

# Query-based Framework for Customized User Selection

## Technical Report and Language Manual

### Abstract

The selection of an adequate set of candidate users based on a repository of profiles is a crucial component in many modern applications, including crowdsourcing platforms, social networks and user-based recommender systems. Criteria for selection vary greatly across applications and depend on semantically rich user information, and so the implementation of user selection modules is highly nontrivial. To assist application developers in this task, we propose a novel declarative framework that includes (1) a query language that extends SPARQL to allow capturing the properties and similarity of (query-defined parts of) user profiles/data, (2) a generic similarity metric allowing to compare (parts of) user profiles and to support soft constraints in a semantically rich environment, and (3) an efficient query evaluation algorithm derived from a theoretical analysis of our novel similarity metric. Our experimental study on real-life data indicates the effectiveness and flexibility of our approach, showing in particular that it outperforms existing task-specific solutions in common user selection tasks.

## 1 Introduction

The effective selection of candidate users from a repository of profiles is a central component in many applications. Examples include crowd-based platforms that aim to choose relevant crowd members to perform a given task, recommender systems that compute recommendations based on the preferences of previous users, and social networks that identify users who share similar properties. The criteria for user selection vary across applications, and include the relevance of their profiles to a query/task (e.g., [9]); their (estimated) expertise (e.g. [13, 19, 26]); their similarity to the current end-users (e.g. [11, 23]), and other considerations, usually combined in ways that are application- or domain-specific. Lacking a *generic* solution, developers often have to invest non-trivial efforts in employing user selection in new applications.

*To this end, we propose a novel declarative framework that supports flexible user selection over wide-ranging applications, from crowdsourcing tools to recommender systems.* Our framework supports semantically rich user profiles, expressible enough to capture the data and semantics of various real-life applications. Over this representation, selection criteria can be specified via a newly developed *query language* (SPARQL-based) called SPARQ-U. An important component of SPARQ-U is a novel *way of measuring similarity* between

```

1 SELECT ?u
2 FROM ontology WHERE
3   {?x instanceOf Place. ?x near Le_Marais,Paris}
4 FROM basic-profile(?u) WHERE
5   {?u livesIn ?x}
6 FROM extended-profile(?u) WHERE
7   {?u like Dogs} WITH SUPPORT >= 0.5
8 SIMILAR basic-profile(?u) TO basic-profile(Isabella)
9   RESTRICTED TO {?y hasProfession ?h}
10   WITH SIMILARITY >= 0.75
11 SIMILAR extended-profile(?u) TO
12   {?u practice Yoga. _ at Park.}
13   WITH SIMILARITY AS querySim >= 0
14 ORDER BY querySim LIMIT 5

```

Figure 1: Example SPARQ-U query  $Q_{\text{Isabella}}$

user profiles or parts thereof, which generically accounts for “soft constraints” such as relevance and expertise. SPARQ-U queries are fed to our query engine, employing in particular an *efficient algorithm for similarity computation*.

SPARQ-U is geared towards being used by *developers* of relevant applications. On the one hand, as different applications require highly different user selection criteria, one cannot expect a one-size-fits-all solution. On the other hand, developing user selection solutions from scratch is a laborious task that requires a high level of expertise. Our declarative framework enables employing customized user selection in wide-ranging applications while abstracting the computation.

We next highlight the challenges in user selection and the way our framework may be employed to alleviate them, via an example.

**Running Example** Consider an online dating service supporting customized search for potential partners. Isabella, for instance, is interested in users who live in the vicinity of her neighborhood (Le Marais, Paris), like dogs, have a similar profession to hers, and who practice yoga at some park. Some of these requirements may be *hard constraints*, e.g., she may insist, as a dog owner, on a partner who likes dogs. Others, *soft constraints*, may still be acceptable if partially satisfied. For instance, Isabella’s notion of an ideal partner may be a person who practices yoga in a park, but she may also consider other potential partners, e.g., a person who practices Pilates in a park (Pilates, similarly to yoga, is a kind of mind-body fitness), or a person who likes hanging out in parks, etc. Also note that while some criteria may be captured by (partial) profile similarity (in Isabella’s case, she seeks partners with similar profession to herself), others may be independent of the end-user’s profile (e.g., Isabella may have never done yoga in a park, yet wish for a partner who will help her acquire this habit).

