



Comparative Preferences in SPARQL

Peter F. Patel-Schneider¹(✉), Axel Polleres^{2,3}, and David Martin¹

¹ NAIL Laboratory, Nuance Communications, Sunnyvale, CA, USA
pfpschneider@gmail.com

² Vienna University of Economics and Business/Complexity Science Hub Vienna,
Wien, Austria

³ Stanford University, Stanford, CA, USA

Abstract. Sometimes one does not want all the solutions to a query but instead only those that are most desirable according to user-specified preferences. If a user-specified preference relation is acyclic then its specification and meaning are straightforward. In many settings, however, it is valuable to support preference relations that are not acyclic and that might not even be transitive, in which case though their handling involves some open questions. We discuss a definition of desired solutions for arbitrary preference relations and show its desirable properties. We modify a previous extension to SPARQL for simple preferences to correctly handle any preference relation and provide translations of this extension back into SPARQL that can compute the desired solutions for all preference relations that are acyclic or transitive. We also propose an additional extension that returns solutions at multiple levels of desirability, which adds additional expressiveness over prior work. However, for the latter we conjecture that an effective translation to a single (non-recursive) SPARQL query is not possible.

1 Introduction

Preferences and the notion of the Semantic Web are tightly interwoven: the seminal vision article often cited as coining the term “Semantic Web” already mentions preferences in several places [1], for instance: “*Pete [...] set his own agent to [...] search with [...] preferences about location and time.*”. The same article also already mentions standardization in terms of languages defining such preferences, such as the Composite Capability/Preference Profile (CC/PP) [2], which allows a user agent (typically a client application) to declare its preferences, e.g., in terms of device capabilities.

Interestingly this early interest in expressing preferences and retrieving Semantic Web data compliant with these preferences has not found its way

A poster of part of this paper is being presented at ISWC 2018. An extended technical report version of this paper, including proofs and the main algorithm, is available at <http://polleres.net/publications/patel-schneider-etal-2018TR.pdf>

Axel Polleres’ work was supported under the Distinguished Visiting Austrian Chair Professors program hosted by The Europe Center of Stanford University.

© Springer Nature Switzerland AG 2018

C. Faron Zucker et al. (Eds.): EKAW 2018, LNAI 11313, pp. 289–305, 2018.

https://doi.org/10.1007/978-3-030-03667-6_19

into later Semantic Web Standards, such as SPARQL. There is no built-in way to express and evaluate queries with preferences in SPARQL, but there have been several proposals [3–5] for adding preferences to SPARQL and defining a meaning for such preferential queries.

We argue that preferences in their full generality are not correctly handled in the proposals so far, and show how this can be addressed with a modified semantics. Additionally, we show that certain kinds of preferences can be expressed in a single SPARQL 1.1 query, although the most recent proposal that translates preferences into standard SPARQL queries fails to work on relatively simple examples due to problems in the SPARQL 1.1 query standard [6].

Example 1 (Running Example). As a running example, let us assume data in a navigation scenario, where a user would be looking for gas stations with preferences among features such as *brand*, *distance* to the user’s location, and different *shops* that sell various *antifreeze* products, given as an RDF Graph G_1 :

```
:p123 a :GasStation; :brand :Mobil; :dist 1.1;
      :shop :TigerMart; :antifreeze :Prestone .
:p456 a :GasStation; :brand :Chevron; :dist 0.5;
      :shop :KwikieMart; :antifreeze :StarBrite .
:p789 a :GasStation; :brand :Shell; :dist 0.8;
      :shop :711; :antifreeze :Zerex .
:p012 a :GasStation; :brand :Citgo; :dist 6 .
```

User preferences could be of different forms, such as:

- P1** “I prefer gas stations within 1 mile distance” (simple boolean)
- P2** “I prefer the closest gas station” (quantitative)
- P3** “I prefer Mobil over Chevron” (comparative)
- P4** “I prefer solutions within 1 mile (**P1**) and among those I prefer Mobil over Chevron gas (**P3**), and Kwikie Mart over 7-11, and otherwise just the closest (**P2**)” (combinations)

So atomic preferences can be *simple boolean*, i.e. stating preferences for solutions fulfilling a certain boolean condition, *quantitative*, i.e., where each solution is given a score from a totally ordered set, or *comparative*, i.e., preferences expressed as a binary relation *between solutions*. Such atomic preferences can be combined in various ways.

A preference query takes the results of a non-preferential subquery (in this case, a subquery that returns gas stations), and selects most preferred ones. (The precise definition of “most preferred” is part of what we are examining in this paper.) The preferences we wish to handle call for a generalization of the skyline operator [7] in databases so we will talk about obtaining the skyline of a preference relation, i.e., the most preferred results based on a preference.¹ Subsidiary preference results can be defined, such as the *n*th skyline, i.e., the

¹ In databases, skyline involves a multiway combination of totally ordered comparisons between the values in tuples; qualitative preferences here instead allow an arbitrary comparison relation.

skyline after the first $n - 1$ skylines have been removed, or returning solutions along with their skyline number (which gives a rank for each solution).

We focus on comparative preferences, partly because (i) comparative preferences are more general than simple boolean or quantitative preferences, (ii) comparative preferences can capture combined preferences as part of their preference relation representation, (iii) user preferences are often comparative [8, 9].

