The screenshot shows a web browser window with the SURF logo and the text "Based on the FURQL Language". Below the logo is a text area containing a FURQL query:

```
Query
DEFINEASC recent AS (2010,2015) defineAsc high AS (2,5)
SELECT ?art1
WHERE { ?art1 <uri:recommends> ?alb . ?alb <uri:date> ?date .
?alb <uri:rating> ?rating .
FILTER { ?date IS recent && ?rating IS high } }
```

To the right of the query input is a "Submit query" button. Below the query input is a table with the following data:

art1	alb	Degree_recommends	date	Degree_date	rating	Degree_rating	Satisfaction degree
uri:beyonce	uri:Home	0.8	2015	1	4,6	1	0,8
uri:shakira	uri:Justified	0.7	2013	1	6	1	0,6
uri:shakira	uri:Euphoria	0.7	2013	1	9	1	0,6
uri:justin	uri:Onedance	0.4	2016	1	9	1	0,4
uri:justin	uri:Home	0.4	2015	1	4,6	1	0,4
uri:beyonce	uri:Butterfly	0.4	2011	1	4	1	0,2

Figure 5.4: Screenshot of SURF

The FURQL prototype and some interactive examples of queries are available and downloadable at <https://www-shaman.irisa.fr/furql/>.

5.2 Experimentations

In order to demonstrate the performances of our approach in the case of fuzzy graph pattern queries, we ran two experiments in order to calculate the execution time of each step of the evaluation for FURQL queries with and without quantified statements and then to assess the cost of adding fuzzy preferences for each type of queries.

In the following, we first present the setup we used for the evaluation and then, we describe in detail each experiment.

³<https://vaadin.com/home>

5.2.1 Experimental Setup

All of the experiments were carried out on a personal computer running Windows 7 (64 bits) with 8GB of RAM.

For these experiments, we used four different sizes of fuzzy RDF datasets containing crisp and fuzzy triples, as described in Table 5.1. Our RDF data is inspired by Musicbrainz⁴ linked data (which is originally crisp), and for representing fuzzy information, we used the reification mechanism that makes it possible to attach fuzzy degrees to triples, as discussed earlier in Subsection 5.1.1

Table 5.1: Fuzzy RDF datasets

Dataset	Size	Reified Triples
DB_1	11796 triples	47185 triples
DB_2	65994 triples	263977 triples
DB_3	112558 triples	450393 triples
DB_4	175416 triples	701665 triples

A java script have been developed to create random fuzzy RDF data of different sizes.

In the following, we first present experiments on nonquantified FURQL queries (Section 5.2.2) and then on quantified ones (Section 5.2.3).

5.2.2 Experiments for nonquantified FURQL Queries

For this experiment, we considered different kinds of nonquantified FURQL queries (summarized in Table 5.2), based on the typology presented in [Umbrich et al., 2015]. Three types have been used. For each kind of queries, we consider two fuzzy subtypes: 1) a subtype for which a condition concerns a value, and 2) a subtype for which a condition concerns the intensity of the relationships. Such subtype is called “Structural” in the following.

- **Edge queries:** that consist in retrieving an entity e by means of a pattern where e may appear either i) in the subject (denoted by $edge-s$), ii) in the object (denoted by $edge-o$), or iii) both (denoted by $edge-so$).

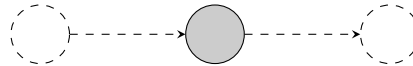


Figure 5.5: Edge query of the form $edge-so$

We consider in the following four edge queries of the form $edge-so$ given in Figure 5.5. Query $Q_{1.2}$ is a fuzzy $edge$ query containing a fuzzy condition that aims to find the *recent* albums recommended by an artist; Its corresponding crisp query, denoted by

⁴<https://musicbrainz.org/>

$Q_{1.1}$, aims to find the albums recommended by an artist and released after 2014. $Q_{1.1}$ is given in Listing 5.2 for which the reification of fuzzy triples is made explicit in the query and $Q_{1.2}$ is given in Listing 5.3, the reification here is implicit and it is performed in the *Query compiler* stage.

```

SELECT ?alb WHERE {
  ?alb date ?d .
  /* reification */
  ?X1 subject ?art1.
  ?X1 predicate recommends.
  ?X1 object ?alb. ?X1 degree ?degree.
  FILTER ( ?d > 2014 ) }

```

Listing 5.2: Crisp *edge* query $Q_{1.1}$

```

DEFINEASC recent AS (2014,2017)
SELECT ?alb WHERE
{
  ?alb date ?d .
  ?art recommends ?alb .
  FILTER ( ?d IS recent )
}

```

Listing 5.3: Fuzzy *edge* query $Q_{1.2}$

Query $Q_{1.4}$ is a fuzzy structural *edge* query containing a fuzzy structural condition that aims to find *highly* recommended albums with a known release date. Its crisp version, denoted by $Q_{1.3}$, aims to find the albums recommended with a degree greater than 0.8 and having a known release date. $Q_{1.3}$ (resp., $Q_{1.4}$) is given in Listing 5.4 (resp., Listing 5.3).

```

SELECT ?alb WHERE {
  ?alb date ?d.
  /* reification */
  ?X1 subject ?art1.
  ?X1 predicate recommends.
  ?X1 object ?alb. ?X1 degree ?degree.
  FILTER ( ?degree > 0.8 ) }

```

Listing 5.4: Crisp structural *edge* query $Q_{1.3}$

```

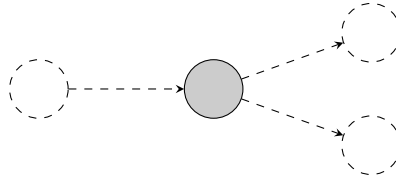
DEFINEASC strong AS (0.7, 0.9)
SELECT ?alb WHERE
{
  ?alb date ?d.
  /* structural condition */
  ?art (recommends | ST IS strong) ?alb.
}

```

Listing 5.5: Fuzzy structural *edge* query $Q_{1.4}$

- **Star queries** (star-shaped queries): consist of three acyclic triple patterns that share the same node (called central node). The central node may appear in different positions; i.e., it can be the subject of the three triples patterns (denoted by *star-s3*), the object of three triples patterns (denoted by *star-o3*), the subject of a triple patterns and the object of the two others (denoted by *star-s1-o2*), or the subject of two triples patterns and the object of the remaining triple pattern (denoted by *star-s2-o1*).

Again we used four queries of the form *star-s2-o1* shown in Figure 5.6.

Figure 5.6: Star query of the form *star-s2-o1*

Query $Q_{2.2}$ is a fuzzy star query containing a fuzzy condition that aims to retrieve the *recent* albums (with a known rating) recommended by an artist. Its corresponding crisp query, denoted by $Q_{2.1}$, aims to retrieve the albums released after 2014 (with a known rating) recommended by an artist. $Q_{2.1}$ (resp. $Q_{2.2}$) is shown in Listing 5.6 (resp. Listing 5.7).

```

SELECT ?alb WHERE {
  ?alb date ?d. ?alb rating ?r.
  /* reification */
  ?X1 subject ?art.
  ?X1 predicate recommends.
  ?X1 object ?alb. ?X1 degree ?degree.
  FILTER ( ?d > 2014 ) }

```

Listing 5.6: Crisp star query $Q_{2.1}$

```

DEFINEASC recent AS (2014,2017)
SELECT ?alb WHERE {
  ?alb date ?d.
  ?alb rating ?r.
  ?art recommends ?alb.
  FILTER ( ?d IS recent )
}

```

Listing 5.7: Fuzzy star query $Q_{2.2}$

Query $Q_{2.4}$ is a fuzzy structural star query containing a fuzzy structural condition that aims to find the *highly* recommended albums (with a known rating and release date) and its corresponding crisp version, denoted by $Q_{2.3}$, aims to find the recommended albums that have a recommendation degree greater than 0.8 (with a known rating and release date). $Q_{2.3}$ is given in Listing 5.8 and $Q_{2.4}$ is given in Listing 5.9.

```

SELECT ?alb WHERE {
  ?alb date ?d. ?alb rating ?r .
  /* reification */
  ?X1 subject ?arti.
  ?X1 predicate recommends.
  ?X1 object ?alb. ?X1 degree ?degree.
  FILTER ( ?degree > 0.8 ) }

```

Listing 5.8: Crisp structural star query
 $Q_{2.3}$

```

DEFINEASC strong AS (0.7, 0.9)
SELECT ?alb WHERE {
  ?alb date ?d.
  ?alb rating ?r.
  /* structural condition */
  ?art (recommends | ST IS strong) ?alb.
}

```

Listing 5.9: Fuzzy structural star query
 $Q_{2.4}$

- **Path queries:** consist of two or three triple patterns that form a path such that two triples share a variable. We may find path shaped queries of length two or three. We consider in the following an example of a path shaped query of length three of the form given in Figure 5.7.

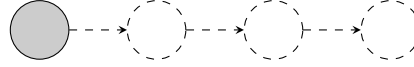


Figure 5.7: Star query of the form *path-3*

Query $Q_{3.2}$ in Listing 5.11 is a fuzzy path query containing a fuzzy condition. It aims to find every artist who has among his friends an artist who created a *recently* released album. Its corresponding crisp query in Listing 5.10, denoted by $Q_{3.1}$, aims to find every artist who has among his friends an artist who created an album released after 2014.

<pre> SELECT ?art1 WHERE { ?art2 creator ?alb. ?alb date ?d. /* reification */ ?X1 subject ?art1. ?X1 predicate friends. ?X1 object ?art2. ?X1 degree ?degree. FILTER (?d > 2014) } </pre>	<pre> DEFINEASC recent AS (2014, 2017) SELECT ?art1 WHERE { ?art2 creator ?alb. ?alb date ?d. ?art1 friend ?art2. FILTER (?d IS recent) } </pre>
---	--

Listing 5.10: Crisp path query $Q_{3.1}$

Listing 5.11: Fuzzy path query $Q_{3.2}$

Query $Q_{3.4}$ is a fuzzy structural simple path query containing a fuzzy structural condition that aims to find every artist who has among his close friends an artist who created an album (cf., Listing 5.13). Its crisp counterpart, denoted by $Q_{3.3}$, aims to find every artist who has among his friends (with a friendship degree greater than 0.8) an artist who created an album (cf., Listing 5.12).

<pre> SELECT ?art1 WHERE { ?art2 creator ?alb. ?alb date ?d . /* reification */ ?X1 subject ?art1. ?X1 predicate friend. ?alb date ?d. ?X1 object ?art2. ?X1 degree ?degree. FILTER (?degree > 0.8) } </pre>	<pre> DEFINEASC strong AS (0.7, 0.9) SELECT ?alb WHERE { ?art2 creator ?alb. /* structural condition */ ?art1 (friend ST IS strong) ?art2 .} </pre>
---	---

Listing 5.12: Crisp structural path query $Q_{3.3}$

Listing 5.13: Fuzzy structural path query $Q_{3.4}$

We evaluated separately each type of queries over the different sizes of database given in Table 5.1 on page 103. The results of these queries are depicted in Figure 5.8. Figure 5.8.(a)

Table 5.2: Different types of FURQL queries

Type	crisp query	Fuzzy Condition	Fuzzy Structural
Edge query	$Q_{1.1}, Q_{1.3}$	$Q_{1.2}$	$Q_{1.4}$
Star query	$Q_{2.1}, Q_{2.3}$	$Q_{2.2}$	$Q_{2.4}$
Simple path query	$Q_{3.1}, Q_{3.3}$	$Q_{3.2}$	$Q_{4.4}$

(resp., Figure 5.8.(b)) presents the execution time in milliseconds of the processing of the *edge* queries (resp., *star* queries) from Table 5.2. Figure 5.8.(c) presents the execution time in milliseconds of the processing of the *path* queries from Table 5.2.

The execution time is the elapsed time between submitting the query to the system and obtaining the query answers, it is measured in milliseconds using the system command time.

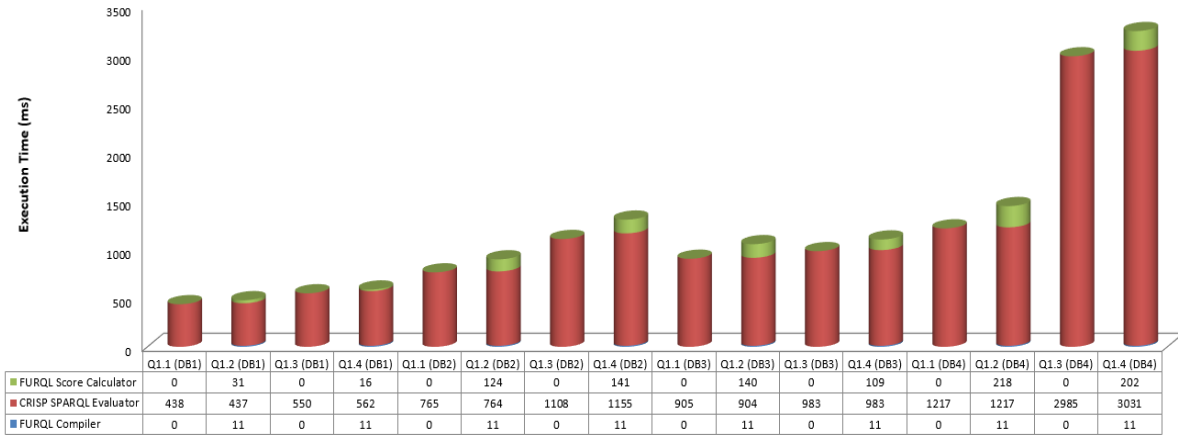
A first (and predictable) observation is that, for each crisp and fuzzy query presented in Table 5.2, the processing time of the overall process is proportional to the size of the dataset, the number of the results and the complexity of the query.

It is straightforward to see that for all the crisp queries the *query compiler* and the *score calculator* modules do not play any role in the processing of the queries. Thus, the corresponding execution times in Figure 5.8 are equal to 0. In the case of fuzzy queries, these modules, which are directly related to the introduction of flexibility into the query language, are strongly dominated in time by the crisp SPARQL evaluator (which includes the time for executing the query and getting the result set). As we can see in Figure 5.8, the time of the evaluation of the initial query by the SPARQL evaluator engine represents at least 89% of the overall process.

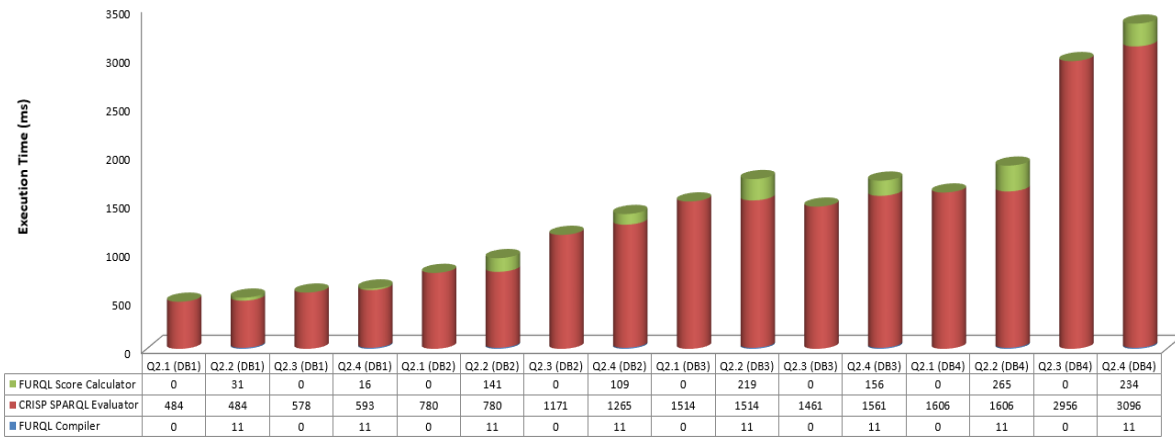
Moreover, the FURQL *compiling* module takes so little time compared to the other two steps that it cannot even be seen in Figure 5.8. This time remains almost constant, and is then independent on the size of the dataset. As to the *score calculation* module, the time used for calculating the final satisfaction degrees is slightly higher than the last step and is dependent on the size of the result set and the nature of the query.

Comparing the pairwise queries ($Q_{i.1}$ with $Q_{i.2}$ and $Q_{i.3}$ with $Q_{i.4}$), we see that the processing time of the fuzzy query is slightly higher than that of its crisp version. The increase is 10% on average.

