Revista Técnica de la Facultad de Ingeniería, Universidad del Zulia
ISSN: 0254-0770
revistatecnica@fing.luz.edu.ve
Universidad del Zulia
República Bolivariana de Venezuela

Marlene Goncalves Da Silva; Ana Isabel Aguilera Faraco
Skyline Queries in SPARQL: An Overview
Revista Técnica de la Facultad de Ingeniería,
Universidad del Zulia, vol. 45, núm. 2, 2022, pp. 133-144
Universidad del Zulia
Maracaibo, República Bolivariana de Venezuela

DOI: https://doi.org/10.22209/rt.v45n2a06

Skyline Queries in SPARQL: An Overview

Marlene Goncalves Da Silva[1], Ana Isabel Aguilera Faraco[*2]

[1]Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Venezuela, Apartado 89000, Caracas, Venezuela.
[2]Escuela de Ingeniería Informática, Facultad de Ingeniería, Universidad de Valparaíso, Valparaíso, C.P. 2340000, Chile

*Autor de correspondencia: ana.aguilera@uv.cl

https://doi.org/10.22209/rt.v45n2a06

Recepción: 02 de Febrero 2022 | Aceptación: 29 de abril de 2022 | Publicación: 01 de mayo de 2022

Abstract

The growth of RDF (Resource Description Framework) datasets and the expansion of their use in conjunction with the definition of SPARQL, a declarative query language, have made RDF data management an active area of research and development. In this regard, mechanisms have been proposed to help users find their desired answers in less time, including ranking methods and preference-based queries. Skyline queries constitute one of the most practical and predominant types of preference-based queries. The aim of this work was to provide a guide to specifying SPARQL skyline queries using syntax proposed in state-of-the-art works, and SPARQL versions 1.0 and 1.1. The results show the possibility of rewriting skyline queries in SPARQL to express preferences. We plan to develop a tool to translate SPARQL skyline queries applying the different grammars proposed, into SPARQL 1.0 and 1.1 with the aim of providing an automatic mechanism of translation.

Keywords: databases; data formats; data processing; orogramming languages; skyline query; SPARQL.

Consultas *Skyline* en SPARQL: Una Visión General

Resumen

El crecimiento de los conjuntos de datos RDF (*Resource Description Framework*) y la expansión de su uso junto con la definición de SPARQL, un lenguaje de consulta declarativo, han convertido la gestión de datos RDF en un área activa de investigación y desarrollo. En este sentido, se han propuesto mecanismos para ayudar a los usuarios a encontrar las respuestas deseadas en menos tiempo, incluidos métodos de clasificación y consultas basadas en preferencias. Las consultas *Skyline* constituyen uno de los tipos más prácticos y predominantes de consultas basadas en preferencias. El objetivo de este trabajo consistió en proporcionar una guía para especificar consultas de *Skyline* SPARQL, utilizando la sintaxis propuesta en trabajos de última generación y SPARQL en las versiones 1.0 y 1.1. Los resultados muestran la posibilidad de reescribir consultas de *Skyline* en SPARQL para expresar preferencias. Se plantea desarrollar una herramienta para traducir las consultas de horizonte SPARQL, aplicando las diferentes gramáticas propuestas, en SPARQL 1.0 y 1.1, con el objetivo de proporcionar un mecanismo automático de traducción.

Palabras clave: bases de datos; formatos de datos; lenguajes de programación; procesamiento de datos, skyline query; SPARQL.

Consultas *Skyline* no SPARQL: uma visão geral

Resumo

O crescimento de conjuntos de dados Resource Description Framework (RDF) e a expansão de seu uso juntamente com a definição de SPARQL, uma linguagem de consulta declarativa, tornaram o gerenciamento de dados RDF uma área ativa de pesquisa e desenvolvimento. Nesse sentido, têm sido propostos mecanismos para ajudar os usuários a encontrar as respostas desejadas em menos tempo, incluindo métodos de classificação e consultas baseadas em preferências. As consultas de horizonte são um dos tipos mais práticos e predominantes de consultas baseadas em preferências. O objetivo deste trabalho foi fornecer um guia para especificar consultas Skyline SPARQL, utilizando a sintaxe proposta em trabalhos de última geração e SPARQL nas versões 1.0 e 1.1. Os resultados mostram a possibilidade de reescrever consultas Skyline no SPARQL para expressar preferências. Propõe-se desenvolver uma ferramenta de tradução de consultas de horizonte SPARQL, aplicando as diferentes gramáticas propostas, em SPARQL 1.0 e 1.1, com o objetivo de fornecer um mecanismo de tradução automática.

Palavras-chave: Bases de dados; formatos de dados; linguagens de programação; processamento de dados, consulta de horizonte; SPARQL.