As for (i), we note that quantitative preferences generalize simple boolean preferences, and comparative preferences generalize quantitative preferences, by preferring solutions with a better score to solutions with a worse score.

2 Foundations and Motivation

We adapt our formal definition of comparative preferences from that of Chomicki [10] as also used by other work in the area such as Troumpoukis et al. [5].

Definition 1. *Given a set of potential solutions P , a preference relation \succ is any relation over $P \times P$. A solution s_1 is dominated by a solution s_2 if $s_2 \succ s_1$.*

Although a preference relation is defined over a universe of *potential* solutions, it is applied to finite sets of *candidate* solutions. Typically, potential solutions are all solutions that are representable under the schema of some information source. Candidate solutions are then solutions returned from the ordinary (non-preferential) part of a query.

Example 2. Referring back to Example 1, the four gas stations shown could be the *candidate* solutions returned by the ordinary SPARQL query, Q_1 :

```
Q1: SELECT * {?X a :GasStation; :brand ?B; :dist ?D. FILTER(?D<=10)}
```

The *potential* solutions would be a much larger set (i.e., all gas stations that could be represented in an information source). The preference relation for **P3** would include all pairs (m, c) where m is a Mobil station and c a Chevron station. After applying this preference relation, for reasons discussed below, one would be left with the Mobil, Shell, and Citgo stations.

In this paper, we ground candidate solutions in results of SPARQL 1.1 queries [6] (that is, multisets of variable bindings). We also allow for solutions from SPARQL extended with external services.² The examples in this paper will not use such service extensions; for simplicity we will just use preferences over solutions of SPARQL queries over a RDF graph, such as G_1 in Example 1.

We do not require any other properties of preference relations. In particular, a preference relation here need not be irreflexive, asymmetric, acyclic, or transitive.³ It may seem that these should be required aspects of a preference relation but we want to study what happens with arbitrary preference relations, such as those likely to be obtained directly from users.

² SPARQL 1.1 provides a basic mechanism for such external services (e.g., to look up or compute current prices or exchange rates), using the SERVICE keyword [11].

³ To review the basic properties of binary preference relations see Chomicki [10].

Example 3. We allow a combination of comparative preferences on different aspects of gas stations. For example **P6a**: a preference for Mobil brand gas stations over Chevron; **P6b**: a preference for gas stations with KwikieMart stores over those with 7-11's; and **P6c**: a preference for gas stations selling Zerex antifreeze over those selling Prestone. (What is important here is that the preferences on the aspects are only *partial* orders, not total orders.)

Example 4. We also allow the obviously cyclic preference relation consisting only of **P3a**: a preference for Mobil brand gas stations over Chevron brand; **P3b**: a preference for Chevron over Citgo; and **P3c**: a preference for Citgo over Mobil.

The implementation of our preference relations in SPARQL will be arbitrary SPARQL expressions. If solutions contain objects from some sort of knowledge repository the relation can depend on anything accessible in the repository. (Troumpoukis et al. [5] call preferences that do not use external information intrinsic preferences.) In SPARQL this means that an expression defining a preference relation can access properties of a solution object from the underlying graph (such as the brand of fuel sold at a gas station, its current distance, or whether it has a roof over its pumps), and other information in the graph (such as whether it is currently raining, etc.), without having this information in the solution itself.

Our preference relations are not examinable in general, and the set of *potential* solutions will generally be very large, or even infinite.

Example 5. For instance, the preference **P5**: “I prefer between two solutions the one closer in distance”, could apply to infinitely many *potential* solutions (not knowing the underlying graph G). Note also that there could be other gasoline brands in G that are not named explicitly in the preferences at hand.

It can thus be practically infeasible to compute the transitive closure of a preference relation, or indeed determine whether a preference relation is irreflexive, asymmetric, acyclic, or transitive.

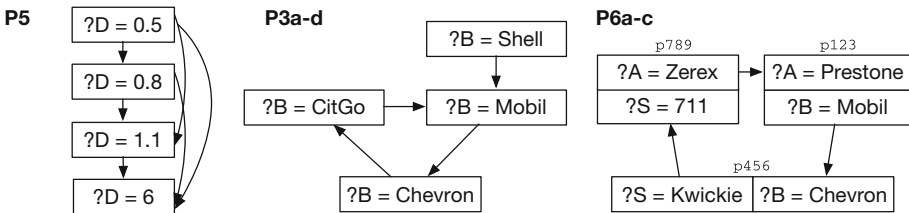


Fig. 1. Preference relations **P5**, **P3a-d**, and **P6a-c** for the solutions of query Q_1 over graph G_1 .

Example 6. For instance, **P5** is transitive on all sets of candidate solutions. However, if we view **P3a-c** as a single preference relation, transitivity does not hold. As well, additional preferences such as **P3d** “I prefer Shell over Mobil” could make the preference relation lose completeness⁴ (every element comparable with every other). See Fig. 1 for a graphical illustration. Similarly, **P6a-c** produces a preference cycle on our example gas stations.

The basic operation in comparative preferences is the winnow operator [10]. The intuitive notion is that given an candidate set of solutions and a preference relation, the winnow operator returns those solutions that have no solution dominating them. Based on its similarity to skylines in databases, we call the result of the winnow operator the skyline of a preference relation. Formally this is (again adapted from Chomicki [10]):

Definition 2. *If S is a finite set of (candidate) solutions and \succ a preference relation over some (potentially infinite) superset of S then the skyline of S with respect to \succ is*

$$\omega_{\succ}(S) = \{s \in S \mid \neg \exists s' \in S. s' \succ s\}$$

We loosely refer to the definition of skyline as the “semantics” of preference relations. Note here that \succ has access to extrinsic information about objects in solutions in S . In the SPARQL setting this means that \succ has access to the graph being queried.