Generally, identifying the potential partners requires: (1) analyzing and comparing user profiles or selected parts thereof (e.g. professions), and (2) designing and employing a notion of *similarity* that is flexible enough to account for different “soft” criteria and that is applicable for such semantically rich data, e.g., for assessing the similarity between practicing yoga at some park and practicing Pilates somewhere outdoors.

**User Model** We support an RDF-based representation of user profiles, in two complementary repositories: an *ontology* that captures application-specific

and/or general semantics of the data (e.g., that yoga and Pilates are forms of mind-body fitness) and *user profiles* that capture facts about individual users. A user’s profile consists of two parts: a *basic profile* that records available user properties (e.g., name, location, etc.) and an *extended profile* where properties also have an associated score, called its *support level*. The latter is used to capture degrees and frequencies of user preferences and habits (e.g. that Isabella likes dogs *a lot* or reads books in the park *frequently*).<sup>1</sup> Such profile data may either be provided by the user or collected/derived by the hosting system, e.g., from previous answers to questions posed by the system [4]; by tools that derive user profiles from social networks [12]; by tools for worker performance assessment [2, 7]; etc. Following the typical situation in many applications, we allow profiles to be highly incomplete and do not assume that missing properties are false (*open-world assumption*).

**Query Language** We illustrate SPARQ-U by continuing the dating service example. The dating service may provide a Web form for customizable search, using which Isabella can specify her preferences. These preferences, according to the logic provided by the service developers, will then be compiled into a SPARQ-U query.

An example query resulting from Isabella’s input is shown in Figure 1. Briefly, the first parts of the query define hard constraints: potential partners must live in the vicinity of Les Marais, Paris (lines 2-5) and like dogs (lines 6-7). The other parts define soft/similarity constraints: the potential partners should resemble Isabella in terms of profession (lines 8-10), and should regularly practice yoga at a park or have a similar habit (lines 11-13). We overview the query syntax and semantics in Section 2, while the full language manual is available in Appendix B. For now, we note that the conjunction of hard and soft constraints in SPARQ-U allows for flexibility of criteria specification.

**Similarity Measure** An important and challenging part of our query language is the **SIMILAR** construct and its interaction with other query parts. In our setting, this measure must account for:

- data semantics (as specified in the input ontology)
- user profiles, i.e., possibly incomplete sets of properties
- extended user profiles, i.e., support scores
- query evaluation, i.e., efficient online comparison of selected profile parts and/or query-specified criteria (e.g. frequent practicing of yoga in a park)

Our user profiles are composed of sets of properties. Each property is composed of (RDF-style) facts, e.g., “Isabella practices yoga”, which in turn are composed of terms/concepts such as “practices” and “yoga”. We thus build our measure gradually, by generalizing existing definitions of semantic similarity for *individual concepts* [11, 28]. These definitions rely on an input taxonomy of concepts to capture data semantics. We first define “lifted” taxonomies of facts and of property sets, capturing their semantics, based on a concept taxonomy that is extracted from our input ontology. We show how this facilitates the comparison of basic user profiles/parts thereof, and in particular explain why this definition

<sup>1</sup>The two parts of a user’s profile can be easily merged by associating the basic profile properties with the maximum support score, to assert their absolute truth. The distinction of profile parts is done to ease the presentation in the sequel.

accounts for the open-world assumption. We then extend the definition to account for numeric support scores. We show that the obtained similarity notions satisfy formal constraints that we claim should hold for any similarity measure in this context. (See Section 3.)

**Query Evaluation** In turn, the main computational challenge in evaluating SPARQ-U queries is in computing semantic similarity. For individual concepts, similarity can be efficiently computed based on the taxonomy they belong to [28]. Our similarity measure could in principle be likewise computed directly over the “lifted” taxonomies, but their size renders such a straightforward extension prohibitively inefficient. Instead, our query evaluation algorithm (see Section 4) avoids the materialization of lifted taxonomies based on unique properties that we prove to hold.