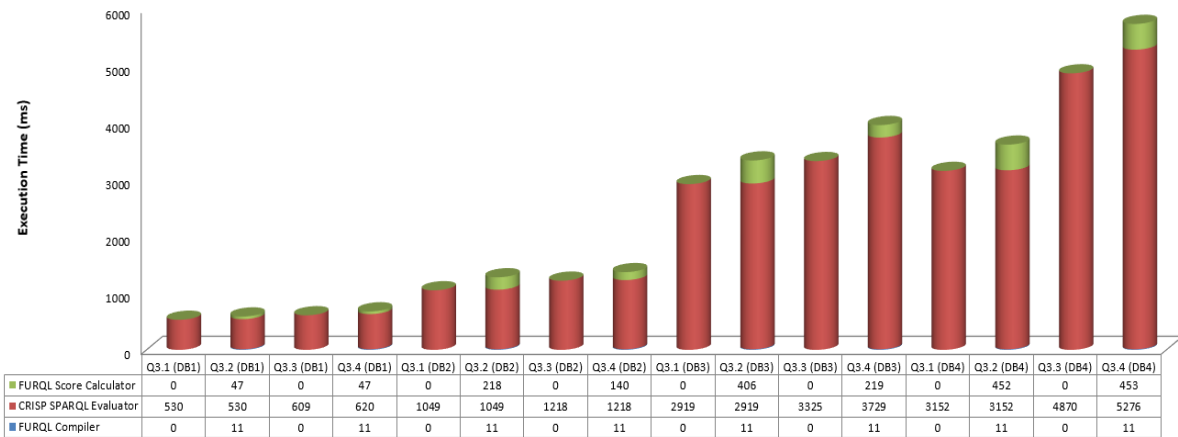
Finally, the results obtained tend to show that introducing fuzziness into a SPARQL query entails a rather small increase of the overall processing time. According to our experimentations, it represents around 11% of the overall time needed for evaluating a FURQL query in the worst case.



(a) Different types of edge queries over different sizes of DB



(b) Different types of star queries over different sizes of DB



(b) Different types of path queries over different sizes of DB

Figure 5.8: Experimental results about the evaluation of FURQL queries

Although these experimental results are preliminary observations, they appear very encouraging since they show that our approach does not entail any important overhead cost.

5.2.3 Experiments for Quantified FURQL Queries

Concerning the evaluation of *fuzzy quantified queries*, two set of experiments were carried out with two different types:

- *Fuzzy quantified queries* involving crisp conditions (see an example in Listing 5.14).

```

DEFINEQRELATIVEASC most AS (0,1)
SELECT ?art1 WHERE {
    ?art1 recommends ?alb . ?alb date ?date .
    FILTER ( ?date > 2014 ) }
GROUP BY ?art1
HAVING most(?alb) ARE ( ?art1 friend ?art2 . ?art2 age ?age .
    ?art2 creator ?alb . ?alb rating ?rating .
    FILTER ( ?rating > 5 && ?age < 30 ) )

```

Listing 5.14: A fuzzy quantified query involving crisp conditions

- *Fuzzy quantified queries* involving fuzzy conditions (see an example in Listing 5.15).

```

DEFINEQRELATIVEASC most AS (0.3,0.8), DEFINEASC high AS (2,5)
DEFINEDESC young AS (25,40), DEFINEASC recent AS (2010,2015)
SELECT ?art1 WHERE {
    ?art1 recommends ?alb . ?alb date ?date .
    FILTER ( ?date IS recent ) }
GROUP BY ?art1
HAVING most(?alb) ARE ( ?art1 friend ?art2 . ?art2 creator ?alb .
    ?alb rating ?rating . ?art2 age ?age .
    FILTER (?rating IS high && ?age IS young) )

```

Listing 5.15: A fuzzy quantified query involving fuzzy conditions

The main objective of these experiments is to assess the cost of each stage involved in the evaluation of *fuzzy quantified queries* and to show that the extra cost due to the introduction of *fuzzy quantified statements* remains limited/acceptable.

Fuzzy quantified query involving crisp conditions

In the first experiment, we processed four *fuzzy quantified queries* with crisp conditions (of the type “ $Q B X$ are A ”) by changing each time the nature of the patterns corresponding to conditions B and A from simple to complex ones. These queries are summarized in Table 5.3.

A complex pattern differs from a simple one by the number of its statements. Here, a complex pattern is composed of nine triple patterns at most, while a simple pattern has

Table 5.3: Set of fuzzy quantified queries with crisp conditions

Query	P_B	P_A	Conditions
$Q1_{crisp}$	simple	simple	crisp
$Q2_{crisp}$	complex	simple	crisp
$Q3_{crisp}$	simple	complex	crisp
$Q4_{crisp}$	complex	complex	crisp

between two or four triple patterns. Each Q_{crisp} contains three crisp conditions. These queries are detailed in Appendix A.

In order to evaluate these queries, we used Yager’s OWA-based interpretation. The results, depicted in Figure 6.8, present the execution time in milliseconds of the processing of the *fuzzy quantified queries* involving crisp conditions from Table 5.3 over the RDF datasets from Table 5.1 on page 103.

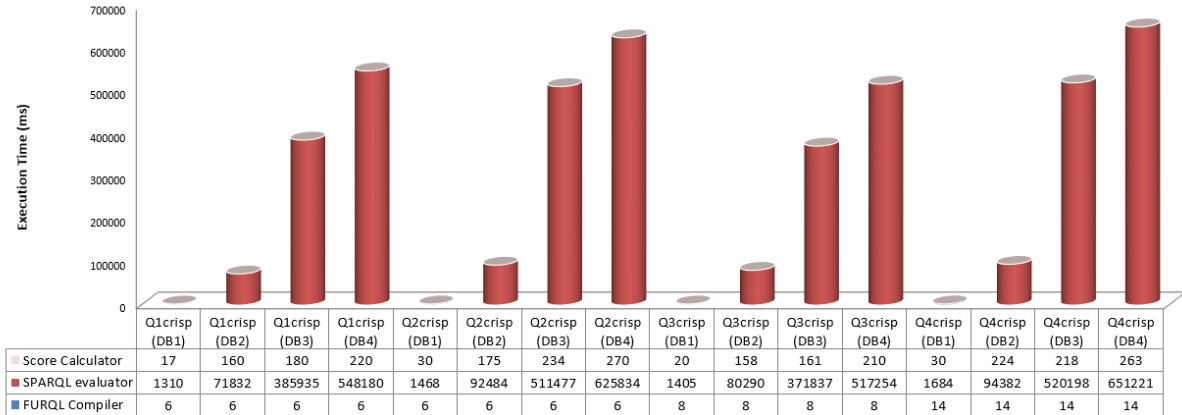


Figure 5.9: Experimental results of Fuzzy Quantified queries involving crisp conditions

These results are commented at the end of the section.

Fuzzy quantified query involving fuzzy conditions

We processed again four *fuzzy quantified queries* with fuzzy conditions (of the type “ $Q B X$ are A ”) by changing each time the nature of the patterns corresponding to conditions B and A from simple to complex ones. Table 5.4 presents these queries.

A complex pattern differs from a simple one by the number and the nature (including structural properties) of its statements. During these experiments, a complex pattern is composed of nine triple patterns at most, while a simple pattern contains between two and four triple patterns. For each complex pattern a fuzzy structural property (e.g., involving the notions of strength or distance) is involved. Each Q_{fuzzy} contains three fuzzy conditions. These queries are detailed in Appendix A.

Table 5.4: Set of fuzzy quantified queries with fuzzy conditions

Query	P_B	P_A	Conditions
$Q1_{fuzzy}$	simple	simple	fuzzy
$Q2_{fuzzy}$	complex	simple	fuzzy
$Q3_{fuzzy}$	simple	complex	fuzzy
$Q4_{fuzzy}$	complex	complex	fuzzy

The results of these experiments, using Yager’s OWA-based interpretation, are depicted in Figure 5.10 that presents the execution time in milliseconds of the processing of the *fuzzy quantified queries* from Table 5.4 over the RDF datasets from Table 5.1 on page 103.

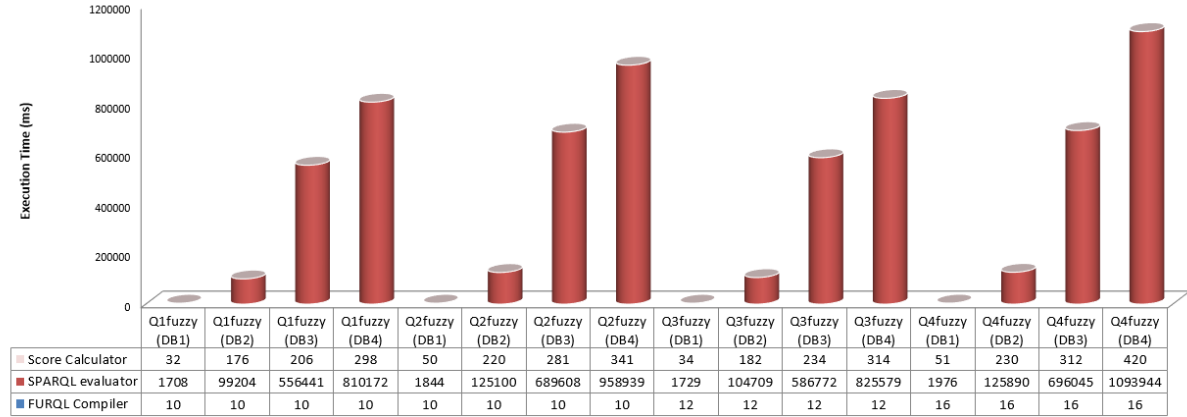


Figure 5.10: Experimental results of Fuzzy Quantified queries involving fuzzy conditions

Results interpretation

A first and obvious observation from Figure 5.9 and Figure 5.10 is that, for all the *fuzzy quantified queries*, the processing time taken by the overall process is proportional to the size of the dataset and the complexity of the pattern in the query.

One can see that, the processing time taken by the *compiling* and the *score calculation* module, which are directly related to the introduction of flexibility into the query language, are very strongly dominated by the time taken by the SPARQL evaluator (which includes the time for executing the query and getting the result set). As it is shown in Figure 5.9 and Figure 5.10, the time of the evaluation of the initial query by the SPARQL evaluator engine represents 99% on average of the overall process.

Indeed, the FURQL *compiling* step takes so little time compared to the *score calculation* and the SPARQL evaluator modules that it cannot even be seen in Figure 5.9 and Figure 5.10. This time remains almost constant, and is independent on the size of the

dataset while slightly increasing in the presence of complex patterns or fuzzy conditions. Moreover, the time needed for calculating the final satisfaction degree in the *score calculator* module is relatively dependent on the size of the result set and the nature of the patterns.

Again, these experimental results, even though preliminary, appear promising. They tend to show that introducing *fuzzy quantified statements* into a SPARQL query does not come with a high price (i.e., entails a very small increase of the overall processing time).

Finally, this conclusion can be extended to the case of Zadeh's interpretation [Zadeh, 1983], inasmuch as it is even more straightforward, in terms of computation, than Yager's OWA-based approach [Yager, 1988]. Thus, the processing time of the score calculating step can only be smaller than in the case of Yager's OWA-based interpretation.

Conclusion

In this chapter, in Section 5.1, we discussed implementation issues related to the FURQL language and we presented an architecture which consists of a software add-on layer (called SURF) over the classical SPARQL engine. Then, in Section 5.2, we performed two set of experiments over different sizes of datasets in order to study the performances of our proposed approach. The first experiments aimed to measure the additional cost induced by the introduction of fuzziness into SPARQL, and the results obtained show the efficiency of our proposal. The second experiments, which concerned *fuzzy quantified queries*, show that the extra cost induced by the fuzzy quantified nature of the queries remains very limited, even in the case of rather complex *fuzzy quantified queries*.

The results of the experiments performed in this chapter are summarized in Table 5.5. Each cell of the table contains three values corresponding to the percentage of time devoted to the *compilation*, the *crisp evaluation* and the *score calculation* stages respectively. They show that in both experiments the *compilation* and the *score calculation* stages are strongly dominated by the *crisp SPARQL evaluation*. The latter represents at least 95% of the overall process. Thus, these results confirm the hypothesis that the **extra cost due to the introduction of fuzziness remains limited/acceptable**.

Finally, these experiments are preliminary and more work is required to further assess FURQL by using different variety of queries (e.g., complex path queries of undetermined length) and considering large databases.

Table 5.5: Experimental results summarization

	DB1	DB2	DB3	DB4	Average
Nonquantified queries					
Nonquantified edge queries	(1.04, 96.66, 2.30)	(0.52, 93.34, 6.15)	(0.51, 93.70, 5.79)	(0.2, 94.40, 5.33)	(0.59, 94.53, 4.89)
Nonquantified star queries	(0.97, 96.92, 2.12)	(0.49, 93.76, 5.75)	(0.32, 94.29, 5.40)	(0.23, 94.50, 5.27)	(0.50, 94.86, 4.63)
Nonquantified path queries	(0.87, 95.40, 3.73)	(0.42, 92.76, 6.82)	(0.15, 95.42, 4.43)	(0.12, 94.78, 5.10)	(0.12, 94.78, 5.10)
Quantified queries					
Quantified queries with crisp conditions	(0.55, 97.85, 1.60)	(0.01, 99.78, 0.21)	(0.00, 99.95, 0.04)	(0.00, 99.96, 0.04)	(0.14, 99.38, 0.47)
Quantified queries with fuzzy conditions	(0.64, 97.14, 2.22)	(0.01, 99.81, 0.18)	(0.00, 99.96, 0.04)	(0.00, 99.96, 0.04)	(0.16, 99.22, 0.62)

Chapter 6

Extensions to General Graph Databases

Contents

Introduction	115
6.1 Background Notions	116
6.1.1 Graph Databases	116
6.1.2 Fuzzy Graphs	117
6.1.3 Fuzzy Graph Databases	120
6.1.4 The FUDGE Query Language	120
6.2 Related Work	122
6.3 Fuzzy Quantified Statements in FUDGE	124
6.3.1 Syntax of a Fuzzy Quantified Query	125
6.3.2 Evaluation of a Fuzzy Quantified Query	126
6.4 About Query Processing	131
6.5 Experimental Results	133
Conclusion	137

Introduction

IN the previous chapters, we addressed mainly the issue of defining an efficient approach for flexible querying in a particular type of graph databases, namely RDF databases. This approach makes it possible to express fuzzy nonquantified and quantified queries into an extension of the SPARQL language.

In the present chapter, we place ourselves in a more general framework: graph database [Angles and Gutierrez, 2008]. An efficient approach for flexible querying of fuzzy

graph databases has been proposed in [Pivert et al., 2014b]. This approach makes it possible to express only fuzzy nonquantified conditions. However, *fuzzy quantified queries* have a high potential in this setting since they can exploit the *structure* of the graph, beside the attribute values attached to the nodes or edges. So far, only one approach from the literature, described in [Castelltort and Laurent, 2014], considered *fuzzy quantified queries* to graph databases but only in a rather limited way.

This chapter is based on our work reported in [Pivert et al., 2016e], in which we showed how it is possible to integrate *fuzzy quantified queries* in a framework named FUDGE that was previously defined in [Pivert et al., 2014a]. FUDGE is a fuzzy extension of Cypher [Cypher, 2017] which is a declarative language for querying (crisp) graph databases. This work is mostly related to the work presented in Chapter 4 in which we deal with the same type of *fuzzy quantified structural query* but in a more specific type of graph databases, called RDF database.

The remainder of this chapter is organized as follows. Section 6.1 presents the different elements that constitute the context of the work. Section 6.2 discusses related work. In Section 6.3, we propose a syntactic format for expressing *fuzzy quantified queries* in the FUDGE language, and we describe its interpretation. Section 6.4 deals with query processing and discusses implementation issues. In Section 6.5, some experimental results showing the feasibility of the approach are presented.

6.1 Background Notions

In this section, we recall important notions about graph databases, fuzzy graph theory, fuzzy graph databases, and the query language FUDGE.

6.1.1 Graph Databases

In the last few years, graph databases has attracted a lot of attention for their ability to handle complex data in many application domains, e.g., social networks, cartographic databases, bibliographic databases, etc [Angles and Gutierrez, 2008, Angles, 2012]. They aim to efficiently manage networks of entities where each node is described by a set of characteristics (for instance a set of attributes), and each edge represents a link between entities.

A graph database management system enables managing data for which the data structure of the schema is modeled as a graph and data is handled through graph-oriented operations and type constructors [Angles and Gutierrez, 2008]. Among the existing systems, let us mention AllegroGraph [allegrograph, 2017], InfiniteGraph [infinitegraph, 2017], Neo4j [Neo4j, 2017] and Sparksee [sparksee, 2017]. Different models of graph databases have been proposed in the

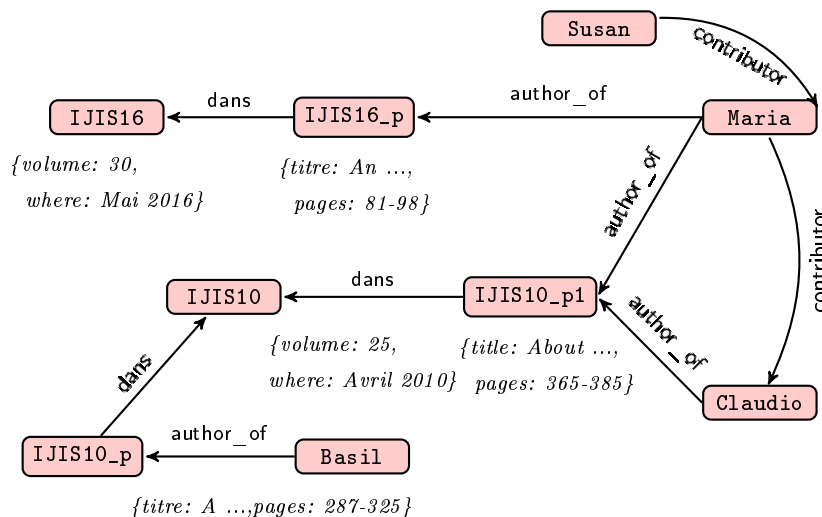


Figure 6.1: An Attributed graph inspired from DBLP

literature (see [Angles and Gutierrez, 2008] for an overview), including the *attributed graph* (aka., *property graph*) aimed to model a network of entities with embedded data. In this model, nodes and edges can be described by data in *attributes* (aka., *properties*).