Introduction

Semantic data on the Web has increased over the past years. This data is mainly based on the Resource Description Framework (RDF) and the current state of technologies and techniques in cloud computing (Elzein *et al.*, 2018); RDF is a data model for representing information about World Wide Web resources. Being a mature widely tested and robust technology for modeling data, RDF provides a foundation for publishing and linking data (Ontotext, 2020). RDF data representation allows information to be identified, disambiguated and interconnected by software agents and different systems. The growth of RDF datasets and the expansion of their use in conjunction with the definition of a declarative query language called SPARQL, defined by W3C (World Wide Web Consortium), have made RDF data management an active area of research and development, and a number of data management systems have been developed for this purpose (Zou and Özsu, 2017). Actually, RDF datasets exceed billions of triples and continue to grow in terms of both number of repositories and their sizes (Elzein *et al.*, 2018). SPARQL, a recursive acronym for SPARQL Protocol and RDF Query Language, is a query language for RDF, also called a semantic query language, used to retrieve data and give precise results (Kostylev *et al.*, 2015). SPARQL was announced as a new standard by RDF Data Access Working Group in 2008 (Prud'hommeaux and Seaborne, 2008). Due to the growing amount of linked data, the importance of semantic search engines for retrieving information has increased. The traditional search model of finding links on the Web is unsatisfactory for the increasingly complex tasks that seek to leverage the diverse, increasingly structured and semantically annotated data sets found on the Web (Sessoms and Anyanwu, 2014). The semantic web search engines that have provided a query language, SPARQL, for processing and running queries on their indexed data, require some mechanisms for ranking SPARQL query results besides the ranking methods applied to keyword queries, in order to help users find their desired answers in less time (Feyznia *et al.*, 2014). Other mechanisms consider preference-based queries, which show encouraging results for personalizing and filtering the massive amount of information residing in today's databases and information systems (Abidi *et al.*, 2018). Among the types of preference-based queries that have been most extensively studied are skyline queries (Borzsonyi *et al.*, 2001), which constitute one of the most practical and predominant types of preference-based queries (Gulzar *et al.*, 2019). They return the most interesting objects according to the user's criteria based on the Pareto dominance operator (Abidi *et al.*, 2017). Skyline queries are typically used in multi-criteria decision-making applications to find answers that are of interest to a user (Keles and Hose, 2019). Other applications include, but are not limited to, decision support systems, recommendation systems, and databases. Even though skyline queries have been extensively studied on relational data in the database community, little attention has yet been paid to research on how the skyline principle can help identify sets of interesting entities in knowledge graphs and, in particular, in RDF queries (Keles and Hose, 2019). The aim of this work is to provide a guide to specifying SPARQL skyline queries both in the syntax proposed by different authors, and in SPARQL 1.0 and 1.1.

The structure of this paper is as follow: section II introduces the background knowledge necessary to understand the topics related to this work (skyline queries, RDF and SPARQL); section III presents related works; section IV describes our approach; and finally section V concludes the paper and gives insights for future work.

Background

In this section, we define the skyline operator, Resource Description Framework (RDF) and the SPARQL query language before explaining the syntax for specifying skyline queries.

Skyline

The skyline operator filters a set of interesting tuples from a relational database. A tuple is interesting if it is not dominated by any other tuple. A tuple dominates another tuple if it is as good or better in all attributes and better in at least one attribute. Börzsönyi *et al.* (2001) incorporated the SKYLINE OF clause in a SQL command as follows:

SELECT <attributes>
FROM <relations>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
SKYLINE OF d_1 [MIN|MAX|DIFF],..., d_n [MIN|MAX|DIFF];
where d_1,…, d_n denote skyline dimensions or attributes.

In addition, MIN, MAX, and DIFF indicate if the dimension value is minimized, maximized, or different respectively. Börzsönyi *et al.* (2001) formalized the dominance relationship and the skyline set in Definitions 1-2.

Definition 1 (dominance): Let SKYLINE OF d_1 MIN, ..., d_l MIN, d_{l+1} MAX, ..., d_m MAX, d_{m+1} DIFF, ..., d_n

DIFF a clause of a skyline query.

A tuple $t = (t_1, \ldots, t_l, t_{l+1}, \ldots, t_m, t_{m+1}, \ldots, t_n)$ dominates a tuple $u = (u_1, \ldots, u_l, u_{l+1}, \ldots, u_m, u_{m+1}, \ldots, u_n)$ if and only if:

- $t_i \leq u_i$ for all $i = 1, \ldots, l$
- $t_i \geq u_i$ for all $i = (l + 1), \ldots, m$
- $t_i = u_i$ for all $i = (m + 1), \ldots, n$

If $t_i = u_i$ for all $i = 1, \ldots, n$, then t and u are incomparable and both are skyline if no DISTINCT is specified.
Definition 2 (skyline): Let T be a set of tuples $t_1, \ldots, t_p$. The skyline S is the set of tuples from M, such that there is no tuple t_i that dominates any tuple in S.

Resource description framework

Resource Description Framework (RDF) is a standard model for data interchange on the Web (RDF, 2021). RDF is a graph data model that formally describes the semantics, or meaning, of information. It consists of a labeled, directed graph of relations between resources and literal values. It is composed by triples based on an Entity-Attribute-Value (EAV) model, in which the subject is the entity, the predicate is the attribute, and the object is the value. Each triple has a unique identifier known as the Internationalized Resource Identifier, or IRI. IRIs look like web page addresses. The parts of a triple, the subject, predicate, and object, represent links in a graph. Figure 1 shows an example of an RDF graph for Twitter data. For this case, the resource https://twitter.com/Commercial_Crew/status/1326274591564718080 is a tweet with the post “Such a privilege to work with people I like & respect so much. I feel blessed” created on 2020-11-11 by Elon Musk. Mr Ellon Musk is a user who joined the social network on June, 2020 and owner of the https://twitter.com/elonmusk account. This account has 39.8 million of followers and 96 followings.