**Implementation and Experiments** We have implemented the above components in a prototype system, along with an extensive experimental study over real data from existing crowdsourcing and social network platforms, demonstrating the effectiveness and flexibility of our approach. Interestingly, we show that our declarative framework outperforms existing, non-declarative and task-specific solutions in common user selection tasks. (See Section 5.)

Our framework has been demonstrated in VLDB [5]. The accompanying short paper provides only a high-level description of the system design and its usage scenario.

**Paper Outline** We present our data representation and query language in Section 2. In Section 3 we formally define similarity notions over complex user data and in Section 4 we propose an efficient algorithm to evaluate them, when executing SPARQ-U queries. Our experimental study is described in Section 5. Related work is presented in Section 6 and we conclude in Section 7. Additional proofs and examples, and a comprehensive language manual are available in Appendix A and Appendix B

## 2 Data Repositories and Queries

We next describe the data representation we use, then describe the novel SPARQL extension of SPARQ-U, for user selection queries. The dating website example from the Introduction will serve to illustrate the constructs we define.

### 2.1 Data Representation and Semantics

To account for various applications, data in our framework is represented using an RDF-style model, based on *facts* in the form of entity-relation-entity, e.g., `{Isabella hasProfession Mathematician}`.

**Definition 2.1** (facts and fact-sets). *Let  $\mathcal{E}$  be a set of element names and  $\mathcal{R}$  a set of relation names. A fact over  $(\mathcal{E}, \mathcal{R})$  is defined as  $f \in \mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \Sigma^*)$ , where  $\Sigma^*$  denotes unrestricted literal terms over some alphabet. A fact-set is a set of facts.*

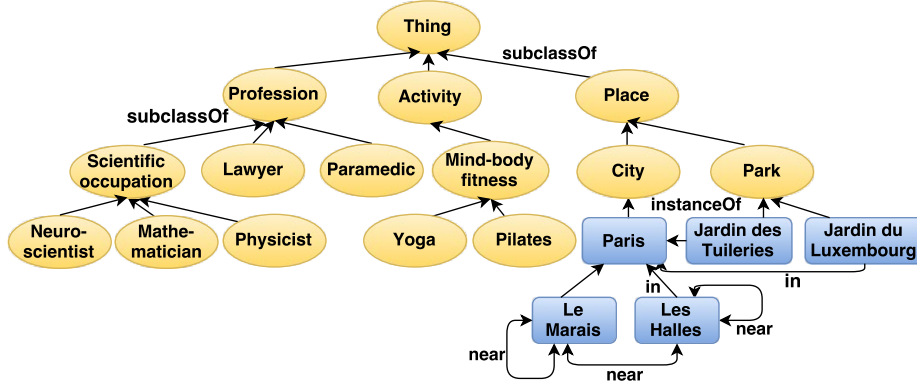


Figure 2: A sample ontology

We denote facts using the RDF notation  $\{e_1 \ r \ e_2\}$  or  $\{e_1 \ r \ l\}$  where  $e_1, e_2 \in \mathcal{E}$ ,  $r \in \mathcal{R}$  and  $l \in \Sigma^*$ .<sup>2</sup> To express complex descriptions that cannot be captured by a single fact, e.g., “Isabella reads books at Jardin des Tuileries”, a fact-set is used (see Table 1). Facts within a fact-set are concatenated using a dot.<sup>3</sup> We use  $\_$  to denote a term (entity or relation) placeholder in facts that use less than three terms. In what follows, we refer to terms in  $\mathcal{E}, \mathcal{R}$ , facts and fact-sets by the collective name *semantic units*. This notation is useful, since some of our following definitions apply to all of these units, and some are defined recursively on the simpler units that compose a compound unit (e.g., a fact is composed of terms).

Our model is used to capture two complementary types of data repositories, as follows.