Example 57 Figure 6.1 is an example of an *attributed graph*, inspired from DBLP¹ with crisp edges.

Nodes are assumed to be typed. If n is a node of V , then $Type(n)$ denotes its type. In Figure 6.1, the nodes IJIS16 and IJIS10 are of type *journal*, the nodes IJIS16-p, IJIS10-p and IJIS10-p1 are of type *paper*, and the nodes Maria, Claudio and Susan are of type *author*. For nodes of type *journal*, *paper* and *author*, a property, called *name*, contains the identifier of the node. Information about the *title* and the *pages* may be attached to node of type *paper* and information about the *volume* and the *date* may be attached to node of type *journal*. In Figure 6.1, the value of the property *name* for a node appears inside the node. \diamond

Such a model may be extended into the notion of a fuzzy graph database where a degree may be attached to edges in order to express the “intensity” of any kind of gradual relationship (e.g., likes, is friends with, is about). In the following section, we introduce the notion of fuzzy graphs.

6.1.2 Fuzzy Graphs

A *graph* G is a pair (V, R) , where V is a set and R is a relation on V . The elements of V (resp. R) correspond to the vertices (resp. edges) of the graph. Similarly, any fuzzy relation ρ on

¹<http://www.informatik.uni-trier.de/~ley/db/>

a set V can be regarded as defining a weighted graph, or fuzzy graph, see [Rosenfeld, 1975], where the edge $(x, y) \in V \times V$ has weight or strength $\rho(x, y) \in [0, 1]$. Having no edge between x and y is equivalent to $\rho(x, y) = 0$.

A fuzzy data graph may contain both fuzzy edges and crisp edges as a fuzzy edge with a degree of 0 or 1 can be considered as crisp. Along the same line, a crisp data graph is simply a special case of fuzzy data graph (where $\rho : V \times V \rightarrow \{0, 1\}$ is Boolean). We then only deal with fuzzy edges and data graphs in the following.

An important operation on fuzzy relations is composition. Assume ρ_1 and ρ_2 are two fuzzy relations on V . Thus, composition $\rho = \rho_1 \circ \rho_2$ is also a fuzzy relation on V s.t. $\rho(x, z) = \max_y \min(\rho_1(x, y), \rho_2(y, z))$. The composition operation can be shown to be associative: $(\rho_1 \circ \rho_2) \circ \rho_3 = \rho_1 \circ (\rho_2 \circ \rho_3)$. The associativity property allows us to use the notation $\rho^k = \rho \circ \rho \circ \dots \circ \rho$ for the composition of ρ with itself $k - 1$ times. In addition, following [Yager, 2013], we define ρ^0 to be s. t. $\rho^0(x, y) = 0, \forall(x, y)$.

Useful notions related to fuzzy graphs are those of *strength* and *length* of a path. These notions were previously used in Chapter 3 in the RDF context, Their definition, drawn from [Rosenfeld, 1975], is recalled hereafter.

Strength of a path. — A path p in G is a sequence $x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_n$ ($n \geq 0$) such that $\rho(x_{i-1}, x_i) > 0, 1 \leq i \leq n$ and where n is the number of links in the path. The *strength* of the path is defined as

$$ST(p) = \min_{i=1..n} \rho(x_{i-1}, x_i). \quad (6.1)$$

In other words, the strength of a path is defined to be the weight of the weakest edge of the path. Two nodes for which there exists a path p with $ST(p) > 0$ between them are called *connected*. We call p a cycle if $n \geq 2$ and $x_0 = x_n$. It is possible to show that $\rho^k(x, y)$ is the strength of the strongest path from x to y containing at most k links. Thus, the strength of the strongest path joining any two vertices x and y (using any number of links) may be denoted by $\rho^\infty(x, y)$.

Length and *distance*. — The *length* of a path $p = x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_n$ in the sense of ρ is defined as follows:

$$Length(p) = \sum_{i=1}^n \frac{1}{\rho(x_{i-1}, x_i)}. \quad (6.2)$$

Clearly $Length(p) \geq n$ (it is equal to n if ρ is Boolean, i.e., if G is a nonfuzzy graph). We can then define the *distance* between two nodes x and y in G as

$$Distance(x, y) = \min_{\text{all paths } p \text{ from } x \text{ to } y} Length(p). \quad (6.3)$$

It is the length of the shortest path from x to y .

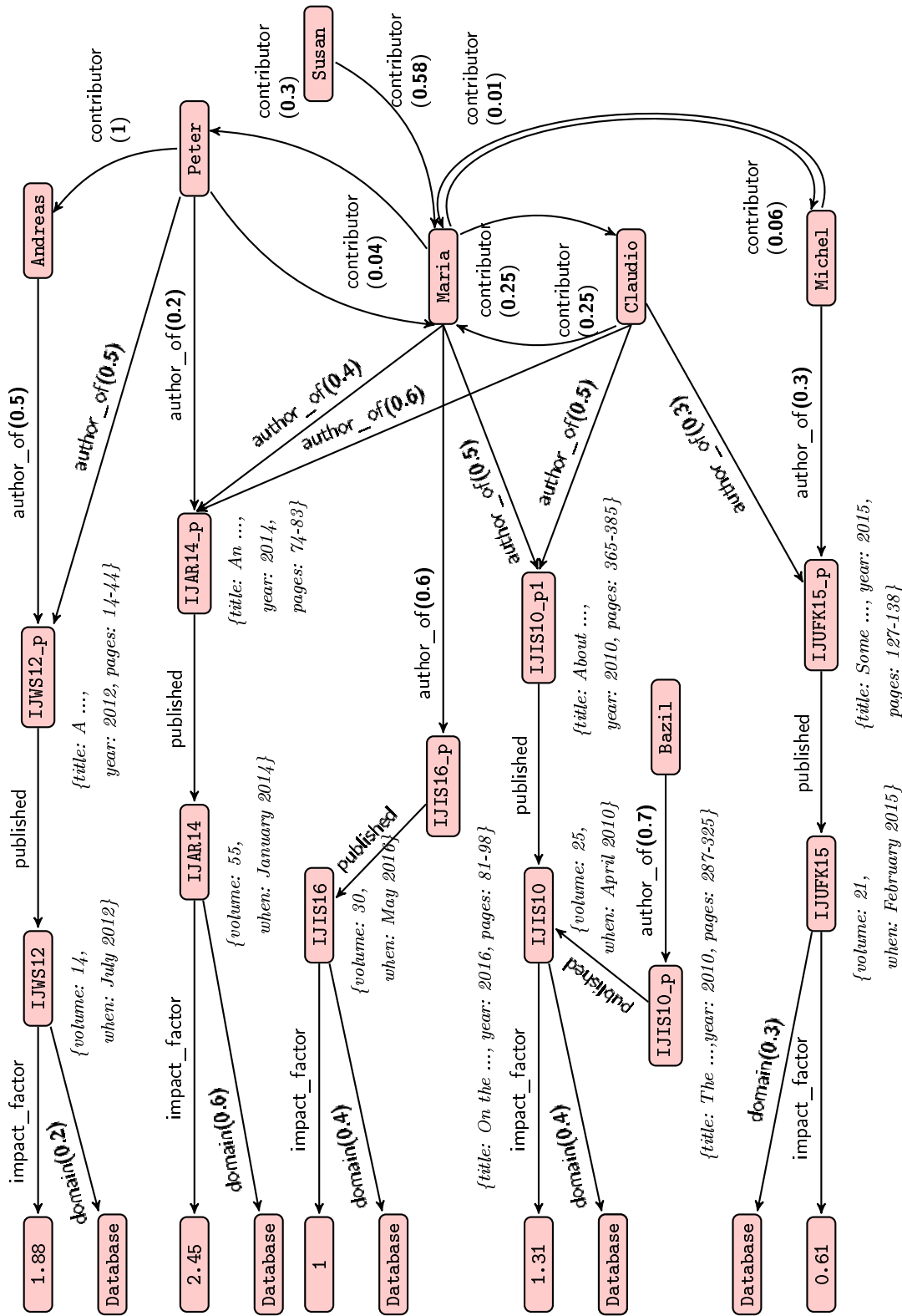


Figure 6.2: Fuzzy data graph DB

6.1.3 Fuzzy Graph Databases

We are interested in fuzzy graph databases where nodes and edges can carry data (e.g., key-value pairs in attributed graphs). So, we consider an extension of the notion of a *fuzzy graph*: the *fuzzy data graph* as defined in [Pivert et al., 2014a].

Definition 14 (Fuzzy data graph). *Let E be a set of labels. A fuzzy data graph G is a quadruple (V, R, κ, ζ) , where V is a finite set of nodes (each node n is identified by $n.id$), $R = \bigcup_{e \in E} \{\rho_e : V \times V \rightarrow [0, 1]\}$ is a set of labeled fuzzy edges between nodes of V , and κ (resp. ζ) is a function assigning a (possibly structured) value to nodes (resp. edges) of G .*

In the following, a *graph database* is meant to be a fuzzy data graph. The following example illustrates this notion.

Example 58 [Fuzzy data graph] Figure 6.2 is an example of a fuzzy data graph, inspired from DBLP² with some fuzzy edges (with a degree in brackets), and crisp ones (degree equal to 1).

In this example, the degree associated with A -contributor-> B is the proportion of journal papers co-written by A and B, over the total number of journal papers written by B. The degree associated with J - domain -> D is the extent to which the journal J belongs to the research domain D.

Nodes are assumed to be typed. If n is a node of V , then $Type(n)$ denotes its type. In Figure 6.2, the nodes IJWS12, IJAR14, IJIS16, IJIS10 and IJUFK15 are of type *journal*, the nodes IJWS12-p, IJAR14-p, IJIS16-p, IJIS10-p, IJIS10-p1 and IJUFK15-p of type *paper*, and the nodes Andreas, Peter, Maria, Claudio, Michel, Bazil and Susan are of type *author*, the nodes named Database are of type *domain* and the other nodes are of type *impact_factor*. For nodes of type *journal*, *paper*, *author* and *domain*, a property, called *name*, contains the identifier of the node and for nodes of type *impact_factor*, a property, called *value*, contains the value of the node. In Figure 6.2, the value of the property *name* or *value* for a node appears inside the node. \diamond

6.1.4 The FUDGE Query Language

FUDGE, based on the algebra described in [Pivert et al., 2015], is an extension of the Cypher language [Cypher, 2017]. The Cypher query language, inspired from ASCII-art for graph representation, is used for querying graph databases in a crisp way in the Neo4j graph DBMS [Neo4j, 2017]. These languages are based on graph pattern matching, meaning that a query Q over a fuzzy data graph \mathcal{DB} defines a graph pattern and answers to Q are its isomorphic subgraphs that can be found in \mathcal{DB} . More concretely, a pattern has the form of a

²<http://www.informatik.uni-trier.de/~ley/db/>

subgraph where variables can occur. An answer maps the variables to elements of \mathcal{DB} .

A fuzzy graph pattern expressed *à la Cypher* consists of a set of expressions $(n1:Type1)-[exp]->(n2:Type2)$ or $(n1:Type1)-[e:label]->(n2:Type2)$ where $n1$ and $n2$ are node variables, e is an edge variable, $label$ is a label of E , exp is a fuzzy regular expression, and $Type1$ and $Type2$ are node types. Such an expression denotes a path satisfying a fuzzy regular expression exp (that is *simple* in the second form e) going from a node of type $Type1$ to a node of type $Type2$. All its arguments are optional, so the simplest form of an expression is $()-[]->()$ denoting a path made of two nodes connected by any edge. Conditions on attributes are expressed on nodes and edges variables in a WHERE clause.

Example 59 [Graph pattern] We denote by \mathcal{P} the graph pattern:

```

1 MATCH
2   (au2)-[:contributor+]->(au1:author),
3   (au1)-[:author_of]->(ar1:paper), (ar1)-[:published]->(j1),
4   (au1)-[:author_of]->(ar2:paper), (ar2)-[:published]->(j2)
5 WHERE j1.name="IJWS12" AND j1.name <> j2.name

```

Listing 6.1: Pattern expressed *à la Cypher*

This pattern “models” information concerning authors ($au2$) who have, among their contributors, an author ($au1$) who published a paper ($ar1$) in IJWS12 and also published a paper ($ar2$) in another journal ($j2$). Figure 6.3 is a graphical representation of \mathcal{P} . \diamond

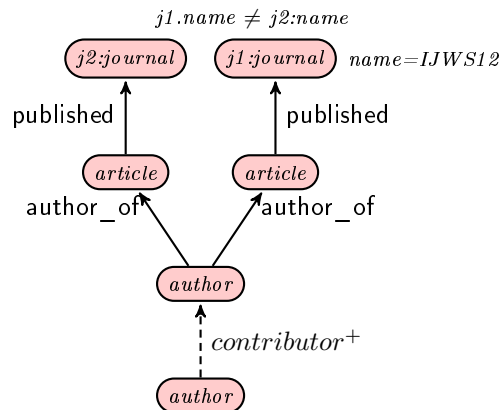


Figure 6.3: Pattern \mathcal{P}

Let us illustrate the way a selection query can be expressed in FUDGE, that embeds fuzzy preferences over the data and the structure specified in the graph pattern. Given a graph database \mathcal{DB} , a selection query expressed in FUDGE is composed of:

1. A list of **DEFINE** clauses for fuzzy term declarations. Again if a fuzzy term **fterm** has a trapezoidal function defined by the quadruple (A-a, A, B and B+b) — meaning that its support is [A-a, B+b] and its core [A, B] —, then the clause has the form **DEFINE fterm AS (A-a,A,B,B+b)**. If **fterm** is a decreasing function, then the clause has the form **DEFINEDESC fterm AS (δ,γ)** meaning that the support of the term is [0, γ] and its core [0, δ] (there is the corresponding **DEFINEASC** clause for increasing functions).
2. A **MATCH** clause, which has the form **MATCH pattern WHERE conditions** that expresses the fuzzy graph pattern.

Example 60 [FUDGE query] Listing 6.2 is an example of a FUDGE query.

```

1  DEFINEDESC short AS (3,5), DEFINEASC high AS (0.5,2) IN
2  MATCH
3  (au2)-[(contributor+)|Length IS short]->(au1:author),
4  (au1)-[:author_of]->(ar1:paper), (ar1)-[:published]->(j1),
5  (au1)-[:author_of]->(ar2:paper), (ar2)-[:published]->(j2),
6  (j2)-[:impact_factor]->(i)
7  WHERE j1.name="IJWS12" AND i.value IS high

```

Listing 6.2: A FUDGE query

This pattern aims to retrieve the authors (**au2**) who have, among their close contributors (connected by a short path — **Length IS short** — made of **contributor** edges), an author (**au1**) who published a paper (**ar1**) in IJWS12 and also published a paper (**ar2**) in a journal (**j2**) which has a *high* impact factor (**i.value IS high**). The fuzzy terms *short* and *high* are defined on line 1. Figure 6.4 is a graphical representation of this pattern where the dashed edge denotes a path and information in italics denotes a node type or an additional condition on node or edge attributes. \diamond

6.2 Related Work

In the last decades, *fuzzy quantified queries* have proved useful in a relational database context for expressing different types of imprecise information needs [Bosc et al., 1995]. Recently, in a graph database context, such statements started to attract increasing attention of many researchers [Yager, 2013, Castelltort and Laurent, 2014, Castelltort and Laurent, 2015] since they can exploit the *structure* of the graph, beside the attribute values attached to the nodes or edges.

In [Yager, 2013], R.R. Yager briefly mentions the possibility of using *fuzzy quantified queries* in a social network database context, such as the question of whether “*most* of the people residing in *western* countries have *strong* connections with each other” and suggests

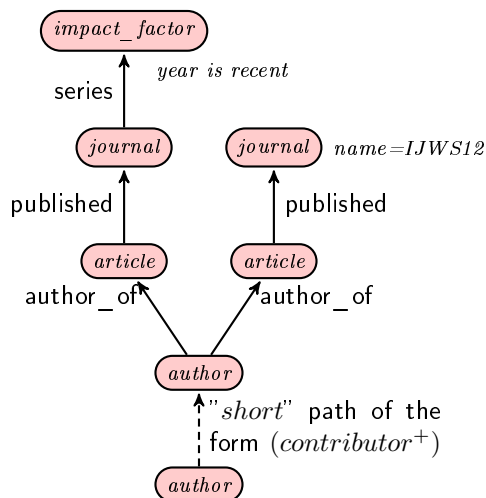


Figure 6.4: Fuzzy pattern

to interpret it using an OWA operator (cf. Subsection 4.1.2). However, the author does not propose any formal language for expressing such queries.