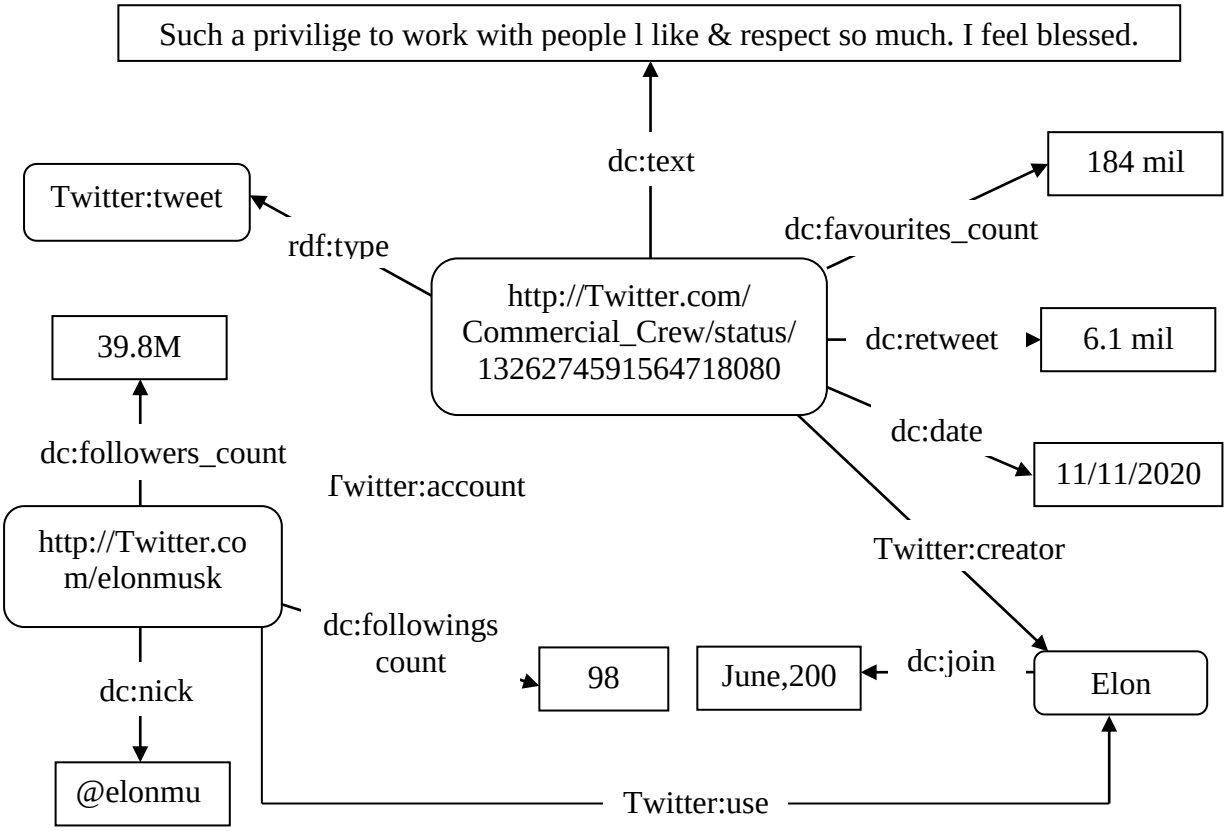

Figure 1. A Twitter resource description framework graph.

SPARQL

SPARQL (SPARQL Protocol and RDF Query Language), is a query language for RDF (Křemen, 2018). SPARQL is a semantic query language for databases able to retrieve and manipulate data stored in RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware.

The structure of a SPARQL query comprises (Feigenbaum, 2009):

- Prefix declarations, for abbreviating IRIs.
- Dataset definition, stating what RDF graph(s) are being queried.
- A result clause, identifying what information to return from the query.
- The query pattern, specifying what to query for in the underlying dataset.
- Query modifiers, slicing, ordering, and otherwise rearranging query results.

A general structure for a SPARQL query is as follow (Feigenbaum, 2009):