We can also define the second skyline as the skyline after the initial skyline has been removed from the solution set, and so on, thus defining “levels” of skyline solutions:

Definition 3. *If S is a finite set of (candidate) solutions and \succ a preference relation over some superset of S then the n th skyline of S with \succ is defined as*

$$\omega_{\succ}^n(S) = \{s \in S \setminus \bigcup_{i=1}^{n-1} \omega_{\succ}^i(S) \mid \neg \exists s' \in S \setminus \bigcup_{i=1}^{n-1} \omega_{\succ}^i(S). s' \succ s\}$$

The rank of a solution, s , in S with respect to \succ is the number of the skyline that it is in, i.e., $\text{rank}_{\succ}^S(s) = n$ for $s \in \omega_{\succ}^n(S)$. If a solution is not in any skyline, then its rank is undefined.

Returning multiple skylines is valuable if all elements of the top skyline might be deemed unsuitable by later processing, or if a minimum number of solutions is required, which could exceed the size of the top skyline.

Example 7. Looking at Fig. 1 let us assume for the moment a slight reformulation of **P5** to be “I prefer between two solutions the one closer or equal in distance”. This obviously is not very intuitive as it makes the preference relation reflexive, and thereby makes *each solution dominate itself* – resulting in an empty skyline. However, as we will see current proposals for encoding preferences in SPARQL allow such preferences.

⁴ Chomicki calls this *connectivity* [10, Definition 2.1].

Example 8. Cyclic preferences can appear in practice by collecting single user preferences, expressed on separate occasions, and with the definitions above can lead to unexpected results. We take here the example of **P3a-d** from Fig. 1. Intuitively, among *all four* candidate solutions the solution with brand **Shell** is most preferred, but then under Definition 3 the second skyline is *empty*, since *each remaining solution is dominated by another candidate solution*. Our intuition is that in such a case all of these three solutions should be equally preferred, and in the second skyline. Note that **P6a-c** also produce a cycle in the 4 candidate solutions, even though their expression does not look obviously cyclic.

Assuming, on the other hand, that **Citgo** was *not* within the candidate solutions, say by changing Q_1 to Q_2 as follows:

```
Q2: SELECT * {?X a :GasStation; :brand ?B; :dist ?D. FILTER(?D<=5)}
```

That would reduce the number of candidate solutions to just **Shell**, **Mobil**, and **Chevron**, which would be in the first, second and third skyline, respectively (this time in accord with our intuitions). So, we see that removing (and likewise adding) candidate solutions can change the rank and ordering of solutions.

To complicate things further, if we consider **P3a-d** or **P6a-c** in combination with **P1** the handling of the combined preference relation becomes even less clear, depending on the semantics of combinations and their respective consideration of candidate solutions.

Given examples such as these, we aim to shed more light on the semantics for preferences in queries, and provide definitions that handle these situations more intuitively and satisfy basic desiderata, such as that each level of skyline should contain at least one solution, even in cases where preferences are not necessarily coherent.

Our primary contributions in the present work are

- an analysis of what goes wrong with a widely used definition of the skyline operator when a preference relation fails to be acyclic and/or transitive,
- a definition that works for arbitrary preference relations,
- a slightly simpler definition that can be used with transitive preference relations, with performance benefits,
- translations to SPARQL 1.1 (based on translations proposed in prior work [5, 12]) to implement these new definitions, and
- a further extension that allows a query to request *multiple levels* of skylines.

In the next section, we will review some earlier approaches to the formulation of preferences in SPARQL, in the light of whether they can express comparative semantics at all and how they would handle the cases above. We will then approach the analysis of preference relations and variations of the definition of the skyline operator more formally in Sects. 4 and 5, by discussing what we call simple (i.e., acyclic and/or transitive) preferences and non-simple preferences separately. Along the way we will refine the notion of skyline operators and provide two more variations thereof. Finally, in Sects. 6 and 7 we will discuss which

of these proposed variations can be implemented in SPARQL 1.1, and which require extensions in terms of bespoke evaluation algorithms. In this topic, we will identify problems in existing approaches to translating preference handling to SPARQL, and propose repairs, where possible.

3 Previous Work

The notion of preference has a central role in many disciplines, including economics, psychology and other social sciences, some areas of philosophy, decision theory and game theory (themselves interdisciplinary topics), and computer science. The formalization of comparative preferences as binary preference relations runs throughout much of this work, although the definitional details vary. Much work on preference relations assumes, mandates, or arranges for them to be acyclic and/or transitive (or constrained in other ways), for a variety of reasons. Nonetheless, there is a large literature showing that cyclic and/or intransitive preference relations do arise naturally in the real world of human judgments [8, 9].

Turning to preferences used with query languages, Rosati et al. [13] show how a set of preferences modeled as a CP-net can be represented in RDF, and how generated SPARQL queries can use the RDF representation to rank the results of an ordinary (non-preferential) SPARQL query. Whereas the preferences semantics discussed here are determined primarily by the definition of skyline, in their approach the semantics are determined by the CP-net formalism. Whereas we are concerned with preferences expressed directly in a SPARQL query, along with non-preferential query clauses, their preferences are acquired and represented independently of the non-preferential SPARQL queries to which they apply⁵.