**Ontology** An ontology is a named fact-set consisting of *general facts* that are not related to a specific user, e.g., facts about geographical locations such as  $\{\text{Le\_Marais in Paris}\}$ . Figure 2 displays a small partial ontology in the form of a graph, where nodes are entities and edges are relations (some edge labels are omitted). Indeed, different large-scale ontologies publish their data in RDF form over the web (e.g. [10]) and different tools enable constructing domain- or application-specific ontologies [31]. To reveal the semantics of knowledge within them, ontologies typically contain a *hierarchical taxonomy* of concepts and instances, e.g., that Paris is a city and that a city is a place. We will leverage such term taxonomies in the sequel to define similarity for *any* type of semantic units.

**User Profiles** Each user is associated with a profile that includes two parts: a basic and an extended profile. The *basic profile* consists of simple properties about users in the form of a fact-set. E.g., Isabella’s profile may include the facts  $\{\text{Isabella livesIn Le\_Marais, Paris. Isabella hasProfession Mathematician}\}$ . The *extended profile* of a user further associates properties (fact-sets) with a numeric score in  $[0, 1]$ , called its *support value*. This score may reflect a rating, habit frequency, likelihood and so on, and is useful in capturing

<sup>2</sup>The curly brackets around facts are added for readability.

<sup>3</sup>Note that the relationships *between* facts are implicit in RDF. Hence, it does not capture information such as fact temporal order. However, using RDF has other advantages that we describe throughout this paper.

| Fact-set   | Support | Prevalence | IC   |
|--|---------|------------|------|
| Isabella like Dogs                                   | 1.0     | 0.8        | 0.08 |
| Isabella read Books                                  | 0.5     | 0.25       | 0.48 |
| Isabella read Books. __ at<br>Jardin_des_Tuileries   | 0.01    | 0.1        | 0.72 |
| Adam like Dogs                                       | 0.75    | 0.8        | 0.08 |
| Adam practice<br>Mind-body_fitness. __ at<br>Park    | 0.28    | 0.45       | 0.3  |
| Adam practice Pilates.<br>__ at Jardin_des_Tuileries | 0.27    | 0.2        | 0.55 |
| Benjamin like Dogs                                   | 0.5     | 0.8        | 0.08 |
| Benjamin read Books                                  | 0.25    | 0.25       | 0.48 |
| Benjamin practice Yoga.<br>__ at Park                | 0.01    | 0.2        | 0.55 |

Table 1: Extended user profiles  
user answers on a scale (such as ratings) or numeric properties derived by the system [2, 4, 12, 25].

**Definition 2.2.** An extended profile database is denoted by  $D = \langle \mathcal{A}, \mathcal{S} \rangle$ , where  $\mathcal{A}$  is a set of fact-sets, and  $\mathcal{S} : \mathcal{A} \rightarrow [0, 1]$  maps every fact-set  $A \in \mathcal{A}$  to a support score  $s_A$ .

Table 1 shows sample extended profiles for 3 users. (Ignore, for now, the “Prevalence” and “IC” columns.) E.g., the support score associated with Isabella’s liking of dogs (first row) may be derived from a rating she provided to the website (normalized to  $[0, 1]$ ).

## 2.2 User Selection Queries

Given the data representation described above, we next overview the SPARQ-U query language constructs, which enable specifying criteria by which users are selected. This language extends SPARQL, the standard RDF query language, to support user selection queries by (i) new soft constraint/similarity constructs, (ii) querying/restricting support values along with their associated fact-sets and (iii) explicitly declaring the knowledge repositories used.<sup>4</sup> The native constructs of SPARQL are mostly used to select relevant repository parts and apply hard constraints. We refer the reader to the SPARQ-U language manual in Appendix B for further details.

We use  $Q_{\text{Isabella}}$  from the Introduction to illustrate the syntax and semantics of SPARQ-U. A query is composed of two main types of clauses: FROM...WHERE and SIMILAR...TO clauses, corresponding, respectively, to hard and soft constraints. Soft constraints form a generic means of capturing useful notions in the context of user selection, such as user similarity, expertise and relevance.

<sup>4</sup>The syntax of SPARQL enables using *prefixes* for querying multiple sources/namespaces. However, since profiles are user-dependent, our query constructs for repository selection admit variables.

The **SELECT** statement (line 1) defines a variable `?u`, which will be assigned names of candidate users and serve as the output.