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
...
}
# query modifiers
ORDER BY …
```

Currently, SPARQL is the standard query language for RDF data. The W3C specification of the first version of SPARQL was SPARQL 1.0 (Prud'hommeaux and Seaborne, 2008), which was published in January 2008. This version defines the fundamental elements of the language, mainly the notion of graph patterns. In March 2013, SPARQL 1.1 (Harris and Seaborne, 2013) was released and its specification defines operators that allow more complex queries such as aggregation, sub-queries and path queries.

SPARQL 1.1 extends SPARQL 1.0 with several advanced features, among the most important we can mention: explicit operators to express the negation of graph patterns, operators to express path queries, aggregate operators, sub-queries and federated queries. Particularly, sub-queries allow expressing queries not supported by SPARQL 1.0. For example, a sub-query allows using the results obtained from the inner query, in particular when aggregate operators are included. The SPARQL 1.0 specification mentions (Prud'hommeaux and Seaborne, 2008), Section 11.4.1) that the negation of graph patterns can be simulated through the combination of an optional pattern and a filter condition of type !bound().

Related Work

Although Bentley *et al.* (1978) proposed the first skyline algorithm, referred to as the maximum vector problem, Börzsönyi *et al.* (2001) defined the skyline operator in the context of databases. In this work, the authors introduced a skyline algorithm based on the divide & conquer principle and the Block Nested Loop (BNL) algorithm where each one of the tuples is compared with non-dominated tuples in a window. Subsequently, SFS (Sort Filter Skyline) (Chomicki *et al.*, 2003), LESS (Linear Elimination Sort for Skyline) (Godfrey *et al.*, 2005), and SaLSa (Sort and Limit Skyline algorithm) (Chomicki *et al.*, 2003) were proposed to improve BNL by means of a monotone preference function that reduces the number of dominance checks. Also, skyline computation algorithms based on index structures were defined where properties of index structures to compute the skyline set were exploited in several works (Tan et al., 2001; Kossmann et al., 2002; Papadias et al., 2005; Lee et al., 2010; Selke and Balke, 2011; Bader, 2012; Endres and Glaser, 2019).

Since continuous growth of the Web, other distributed algorithms have been presented to efficiently compute the skyline over Web data sources (Balke and Guntzer, 2004; Balke *et al.*, 2004; Alvarado *et al.*, 2013). These algorithms are twofold, i.e., they build the skyline in two phases: first a superset is constructed, and then, dominated points are eliminated in a second phase. Each algorithm exploits a specific stopping condition to terminate the first phase, so as to avoid a full scan of Web data sources. Similarly, Chen *et al.* (2011) proposed an algorithm to compute the skyline on RDF documents that have been represented as VTPs (Vertical Table Partitioning).

More recently, there are some works related to extensions of SPARQL but with qualitative preferences (Siberski *et al.*, 2006; Troumpoukis *et al.*, 2017; Patel-Schneider *et al.*, 2018) which are more general than the skyline, being the skyline a particular case of them. Siberski *et al.* (2006) included preference-based querying capabilities to SPARQL incorporating the PREFERRING clause into the SPARQL syntax. SPREFQL (Troumpoukis et al., 2017) is another extension of SPARQL for qualitative preferences. Unlike Siberski *et al.* (2006), they support conditional preferences (if-then-else). At the implementation level, they presented a query rewriting technique that maps from a SPREFQL query to an equivalent SPARQL query by means of the NOT EXISTS operator. Unfortunately, their solution based on query rewriting does not work correctly due to the fact that it is based on the SPARQL EXISTS, which has many known problems (Patel-Schneider and Martin, 2016). Thus, Patel-Schneider *et al.* (2018) identified and fixed the problem in the previous proposals for acyclic and transitive preference relations. Finally, Keles and Hose (2019) presented a set of client-based algorithms to evaluate skyline queries over knowledge graphs using standard query interfaces for RDF, but they did not consider extending SPARQL. In this work, we focus on the proposals of Siberski *et al.* (2006), Troumpoukis *et al.* (2017) and Patel-Schneider *et al.* (2018) to specify SPARQL skyline queries in both their syntax, and SPARQL 1.0 and 1.1.

Approaches

Some works that extend SPARQL with qualitative preferences are Siberski *et al.* (2006), Gueroussova *et al.* (2013), Gueroussova *et al.* (2013b), Troumpoukis *et al.* (2017), Patel-Schneider *et al.* (2018). These works are based on the winnow operator (Chomicki, 2002), which is a more general operator than skyline. In this section, we will illustrate how these approaches can be used to express SPARQL skyline queries by using an example based on Twitter data. Suppose a database containing data from Twitter spambots (MIB, 2016) and a table named *users* storing the number of followers (followers_count) and the number of tweets each user has

liked in the account's lifetime (favourites_count), among other data. Also consider that someone wants to identify the most followed users who have the highest number of tweets he has liked. A subset of our knowledge base in Turtle syntax is the following:

```
@prefix : <http://www.example.org/>.
:LOUISHAIRY a :user; :followers_count 20004;
:favourites_count 15958 .
:CBS6Albany a :user; :followers_count 27856;
:favourites_count 291 .
:BryanBroaddus a :user; :followers_count 52287;
:favourites_count 19 .
:KingKhanBeats a :user; :followers_count 1824;
:favourites_count 36945 .
:lilyfan_ a :user; :followers_count 482;
:favourites_count 9909 .
:Adam_Loko116 a :user; :followers_count 943;
:favourites_count 9355 .
:bakkedahla :user; :followers_count 4558;
:favourites_count 1552 .
:myltuazona :user; :followers_count 498;
:favourites_count 13415 .
```

According to the interested person, both followers_count and favourites_count are equally important and relevant; hence, a predefined score function cannot be assigned to be used in a query. A user can be chosen if and only if there is no other user with a higher number of followers and a higher favourites_count. To select a user, we must identify the set of all the users that are non-dominated by any other user in terms of two criteria: maximizing followers_count and maximizing favourites_count; this is our skyline. Following these criteria, the computed skyline is composed by the users :LOUISHAIRY, :CBS6Albany, :BryanBroaddus, and :KingKhanBeats are the non-dominated ones, i.e., there is no other user with values better than them in these two attributes. Additionally, a user u_1 dominates a user u_2, if u_1 has better or equal values and at least one better in followers_count and favourites_count than s_2, e.g., the user :KingKhanBeats dominates the user :lilyfan_.

Next, we will describe how to specify the SPARQL query for the most followed users who have the highest number of tweets he has liked, following the syntax for each proposal (Siberski *et al.*, 2006; Troumpoukis *et al.*, 2017; Patel-Schneider *et al.*, 2018) and then we will detail how to express in an equivalent SPARQL query. Siberski *et al.* (2006) were the first to propose the addition of qualitative preference-based querying capabilities to SPARQL by means of the PREFERRING clause, which contains criteria separated by the AND construct. The CASCADE keyword can be used to prioritize a preference criterion over another one. The authors did not deal with query processing/optimization issues although they extended the ARQ query engine (The Apache Software Foundation, 2019) with BNL as a proof of concept. This implementation is not available.

The basic SPARQL query structure provides solution modifiers such as group by, order by, limit, offset, etc. Based on these solution modifiers, Siberski *et al.* (2006) extends them with a preferring clause. As our focus is on skyline queries, a preferring clause can be expressed in BNF according to Siberski *et al.* (2006) as follows in Algorithm 1.

Algorithm 1. Grammar for SPARQL skyline queries according to Siberski *et al.* (2006).
‹PreferringClause› ::= 'PREFERRING' ‹MultiDimPref›
‹MultiDimPref› ::= ‹AtomicPref› ('AND' ‹AtomicPref ›)*
‹AtomicPreference› ::= ‹HighestPref› | ‹LowestPref›
‹HighestPref› ::= 'HIGHEST' ‹Expression›
‹LowestPref› ::= 'LOWEST' ‹Expression›

Our example skyline SPARQL query can be expressed in terms of Siberski *et al.* (2006)'s syntax as shown Figure 2.