Chomicki [10] analyzes intrinsic comparative preferences in a relational database setting. He considers preferences as multidimensional combinations of atomic preferences where each atomic preference is a built-in SQL function (such as numeric ordering). A preference function could be to consider distance and fuel brand separately, which ends up with the closest gas station for each brand, or to first consider distance and then some ordering of brands, which ends up with the closest gas station but if there is a tie for closest then chooses by brand. He analyzes the properties of his winnow operator with an eye to how it can be optimized in SQL queries.

Siberski et al. [3] transform a subset of these preferences into an early version of SPARQL, producing a SPARQL query extension for conjunctive (combine preferences over two different values) cascaded (consider preference over one value before preference over another) intrinsic preferences. They implement these preferences as an extension to ARQ⁶, with a syntax inspired by an early version of the Preference SQL language [14].

⁵ In a later version of [13], available on semanticscholar.org, the approach is extended to handle CP-theories.

⁶ <https://jena.apache.org/documentation/query/>.

Gueroussova et al. [4, 12] transform a version of Chomicki’s preferences (which they call conditional preferences) to SPARQL, extending the work of Siberski et al. by adding more combination operators. They define an extension to SPARQL, which they call PrefSPARQL (after the later version of Preference SQL [15] which incorporates these features) and provide a mapping from PrefSPARQL to both SPARQL 1.0 queries [16] and SPARQL 1.1 queries.

Unfortunately their mapping to SPARQL 1.1 uses the SPARQL EXISTS operator. SPARQL EXISTS has many known problems [17], and their solution falls prey to one them, namely that the SPARQL BOUND operator does not work correctly inside EXISTS. Their mapping to SPARQL 1.0 is not affected by this problem, as shown by the next example.

Both the approaches by Siberski et al. and Gueroussova et al. focus on combined preferences over boolean preferences, that is, while they allow the combination of different preferences, these preferences do not give the full flexibility of comparing arbitrary solutions pairwise, but only by declaring boolean “preferred conditions” C , that divide the solution space into preferred and non-preferred solutions according to the preference, according to the schema in Eq. (1).

$$\text{SELECT } V \text{ WHERE } \{ P \text{ PREFERRED } C \} \quad (1)$$

Example 9. In both the approaches above preferences like **P1** can be written by extending queries like Q_1 . In PrefSPARQL this is:

```
Q3: SELECT * { ?X a :GasStation; :brand ?B; :dist ?D .
  PREFERRED (?D <= 1) }
```

Combined preferences can be expressed (so long as they do not involve comparisons between different attributes of solutions), such as this variant of **P4**:

P4' “I prefer solutions within 1 mile and among those I prefer Mobil, and otherwise just the closest.”

which in PrefSPARQL could be written as the following query:

```
Q4: SELECT * { ?X a :GasStation; :brand ?B; :dist ?D .
  PREFERRED ((?D <= 1) AND ?B="Mobil") PRIOR TO LOWEST ?D }
```

While referring for details to [12], we illustrate PrefSPARQL’s translation back to “vanilla” SPARQL by the example of Q_3 which in SPARQL 1.1 yields $Q_3^{1.1}$:

```
Q31.1: 1 SELECT * { {?X a :GasStation; :brand ?B; :dist ?D .
  2   FILTER NOT EXISTS { ?X' a :GasStation; :brand ?B'; :dist ?D' .
  3     FILTER ((?D' <= 1) > (?D <= 1)) } }
```

That is, the translation relies in principle on creating a copy of the query pattern within a SPARQL 1.1 NOT EXISTS clause (line 2) and then encoding dominance of the solutions to this copy in an inner FILTER expression (line 3).

As shown in [12], this principle can be easily extended to combined preference relations (through encoding AND and PRIOR TO, and adding conditional IF-THEN-ELSE preferences from Preference SQL to more complex inner

FILTER expressions). Specific quantitative comparative preference relations are also expressible with the keyword HIGHEST or LOWEST, but not general comparative preferences.

Example 10. The quantitative preference **P2** would be expressible in PrefSPARQL as query

```
Q4: SELECT ?X ?B ?D { ?X a :GasStation; :brand ?B; :dist ?D.
      PREFERRING LOWEST ?D }
```

However, there exists as mentioned above a problem with this translation due to the semantics of NOT EXISTS, as illustrated with the following example:

Example 11. Let us assume **P7** saying “I prefer gas stations with a shop”, which in PrefSPARQL could be expressed as

```
Q5: SELECT * { ?X a :GasStation. OPTIONAL { ?X :shop ?S. }
      PREFERRING ( BOUND(?S) ) }
```

in which case the translation from [12] no longer works, as shown in $Q_5^{1:1}$:

```
Q51:1: 1  SELECT * { { ?X a :GasStation. OPTIONAL { ?X :shop ?S. }
2          FILTER NOT EXISTS { ?X' a :GasStation. OPTIONAL { ?X' :shop ?S'. }
3          FILTER(BOUND(?S') > BOUND(?S)) } }
```