A first attempt to extend Cypher with *fuzzy quantified queries* — in the context of a *regular* (crisp) graph database — is described in [Castelltort and Laurent, 2014, Castelltort and Laurent, 2015]. In [Castelltort and Laurent, 2014], the authors take as an example a graph database representing hotels and their customers and consider the following fuzzy quantified query:

```
1 MATCH (c1:customer)-[:knows**almost3]->(c2:customer)
2 RETURN c1,c2
```

looking for pairs of customers linked through *almost 3* hops. The syntax ****** is used for indicating what the authors call a *fuzzy linker*. However, the interpretation of such queries is not formally given. The authors give a second example that involves the fuzzy concept *popular* applied to hotels. They assume that a hotel is popular if a large proportion of customers visited it. First, they consider a crisp interpretation of this concept (*large* being seen as equivalent to *at least n*) and recall how the corresponding query can be expressed in Cypher:

```
1 MATCH (c:customer)-[:visit]->(h:hotel)
2 WITH h, count(*) AS cpt
3 WHERE cpt > n - 1
4 RETURN h
```

Then, the authors switch to a fuzzy interpretation of the term *popular* and propose the expression:

```
1 MATCH (c:customer)-[:visit]->(h:hotel)
2 WITH h, count(*) AS cpt
3 WHERE popular(cpt) > 0
4 RETURN h
```

In [Castelltort and Laurent, 2015], the same authors propose an approach aimed to summarize a (crisp) graph database by means of *fuzzy quantified statements* of the form $\mathcal{Q} X \text{ are } A$, in the same spirit as what [Rasmussen and Yager, 1997] did for relational databases. Again, they consider that the degree of truth of such a statement is obtained by a sigma-count (according to Zadeh’s interpretation) and show how the corresponding queries can be expressed in Cypher. More precisely, given a graph database G and a summary $S = a-[r]->b$, \mathcal{Q} , the authors consider two degrees of truth of S in G defined as follows:

$$\text{truth}_1(S) = \mu_{\mathcal{Q}}\left(\frac{\text{count}(\text{distinct } S)}{\text{count}(\text{distinct } a)}\right) \quad (6.4)$$

$$\text{truth}_2(S) = \mu_{\mathcal{Q}}\left(\frac{\text{count}(\text{distinct } S)}{\text{count}(\text{distinct } a-[r]->(??))}\right) \quad (6.5)$$

They illustrate these notions using a database representing students who rent or own a house or an apartment. The degree of truth (in the sense of the second formula above) of the summary “ $S = \text{student}-[\text{rent}]->\text{apartment}, \text{most}$ ” — meaning “most of the students rent an apartment” (as opposed to a house) — is given by the membership degree to the fuzzy quantifier *most* of the ratio: (number of times a relationship of type *rents* appears between a student and an apartment) over (number of relations of type *rents* starting from a *student* node). The corresponding Cypher query is:

```

1 MATCH (s:student)-[rents]->(a:apartment)
2 WITH toFloat(count(*)) AS countS
3 MATCH (s1:student)-[rents]->(m)
4 WITH toFloat(count(*)) AS count2
5 RETURN MuMost(countS/count2)

```

A limitation of this approach is that only the quantifier is fuzzy (whereas in general, in a fuzzy quantified statement of the form “ $\mathcal{Q} B X \text{ are } A$ ”, the predicates A and B may be fuzzy too).

The work the most related to that presented here is [Pivert et al., 2017] described in Chapter 4, where we introduced the notion of *fuzzy quantified statements* in a (fuzzy) RDF database context. We showed how this statement could be expressed in the FURQL language (which is a flexible extension of the SPARQL query language) that we previously proposed in [Pivert et al., 2016c].

6.3 Fuzzy Quantified Statements in FUDGE

In this section, we show how a specific type of *fuzzy quantified statements* may be expressed in the FUDGE query language. We first propose a syntactic format for these queries, then we show how they can be efficiently evaluated.

6.3.1 Syntax of a Fuzzy Quantified Query

In the following, we consider *fuzzy quantified queries* involving fuzzy predicates (beside the quantifier) over *fuzzy graph databases*. The *fuzzy quantified statements* considered are of the same type as those used in Chapter 4 in the context of RDF databases. They are of the form “ $Q B X$ are A ”, where the quantifier Q is represented by a fuzzy set and denotes either an increasing/decreasing relative quantifier (e.g., *most*) or an increasing/decreasing absolute one (e.g., *at least three*), where B is the fuzzy condition “to be connected (according to a given pattern) to a node x ”, X is the set of nodes in the graph, and A is the fuzzy (possibly compound) condition.

An example of such a statement is: “*most* of the *recent* papers of which x is a *main* author, have been published in a *renowned* database journal”.

The general syntactic form of a *fuzzy quantified query* of the form “ $Q B X$ are A ” in the FUDGE language is given in Listing 6.3.

```

1 DEFINE... IN
2 MATCH B(res, x)
3 WITH res HAVING Q(x) ARE A(x)
4 RETURN res

```

Listing 6.3: Syntax of a fuzzy quantified query

This query contains a list of **DEFINE** clauses for the fuzzy quantifiers and the fuzzy terms declarations, a **MATCH** clause for fuzzy graph pattern selection, a **HAVING** clause for the fuzzy quantified statement definition, and a **RETURN** clause for specifying which elements should be returned in the resultset. $B(\text{res}, x)$ denotes the fuzzy graph pattern involving the nodes res and x and expressing the (possibly fuzzy) conditions in B . $B(\text{res}, x)$ takes the form of a fuzzy graph pattern expressed *à la Cypher* by P_B **WHERE** C_B (see Section 6.1.4). $A(x)$ denotes the fuzzy graph pattern involving the node x and expressing the (possibly fuzzy) conditions in A . $A(x)$ takes the form of a fuzzy graph pattern expressed *à la Cypher* by P_A **WHERE** C_A (see Section 6.1.4).

Example 61 [Fuzzy Quantified Query] The query, denoted by $Q_{\text{mostAuthors}}$, that consists in “retrieving every author (a) such that *most* of the recent papers (p) of which he/she is a *main* author, have been published in a *renowned* database journal (j)” may be expressed in FUDGE as follows:

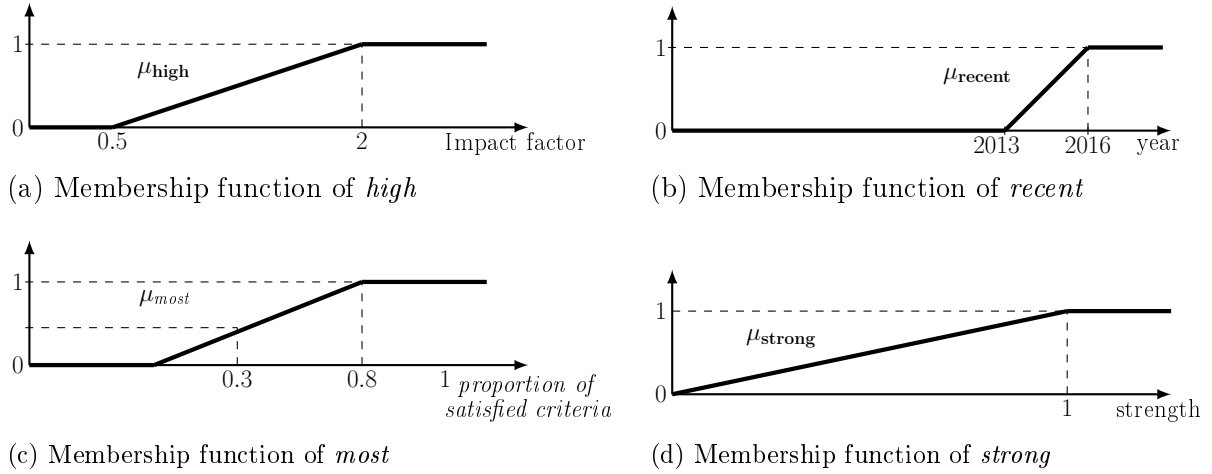


Figure 6.5: Membership functions of Example 61

```

1 DEFINEQRELATIVEASC most AS (0.3,0.8) DEFINEASC recent AS (2013,2016)
2 DEFINEASC strong AS (0,1) DEFINEASC high AS (0.5,2) IN
3 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
4 WHERE p.year IS recent
5 WITH a
6 HAVING most(p) ARE ( (p)-[:published]->(j:journal),
7 (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:dom)
8 WHERE i.value IS high AND d.name="database" )
9 RETURN a

```

Listing 6.4: Syntax of the fuzzy quantified query $Q_{mostAuthors}$

where the `DEFINEQRELATIVEASC` clause defines the fuzzy relative increasing quantifier *most* of Figure 6.5.(c), and the next `DEFINEASC` clauses define the increasing fuzzy terms *recent*, *strong* and *high* of Figures 6.5.(b), 6.5.(d), and 6.5.(a) respectively. In this query, *a* corresponds to *res*, *p* corresponds to *x*, lines 3 and 4 correspond to *B* and lines 6 to 8 correspond to *A*.

According to the general syntax introduced in Listing 6.3, the variable *a* instantiates *res* and the variable *p* instantiates *x*. \diamond

6.3.2 Evaluation of a Fuzzy Quantified Query

From a conceptual point of view, the interpretation of such a query can be based on one of the formulas (4.3), (4.8), and (4.17) already presented in Chapter 4. Its evaluation involves three stages :

1. the compiling of the fuzzy quantified query Q into a crisp query denoted by $Q_{derivedBoolean}$,

2. the interpretation of the crisp query $Q_{derivedBoolean}$,
3. the calculation of the answers to Q based on the answers to $Q_{derivedBoolean}$.

Compiling

The compiling stage translates the fuzzy quantified query Q into a crisp query denoted by $Q_{derivedBoolean}$. This compilation involves two translation steps.

First, Q is transcribed into a derived query $Q_{derived}$ whose aim is to retrieve the elements necessary to the interpretation of the fuzzy quantified statement from Q . The query $Q_{derived}$, whose general form³ is given in Listing 6.5, makes it possible to get the elements of the B part of the initial query, matching the variables `res` and `x`, for which we will then need to calculate the final satisfaction degree. It is obtained by removing the `WITH` and `HAVING` clauses from the initial query, and adding the `OPTIONAL MATCH` clause before the fuzzy graph pattern in condition A .

```

1 MATCH B(res, x)
2 OPTIONAL MATCH A(x)
3 RETURN res x  $I_A$   $I_B$ 

```

Listing 6.5: Derived query $Q_{derived}$

Such a query allows to retrieve the pairs $\{res, x\}$ that belong to the graph and all the information needed for the calculation of μ_B and μ_A , i.e., the combination of fuzzy degrees associated with relationships and node attribute values involved in `B(res,x)` and in `A(x)`, respectively denoted by I_B and I_A . The Listing 6.6 of Example 62 below presents the derived query associated with the query $Q_{mostAuthors}$.

The processing of $Q_{derived}$ is based on the *derivation principle* introduced by [Pivert and Bosc, 2012] in the context of relational databases: $Q_{derived}$ is in fact derived into another query denoted by $Q_{derivedBoolean}$. The derivation step translates the fuzzy query into a crisp one by transforming its fuzzy conditions into Boolean ones that select the support of the fuzzy statements. For instance, following this principle, the fuzzy condition `p.year IS recent` (where *recent* is defined as `DEFINEASC recent AS (2013,2016)`) becomes the crisp condition `p.year > 2013` in order to remove the answers that do not belong to the support of the answer. Listing 6.7 of Example 62 below is an illustration of the derivation of the query $Q_{derived}$. The derivation principle applied to the FUDGE language is detailed in [Pivert et al., 2015].

Crisp interpretation

The previous compiling stage translates the fuzzy quantified query Q embedding fuzzy quantifiers and fuzzy conditions into a crisp query $Q_{derivedBoolean}$, that can be processed by a

³Hereafter, the `DEFINE` clauses are omitted for the sake of simplicity.

classical graph DBMS (e.g., Neo4j).

For the sake of simplicity, we consider in the following that the result of $Q_{derived}$, denoted by $\llbracket Q_{derived} \rrbracket$, is made of the quadruples $(\mathbf{res}_i, \mathbf{x}_i, \mu_{Bi}, \mu_{Ai})$ matching the query.

Final result calculation

The last stage of the evaluation calculates the satisfaction degrees μ_B and μ_A according to I_B and I_A . If the optional part does not match a given answer, then $\mu_A = 0$. The answers of the initial fuzzy quantified query Q (involving the fuzzy quantifier \mathcal{Q}) are answers of the query $Q_{derived}$ derived from Q , and the final satisfaction degree associated with each element e can be calculated according to the three different interpretations mentioned earlier in Section 4.1. Hereafter, we illustrate this using [Zadeh, 1983] and [Yager, 1988]’s approaches (which are the most commonly used when it comes to interpreting *fuzzy quantified statements*).

Following Zadeh’s Sigma-count-based approach (cf. Subsection 4.1.2.1) we have:

$$\mu(e) = \mu_{\mathcal{Q}} \left(\frac{\sum_{\{(res_i, x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket Q_{derived} \rrbracket \mid res_i = e\}} \min(\mu_{Ai}, \mu_{Bi})}{\sum_{\{(res_i, x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket Q_{derived} \rrbracket \mid res_i = e\}} \mu_{Bi}} \right) \quad (6.6)$$

In the case of a fuzzy absolute quantified query, the final satisfaction degree associated with each element e is simply

$$\mu(e) = \mu_{\mathcal{Q}} \left(\sum_{\{(res_i, x_i, \mu_{Bi}, \mu_{Ai}) \in \llbracket Q_{derived} \rrbracket \mid res_i = e\}} \mu_{Ai} \right).$$

Example 62 [Evaluation of a Fuzzy Quantified Query] Let us consider the fuzzy quantified query $Q_{mostAuthors}$ of Listing 6.4. We evaluate this query according to the fuzzy data graph \mathcal{DB} of Figure 6.2. In order to interpret $Q_{mostAuthors}$, we first derive the following query $Q_{derived}$ from $Q_{mostAuthors}$, that retrieves “the authors (a) who highly contributed to at least one recent paper (p) (corresponds to B(a, p) in lines 1 and 2) possibly (OPTIONAL) published in a *renowned* database journal (corresponds to A(p) in lines 3 to 5)”.

```

1 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2 WHERE p.year IS recent
3 OPTIONAL MATCH (p)-[:published]->(j:journal),
4   (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:dom)
5 WHERE i.value IS high AND d.name="database"
6 RETURN a p  $\mu_A$   $\mu_B$ 

```

Listing 6.6: Query $Q_{derived}$ derived from $Q_{mostAuthors}$

Then, we evaluate the Cypher query $Q_{derivedBoolean}$ given in Listing 6.7, derived from the FUDGE nonquantified query $Q_{derived}$ of Listing 6.6.

```

1 MATCH fudge_p0 = (a:author)-[:author_of]->(p:paper)
2 WITH
3 REDUCE(min=1.0, edge IN relationships(fudge_p0) |
4     case when edge.fdegree<min then edge.fdegree else min end) AS fudge_p0,
5     a AS a, p AS p
6 WHERE fudge_p0>0.0 AND p.year>2013
7 OPTIONAL MATCH (p)-[:published]->(j:journal),
8     (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:domain)
9     WHERE i.value>0.5 AND d.name='database'
10 RETURN a p  $\mu_A$   $\mu_B$ 

```

Listing 6.7: Query $Q_{derivedBoolean}$ derived from $Q_{derived}$

Line 1 refers to the graph pattern structure related to condition B . Lines 3 to 5 perform the calculation of the *strength* of the degree connecting a to p . The `WHERE` clause in Line 6 implements conditions induced by the derivation of fuzzy preferences in condition B of the initial query (Line 2 of Listing 6.6). Line 7 to 8 refer to the graph pattern structure related to condition A . The `WHERE` clause in Line 9 implements crisp conditions of the initial query and conditions induced by the derivation of fuzzy preferences in condition A (Line 5 of Listing 6.6). The `WHERE` clause returns the isomorphic subgraphs that belong to the answer and complementary information (μ_A and μ_B) needed for the Score Calculation stage.

This query returns a list of authors (a) with their papers (p), satisfying the conditions of query $Q_{derived}$, along with their respective satisfaction degrees. Here,

$$\mu_B = \min(\mu_{strong}(\rho_{author}(a, p)), \mu_{recent}(p.year)) \text{ and}$$

$$\mu_A = \mu_{high}(i.value).$$

where μ_q denotes the membership degree of the predicate q and $\rho_e(x, y)$ denotes the weight of the edge (x, y) .

Let us consider $\llbracket Q_{derived}(a) \rrbracket$ the set of answers of the query $Q_{derived}$ for a given author a . The set $\llbracket Q_{derived}(a) \rrbracket$ provides a list of papers with their respective satisfaction degrees. This result set is of the form

$$\llbracket Q_{derived}(a) \rrbracket = \{((\mu_B, \mu_A)/p_1), \dots, ((\mu_B, \mu_A)/p_n)\}.$$

For the running example, $Q_{derived}$ returns the four answers $\{\text{Peter, Maria, Claudio, Michel}\}$. The authors **Andreas, Susan** and **Bazil** do not belong to the result of $Q_{mostAuthors}$ because **Susan** has not written a journal paper yet and **Andreas** and **Bazil** do not have a recent paper.