```
1  SELECT ?u ?followers_count ?favourites_count
2  WHERE { ?u a :user;:followers_count ?followers_count
3  ;:favourites_count ?favourites_count. }
4  PREFERRING HIGHEST(?followers_count) AND
5                  HIGHEST(?favourites_count)
```

Figure 2. SPARQL skyline queries. The skyline of the most followed users who have the highest number of tweets he has liked according to Siberski *et al.* (2006)'s syntax.

Based on Siberski *et al.* (2006)'s work, the authors Gueroussova *et al.* (2013) and Gueroussova *et al.* (2013b) proposed an extension of the SPARQL query language called PrefSPARQL, which includes the expression of conditional preferences and additional atomic preference constructs such as 'AROUND', 'MORE THAN', 'LESS THAN', and 'BETWEEN'. Since preferences semantically filter the solution set, they add preferences at the level of filters instead of solution modifiers. A preferring clause for skyline queries can be expressed in BNF as in Algorithm 2.

```
1  SELECT ?u ?followers_count ?favourites_count
2  WHERE {?u a :user;:followers_count ?followers_count;
3  :favourites_count ?favourites_count.
4  PREFERRING HIGHEST(?followers_count) AND
5                  HIGHEST(?favourites_count)}
```

Figure 3. SPARQL skyline queries. The skyline of the most followed users who have the highest number if tweets he has liked, following the PrefSPARQL grammar.

Algorithm 2. PrefSPARQL grammar.
‹Filter› ::= 'FILTER' ‹Constraint› |
'PREFERRING' '(' ‹MultiDimPref› ')'
‹MultiDimPref› ::= ‹AtomicPref› ('AND' ‹AtomicPref ›)*
‹AtomicPref› ::= ‹HighestPref› | ‹LowestPref ›
‹HighestPref› ::= 'HIGHEST' ‹Expression›
‹LowestPref› ::= 'LOWEST' ‹Expression›

Following the PrefSPARQL grammar, our example skyline SPARQL query is specified in Figure 3. They also show how queries can be rewritten in SPARQL 1.1 and SPARQL 1.0 in order to perform skyline queries using existing SPARQL query engines. P PREFERRING Pref can be expressed in SPARQL 1.1 as P FILTER NOT EXISTS {P' FILTER (tr(P, P', Pref))} where P is a SPARQL pattern, Pref represents preference criteria, P' is the same graph pattern than P but with all variables renamed as fresh variables, and tr is a translation function that translates the dominance check condition according to Definition 1. Similar to nested SQL query proposed by Börzsönyi *et al.* (2001), the condition within FILTER identifies the dominated ones and FILTER NOT EXIST discards them from the answer. Figure 4 illustrate our example skyline SPARQL query translated to SPARQL 1.1. In this example, P is "?u a :user ;:followers_count ?followers_count;:favourites_count ?favourites_count" (lines 2-4); P' is ?u_ a :user;:followers_count ?followers_count_;:favourites_count ?favourites_count_" (lines 6-8); and tr(P, P', Pref) is "?followers_count_ >= ?followers_count && ?favourites_count_ >= ?favourites_count && (?followers_count_ > ?followers_count ||?favourites_count_ > ?favourites_count)" (lines 9-12).