This fails because the substitution semantics of EXISTS here produces algebra expressions like BOUND (:TigerMart) that are undefined in SPARQL 1.1; we refer to details in [17]. We note though, that Guerousova et al.’s [12] SPARQL 1.0-based way of encoding non-existence through a combination of OPTIONAL and !BOUND(), as illustrated in the following query $Q_5^{1:0}$, works as intended:

```
Q51:0: 1  SELECT * { { ?X a :GasStation. OPTIONAL { ?X :shop ?S. }
2          OPTIONAL { ?X' a :GasStation. OPTIONAL { ?X' :shop ?S'. }
3          FILTER ( BOUND(?S') > BOUND(?S) ) }
4          FILTER ( !BOUND(?X') ) }
```

Troumpoukis et al. [5] expand on this work to allow arbitrary SPARQL expressions as the \succ operator; that is, they define an extension to SPARQL 1.1 that can express full comparative preferences called SPREFQL, and also provide a mapping from SPREFQL into SPARQL 1.1 queries. They implemented SPREFQL and compared its performance to the performance of their mapping.

As opposed to PrefSPARQL, in SPREFQL one can express comparative preferences by explicitly referring to variables in the “copy” of the query pattern over which preferences are defined using a clause PREFER-TO-IF which creates two explicit copies V_1, V_2 of the variables V in the SELECT clause that can be referenced in a comparative condition C :

$$\text{SELECT } V \text{ WHERE } \{ P \} \text{ PREFER } V_1 \text{ TO } V_2 \text{ IF } C \quad (2)$$

Example 12. The quantitative preference **P2** would be expressible in SPREFQL as a comparative preference in query Q_6 :

```
Q6: SELECT ?X ?B ?D { ?X a :GasStation; :brand ?B; :dist ?D. }
      PREFER ?X1 ?B1 ?D1 TO ?X2 ?B2 ?D2 IF (?D1 < ?D2)
```

More general comparative preferences such as **P3** can also be expressed in SPREFQL, as in query Q_7 :

```
Q7: SELECT ?X ?B ?D { ?X a :GasStation; :brand ?B; :dist ?D . }
    PREFER ?X1 ?B1 ?D1 TO ?X2 ?B2 ?D2
    IF (?B1 = :mobil && ?B2 = :chevron)
```

Combined preferences are also supported, through AND and PRIOR TO clauses. Unfortunately, however, with the syntactic expansion over PrefSPARQL, it is quite possible for the preference relation to be non-transitive or to have loops.

Example 13. For instance, imagine Q_6 modified by replacing $<$ with \leq (which we will refer to as Q'_6 below), so that each solution dominates itself. Another example involves expressing **P6a-c** as:

```
Q8: SELECT ?X ?B ?S ?T {?X a :GasStation; :brand ?B; :shop ?S; :antifreeze ?A.}
    PREFER ?X1 ?B1 ?S1 ?A1 TO ?X2 ?B2 ?S2 ?A2
    IF ( (?B1 = :Mobil && ?B2 = :Chevron) ||
        (?S1 = :KwikieMart && ?S2 = :711) ||
        (?A1 = :Zerex && ?A2 = :Prestone) )
```

This is problematic, as Troumpoukis et al. only use the simple definition of winnowing above. If there is a preference loop in the candidate solutions, as for Q_8 on the example data, then none of the solutions in the loop will ever be returned, because each of them is dominated by another solution. The preference combination operators do not help: AND will just produce the same cyclic preference and PRIOR TO does not have a suitable meaning. This problem also occurs for reflexive loops, as for Q'_6 where each solution dominates itself. We provide a simple but general solution for these problems below. As we will see, the absence of transitivity does not cause a problem by itself, but does complicate the problem of loops.

Also, note that Troumpoukis et al. provide a mapping to SPARQL 1.1 where the schema of Eq. (2) is replaced by (again, we refer for details to [5]):

$$\text{SELECT } V \text{ WHERE } \{ P \text{ FILTER NOT EXISTS } \{ P_{(V/V_1)} \text{ FILTER } C_{(V_2/V)} \} \} \quad (3)$$

This is a straightforward extension of the PrefSPARQL translation, which however depends again on SPARQL EXISTS and also falls prey to the problem with BOUND, as in Example 11.

4 Simple Comparative Preferences

We first establish that acyclic preference relations are non-problematic: they are guaranteed to determine a nonempty skyline over any finite, non-empty set of solutions and ranks behave well. In addition, as we discuss in detail in Sect. 6, a skyline query over an acyclic preference relation is specifiable in SPARQL 1.1. As every transitive and irreflexive relation is acyclic, handling acyclic preference relations also handles this common requirement imposed on preference relations.

Theorem 1. *If the preference relation \succ is acyclic over a finite, non-empty set of candidate solutions S , i.e., there is no candidate solution s for which there is a sequence of candidate solutions starting and ending with s such that each dominates the next, then $\omega_{\succ}(S)$ is non-empty.*

Because each skyline is non-empty for a finite, non-empty set of solutions if the preference relation is acyclic, each solution has a uniquely defined rank.

Corollary 1. *If the preference relation \succ is acyclic over a finite, non-empty set of candidate solutions S then $\mathbf{rank}_{\succ}^S(s)$ is defined for each $s \in S$.*

Dominance between solutions is reflected in their relative ranks.