For the running example, we then have

$$\begin{aligned} \llbracket Q_{derived}(\text{Peter}) \rrbracket &= \{(0.2, 1)/\text{IJAR14_p}\}, \\ \llbracket Q_{derived}(\text{Maria}) \rrbracket &= \{(0.33, 1)/\text{IJAR14_p}\}, \{(0.6, 0.33)/\text{IJIS16_p}\}, \\ \llbracket Q_{derived}(\text{Claudio}) \rrbracket &= \{(0.33, 1)/\text{IJAR14_p}\}, \{(0.3, 0.07)/\text{IJUFK15_p}\}, \text{ and} \\ \llbracket Q_{derived}(\text{Michel}) \rrbracket &= \{(0.3, 0.07)/\text{IJUFK15_p}\}. \end{aligned}$$

Finally, assuming for the sake of simplicity that $\mu_{most}(x) = x$, the final result of the query $Q_{mostAuthors}$ evaluated on \mathcal{DB} using Formula 6.6 is

$$\begin{aligned} \llbracket Q_{mostAuthors} \rrbracket &= \{ \\ \mu(\text{Peter}) &= \mu_{most}\left(\frac{0.2}{0.2}\right) = 1, \mu(\text{Maria}) = \mu_{most}\left(\frac{0.66}{0.93}\right) = 0.71, \\ \mu(\text{Claudio}) &= \mu_{most}\left(\frac{0.4}{0.63}\right) = 0.63, \mu(\text{Michel}) = \mu_{most}\left(\frac{0.07}{0.3}\right) = 0.23\}. \diamond \end{aligned}$$

Using Yager's OWA-based approach (cf. subsection 4.1.2.2), for each element e returned by $Q_{derived}$ we calculate

$$\mu(e) = \sum_{\{(res_i, x_i, \mu_{B_i}, \mu_{A_i}) \in \llbracket Q_{derived} \rrbracket \mid res_i = e\}} w_i \times c_i. \quad (6.7)$$

Let us consider the fuzzy set $B = \{\mu_{B_1}/x_1, \dots, \mu_{B_n}/x_n\}$ such that $\mu_{B_1} \leq \dots \leq \mu_{B_n}$, the fuzzy set $A = \{\mu_{A_1}/x_1, \dots, \mu_{A_n}/x_n\}$ and $d = \sum_{i=1}^n \mu_{B_i}$.

The weights of the OWA operator are defined by

$$w_i = \mu_Q(S_{x_i}) - \mu_Q(S_{x_{i-1}})$$

with

$$S_{x_i} = \sum_{j=1}^i \frac{\mu_{B_j}}{d}.$$

The implication values are denoted by

$$c_{x_i} = \max(1 - \mu_{B_i}, \mu_{A_i})$$

and ordered decreasingly such that $c_1 \geq \dots \geq c_n$.

Example 63 In order to calculate $\mu(\text{Maria})$ from $Q_{derived}$, let us consider B (resp. A) the set of satisfaction degrees corresponding to condition B (resp. A) of element **Maria** as follows $B = \{0.33/\text{IJAR14}, 0.6/\text{IJIS16}\}$ and $A = \{1/\text{IJAR14},$

$0.33/\{IJIS16\}$. We have $d = 0.93$ and:

$$S_{IJAR14} = \frac{0.33}{0.93} = 0.35, \text{ and } S_{IJIS16} = \frac{0.33 + 0.6}{0.93} = 1.$$

Then, with $\mu_{most}(x) = x$, we get $\mu_Q(S_{IJAR14}) = 0.35$ and $\mu_Q(S_{IJIS16}) = 1$.

Therefore, the weights of the OWA operator are:

$$W_1 = \mu_Q(S_{IJAR14}) - \mu_Q(S_0) = 0.35 \text{ and } W_2 = \mu_Q(S_{IJIS16}) - \mu_Q(S_{IJAR14}) = 0.65.$$

The implication values are:

$$c_{IJAR14} = \max(1 - 0.33, 1) = 1, \text{ and } c_{IJIS16} = \max(1 - 0.6, 0.33) = 0.4.$$

Thus, $c_1 = 1$ and $c_2 = 0.4$. Finally, we get:

$$\mu(\text{Maria}) = 0.35 \times 1 + 0.65 \times 0.4 = 0.35 + 0.26 = 0.61.$$

Lastly, the final result of the query $Q_{mostAuthors}$ evaluated on \mathcal{DB} , given by Formula 6.7, is:

$$\llbracket Q_{mostAuthors} \rrbracket = \{ \mu(\text{Peter}) = 1, \mu(\text{Claudio}) = 0.84, \mu(\text{Michel}) = 0.7, \mu(\text{Maria}) = 0.61 \}. \diamond$$

6.4 About Query Processing

For the implementation of these quantified queries, we updated the SUGAR software described in [Pivert et al., 2014a, Pivert et al., 2016b], which is a software add-on layer that implements the FUDGE language over the *Neo4j* graph DBMS. This software efficiently evaluates FUDGE queries that contain fuzzy preferences, but its initial version did not support *fuzzy quantified statements*.

The SUGAR software basically consists of two modules, which implement the *Compiling* and *Final result calculation* stages defined in Section 6.3.2. These modules interact with a Neo4j engine, which implements the *Crisp implementation* stage defined in Section 6.3.2.

1. In a pre-processing step, the *Query compiler* module produces
 - the query-dependent functions that allow us to compute μ_B , μ_A and μ , for each returned answer, according to the chosen interpretation, and,
 - the (crisp) Cypher query $Q_{derivedBoolean}$, which is then sent to the Neo4j engine for retrieving the information needed to calculate μ_B and μ_A .

The compilation uses the derivation principle introduced in [Bosc and Pivert, 2000] in the context of relational databases.

2. In a post-processing step, the *Score calculator* module performs a grouping (according to the `WITH` clause of the initial query) of the elements, then calculates μ_B , μ_A and μ for each returned answer, and finally ranks the answers.

Figure 6.6 illustrates this architecture.

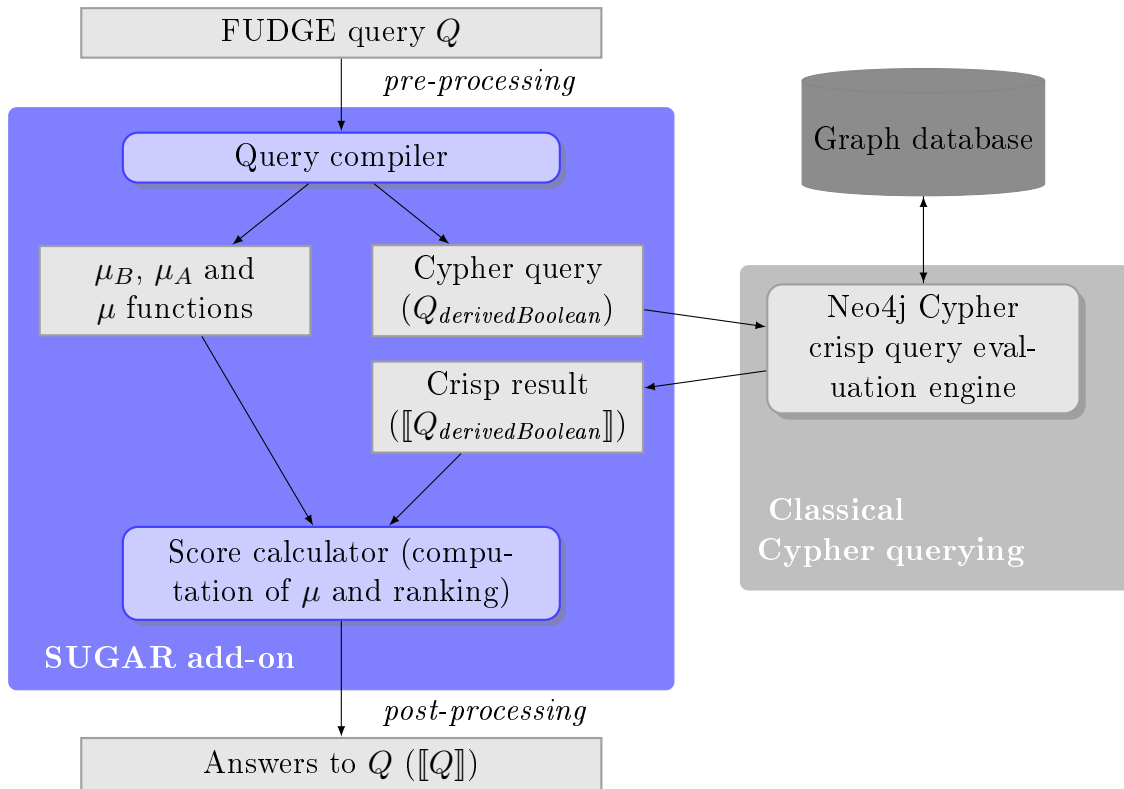


Figure 6.6: SUGAR software architecture

For quantified queries of the type introduced in the previous sections (i.e., using relative quantifiers), the principle is to first evaluate the fuzzy query $Q_{derivedBoolean}$ derived from the original query. For each element $x \in \llbracket Q_{derivedBoolean} \rrbracket$, we return the satisfaction degrees related to conditions A and B , denoted respectively by μ_A and μ_B . The final satisfaction degree μ can be calculated according to Formulas (4.3), (4.8) or (4.17) (presented in Chapter 4 Subsection 4.1.2) using the values of μ_B and μ_A . At the current time, [Zadeh, 1983]’s approach and [Yager, 1988]’s OWA-based approach have been implemented, and the choice of the interpretation to be used is made through the system configuration tool. Finally, a set of answers ranked in decreasing order of their satisfaction degree is returned.

As a proof-of-concept of the proposed approach, the FUDGE prototype is available at www-shaman.irisa.fr/fudge-prototype.

A screenshot of this prototype is shown in Figure 6.7 which contains the final result of the evaluation of the query $Q_{mostAuthors}$ of Example 61. The GUI is composed of two frames:

- a central frame for visualizing the graph and the results of a query, and
- an input field frame (placed under the central one), for entering and running a FUDGE query.

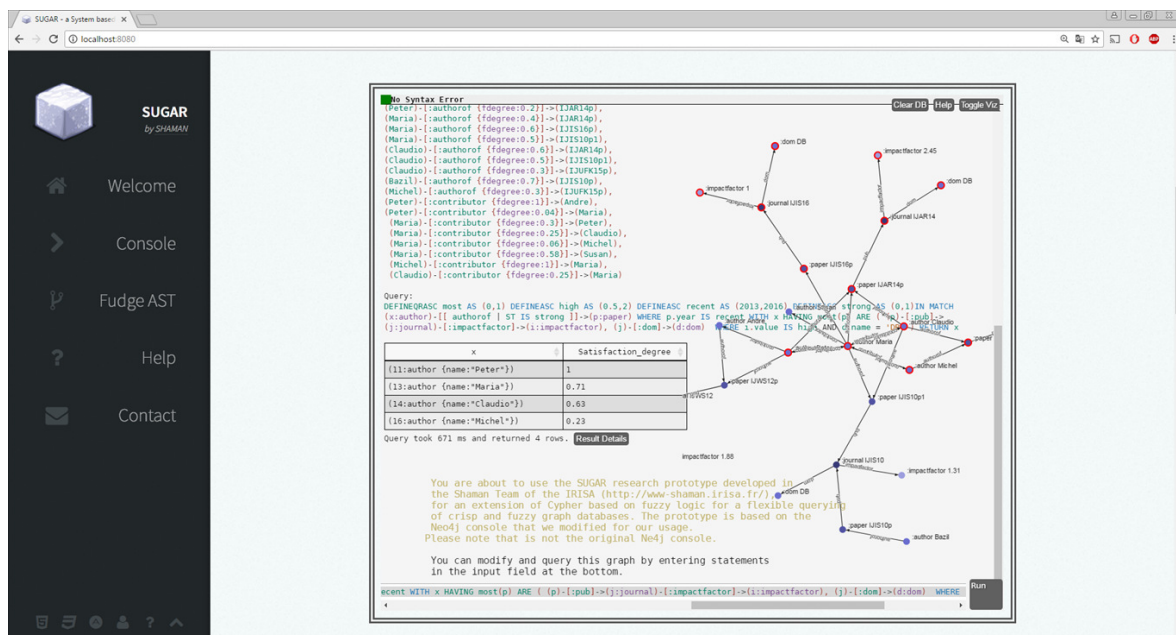


Figure 6.7: Screenshot of the FUDGE prototype

6.5 Experimental Results

In order to confirm the effectiveness and efficiency of the approach, we carried out some experiments on a computer running on Windows 7 (64 bits) with 8 Gb of RAM. The queries used in these experiments are based on the typology of [Angles, 2012] that considers three categories of queries:

- *Adjacency query*: tests whether two nodes are adjacent (or neighbors) when there exists an edge between them or whether two edges are adjacent when they have a common node.
- *Reachability query*: tests whether two given nodes are connected by a path. Two types of paths are considered: *fixed length paths*, which contain a fixed number of nodes and

edges; and *regular simple paths*, which allow some node and edge restrictions (e.g., regular expressions).

- *Pattern matching query*: graph pattern matching consists in finding all subgraphs of a data graph that are isomorphic to a graph pattern.

During our experiments, we considered four queries with various forms of condition A .

- The first query Q_1 (Listing 6.8), where A is an *adjacency pattern*, aims to find the authors such that *most* of the *recent* papers of which they are *main* authors, have been published in a journal.

```

1 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2 WHERE p.year IS recent
3 WITH a
4 HAVING most(p) ARE ( (p)-[:published]->(j:journal) )
5 RETURN a

```

Listing 6.8: Fuzzy quantified query with adjacency pattern Q_1

- The second query Q_2 (Listing 6.9), where A is an *reachability pattern* involving *fixed length path*, aims to find the authors such that *most* of the *recent* papers of which they are *main* authors, have been published in a ranked journal.

```

1 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2 WHERE p.year IS recent
3 WITH a
4 HAVING most(p) ARE ( (p)-[:published]->(j:journal),
5                       (j)-[:impact_factor]->(i:impact_factor) )
6 RETURN a

```

Listing 6.9: Fuzzy quantified query with reachability pattern Q_2

- The third query Q_3 (Listing 6.10), where A is an *reachability pattern* involving *regular simple path*, aims to find the authors a such that *most* of the *recent* papers of which they are *main* authors, have been co-authored by another author b who is a contributor (not necessarily direct) of *Claudio*.

```

1 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2 WHERE p.year IS recent
3 WITH a
4 HAVING most(p) ARE ( (b:author)-[:author_of]->(p),
5                       (b)-[:contributor*]->(c:author) )
6 WHERE c.name="Claudio"
7 RETURN a

```

Listing 6.10: Fuzzy quantified with reachability query Q_3

- The fourth query Q_4 (Listing 6.11), where A is a *pattern matching*, aims to find the authors (a) such that *most* of the *recent* papers (p) of which they are *main* authors, have been published in a *renowned* database journal (j)”.

```

1 DEFINEQRELATIVEASC most AS (0.3,0.8) DEFINEASC recent AS (2013,2016)
2 DEFINEASC strong AS (0,1) DEFINEASC high AS (0.5,2) IN
3 MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
4 WHERE p.year IS recent
5 WITH a
6 HAVING most(p) ARE ( (p)-[:published]->(j:journal),
7 (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:dom)
8 WHERE i.value IS high AND d.name="database" )
9 RETURN a

```

Listing 6.11: Fuzzy quantified query with *pattern matching*

Our experiments have been performed on a database inspired from DBLP containing crisp (e.g., *published*) and fuzzy edges (e.g., *contributor*). A java script have been developed to create random graph data of different sizes. In these experiments, four database sizes have been considered, see Table 6.1.

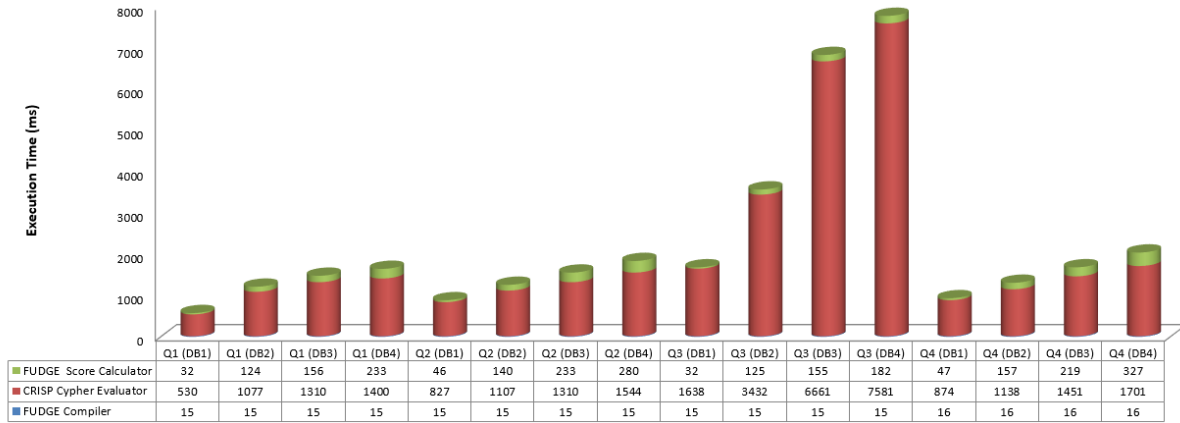
Table 6.1: Fuzzy graph datasets

Dataset	Size
DB_1	700 nodes & 1447 edges
DB_2	2100 nodes & 4545 edges
DB_3	3500 nodes & 7571 edges
DB_4	4900 nodes & 10494 edges

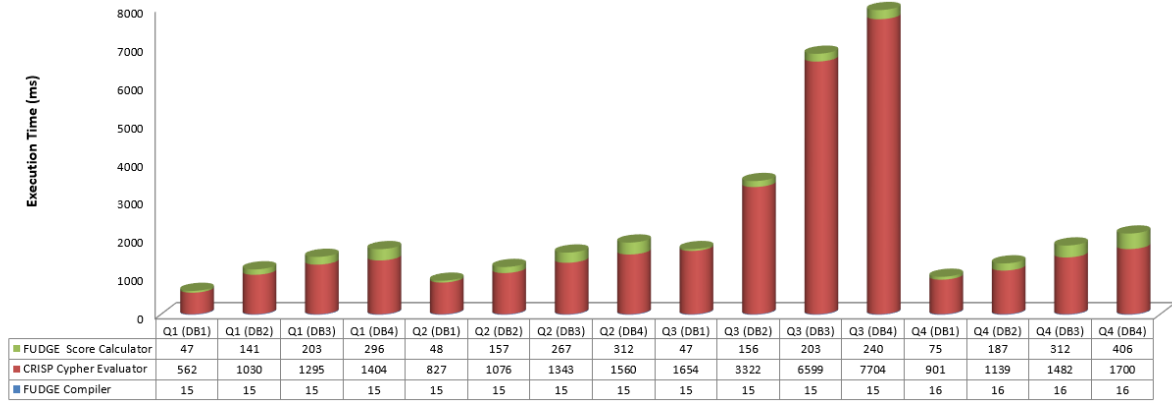
The results of the processing of these queries over the RDF datasets from Table 6.1 are depicted in Figure 6.8 where Figure 6.8.(a) (resp., Figure 6.8.(b)) presents the execution time in milliseconds using Zadeh’s interpretation (resp., Yager’s OWA-based interpretation).