```
1 SELECT ?u ?followers_count ?favourites_count
2  WHERE {?u a :user;:followers_count
3          ?followers_count;:favourites_count
4          ?favourites_count .
5          FILTER NOT EXISTS {
6          ?u_ a :user;:followers_count
7          ?followers_count_;:favourites_count
8          ?favourites_count_ .
9          FILTER (?followers_count_ >= ?followers_count
10         && ?favourites_count_ >= ?favourites_count
11         && (?followers_count_ > ?followers_count ||
12         ?favourites_count_ > ?favourites_count)) } }
```

Figure 4. SPARQL 1.1. skyline queries. The skyline of the most followed users who have the highest number of tweets he has liked.

For P', the character "_" was added to each variable name of P. Lines 9-12 specify the dominance check. Lines 5-12 filters a set of dominated instances. An instance dominates another instance if it is as good or better in all attributes and better in at least one attribute (lines 9-12). In addition, to translate skyline queries in SPARQL 1.0, we can replace NOT EXISTS by a combination of OPTIONAL and FILTER(!bound). P PREFERRING Pref can be expressed in SPARQL 1.0 as: P OPTIONAL {P' FILTER (tr(P, P', Pref)) [ ] ?check [ ]} FILTER (!bound(?check)) where {[ ] ?check [ ]} is an auxiliary triple pattern that represents any predicate in P' and ?check is a fresh variable that is used to bind and thus, to verify for instance, the non-existence of instances better than it. As with SPARQL 1.1, P and P' represent SPARQL patterns, Pref the preference criteria, and tr is the translation function. Figure 5 illustrates our example skyline SPARQL query translated to SPARQL 1.0. FILTER within the OPTIONAL clause allows performing pairwise dominance checks for each pair of instances (lines 11-15) and the FILTER in line 16 verifies the instance is not dominated. If ?u_ is bound, this means that it is dominated because lines 11-15 found a better instance than ?u). Similar to nested SQL queries proposed by Börzsönyi *et al.* (2001), the condition within FILTER identifies the dominated ones and FILTER NOT EXIST discards them from the answer.

In this example, P is "?u a :user ;:followers_count ?followers_count;:favourites_count ?favourites_count" (lines 2-4); P' is ?u_ a :user;:followers_count ?followers_count_;:favourites_count ?favourites_count_" (lines 8-10); and tr(P, P', Pref) is "?followers_count_ >= ?followers_count && ?favourites_count_ >= ?favourites_count && (?followers_count_ > ?followers_count ||?favourites_count_ > ?favourites_count)" (lines 11-15).

```
1  SELECT ?u ?followers_count ?favourites_count
2  WHERE {?u a :user;:followers_count
3          ?followers_count;:favourites_count
4          ?favourites_count .
5        OPTIONAL { ?u a :user;:followers_count
6        ?followers_count;:favourites_count
7        ?favourites_count .
8          ?u_ a :user;:followers_count
9          ?followers_count_;:favourites_count
10         ?favourites_count_ .
11         FILTER (?followers_count_ >=
12         ?followers_count
13         && ?favourites_count_ >= ?favourites_count
14         && (?followers_count_ > ?followers_count_ ||
15         ?favourites_count > ?favourites_count)) }
16       FILTER (!BOUND(?u_))}
```

Figure 5. SPARQL 1.0 skyline queries. The skyline of the most followed users who have the highest number of tweets he has liked.

Subsequently, Troumpoukis *et al.* (2017) proposed SPREFQL as another extension of SPARQL for qualitative preferences. Their work comes nearer to Chomicki (2002)'s framework than Siberski *et al.* (2006) because it allows the expression of extrinsic preferences whose formulas may refer both to built-in predicates (e.g., equality, inequality, and arithmetic comparison operations) on the basis of tuples and to other constructors such as database relations. Although any query in Siberski *et al.* (2006) and Gueroussova *et al.* (2013) can be expressed in SPREFQL, reverse translation is not always possible. The authors introduced in Troumpoukis *et al.* (2017) a couple of cases where a query expressed in SPREFQL cannot be specified in Siberski *et al.* (2006) and Gueroussova *et al.* (2013). At the implementation level, they presented a query rewriter that maps from a SPREFQL query to an equivalent SPARQL query by means of the NOT EXISTS operator. Also, they experimentally study the performance of NL (Nested Loops), BNL and query rewriting; NL is a naive algorithm that compares each input tuple against all input tuples and whose computational complexity is quadratic. NL has the worst performance while BNL outperforms query rewriting in 6 out of 7 queries. They implemented an open-source prototype of SPREFQL (Bitbucket, 2021) which is available.

The PREFER clause is after the group-by/having clauses and before the limit/offset clauses. A PREFER clause for skyline queries can be expressed in EBNF as in Algorithm 3. All non-terminals that are not defined in this table are defined by standard SPARQL syntax.

Algorithm 3. Prefer grammar.
‹SolutionModifier› ::= [‹GroupClause›] [‹HavingClause›] [‹PreferClause›] [‹OrderClause›] [‹LimitOfsetClauses›]
‹PreferClause› ::= 'PREFER' ‹VarList› 'TO' ‹VarList› 'IF'
‹ParetoPref›
‹VarList› ::= ‹Var› | '(' ‹Var› + ')'
‹ParetoPref› ::= ‹SimplePref› ['AND' ‹ParetoPref›]
‹SimplePref› ::= ‹Constraint›

Expressing a skyline query in SPREFQL is quite similar to specifying it with the condition of Gueroussova *et al.* (2013) and Gueroussova *et al.* (2013b) proposal to rewrite a preference-based query in SPARQL. The condition for pair-wise dominance checks within the FILTER NOT EXISTS or OPTIONAL FILTER(!BOUND) is the same as that expressed in the condition of the IF.

Variable names are assigned to two binding sets that can be distinguished from each other through the PREFER clause. The first binding set refers to the preferred ones while the second is the dominated ones. Then, the "IF" clause expresses the conditions that make the first binding set dominate the second one. Each variable name in the PREFER clause maps to variables in order of appearance. For example, there are four bindings in each result, (?u ?followers_count ?favourites_count), in the query of Figure 6. Variables in (?u1 ?followers_count1?favourites_count1) are assigned to the first binding set while (?u2 ?followers_count2 ?favourites_count2) includes variables for the second binding set. All these variables are used in the IF clause to check the dominance of the first binding set over the second.