Theorem 2. *If the preference relation \succ is acyclic over a finite, non-empty set of candidate solutions S then $s_1 \succ s_2$ implies $\mathbf{rank}_{\succ}^S(s_1) < \mathbf{rank}_{\succ}^S(s_2)$.*

As Troumpoukis et al. [5] use ω_{\succ} as their winnow operator, their solution performs correctly on acyclic preference relations. However, if the preference relation has loops, then none of the solutions in the loop will be in any skyline (and thus none of them will have a rank), even if there is no other solution dominating any solution in the loop. Even a simple reflexive loop, such as accidentally writing \leq instead of $<$ as in Example 13, query Q'_6 above, causes problems for Definition 2. A preference written like this seems unintuitive, but it is not forbidden in SPREFQL. As arbitrarily complex SPARQL expressions can occur in the IF clause it might not be obvious or even possible beforehand to determine whether a preference is irreflexive.

5 Non-Simple Comparative Preferences

To address such cases, we begin by addressing the empty-skyline problem mentioned above, which can be done by modifying the definition of skyline.

What makes intuitive sense in the presence of preference loops *in the candidate solutions* is to consider all the solutions in the loop as if they were equally preferred, i.e., they don't count as dominating each other or themselves. This regains the desirable property that a finite, non-empty set of candidate solutions has a non-empty skyline.

Formally, we modify the definition of skyline (Definition 2) as follows:

Definition 4. *If S is a finite set of solutions and \succ a preference relation over some superset of S then the skyline of S with \succ is*

$$\omega_{\succ}^l(s) = \{s \in S \mid \neg \exists s' \in S. (s' \succ^* s \wedge \neg (s \succ^* s'))\}$$

where \succ^* is the transitive closure of \succ over candidate solutions, i.e., there is a sequence of candidate solutions, each dominating the next.

In English, this says that a solution is in the skyline if there is no solution that transitively dominates it and that it does not transitively dominate.

This definition regains the desirable properties from above, slightly modified.

Theorem 3. $\omega_{\succ}^l(S)$ is non-empty for S any finite, non-empty set of candidate solutions.

Corollary 2. $\text{rank}_{\succ}^S(s)$ is defined for each $s \in S$ for S any finite, non-empty set of candidate solutions.

Loops cause the rank of solutions in the loop to be the same, so dominance only produces a rank at least as large.

Theorem 4. For any preference relation \succ over a finite, non-empty set of candidate solutions S , $s_1 \succ s_2$ implies $\text{rank}_{\succ}^S(s_1) \leq \text{rank}_{\succ}^S(s_2)$.

Note that, in Definition 4, transitive closure is needed for both the ancestor and the descendant of s . Consider a solution s that dominates only a single element s_1 of a minimal domination cycle⁷ s_1, \dots, s_n, s_1 , with $n > 2$. Now s should knock each s_i out of a skyline but no s_j should. To get to *only* s requires looking at the transitive dominators of s_i , not just its direct dominators.

Example 14. Getting back to the preference relation from the right-hand-side of Fig. 1, the solution s_{Shell} dominates s_{Mobil} directly, but $s_{Chevron}$ and s_{Citgo} are dominated only indirectly. Thus, if in Definition 4 $s' \succ^* s$ were replaced by $s' \succ s$, then $s_{Chevron}$ and s_{Citgo} – counter to our intuition – would end up in the first skyline. On the other hand, if we replaced $\neg(s \succ^* s')$ with $\neg(s \succ s')$, it is easy to see that the second skyline would be empty, again going counter to our intuition.

As a special case (specifically considered here because of its relationship to SPARQL), if the preference relation is known to be transitive then there is no need to compute its transitive closure; that is, the following simpler definition suffices to deal such known transitive (including reflexive) preferences, even if they are cyclic. Here, in English, a solution is in the skyline if there is no solution that *directly* dominates it and that it does not *directly* dominate.

Definition 5. If S is a finite set of (candidate) solutions and \succ a transitive preference relation over some superset of S then the skyline of S with \succ is

$$\omega_{\succ}^t(S) = \{s \in S \mid \neg \exists s' \in S. (s' \succ s \wedge \neg(s \succ s'))\}$$

This definition maintains the desirable properties from Theorems 2, 3, and 4, for transitive preference relations. They all come from the simple observation that the transitive closure of a transitive relation is itself, so Definitions 4 and 5 coincide for transitive preference relations. Note that transitivity is not actually required in general for Definition 5 to suffice, but only transitivity into loops, i.e., a direct dominator of any solution in a loop is a direct dominator of every solution in the loop. We call the preference relation \succ *clique-cyclic* in this case.

Theorem 5. Let S be any finite set of (candidate) solutions. Let \succ be clique-cyclic, i.e., for any solutions s_1 and s_2 , if $s_1 \succ^* s_2$ and $s_2 \succ^* s_1$ then for any solution s if $s \succ s_1$ then $s \succ s_2$. Then $\omega_{\succ}^t(S) = \omega_{\succ}^l(S)$.

⁷ A domination cycle is minimal if the only domination relationships between elements of the cycle are those from one element of the cycle to the next.

6 Comparative Preferences in SPARQL

As discussed in Sect. 3 above, languages like PrefSPARQL and SPREFQL have suggested translations to native SPARQL, thus showing a – not necessarily very efficient – implementation path in terms of off-the-shelf engines, and proving that the respective languages do not add expressivity on top of SPARQL. However, since both these translations only implement the simple winnow operator from Definition 2, we now turn to the question whether $\omega_{\succ}^t(S)$ and $\omega_{\succ}^l(S)$ can likewise be expressed in SPARQL 1.1.