The main result is that the processing time taken by the *compiling* and the score calculation stages, which are related to the introduction of flexibility into the query language, are very strongly dominated by the time taken by the crisp Cypher evaluator.

Moreover, the FUDGE *compiling* stage takes so little time compared to the other two stages that it cannot even be seen in Figure 6.8. This time remains almost constant, and is independent on the size of the dataset while slightly increasing in the presence of complex patterns or fuzzy conditions. As to the *score calculation* stage, it represents around 9% of the time needed for evaluating a fuzzy quantified FUDGE query. The time used for calculating the final satisfaction degree is of course dependent on the size of the result set and the nature of the patterns.



(a) Zadeh's interpretation over different size of DB



(b) Yager's interpretation over different size of DB

Figure 6.8: Experimental results of fuzzy quantified queries in FUDGE

Finally, these results show that introducing *fuzzy quantified statements* into a FUDGE query entails a reasonable increase of the overall processing time in the case of selection graph pattern queries. It represents, in the worst case, around 11% of the time needed for evaluating a fuzzy quantified FUDGE query.

Conclusion

In this chapter, we have dealt with *fuzzy quantified structural queries*, addressed to fuzzy graph databases. We have first defined the syntax and semantics of a fuzzy extension of the query language Cypher. This extension makes it possible to express and interpret such queries with different approaches from the literature. A query processing strategy based on the derivation of nonquantified fuzzy queries has also been proposed. Then, we updated the software SURF described in [Pivert et al., 2015, Pivert et al., 2016b] to be able to express such queries and performed some experiments using different sizes of fuzzy graphs in order to study its performances. The results of these experiments show that the cost of dealing with fuzzy quantification in a query is reasonable w.r.t. the cost of the overall evaluation.

Conclusion

THE last decade has witnessed an increasing interest in expressing preferences inside database queries for their ability to provide the user with the best answers, according to his/her information need. Even though most of the work in this area has been devoted to relational databases, several proposals have also been made in the Semantic Web area in order to query RDF databases in a flexible way. However, it appears that these approaches are mainly straightforward adaptations of proposals made in the relational database context: they make it possible to express preferences on the *values* of the nodes, but not on the **structure of the RDF graph**. Structural preferences are quite important in a graph database context and may concern the *strength* of a path, the *distance* between two nodes, etc. Moreover, these approaches consist of flexible extensions of the SPARQL query language that only deal with *crisp* RDF data. In the real world, though, semantic Web data often carry gradual notions such as friendship in social networks, aboutness in a bibliographic context, etc. Such notions can be modeled by fuzzy sets, which leads to attaching a degree in $[0, 1]$ to the edges of the graph.

Motivated by these concerns, we addressed in this thesis the issue of efficient querying of (fuzzy) RDF data with the aim of extending the SPARQL query language so as to be able to express i) fuzzy preferences on *data* (e.g., the release year of a movie is *recent*) and on the *structure* of the data graph (e.g., the path between two friends is required to be *short*). and ii) fuzzy *quantified* preferences (e.g., *most* of the albums that are recommended by an artist, are *highly* rated and have been created by a *young* friend of this artist).

To the best of our knowledge, this thesis is the first attempt in this direction in which we provide solutions for these different issues. After motivating our work, we presented in Chapter 1 basic notions related to our thesis, namely the RDF data model, the SPARQL query language and fuzzy set theory.

In Chapter 2 we provided an overview of the main proposals made in the literature that propose a flexible extension of SPARQL based on user preferences queries, relaxation

techniques and approximate matching. We discussed these approaches, classified them and pointed out their limits.

Chapter 3 was dedicated to the definition of a fuzzy extension of SPARQL that goes beyond the previous proposals in terms of expressiveness inasmuch as it makes it possible i) to deal with both crisp and fuzzy RDF databases, and ii) to express fuzzy structural conditions beside more classical fuzzy conditions on the values of the nodes present in the RDF graph. The language, called FURQL, is based on the notion of fuzzy graph pattern which extends Boolean graph patterns introduced by several authors in a crisp querying context.

Then, in Chapter 4 we proposed to integrate more complex conditions, namely, *fuzzy quantified statements* of the type “ $Q B X$ are A ” into the FURQL language (addressed to fuzzy RDF database) previously introduced in Chapter 3. We defined the syntax and semantics of an extension of the FURQL query language, that makes it possible to deal with such queries. A query processing strategy based on the derivation of nonquantified fuzzy queries has also been proposed.

These functionalities were successfully implemented using a prototype called SURF. Experimental results, described in Chapter 5, show the validity of the approach. In the case of fuzzy nonquantified queries, the results obtained indicate that introducing fuzziness into a SPARQL query comes with a very limited cost. And in the case of *fuzzy quantified queries*, the results show that the extra cost induced by the fuzzy nature of the queries remains also very limited, even in the case of rather complex *fuzzy quantified queries*.

Finally, the last chapter was devoted to integrating *fuzzy quantified queries* in an extension of the Neo4j Cypher language, called FUDGE, (described in [Pivert et al., 2014b, Pivert et al., 2016b]) in a more general (fuzzy) graph database context (of which fuzzy RDF databases are a special case). We first proposed a syntactic format for expressing these queries in the FUDGE language, and we described their interpretation using different approaches from the literature. Then, we carried out some experimentations in order to assess the performances of the evaluation method. The results of these experiments show that the cost of dealing with fuzzy quantification in a query is very reasonable w.r.t. the cost of the overall evaluation.

Future Work

In this thesis, we have proposed a fuzzy extension of the SPARQL query language that makes it possible to express *fuzzy structural conditions* and *fuzzy quantified statements* in an efficient way. This work serves as a baseline and leaves some open questions to solve and sets the basis for further extensions. Different perspectives on short-term and long-term work have been

identified and are outlined hereafter.

Extend the FURQL and FUDGE languages with more sophisticated preferences

In this thesis, we limited fuzzy structural properties to the *distance* and the *strength* where the distance between two nodes is the length of the shortest path between these two nodes and the strength of a path is defined to be the weight of the weakest edge of the path. It is also worth to consider *other structural properties*, like:

- the *centrality*, the *prestige* and the *influence* used in social networks analysis [Rusinowska et al., 2011]. For instance, the degree of *centrality* of a node measures the extent to which this node is connected with other nodes in a given social network. The question to answer is how *central* this node is in this network. The degree of *prestige* measures the extent to which a social actor in a network receives or serves as the object of relations sent by others in the network. Persons, who are chosen as friends by many others have a special position (prestige) in the group.
- the *clique* which is one of the basic concepts of classical graph theory. Ronald R. Yager in [Yager, 2014] redefined this notion in the case of a fuzzy graph.

Moreover, we introduced a specific type of *fuzzy quantified queries* of the form: Q nodes, among those that are connected to a node x according to a certain pattern, satisfy a fuzzy condition c . An example of such a statement is “*most* of the papers whose x is a *main* author, have been published in a *renowned* database journal”. It would be interesting to study other types of *fuzzy quantified queries*, in particular, those that aim to find the nodes x such that x is connected (by a path) to Q nodes reachable by a given pattern and satisfying a given condition c . An example of such a query is “find the authors x that had a paper published in *most* of the *renowned* database journals”. And also those that aim to find if there exists a path from x to a node satisfying c such that this path contains Q nodes (where Q is an absolute quantifier).

Make SURF and SUGAR more user-friendly

The softwares that we developed make it possible to express fuzzy user preferences where the query is explicitly written in the syntax of the formal query language (FURQL for RDF data and FUDGE for graph data) and the fuzzy terms are defined in the query by a predefined clause `DEFINE`. These softwares may be improved further in order to make them more *user-oriented*.

One can think first about proposing a way to help non-expert users define fuzzy terms easily. There is, therefore, a definite need about developing a *user interface* in order to help casual users define their preferences and the underlying fuzzy membership functions in a more

easy way, following the work of [Smits et al., 2013] in which the authors described *ReqFlex*, an intuitive user interface to the definition of preferences and the construction of fuzzy queries in a relational context.

Moreover, we may think also about integrating and analysing **user profiles** in order to focus more on the user's interest and preferences and take into account the user's context in order to personalise the retrieved information.

Add Quality-Related Metadata

Another important perspective concerns the management of **quality-related metadata** [Fürber and Hepp, 2010]. Since the RDF model that we used in this thesis makes it possible to model fuzzy notions, we can extend this model to represent data **quality dimensions** (e.g., accuracy, completeness, timeliness, consistency and so on). Indeed, these dimensions are of fuzzy nature and the values returned by the associated metrics may be viewed as satisfaction degrees.

Then, it would be also worth investigating the way our framework FURQL could be extended:

1. to express *fuzzy preferences queries* concerning some *quality dimensions*.
2. to associate quality information with the answers to a query. This would make it possible to rank-order the answers according to their quality level (on one or several dimensions) and to warn the user about the presence of “suspect” answers, for instance.

Develop Real-World RDF Databases (Benchmark)

Several RDF benchmarks have been developed (e.g., Lehigh University Benchmark (LUBM) [Guo et al., 2005], SPARQL Performance Benchmark (*SP²Bench*) [Schmidt et al., 2009], Berlin SPARQL Benchmark (BSBM) [Bizer and Schultz, 2009], DBpedia SPARQL Benchmark (DBPSB) [Morse et al., 2011], etc.) for data generator and benchmark queries in order to evaluate the performance of RDF stores. However, none of the existing benchmarks provides fuzzy RDF data or explicitly deals with fuzzy user preferences.

For that purpose, in this thesis, we initially performed the evaluation of our approaches using a fuzzy RDF database inspired by Musicbrainz⁴ with synthetic data generated by a script that allowed us to create datasets of different sizes.

A future work would be to consider further evaluation using some of the existing real-world data benchmarks or ideally create our own Fuzzy RDF Benchmark.

⁴<https://musicbrainz.org/>

Obviously, many research problems remain open and this thesis is only a first step which will help, we hope, to convince the databases community of the interest of using fuzzy logic for the flexible/intelligent management of data in information systems.

List of Figures

1.1	Sample RDF graph extracted from MusicBrainz	23
1.2	A graphical representation of the graph pattern from Listing 1.3	26
1.3	Possible subgraphs from Figure 1.1	26
1.4	Trapezoidal membership function	33
1.5	Graphical representation of the predicate <i>tall</i> (crisp set)	34
1.6	Graphical representation of the predicate <i>tall</i> (fuzzy set)	34
2.1	Membership function of <i>recent</i>	43
2.2	Membership function of the fuzzy number “at least Y”	43
2.3	CP-net of Example 34	53
2.4	An RDFS Ontology	54
3.1	Fuzzy RDF graph G_{MB} inspired by MusicBrainz	63
3.2	Cycle-free paths from G_{MB} connecting Beyonce to Euphoria	65
3.3	A possible representation of the fuzzy term <i>short</i>	68
3.4	Graphical representation of the pattern P_{rec_low}	69
3.5	Representation of the fuzzy term <i>low</i> applied to a <i>rating</i> value	70
3.6	Some paths from G_{MB}	72
3.7	Graphical representation of pattern P_{rec_low}	78
3.8	Subgraphs satisfying P_{rec_low}	78
4.1	Quantifiers “at most 2” (left) and “at least 3” (right)	81
4.2	The fuzzy quantifier “at least five”	82
4.3	Membership functions of Example 51	89
4.4	Membership function of the fuzzy term <i>strong</i>	90
4.5	Fuzzy RDF graph G_{MB} inspired by MusicBrainz	92
5.1	Reification of fuzzy triple of Example 55	99
5.2	Implementation of a specific FURQL query evaluation engine	100
5.3	SURF software architecture	101
5.4	Screenshot of SURF	102

5.5	Edge query of the form <i>edge-so</i>	103
5.6	Star query of the form <i>star-s2-o1</i>	105
5.7	Star query of the form <i>path-3</i>	106
5.8	Experimental results about the evaluation of FURQL queries	108
5.9	Experimental results of Fuzzy Quantified queries involving crisp conditions . . .	110
5.10	Experimental results of Fuzzy Quantified queries involving fuzzy conditions . .	111
6.1	An Attributed graph inspired from DBLP	117
6.2	Fuzzy data graph \mathcal{DB}	119
6.3	Pattern \mathcal{P}	121
6.4	Fuzzy pattern	123
6.5	Membership functions of Example 61	126
6.6	SUGAR software architecture	132
6.7	Screenshot of the FUDGE prototype	133
6.8	Experimental results of fuzzy quantified queries in FUDGE	136

List of Tables

1.1	Results of Listing 1.5	27
1.2	Results of Listing 1.6	27
1.3	Tall men	34
1.4	Properties of <i>t-norm</i> and <i>t-conorm</i> operators	38
2.1	The set of annotated RDF statements	49
2.2	RDFS Inferences Rules	54
2.3	Main features of the preference query approaches	59
4.1	Characteristics of monotonous fuzzy quantifiers	80
5.1	Fuzzy RDF datasets	103
5.2	Different types of FURQL queries	107
5.3	Set of fuzzy quantified queries with crisp conditions	110
5.4	Set of fuzzy quantified queries with fuzzy conditions	111
5.5	Experimental results summarization	113
6.1	Fuzzy graph datasets	135

Appendix A

Sample of Queries

The following listing is an example of a derived nonfuzzy query.

```
SELECT ?art1 ?alb ?Degree_recommends ?date ?art2 ?Degree_friend ?age ?rating WHERE {  
  ?X1 subject ?art1 . ?X1 predicate recommends . ?X1 object ?alb .  
  ?X1 degree ?Degree_recommends . ?alb <uri:date> ?date .  
  FILTER ( ?date > 2010.0 )  
  OPTIONAL {  
    ?X2 subject ?art1 . ?X2 predicate friend .  
    ?X2 object ?art2 . ?X2 degree ?Degree_friend .  
    ?art2 <uri:age> ?age . ?art2 <uri:creator> ?alb . ?alb <uri:rating> ?rating .  
    FILTER ( ?rating > 2.0 && ?age < 40.0 ) }  
  }
```

Listing A.1: Query $R_{flatBoolean}$ derived from R_{flat}

In the following, we give all the queries that were used in the experiments of Subsection 5.2.3.