The **FROM...WHERE** clauses serve to apply SPARQL selection over chosen repositories. In lines 2-3 the query selects places near Le Marais, which are bound to the variable `?x`. In lines 4-5 the basic profile of `?u` is examined to ensure the candidate users live near Le Marais. In lines 6-7 the extended profile is examined to ensure that candidates like dogs, i.e., that they rated dogs highly enough, via a support lower bound.<sup>5</sup>

The following clauses define soft constraints. Lines 8-10 define a similarity constraint on the basic profiles of `?u`. The **RESTRICTED TO** construct (line 9) restricts the similarity evaluation to a subset of the profiles – in this example, to user professions. Line 10 requires that the similarity score for these subsets is  $\geq 0.75$  (similarity scores are defined in the next section). The clause in lines 11-13 considers the desired partner property “practices yoga at some park” and identifies users whose extended profile contains a similar fact-set. The maximal score of such fact-set per user is computed and given the alias name `querySim`. This score is then used in line 14 (an optional **ORDER BY** clause) to rank the five most adequate users, sorted by their `querySim` score. I.e., among the users that match the other parts of the query, those who most frequently practice yoga at some park or have a highly similar habit will be returned. The language enables combining the similarity scores of multiple clauses using standard aggregate functions (see, e.g., Figure 3).

### 3 Similarity Notions

We next propose a novel formulation for semantically-aware similarity computation over complex user data.

Different similarity measures are considered in previous work (e.g., [11, 14, 17, 28]), yet none of them can fully account for the semantically-rich representation of user data in our setting. First, SPARQ-U requires a *semantically aware comparison of fact-sets*, e.g., quantifying the relevance to practicing yoga at a park (Figure 1 lines 11-13) of a user like Adam who practices Pilates, another form of mind-body fitness, at Jardin des Tuileries (Table 1) or of users who play chess and basketball outdoors, practice yoga indoors, etc. Second, our similarity measure must operate under the assumption that *profiles are incomplete*, e.g., it is possible that Adam does practice yoga in the park, but failed to mention it. Third, *support scores* play an important role in similarity evaluation, and e.g., our measure should allow comparing Adam to Benjamin who does practice yoga at a park, but very rarely.

Our contribution in this section is thus to define *formal constraints* on any similarity function that takes into consideration the semantics of fact-sets that describe user data as well as support scores. Then, by extending and adapting standard similarity functions to our data representation, we develop a *generic formulation* that (i) satisfies the constraints, as we prove below, (ii) can be computed via an efficient algorithm, as we show in Section 4, and (iii) works well in real-life scenarios, as we show in Section 5.

---

<sup>5</sup>The choice of support bound may be done similarly to how users have entered their support to begin with: e.g., Isabella can specify, via a web form, that she seeks users that rated their liking of dogs at least 5 out of 10.

### 3.1 Preliminaries

We start by formally defining two auxiliary notions used in our construction: *semantic subsumption*, which will serve to identify the common information between semantic units and to “lift” term taxonomies to facts and fact-sets; and *information content*, which will serve to quantify the common information based on its prevalence.

**Semantic Subsumption [3, 4]** Ontologies often define a taxonomy of concepts, using relations such as `subclassOf`, as in the fact `{Mathematician subclassOf Scientific_Occupation}`. E.g., in Figure 2 there is a taxonomy of concepts (marked by rounded nodes and connected by `subclassOf` relation) with instances (rectangular nodes, connected by `instanceOf` relation). This taxonomy can be interpreted as a subsumption relation between elements: e.g., every occurrence of `Mathematician` subsumes `Scientific_Occupation`.<sup>6</sup> This relation between terms lifts to subsumption between facts and fact-sets that contain these terms [3, 4]: for instance, in every occasion when Adam practices Pilates, he practices (some form of) mind-body fitness. By this lifting, one can identify the common information of e.g. two basic user profiles, by taking a common ancestor of their fact-sets. Formally, given two terms  $t_1, t_2$ , we denote by  $t_1 \leq t_2$  the fact that  $t_1$  is *subsumed* by every occurrence of  $t_2$  (e.g., `Scientific_Occupation`  $\leq$  `Mathematician`). The relation  $\leq$  forms a partial order over the elements of  $\mathcal{E}$  and relations of  $\mathcal{R}$ . Using the definition of [4] we can then “lift”  $\leq$  into subsumption partial orders over *compound* semantic units, namely, facts and fact-sets. First, fact-sets are redefined to be *non-redundant*: any fact-set should not contain two facts  $f_1, f_2$  such that  $f_1 \leq f_2$ . Then, for compound semantic units  $X_1, X_2$  it holds that  $X_1 \leq X_2$  iff for every simpler unit  $P_1$  composing  $X_1$  (facts in a fact-set, terms in a fact) there exists a corresponding unit  $P_2$  in  $X_2$  s.t.  $P_1 \leq P_2$ .