As we want to allow for general comparative preferences, we adopt the SPREFQL syntax [5] as opposed to the syntax of earlier work such as PrefSPARQL [4].

We recall the mapping from SPREFQL to SPARQL 1.1 by the schema given in Eq. (3). While Troumpoukis et al. already suggest that evaluating this translation is potentially more expensive than directly implementing PREFER, for now we are only concerned with the expressibility of the different variations of skyline operators we introduced in SPARQL.

We recall there are two problems in the original translation: first, the use of NOT EXISTS in the translation into SPARQL, and second, that the reliance on the skyline operator of Definition 2 only works for the simple case of acyclic preference relations.

The first problem can be overcome using the translation that uses OPTIONAL and !BOUND() instead of NOT EXISTS from Guerousova et al. [12], cf. Example 11. This idea can be generalized to SPREFQL with the following mapping for Eq. (2).

Mapping 1 (Simple Mapping to SPARQL)

```
SELECT V WHERE { P
  OPTIONAL { P(V/V1) FILTER ( C(V2/V) ) BIND (1 TO ?exists) }
  FILTER (!BOUND(?exists)) }
```

Theorem 6. *Mapping 1 correctly implements ω_{\succ} .*

Proof. (Sketch) The OPTIONAL part only binds a value to ?exists when a dominator exists so the final filter only lets through solutions that are not dominated.

Handling the second problem however requires changing the semantics of PREFER. When we view the intuitive meaning of Eq. (2) decoupled from the mapping to SPARQL it conceptually reduces to first constructing the solution set S for

```
SELECT V WHERE { P }
```

and then eliminating non-dominated solutions of this query according to the chosen winnow operator. So we need to repair the semantics to use our winnow operator ω^l , which as we showed above produces desirable results for any preference relation, instead of the original ω . Of course this is quite a significant

change. On the plus side, it doesn't have problems with loops. On the negative side, it may require computing (a part of) the transitive closure of \succ . We will discuss next whether and how this computation can also be realized within SPARQL itself, or by means of bespoke algorithms.

7 Implementing SPARQL Preferences

This repaired version of comparative preferences semantics can be efficiently implemented. Instead of just checking whether a solution has a direct dominator we have to check its transitive dominators and see whether they are in a loop. This sounds expensive, going around the loops repeatedly, but can actually be done relatively efficiently.

The basic idea⁸ behind the algorithm is to check \succ between each pair of candidate solutions. The algorithm keeps track of a representative for each solution which represents all the solutions that are in loops involving the solution. When a new loop is found, the solutions in the loop are given the same representative. This operation has to be done efficiently over the entire exploration. Fortunately, the union-find algorithm [18] does precisely this in time $O(n \log^*(n))$, where n is the number of candidate solutions. After the representatives are found, all that is needed is to check whether a solution has a direct dominator with a different representative.

The algorithm checks \succ between each pair of solutions so its running time is dominated by the n^2 computations of \succ ; the union-find algorithm only adds $O(n \log^*(n))$. The most significant change in actual running time between the computation of ω_\succ and ω_\succ^t will be due to not being able to quit checking for dominators of a solution when the first one is found.

If the preference relation is known to belong to either of the special cases discussed in the context of Definition 5, i.e., if either the preference relation is acyclic, transitive or clique-cyclic, then it is possible to translate preferences back into SPARQL itself. For an *acyclic* preference relation the translation is the one in Mapping 1. For transitive preference relations (or, likewise, if \succ is clique-cyclic) a slightly more complex translation is needed.

Mapping 2 (Mapping to SPARQL for transitive Preferences)

```
SELECT V WHERE { P
  OPTIONAL { P_{(V/V_1)} FILTER ( C_{(V_2/V)} && ! C_{(V_1/V,V_2/V_1)} )
    BIND (1 TO ?exists) }
  FILTER (!BOUND(?exists)) }
```

Theorem 7. *Mapping 2 correctly implements the winnow operator ω_\succ^t .*

The general case is much tougher to translate back to a single SPARQL query as it has to compute (part of) the transitive closure of \succ over the solutions. If, however, we allow for multiple queries we can first construct a graph that reifies

⁸ The full algorithm is in the extended technical report version of the paper.

the solutions and asserts \succ between them and then determine the skyline with a subsequent query against the union of that constructed graph and the original graph.

8 Multiple Skylines

There are situations where more solutions than the top skyline are desired, and it is inconvenient to submit multiple different queries to get those additional solutions, manually eliminating prior solutions. Thus, a SPARQL extension that returns multiple levels of skylines, possibly including an indicator of each solution's rank, seems natural.

Siberski et al. [3] propose that a use of SPARQL's existing keyword pattern "LIMIT k ", in combination with their proposed keyword `PREFERRING`, can inform the query evaluator to retrieve enough levels of skyline to produce solutions numbering at least k . However, this approach has 3 weaknesses: it gives the LIMIT modifier a counterintuitive special meaning when it is used in that combination; it precludes the use of LIMIT with its usual meaning; and it does not support the specification of an explicit number of complete skylines to be returned.