```
1 SELECT ?u ?followers_count ?favourites_count
2 WHERE { ?u a :user;:followers_count
3       ?followers_count;:favourites_count ?favourites_count }
4           PREFER (?u1 ?followers_count1 ?favourites_count1)
5                 TO (?u2 ?followers_count2 ?favourites_count2)
6           IF ( ?followers_count1 >= ?followers_count2 && ?
7                 favourites_count1 >= ?favourites_count2 &&
8              ( ?followers_count1 > ?followers_count2 || ?
9                    favourites_count1 > ?favourites_count2) )
```

Figure 6. SPARQL skyline queries. The skyline of the most followed users who have the highest number of tweets he has liked, following SPREFQL grammar.

Similar to Gueroussova *et al.* (2013), Troumpoukis *et al.* (2017) proposed the translation from a SPREFQL query to SPARQL 1.1. A query SELECT L WHERE {P} PREFER L_1 TO L_2 IF C can be expressed SPARQL 1.1. as SELECT L WHERE {P FILTER NOT EXISTS {P{L/L_2} FILTER C{L_2/L} where P{L/L_1} is equal to P but replacing all variable names of P that appear in L with its corresponding variable in L_1, and

$C\{L_2/L\}$ is equal to C but replacing all variable names of L_2 with its corresponding variable in L. For our motivational example, the query is translated as in Figure 4.

Conclusion

In this article, we have described the syntax for specifying SPARQL skyline queries following the grammar proposed by authors of state-of-art works. Each author proposes a different grammar and implements his own tool to evaluate this type of query. Despite the fact that some proposals have been made in recent years, there is no standard language for expressing skyline queries in SPARQL. Therefore, if a user wants to evaluate a SPARQL skyline query, he must select the grammar and the tool to execute it. An alternative is to rewrite the query in SPARQL in version 1.0 or 1.1 and have it executed by any SPARQL engine, giving the user a range of options among the tools, from which to choose. This article summarises a guide to specifying SPARQL skyline queries to express preferences with different alternatives at the user's convenience. Finally, we plan to develop a tool to translate SPARQL skyline queries using the different grammars proposed, into SPARQL 1.0 and 1.1 with the aim of providing an automatic mechanism of translation.

References

Abidi, A., Elmi, S., Tobji, M. A. B., HadjAli, A., Yaghlane, B. B. (2018). Skyline queries over possibilistic RDF data. *International Journal of Approximate Reasoning*, 93, 277-289.

Abidi, A., Tobji, M. A. B., Hadjali, A., Yaghlane, B. B. (2017). *Skyline modeling and computing over trust RDF data.* Proceedings of the 19th international conference on enterprise information systems (ICEIS 2017). Setúbal: Science and Technology Publications, 634-643.

Alvarado, A., Baldizán, O., Vidal, M., Goncalves, M. (2013). FOPA: a final object pruning algorithm to efficiently produce skyline points. Database and Expert Systems Applications. DEXA 2013. Berlin: Springer, 334-348.

Bader, M. (2012). *Space-filling curves: an introduction with applications in scientific computing.* Suisse: Springer Publishing Company, Inc.

Balke W., Guntzer, U. (2004). *Multi-objective query processing for database systems*. Proceedings of the 30th international conference on very large data bases. New York: ACM Digital Library, 936-947.

Balke, W. T., Guntzer, U., Zheng, J. X. (2004). *Efficient distributed skylining for Web information systems*. Advances in Database Technology - EDBT 2004. Berlin: Springer, 256-273.

Bartolini, I., Ciaccia, P., Patella, M. (2008). Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems*, 33, 1-49.

Bentley, J., Kung, H., Schkolnick, M., Thompson, C. (1978). On the average number of maxima in a set of vectors and applications. *Journal of the ACM*, 25, 536-543.

Bitbucket. (2021). SPREFQL dataengineering/sprefql – Bitbucket [online] available in: https://bitbucket.org/dataengineering/sprefql/src/master/ [accessed: 1 March 2021].

Borzsonyi, S., Kossmann, D., Stocker, K. (2001). *The skyline operator*. Proceedings of the 17th international conference on data engineering. Heidelberg: IEEE Computer Society, 421-430.

Chen, L., Gao, S., Anyanwu, K. (2011). Efficiently evaluating skyline queries on RDF databases. The Semanic Web: Research and Applications. ESWC 2011. Berlin: Springer, 123-138.

Chomicki, J. (2002). Querying with intrinsic preferences. In: *Advances in Database Technology — EDBT 2002*. Eds. Jensen, C. S., Šaltenis, S., Jeffery, K. G., Pokorny, J., Bertino, E., Böhn, K., Jarke, M. Berlin: Springer, 34-51.

Chomicki, J., Godfrey, P., Gryz, J., Liang, D. (2003). *Skyline with presorting*. Proceedings of the 19th international conference on data engineering (ICDE 2003). Bangalore: IEEE Computer Society, 717-719.

Elzein, N. M., Majid, M. A., Hashem, I. A. T., Yaqoob, I., Alaba, F. A., Imran, M. (2018). Managing big RDF data in clouds: challenges, opportunities, and solutions. *Sustainable Cities and Society*, 39, 375-386.

Endres, M., Glaser, E. (2019). Indexing for skyline computation. In: *Flexible Query Answering Systems*. Eds. Cuzzocrea, A., Greco, S., Larsen, H. L., Saccà, D., Andreasen, T., Christiansen, H. Suisse: Springer International Publishing, 31-42.