- Q_{1crisp} : A fuzzy quantified query with simple pattern in B and simple pattern in A , involving crisp conditions (see Listing A.2),

```
DEFINEQRASC most AS (0,1)  
SELECT ?art1 WHERE {  
  ?art1 <uri:recommends> ?alb2 . ?alb <uri:date> ?date2 .  
  FILTER ( ?date2 > 2014 ) }  
GROUP BY ?art1  
HAVING most(?alb2) ARE (  
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .  
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .  
  FILTER ( ?rating2 > 5 && ?age2 < 30 ) )
```

Listing A.2: A fuzzy quantified query Q_{1crisp}

- Q_{2crisp} : A fuzzy quantified query with complex pattern in B and simple pattern in A , involving crisp conditions (see Listing A.3),

```

DEFINEQRASC most AS (0,1) DEFINEASC strong AS (0.3,0.6)
SELECT ?art1 WHERE {
  ?art1 <uri:recommends> ?alb2 . ?alb2 <uri:date> ?date2 .
  ?art1 <uri:rating> ?r1 . ?art1 <uri:memberOf> ?m1 .
  ?art1 <uri:gender> ?g1 . ?art1 <uri:age> ?age1 .
  ?art1 <uri:type> ?t11 . ?alb2 <uri:type> ?t22 .
  FILTER ( ?date2 > '2014' ) }
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .
  FILTER ( ?rating2 > 5 && ?age2 < 30 ) )

```

Listing A.3: A fuzzy quantified query Q_{2crisp}

- Q_{3crisp} : A fuzzy quantified query with simple pattern in B and complex pattern in A , involving crisp conditions (see Listing A.4),

```

DEFINEQRASC most AS (0,1) DEFINEASC strong AS (0.3,0.6)
SELECT ?art1 WHERE {
  ?art1 <uri:recommends> ?alb2 . ?alb2 <uri:date> ?date2 .
  FILTER ( ?date2 > 2014 )}
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .
  ?art2 <uri:rating> ?r2 . ?art2 <uri:memberOf> ?m2 .
  ?art2 <uri:gender> ?g2 . ?art2 <uri:type> ?t21 . ?
  alb2 <uri:type> ?t22 .
  FILTER ( ?rating2 > 5 && ?age2 < 30 ) )

```

Listing A.4: A fuzzy quantified query Q_{3crisp}

- Q_{4crisp} : A fuzzy quantified query with complex pattern in B and complex pattern in A , involving crisp conditions (see Listing A.5),

```

DEFINEQRASC most AS (0,1) DEFINEASC strong AS (0.3,0.6)
SELECT ?art1 WHERE {
  ?art1 <uri:recommends> ?alb2 . ?alb2 <uri:date> ?date2 .
  ?art1 <uri:rating> ?r1 . ?art1 <uri:memberOf> ?m1 .
  ?art1 <uri:gender> ?g1 . ?art1 <uri:age> ?age1 .
  ?art1 <uri:type> ?t11 .
  FILTER ( ?date2 > '2014' ) }
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .
  ?art2 <uri:rating> ?r2. ?art2 <uri:memberOf> ?m2 .
  ?art2 <uri:gender> ?g2 . ?art2 <uri:type> ?t21 .
  ?alb2 <uri:type> ?t22 .
  FILTER ( ?rating2 > 5 && ?age2 < 30 ) )

```

Listing A.5: A fuzzy quantified query Q_{4crisp}

- Q_{1fuzzy} : A fuzzy quantified query with simple pattern in B and simple pattern in A , involving fuzzy conditions (see Listing A.6),

```

DEFINEQRASC most AS (0,1) DEFINEASC recent AS (2014,2016)
DEFINDESC young AS (25,32) DEFINEASC high AS (3,6)
SELECT ?art1 WHERE {
  ?art1 <uri:recommends> ?alb2 . ?alb <uri:date> ?date2 .
  FILTER ( ?date2 IS recent ) }
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .
  FILTER ( ?rating2 IS high && ?age2 IS young ) )

```

Listing A.6: A fuzzy quantified query Q_{1fuzzy}

- Q_{2fuzzy} : A fuzzy quantified query with complex pattern in B and simple pattern in A , involving fuzzy conditions (see Listing A.7),

```

DEFINEQRASC most AS (0,1) DEFINEASC recent AS (2014,2016)
DEFINEDESC young AS (25,32) DEFINEASC high AS (3,6)
DEFINEASC high AS (2,5)
SELECT ?art1 WHERE {
  ?art1 (recommends | ST IS strong) ?alb2 .
  ?alb2 <uri:date> ?date2 . ?art1 <uri:rating> ?r1 .
  ?art1 <uri:memberOf> ?m1 . ?art1 <uri:gender> ?g1 .
  ?art1 <uri:age> ?age1 . ?art1 <uri:type> ?t11 .
  ?alb1 <uri:type> ?t12 .
  FILTER ( ?date2 IS recent ) }
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 <uri:friend> ?art2 . ?art2 <uri:age> ?age2 .
  ?art2 <uri:creator> ?alb2 . ?alb2 <uri:rating> ?rating2 .
  FILTER ( ?rating2 IS high && ?age2 IS young ) )

```

Listing A.7: A fuzzy quantified query Q_{2fuzzy}

- Q_{3fuzzy} : A fuzzy quantified query with simple pattern in B and complex pattern in A , involving fuzzy conditions (see Listing A.8),

```

DEFINEQRASC most AS (0,1) DEFINEASC recent AS (2014,2016)
DEFINEDESC young AS (25,32) DEFINEASC high AS (3,6)
DEFINEASC high AS (2,5)
SELECT ?art1 WHERE {
  ?art1 <uri:recommends> ?alb2 . ?alb2 <uri:date> ?date2 .
  FILTER ( ?date2 IS recent )}
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 ( <uri:friend> | ST IS strong ) ?art2 .
  ?art2 <uri:age> ?age2 . ?art2 <uri:creator> ?alb2 .
  ?alb2 <uri:rating> ?rating2 . ?art2 <uri:rating> ?r2 .
  ?art2 <uri:memberOf> ?m2 . ?art2 <uri:gender> ?g2 .
  ?art2 <uri:type> ?t21 . ?alb2 <uri:type> ?t22 .
  FILTER ( ?rating2 IS high && ?age2 IS young ) )

```

Listing A.8: A fuzzy quantified query Q_{3fuzzy}

- Q_{4fuzzy} : A fuzzy quantified query with complex pattern in B and complex pattern in A , involving fuzzy conditions (see Listing A.9).

```

DEFINEQRASC most AS (0,1) DEFINEASC recent AS (2014,2016)
DEFINEDASC young AS (25,32) DEFINEASC high AS (3,6)
DEFINEASC high AS (2,5)
SELECT ?art1 WHERE {
  ?art1 (recommends | ST is strong) ?alb2 .
  ?alb2 <uri:date> ?date2 . ?art1 <uri:rating> ?r1 .
  ?art1 <uri:memberOf> ?m1 . ?art1 <uri:gender> ?g1 .
  ?art1 <uri:age> ?age1. ?art1 <uri:type> ?t11.
  FILTER ( ?date2 is recent )}
GROUP BY ?art1
HAVING most(?alb2) ARE (
  ?art1 ( <uri:friend> | ST IS strong ) ?art2 .
  ?art2 <uri:age> ?age2 . ?art2 <uri:creator> ?alb2 .
  ?alb2 <uri:rating> ?rating2 . ?art2 <uri:rating> ?r2 .
  ?art2 <uri:memberOf> ?m2 . ?art2 <uri:gender> ?g2 .
  ?art2 <uri:type> ?t21 . ?alb2 <uri:type> ?t22 .
  FILTER ( ?rating2 is high && ?age2 is young ) )

```

Listing A.9: A fuzzy quantified query Q_{4fuzzy}

Bibliography

- [Akbarinia et al., 2007] Akbarinia, R., Pacitti, E., and Valduriez, P. (2007). Best position algorithms for top-k queries. In *Proceedings of the 33rd international conference on Very large data bases*, pages 495–506. VLDB Endowment.
- [Alkhateeb et al., 2009] Alkhateeb, F., Baget, J.-F., and Euzenat, J. (2009). Extending SPARQL with regular expression patterns (for querying RDF). *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):57–73.
- [allegrograph, 2017] allegrograph (consulted in 2017). AllegroGraph web site. franz.com/agraph/allegrograph.
- [Angles, 2012] Angles, R. (2012). A comparison of current graph database models. In *Proceedings of the 28th IEEE International Conference on Data Engineering Workshops (ICDEW)*, pages 171–177. IEEE.
- [Angles and Gutierrez, 2008] Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39.
- [Anyanwu et al., 2007] Anyanwu, K., Maduko, A., and Sheth, A. (2007). SPARQ2L: towards support for subgraph extraction queries in RDF databases. In *Proceedings of the 16th international conference on World Wide Web*, pages 797–806. ACM.
- [Arenas and Pérez, 2011] Arenas, M. and Pérez, J. (2011). Querying semantic web data with SPARQL. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316.
- [Berkeley, 2010] Berkeley, D. (2010). Oracle embedded database. Located at: <http://www.oracle.com/database/berkeley-db>.
- [Bizer and Schultz, 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24.
- [Borzsony et al., 2001] Borzsony, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430. IEEE.

- [Bosc et al., 1995] Bosc, P., Liétard, L., and Pivert, O. (1995). Quantified statements and database fuzzy querying. In Bosc, P. and Kacprzyk, J., editors, *Fuzziness in database management systems*, pages 275–308. Springer.
- [Bosc and Pivert, 1995] Bosc, P. and Pivert, O. (1995). SQLf: a relational database language for fuzzy querying. *IEEE transactions on Fuzzy Systems*, 3(1):1–17.
- [Bosc and Pivert, 2000] Bosc, P. and Pivert, O. (2000). SQLf query functionality on top of a regular relational database management system. In *Knowledge Management in Fuzzy Databases*, pages 171–190. Springer.
- [Bosc and Pivert, 2012] Bosc, P. and Pivert, O. (2012). On four noncommutative fuzzy connectives and their axiomatization. *Fuzzy Sets and Systems*, 202:42–60.
- [Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR)*, 21:135–191.
- [Bozzon et al., 2011] Bozzon, A., Della Valle, E., and Magliacane, S. (2011). Towards and efficient SPARQL top-k query execution in virtual RDF stores. In *Proceedings of the 5th International Workshop on Ranking in Databases (DBRANK'11)*, pages 1–6.
- [Bozzon et al., 2012] Bozzon, A., Della Valle, E., and Magliacane, S. (2012). Extending SPARQL algebra to support efficient evaluation of top-k SPARQL queries. In *Search Computing – Broadening Web Search*, pages 143–156. Springer.
- [Bruno et al., 2002] Bruno, N., Chaudhuri, S., and Gravano, L. (2002). Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems (TODS)*, 27(2):153–187.
- [Bry et al., 2010] Bry, F., Furche, T., Marnette, B., Ley, C., Linse, B., and Poppe, O. (2010). SPARQLog: SPARQL with rules and quantification. In *Semantic Web Information Management*, pages 341–370. Springer.
- [Buche et al., 2009] Buche, P., Dibie-Barthélemy, J., and Chebil, H. (2009). Flexible SPARQL querying of web data tables driven by an ontology. In *Flexible Query Answering Systems*, pages 345–357. Springer.
- [Buche et al., 2008] Buche, P., Dibie-Barthélemy, J., and Hignette, G. (2008). Flexible querying of fuzzy RDF annotations using fuzzy conceptual graphs. In *Conceptual Structures: Knowledge Visualization and Reasoning*, pages 133–146. Springer.
- [Buche et al., 2013] Buche, P., Dibie-Barthelemy, J., Ibanescu, L., and Soler, L. (2013). Fuzzy web data tables integration guided by an ontological and terminological resource. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):805–819.

- [Cali et al., 2014] Cali, A., Frosini, R., Poulouvasilis, A., and Wood, P. T. (2014). Flexible querying for SPARQL. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 473–490. Springer.
- [Carroll, 2002] Carroll, J. J. (2002). Matching RDF graphs. In *The Semantic Web ISWC 2002*, pages 5–15. Springer.
- [Castelltort and Laurent, 2014] Castelltort, A. and Laurent, A. (2014). Fuzzy queries over NoSQL graph databases: Perspectives for extending the Cypher language. In *Proceedings of the 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'14)*, pages 384–395.
- [Castelltort and Laurent, 2015] Castelltort, A. and Laurent, A. (2015). Extracting fuzzy summaries from NoSQL graph databases. In *Proceedings of the 11th International Conference on Flexible Query Answering Systems (FQAS'15)*, pages 189–200. Springer.
- [Cedeño and Candan, 2011] Cedeño, J. P. and Candan, K. S. (2011). R2DF framework for ranked path queries over weighted RDF graphs. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 40. ACM.
- [Chang et al., 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.
- [Chen et al., 2011] Chen, L., Gao, S., and Anyanwu, K. (2011). Efficiently evaluating skyline queries on RDF databases. In *The Semantic Web: Research and Applications*, pages 123–138. Springer.
- [Cheng et al., 2010] Cheng, J., Ma, Z., and Yan, L. (2010). f-SPARQL: a flexible extension of SPARQL. In *Database and Expert Systems Applications*, pages 487–494. Springer.
- [Chomicki, 2002] Chomicki, J. (2002). Querying with intrinsic preferences. In *Advances in Database Technology—EDBT 2002*, pages 34–51. Springer.
- [Choquet, 1954] Choquet, G. (1954). Theory of capacities. In *Annales de l'institut Fourier*, volume 5, pages 131–295.
- [Chu et al., 1996] Chu, W. W., Yang, H., Chiang, K., Minock, M., Chow, G., and Larson, C. (1996). Cobase: A scalable and extensible cooperative information system. In *Intelligent Integration of Information*, pages 135–171. Springer.
- [Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.
- [Cypher, 2017] Cypher (2017). The Neo4j Developer Manual v3.2.

- [Das et al., 2006] Das, G., Gunopulos, D., Koudas, N., and Tsirogiannis, D. (2006). Answering top-k queries using views. In *Proceedings of the 32nd international conference on Very large data bases*, pages 451–462. VLDB Endowment.
- [De Virgilio et al., 2013] De Virgilio, R., Maccioni, A., and Torlone, R. (2013). A similarity measure for approximate querying over RDF data. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 205–213. ACM.
- [De Virgilio et al., 2015] De Virgilio, R., Maccioni, A., and Torlone, R. (2015). Approximate querying of RDF graphs via path alignment. *Distributed and Parallel Databases*, 33(4):555–581.
- [DeCandia et al., 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon’s highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM.
- [Dividino et al., 2012] Dividino, R., Gröner, G., Scheglmann, S., and Thimm, M. (2012). Ranking RDF with provenance via preference aggregation. In *Knowledge Engineering and Knowledge Management*, pages 154–163. Springer.
- [Dividino et al., 2009] Dividino, R., Sizov, S., Staab, S., and Schueler, B. (2009). Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *Journal of Web Semantics*, 7(3):204–219.
- [Dolog et al., 2006] Dolog, P., Stuckenschmidt, H., and Wache, H. (2006). Robust query processing for personalized information access on the semantic web. In *Proceedings of the 7th International Conference on Flexible Query Answering Systems (FQAS’06)*, pages 343–355. Springer.
- [Dolog et al., 2009] Dolog, P., Stuckenschmidt, H., Wache, H., and Diederich, J. (2009). Relaxing RDF queries based on user and domain preferences. *Journal of Intelligent Information Systems*, 33(3):239–260.
- [Dubois and Prade, 1986] Dubois, D. and Prade, H. (1986). Weighted minimum and maximum operations in fuzzy set theory. *Information Sciences*, 39(2):205–210.
- [Dubois and Prade, 1997] Dubois, D. and Prade, H. (1997). Using fuzzy sets in flexible querying: Why and how? In *Flexible query answering systems*, pages 45–60. Springer.
- [Dubois, 1980] Dubois, D. J. (1980). *Fuzzy sets and systems: theory and applications*, volume 144. Academic press.

- [Elbassuoni et al., 2011] Elbassuoni, S., Ramanath, M., and Weikum, G. (2011). Query relaxation for entity-relationship search. In *Extended Semantic Web Conference*, pages 62–76. Springer.
- [Fagin et al., 2003] Fagin, R., Lotem, A., and Naor, M. (2003). Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656.
- [Fan et al., 2016] Fan, W., Wu, Y., and Xu, J. (2016). Adding counting quantifiers to graph patterns. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1215–1230. ACM.
- [Faye et al., 2012] Faye, D. C., Curé, O., and Blin, G. (2012). A survey of RDF storage approaches. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 15:11–35.
- [Fodor and Yager, 2000] Fodor, J. and Yager, R. (2000). Fuzzy-set theoretic operators and quantifiers. In Dubois, D. and Prade, H., editors, *The Handbooks of Fuzzy Sets Series, vol. 1: Fundamentals of Fuzzy Sets*, pages 125–193. Kluwer Academic Publishers.
- [Fokou et al., 2014] Fokou, G., Jean, S., and Hadjali, A. (2014). Endowing semantic query languages with advanced relaxation capabilities. In *Foundations of Intelligent Systems*, pages 512–517. Springer.
- [Fokou et al., 2015] Fokou, G., Jean, S., Hadjali, A., and Baron, M. (2015). Cooperative techniques for SPARQL query relaxation in RDF databases. In *Proceedings of the 12th European Semantic Web Conference on The Semantic Web*, pages 237–252. Springer.
- [Fokou et al., 2017] Fokou, G., Jean, S., Hadjali, A., and Baron, M. (2017). Handling failing RDF queries: from diagnosis to relaxation. *Knowledge and Information Systems*, 50(1):167–195.
- [Fortin, 1996] Fortin, S. (1996). The graph isomorphism problem. Technical report, 96-20, University of Alberta, Edmonton, Alberta, Canada.
- [Frosini et al., 2017] Frosini, R., Cali, A., Poulouvasilis, A., and Wood, P. T. (2017). Flexible query processing for SPARQL. *Semantic Web*, 8:533–563.
- [Fürber and Hepp, 2010] Fürber, C. and Hepp, M. (2010). Using semantic web resources for data quality management. *Knowledge Engineering and Management by the Masses*, pages 211–225.
- [Gaasterland et al., 1992] Gaasterland, T., Godfrey, P., and Minker, J. (1992). Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3-4):293–321.