To address these issues, we propose the addition of the keyword `SKYLINE`, to be used in conjunction with `LIMIT`, in either of the following patterns:

- `LIMIT SKYLINE n [TO m] [AS v_{rank}] ...` return complete skyline(s) with rank n (or with ranks n to m , with $1 \leq n \leq m$).
- `LIMIT SKYLINE ALL AS v_{rank} ...` return all skylines.

The `AS v_{rank}` is optional if explicit limits are given. If present it adds a new binding to the solution bindings of the query assigning to variable v_{rank} the rank of the solution. Ranks are counted starting at 1, and the absence of a `LIMIT SKYLINE` clause is equivalent to `LIMIT SKYLINE 1`.

Our implementation of ω_l can be simply extended to compute which skyline a solution is in. Each solution is initially given a tentative skyline of 1. When a non-looping dominator of a solution is found the solution's representative is assigned to the tentative skyline that is the maximum of its previous tentative skyline and one plus the tentative skyline of the dominator's representative. As non-looping dominators are only found when their dominators have been completely processed their tentative skyline is their final one. This way each solution representative is assigned to their skyline number so the algorithm can produce the first (top) skyline, solutions in skyline(s) ranked between n and m along with their skyline number, or all solutions with their skyline number.

9 Conclusions

We have considered the semantics of queries that rely on user preferences, extensions of SPARQL that would allow for handling such queries, and their possible implementation by translation into SPARQL. We identified several categories of

preference relations with different characteristics that are significant in terms of specifying preference query semantics and specifying translations to SPARQL.

We summarize our conclusions, in order of increasing generality of the allowed preference relations: The semantics of *acyclic* preference relations are as (implicitly) indicated by Troumpoukis et al. [5]. However, we identified a problem with their translation to SPARQL, and showed how it can be repaired. *Transitive, irreflexive* preference relations, which occur in many applied settings, as a special case of acyclic preference relations, are subject to the same observations. For *transitive* preference relations in general, a modified semantics is needed, as well as a slightly more complex translation into SPARQL. This semantics allows for more efficient processing than the most general semantics mentioned below. We defined a category of *clique-cyclic* preference relations (a superset of transitive preference relations, cf. Theorem 5), which can be handled with the same semantics as transitive preference relations. Finally, for *arbitrary* preference relations, we gave a semantics that is slightly more complex than for transitive preference relations, and showed that preference queries can be implemented by translation to multiple (sequentially executed) SPARQL queries.

In addition, we showed that our proposed semantics and implementation for each of these categories satisfies basic desiderata for the results of queries with preferences, and we discussed an algorithm that would be more efficient than the implementation by multiple, nested SPARQL queries. Finally, we proposed an additional SPARQL extension that provides a

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. *Sci. Am.* **284**(5), 28–37 (2001)
2. Klyne, G., et al.: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation, January 2004
3. Siberski, W., Pan, J.Z., Thaden, U.: Querying the semantic web with preferences. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 612–624. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_44
4. Guerousova, M., Polleres, A., McIlraith, S.A.: SPARQL with qualitative and quantitative preferences. In: 2nd International Workshop on Ordering and Reasoning (OrdRing 2013), at ISWC 2013. CEUR Workshop Proceedings, vol. 1059 (2013)
5. Troumpoukis, A., Konstantopoulos, S., Charalambidis, A.: An extension of SPARQL for expressing qualitative preferences. In: d’Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 711–727. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_42
6. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation, March 2013. <http://www.w3.org/TR/sparql11-query/>
7. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings 17th International Conference on Data Engineering, pp. 421–430 (2001)
8. Tversky, A.: Intransitivity of preferences. *Psychol. Rev.* **76**(1), 31–48 (1969)

9. Nurmi, H.: Making sense of intransitivity, incompleteness and discontinuity of preferences. In: Zaraté, P., Kersten, G.E., Hernández, J.E. (eds.) GDN 2014. LNBIP, vol. 180, pp. 184–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07179-4_21
10. Chomicki, J.: Preference formulas in relational queries. *ACM Trans. Database Syst.* **28**(4), 427–466 (2003)
11. Buil-Aranda, C., Arenas, M., Corcho, O., Polleres, A.: Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Semant.* **18**(1), 1–17 (2013)
12. Guerousova, M., Polleres, A., McIlraith, S.A.: SPARQL with qualitative and quantitative preferences (extended report). University of Toronto CSRG Report 619 (2013)
13. Rosati, J., Di Noia, T., Lukasiewicz, T., De Leone, R., Maurino, A.: Preference queries with ceteris paribus semantics for linked data. In: Debruyne, C., et al. (eds.) On the Move to Meaningful Internet Systems: OTM 2015 Conferences. OTM 2015. Lecture Notes in Computer Science, vol. 9415. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26148-5_28
14. Kießling, W., Köstler, G.: Preference SQL - design, implementation, experiences. In: 28th International Conference on Very Large Data Bases, pp. 990–1001 (2002)
15. Kießling, W., Endres, M., Wenzel, F.: The preference SQL system - an overview. *IEEE Data Eng. Bull.* **34**(2), 11–18 (2011)
16. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008). <https://www.w3.org/TR/rdf-sparql-query/>
17. Patel-Schneider, P.F., Martin, D.: EXISTStential aspects of SPARQL. In: The 15th International Semantic Web Conference (ISWC 2016), October 2016
18. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. *J. ACM* **22**(2), 215–225 (1975)