Feigenbaum, L. (2009). SPARQL by example [online] available in: https://www.w3.org/2009/Talks/0615-qbe/ [accessed: 12 October 2020].

Feyznia, A., Kahani, M., Zarrinkalam, F. (2014). *COLINA: a method for ranking SPARQL query results through content and link analysis*. Proceedings of the 13th international semantic Web conference (ISWC 2014). New York: ACM Digital Library, 273-276.

Godfrey, P., Shipley, R., Gryz, J. (2005). *Maximal vector computation in large data sets*. Proceedings of the 31st international conference on very large data bases. New York: ACM Digital Library, 229-240.

Gueroussova, M. Polleres, A., McIlraith, S. (2013). *SPARQL with qualitative and quantitative preferences*. Proceedings of the 2nd international conference on ordering and reasoning. New York: ACM Digital Library, 2-8.

Gueroussova, M., Polleres, A., McIlraith, S. (2013b). SPARQL with qualitative and quantitative preferences (extended report). Tech. Rep. CSRG-619. Toronto: University of Toronto.

Gulzar, Y., Alwan, A. A., Abdullah, R. M., Xin, Q., Swidan, M. B. (2019). SCSA: evaluating skyline queries in incomplete data. *Applied Intelligence*, 49, 1636-1657.

Harris, S., Seaborne, A. (2013). SPARQL 1.1 query language. W3C Recommendation [online] available in: http://www.w3.org/TR/2013/REC-sparql11-query20130321/ [accessed: 1 March 2021].

Keles, I., Hose, K. (2019). *Skyline queries over knowledge graphs*. Proceedings of the 18th international semantic Web conference. Berlin: Springer, 293-310.

Keles, I., Hose, K. (2019). Skyline queries over knowledge graphs. In: *The Semantic Web – ISWC 2019*. Eds. Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I., Hogan, A., Song, J., Lefrançois, M., Gandon, F. Berlin: Springer International Publishing, 293-310.

Kossmann, D., Ramsak, F., Rost, S. (2002). *Shooting stars in the sky: an online algorithm for skyline queries*. Proceedings of the 28th international conference on very large data bases. New York: ACM Digital Library, 275-286.

Kostylev, E. V., Reutter, J. L., Ugarte, M. (2015). *Expressiveness of construct queries in SPARQL*. 18th international conference on database theory (ICDT'15). Eds. Arenas, M., Ugarte, M. Brussels: Dagstuhl Publishing, 1-25.

Křemen, P. (2018). SPARQL query language for RDF [online] available in: https://cw.fel.cvut.cz/b181/_media/courses/osw/lecture-03sparql-s.pdf [accessed: 1 March 2021].

Lee, K., Lee, W. C., Zheng, B., Li, H., Tian, Y. (2010). Z-sky: an efficient skyline query processing framework based on z-order. *The VLDB Journal*, 19, 333-362.

MIB. (2016). My information bubble project [online] available in: http://mib.projects.iit.cnr.it/ [accessed: 1 March 2021].

Ontotext. (2020). What is RDF and why to use it? Ontotext Fundamentals Series [online] available in: https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf/ [accessed: 8 December 2020].

Papadias, D., Tao, Y., Fu, G., Seeger, B. (2005). Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS) - Special Issue: SIGMOD/PODS*, 30, 41-82.

Patel-Schneider, P. F., Martin, D. (2016). *EXISTStential aspects of SPARQL*. Proceedings of 15th international semantic Web conference. Kobe: Computer Science Bibliography, 1-4.

Patel-Schneider, P. F., Polleres, A., Martin, D. (2018). Comparative preferences in SPARQL. In: *Knowledge Engineering and Knowledge Management*. Eds. Zucker, C. F., Ghidini, C., Napoli, A., Toussaint, Y. Berlin: Springer International Publishing, 289-305.

Prud'hommeaux, E., Seaborne, A. (2008). SPARQL query language for RDF. W3C Recommendation [online] available in: https://www.w3.org/TR/rdf-sparql-query/ [accessed: 2 December 2020].

RDF. (2021). RDF - semantic Web standards [online] available in: https://www.w3.org/ [accessed: 1 March 2021].

Selke, J., Balke, W. T. (2011). Skymap: a trie-based index structure for high-performance skyline query processing. Database and Expert Systems Applications. DEXA 2011. Berlin: Springer, 350-365.

Sessoms, M., Anyanwu, K. (2014). Enabling a package query paradigm on the semantic Web: model and algorithms, Transactions on Large-Scale Data -and Knowledge- Centered Systems XIII. Berlin: Springer, 1-32.

Siberski, W., Pan, J. Z., Thaden, U. (2006). Querying the semantic web with preferences. In: *The Semantic Web - ISWC 2006*. Eds. Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. M. Berlin: Springer, 612-624.

Tan, K., Eng, P., Ooi, B. (2001). *Efficient progressive skyline computation*. Proceedings of the 27th international conference on very large data bases. San Francisco: Morgan Kaufmann Publishers Inc., 301-310.

The Apache Software Foundation (2019). Arq – A SPARQL processor for jena [online] available in: http://jena.apache.org/documentation/ query/index.html [accessed: 8 November 2019].

Troumpoukis, A., Konstantopoulos, S., Charalambidis, A. (2017). An extension of SPARQL for expressing qualitative preferences. In: *The Semantic Web – ISWC 2017*. Eds. d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. Berlin: Springer International Publishing, 711-727.

Zou, L., Özsu, M. T. (2017). Graph-based RDF data management. *Data Science Engineering*, 2, 56-70.