- [Gearon et al., 2012] Gearon, P., Passant, A., and Polleres, A. (2012). SPARQL 1.1 Update. *Working draft WD-sparql11-update-20110512, W3C (May 2011)*.
- [Godfrey, 1997] Godfrey, P. (1997). Minimization in cooperative response to failing database queries. *International Journal of Cooperative Information Systems*, 6(02):95–149.
- [Grabisch et al., 1992] Grabisch, M., Murofushi, T., and Sugeno, M. (1992). Fuzzy measure of fuzzy events defined by fuzzy integrals. *Fuzzy Sets and Systems*, 50:293–313.
- [Gueroussova et al., 2013] Gueroussova, M., Polleres, A., and McIlraith, S. A. (2013). SPARQL with qualitative and quantitative preferences. In *Proceedings of the Intl. Workshop OrdRing, co-located with ISWC*, pages 2–8.
- [Guo et al., 2005] Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182.
- [Gutierrez et al., 2004] Gutierrez, C., Hurtado, C., and Mendelzon, A. O. (2004). Foundations of semantic web databases. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 95–106. ACM.
- [Gutierrez et al., 2007] Gutierrez, C., Hurtado, C. A., and Vaisman, A. (2007). Introducing time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218.
- [Haase et al., 2004] Haase, P., Broekstra, J., Eberhart, A., and Volz, R. (2004). A comparison of RDF query languages. In *The Semantic Web—ISWC 2004*, pages 502–517. Springer.
- [Hadjali et al., 2011] Hadjali, A., Kaci, S., and Prade, H. (2011). Database preference queries—a possibilistic logic approach with symbolic priorities. *Annals of Mathematics and Artificial Intelligence*, 63(3-4):357–383.
- [Harris and Seaborne, 2013] Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. W3C Recommendation <http://www.w3.org/TR/sparql11-query>.
- [Hartig, 2009] Hartig, O. (2009). Querying trust in RDF data with tSPARQL. In *Proceedings of the 6th European Semantic Web Conference (ESWC)*, pages 5–20. Springer.
- [Hogan et al., 2012] Hogan, A., Mellotte, M., Powell, G., and Stampouli, D. (2012). Towards fuzzy query-relaxation for RDF. In *Proceedings of the 9th Extended Semantic Web Conference (ESWC)*, pages 687–702. Springer.
- [Huang et al., 2008] Huang, H., Liu, C., and Zhou, X. (2008). Computing relaxed answers on RDF databases. In *In Proceedings of the 9th international conference on Web Information Systems Engineering (WISE’08)*, pages 163–175. Springer.

- [Hurtado et al., 2006] Hurtado, C. A., Poulouvasilis, A., and Wood, P. T. (2006). A relaxed approach to RDF querying. In *International Semantic Web Conference*, pages 314–328. Springer.
- [Hurtado et al., 2008] Hurtado, C. A., Poulouvasilis, A., and Wood, P. T. (2008). Query relaxation in RDF. In *Journal on data semantics*, pages 31–61. Springer.
- [Ilyas et al., 2008] Ilyas, I. F., Beskales, G., and Soliman, M. A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11.
- [infinitegraph, 2017] infinitegraph (consulted in 2017). InfiniteGraph web site. www.objectivity.com/infinitegraph.
- [Jannach, 2009] Jannach, D. (2009). Fast computation of query relaxations for knowledge-based recommenders. *Ai Communications*, 22(4):235–248.
- [Kacprzyk et al., 1989] Kacprzyk, J., Zadrozny, S., and Ziolkowski, A. (1989). FQUERY III +: a "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers. *Information Systems*, 14(6):443–453.
- [Kießling, 2002] Kießling, W. (2002). Foundations of preferences in database systems. In *Proceedings of the 28th international conference on Very Large Data Bases (VLDB)*, pages 311–322.
- [Kießling et al., 2011] Kießling, W., Endres, M., and Wenzel, F. (2011). The preference SQL system-an overview. *IEEE Data Eng. Bull.*, 34(2):11–18.
- [Kleinberg, 1999] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632.
- [Kochut and Janik, 2007] Kochut, K. J. and Janik, M. (2007). SPARQLer: Extended SPARQL for semantic association discovery. In *Proceedings of the 4th European Semantic Web Conference (ESWC '07)*, pages 145–159. Springer.
- [Lakshman and Malik, 2009] Lakshman, A. and Malik, P. (2009). Cassandra: structured storage system on a P2P network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 5–5. ACM.
- [Leavitt, 2010] Leavitt, N. (2010). Will nosql databases live up to their promise? *Computer*, 43(2).
- [Li et al., 2005] Li, C., Chang, K. C.-C., Ilyas, I. F., and Song, S. (2005). RankSQL: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 131–142. ACM.

- [Lv et al., 2008] Lv, Y., Ma, Z., and Yan, L. (2008). Fuzzy RDF: A data model to represent fuzzy metadata. In *Proceedings of the 6th International Conference on Fuzzy Systems (FUZZ-IEEE'08)*, pages 1439–1445.
- [Ma et al., 2016] Ma, R., Jia, X., Cheng, J., and Angryk, R. A. (2016). SPARQL queries on RDF with fuzzy constraints and preferences. *Journal of Intelligent & Fuzzy Systems*, 30(1):183–195.
- [Magliacane et al., 2012] Magliacane, S., Bozzon, A., and Della Valle, E. (2012). Efficient execution of top-k SPARQL queries. In *The Semantic Web–ISWC 2012*, pages 344–360. Springer.
- [Mazzieri and Dragoni, 2008] Mazzieri, M. and Dragoni, A. (2008). A fuzzy semantics for the resource description framework. In da Costa, P., d’Amato, C., Fanizzi, N., Laskey, K., Laskey, K., Lukasiewicz, T., Nickles, M., and Pool, M., editors, *Uncertainty Reasoning for the Semantic Web I*, volume 5327 of *Lecture Notes in Computer Science*, pages 244–261. Springer Berlin Heidelberg.
- [Mazzieri and Dragoni, 2005] Mazzieri, M. and Dragoni, A. F. (2005). A fuzzy semantics for semantic web languages. In *Proceedings of the 2005 International Conference on Uncertainty Reasoning for the Semantic Web–Volume 173*, pages 12–22.
- [Morsey et al., 2011] Morsey, M., Lehmann, J., Auer, S., and Ngonga Ngomo, A.-C. (2011). DBpedia SPARQL Benchmark–Performance Assessment with Real Queries on Real Data. *The Semantic Web–ISWC 2011*, pages 454–469.
- [Murofushi and Sugeno, 1989] Murofushi, T. and Sugeno, M. (1989). An interpretation of fuzzy measures and the Choquet integral as an integral with respect to a fuzzy measure. *Fuzzy sets and Systems*, 29(2):201–227.
- [Neo4j, 2017] Neo4j (consulted in 2017). Neo4j web site. www.neo4j.org.
- [Pérez et al., 2006] Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of SPARQL. In *International semantic web conference (ISWC)*, pages 30–43. Springer.
- [Pérez et al., 2008] Pérez, J., Arenas, M., and Gutierrez, C. (2008). nSPARQL: A navigational language for RDF. In *International Semantic Web Conference*, pages 66–81. Springer.
- [Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16:1–16:45.
- [Pérez et al., 2010] Pérez, J., Arenas, M., and Gutierrez, C. (2010). nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270.

- [Pivert and Bosc, 2012] Pivert, O. and Bosc, P. (2012). *Fuzzy Preference Queries to Relational Databases*. Imperial College Press, London, UK.
- [Pivert et al., 2016a] Pivert, O., Slama, O., Smits, G., and Thion, V. (2016a). A fuzzy extension of SPARQL for querying gradual RDF data. In *Proceedings of the 10th IEEE International Conference on Research Challenges in Information Science (RCIS'16)*, pages 1–2.
- [Pivert et al., 2016b] Pivert, O., Slama, O., Smits, G., and Thion, V. (2016b). SUGAR: A graph database fuzzy querying system. In *Proceedings of the 10th IEEE International Conference on Research Challenges in Information Science (RCIS'16)*, pages 1–2.
- [Pivert et al., 2016c] Pivert, O., Slama, O., and Thion, V. (2016c). An extension of SPARQL with fuzzy navigational capabilities for querying fuzzy RDF data. In *Proceedings of the 25th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'16)*, pages 2409–2416. IEEE.
- [Pivert et al., 2016d] Pivert, O., Slama, O., and Thion, V. (2016d). FURQL : une extension floue du langage SPARQL. In *Actes des 32èmes Journées Bases de Données Avancées (BDA'16)*.
- [Pivert et al., 2016e] Pivert, O., Slama, O., and Thion, V. (2016e). Fuzzy quantified structural queries to fuzzy graph databases. In *Proceedings of the 10th International Conference on Scalable Uncertainty Management (SUM'16)*, pages 260–273. Springer.
- [Pivert et al., 2016f] Pivert, O., Slama, O., and Thion, V. (2016f). Requêtes quantifiées floues structurelles sur des bases de données graphe. In *Actes des Rencontres Francophones sur la Logique Floue et ses Applications (LFA'16)*, pages 9–16.
- [Pivert et al., 2016g] Pivert, O., Slama, O., and Thion, V. (2016g). SPARQL extensions with preferences: a survey. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1015–1020. ACM.
- [Pivert et al., 2017] Pivert, O., Slama, O., and Thion, V. (2017). Fuzzy quantified queries to fuzzy RDF databases. In *Proceedings of the 26th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'17)*. IEEE.
- [Pivert and Smits, 2015] Pivert, O. and Smits, G. (2015). How to efficiently diagnose and repair fuzzy database queries that fail. In *Fifty Years of Fuzzy Logic and its Applications*, pages 499–517. Springer.
- [Pivert et al., 2015] Pivert, O., Smits, G., and Thion, V. (2015). Expression and efficient processing of fuzzy queries in a graph database context. In *Proceedings of the 24th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'15)*, pages 1–8.

- [Pivert et al., 2014a] Pivert, O., Thion, V., Jaudoin, H., and Smits, G. (2014a). On a fuzzy algebra for querying graph databases. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14)*, pages 748–755.
- [Pivert et al., 2014b] Pivert, O., Thion, V., Jaudoin, H., and Smits, G. (2014b). On a fuzzy algebra for querying graph databases. In *Proceedings of the 26th International Conference on Tools with Artificial Intelligence (ICTAI'14)*, pages 748–755. IEEE.
- [Poulovassilis and Wood, 2010] Poulovassilis, A. and Wood, P. T. (2010). Combining approximation and relaxation in semantic web path queries. In *Proceedings of the 9th international semantic web conference on The semantic web (ISWC'10)*, pages 631–646. Springer.
- [Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation. <http://www.w3.org/TR/rdf-sparql-query/>.
- [Pugliese et al., 2008] Pugliese, A., Udreă, O., and Subrahmanian, V. (2008). Scaling RDF with time. In *Proceedings of the 17th international conference on World Wide Web (WWW)*, pages 605–614. ACM.
- [Rasmussen and Yager, 1997] Rasmussen, D. and Yager, R. R. (1997). Summary SQL - A fuzzy tool for data mining. *Intelligent Data Analysis*, 1(1-4):49–58.
- [Read and Corneil, 1977] Read, R. C. and Corneil, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363.
- [Reddy and Kumar, 2010] Reddy, B. and Kumar, P. S. (2010). Efficient approximate SPARQL querying of web of linked data. In *Proceedings of the 6th International Conference on Uncertainty Reasoning for the Semantic Web-Volume 654*, pages 37–48.
- [Robinson et al., 2015] Robinson, I., Webber, J., and Eifrem, E. (2015). New opportunities for connected data.
- [Rosati et al., 2015] Rosati, J., Di Noia, T., Lukasiewicz, T., De Leone, R., and Maurino, A. (2015). Preference queries with ceteris paribus semantics for linked data. In *Proceedings of the Confederated International Conferences on On the Move to Meaningful Internet Systems (OTM'15)*, pages 423–442. Springer.
- [Rosenfeld, 2014] Rosenfeld, A. (2014). Fuzzy graphs. In *Proceedings of the US–Japan Seminar on Fuzzy Sets and Their Applications*, page 77. Academic Press.
- [Rosenfeld, 1975] Rosenfeld, A. (1975). Fuzzy graphs. In *Fuzzy Sets and their Applications to Cognitive and Decision Processes*, pages 77–97. Academic Press.

- [Rusinowska et al., 2011] Rusinowska, A., Berghammer, R., De Swart, H., and Grabisch, M. (2011). Social networks: prestige, centrality, and influence. In *Proceedings of the 12th international conference on Relational and algebraic methods in computer science*, pages 22–39. Springer-Verlag.
- [Schmidt et al., 2009] Schmidt, M., Hornung, T., Lausen, G., and Pinkel, C. (2009). SP2Bench: a SPARQL performance benchmark. In *Proceedings of the 25th International Conference on Data Engineering (ICDE'09)*, pages 222–233. IEEE.
- [Siberski et al., 2006] Siberski, W., Pan, J. Z., and Thaden, U. (2006). Querying the semantic web with preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC)*, pages 612–624. Springer.
- [Smits et al., 2013] Smits, G., Pivert, O., and Girault, T. (2013). ReqFlex: Fuzzy Queries for Everyone. *Proceedings of the VLDB Endowment (PVLDB)*, 6(12):1206–1209.
- [sparksee, 2017] sparksee (consulted in 2017). Sparksee web site. sparsity-technologies.com.
- [Straccia, 2009] Straccia, U. (2009). A minimal deductive system for general fuzzy RDF. In *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems (RR'09)*, pages 166–181.
- [Sugeno, 1974] Sugeno, M. (1974). *Theory of fuzzy integrals and its applications*. PhD thesis, Tokyo Institute of Technology.
- [Tahani, 1977] Tahani, V. (1977). A conceptual framework for fuzzy query processing a step toward very intelligent database systems. *Information Processing and Management*, 13(5):289–303.
- [Tappolet and Bernstein, 2009] Tappolet, J. and Bernstein, A. (2009). Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *Proceedings of the 6th European Semantic Web Conference (ESWC'09)*, pages 308–322. Springer.
- [Torlone and Ciaccia, 2002] Torlone, R. and Ciaccia, P. (2002). Finding the best when it's a matter of preference. In *Proceedings of the 10th Italian National Conference on Advanced Data Base Systems (SEBD'02)*, pages 347–360.
- [Udrea et al., 2006] Udrea, O., Recupero, D., and Subrahmanian, V. (2006). Annotated RDF. In Sure, Y. and Domingue, J., editors, *Proceedings of the 3rd European Semantic Web Conference (ESWC'06)*, volume 4011 of *Lecture Notes in Computer Science*, pages 487–501. Springer Berlin Heidelberg.
- [Udrea et al., 2010] Udrea, O., Recupero, D. R., and Subrahmanian, V. S. (2010). Annotated RDF. *ACM Transactions on Computational Logic (TOCL)*, 11(2):10.

- [Umbrich et al., 2015] Umbrich, J., Hogan, A., Polleres, A., and Decker, S. (2015). Link traversal querying for a diverse web of data. *Semantic Web journal (SWJ)*, 6(6):585–624.
- [W3C, 2014] W3C (2014). RDF overview and documentations. <http://www.w3.org/RDF/>.
- [Wang et al., 2015] Wang, D., Zou, L., and Zhao, D. (2015). Top-k queries on RDF graphs. *Information Sciences*, 316:201–217.
- [Wang et al., 2012] Wang, H., Ma, Z., and Cheng, J. (2012). fp-Sparql: an RDF fuzzy retrieval mechanism supporting user preference. In *Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 443–447.
- [Xie et al., 2013] Xie, M., Lakshmanan, L. V., and Wood, P. T. (2013). Efficient top-k query answering using cached views. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 489–500. ACM.
- [Yager, 2013] Yager, R. (2013). Social network database querying based on computing with words. In *Flexible Approaches in Data, Information and Knowledge Management*, Studies in Computational Intelligence. Springer.
- [Yager, 1984] Yager, R. R. (1984). General multiple-objective decision functions and linguistically quantified statements. *International Journal of Man-Machine Studies*, 21:389–400.
- [Yager, 1988] Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on systems, Man, and Cybernetics*, 18(1):183–190.
- [Yager, 2014] Yager, R. R. (2014). Social network database querying based on computing with words. In *Flexible approaches in data, information and knowledge management*, pages 241–257. Springer.
- [Zadeh, 1965] Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.
- [Zadeh, 1983] Zadeh, L. A. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics with applications*, 9(1):149–184.
- [Zahmatkesh et al., 2014] Zahmatkesh, S., Valle, E. D., Dell’Aglia, D., and Bozzon, A. (2014). Towards a top-k SPARQL query benchmark generator. In *Proceedings of the 3rd International Conference on Ordering and Reasoning-Volume 1303*, pages 35–46.
- [Zaveri et al., 2016] Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., and Auer, S. (2016). Quality assessment for linked data: A survey. *Semantic Web Journal*, 7(1):63–93.
- [Zheng et al., 2016] Zheng, W., Zou, L., Peng, W., Yan, X., Song, S., and Zhao, D. (2016). Semantic SPARQL similarity search over RDF knowledge graphs. *Proceedings of the VLDB Endowment*, 9(11):840–851.

-
- [Zhu et al., 2002] Zhu, H., Zhong, J., Li, J., and Yu, Y. (2002). An approach for semantic search by matching RDF graphs. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 450–454.
- [Zimmermann et al., 2012] Zimmermann, A., Lopes, N., Polleres, A., and Straccia, U. (2012). A general framework for representing, reasoning and querying with annotated semantic web data. *Journal of Web Semantics*, 11:72–95.
- [Zimmermann, 2011] Zimmermann, H.-J. (2011). *Fuzzy set theory—and its applications*. Springer Science & Business Media.