**Example 3.1.** `Scientific_Occupation`  $\leq$  `Mathematician` lifts to `{Isabella hasProfession Scientific_Occupation}`  $\leq$  `{Isabella hasProfession Mathematician}` in the partial order over facts. When lifting to fact-sets, we have that `{Isabella hasProfession Mathematician}`  $\leq$  `{Isabella hasProfession Mathematician, Isabella hasProfession Paramedic.}`.

We conclude with a remark on literals. Literal values are typically not included in the taxonomy, yet one often wants to estimate the similarity using them, e.g., `{Isabella bornAt “1987.04.29”}`. A simple solution (used in our implementation) is to generate a taxonomy of ranges of values, e.g., `{“1980-1989”  $\leq$  “1987”  $\leq$  “1987.04.29”}`. This enables discovering, e.g., that two users are similar by being born in the same year/decade.

**Information Content (IC) [28]** We next provide the formal formulation for IC, which captures the informativeness of semantic units. The *prevalence* of a semantic unit  $p(X)$  is the fraction of users whose profiles *subsume* this unit. E.g., the prevalence of `{User hasProfession Scientific_Occupation}` will be

<sup>6</sup>In terms of concepts, scientific occupations *include* the mathematician profession. However, we are interested in *occurrences* of these concepts within facts, where every occurrence of mathematician implies an occurrence of a (type of) scientific occupation.



measured as fraction of profiles including this fact or more specific facts, such as `{User hasProfession Mathematician}`.

**Definition 3.2** (Information content (IC)). *Let  $X$  be a semantic unit with prevalence  $p(X)$ . The IC of  $X$  is computed as*  

$$\text{ic}(X) := -\log\left(\frac{9p(X)}{10} + 0.1\right)$$

Our definitions of prevalence and IC extend those of [28] to reflect subsumption (in  $p(X)$ ), with additional normalization (inside the log function) to ensure the IC value lies within  $[0, 1]$ . Importantly, this definition is *monotonous* with respect to the partial order on semantic units:  $X \leq Y$  entails that  $p(X) \geq p(Y)$ , hence  $\text{ic}(X) \leq \text{ic}(Y)$ . This property will be useful in the sequel.

Note that this definition may suffer from a *cold start* problem: when the set of users is still small, their data may not correctly reflect the prevalence of semantic units. To overcome this, one can initially use statistics from external sources, e.g., Google  $n$ -grams (<https://books.google.com/ngrams>), until sufficient data is gathered.

### 3.2 Semantic Similarity

Given a subsumption partial order over semantic units (terms, facts or fact-sets) and a function for computing their IC, we impose the following natural constraints that any similarity measure  $\text{sim}(\cdot, \cdot)$  between pairs of semantic units should satisfy.

- (1) **Maximum self-similarity.**  $\forall X, Y \text{ } \text{sim}(X, X) \geq \text{sim}(X, Y)$
- (2) **Symmetry.**  $\forall X, Y \text{ } \text{sim}(X, Y) = \text{sim}(Y, X)$
- (3) **Fixed value range.**  $\forall X, Y \text{ } \text{sim}(X, Y) \in [0, 1]$
- (4) **IC monotonicity.**  $\forall X, Y, Z \text{ } X \leq Y \Rightarrow \text{sim}(X, Z) \leq \text{sim}(Y, Z)$

We call a function  $\text{sim}(\cdot, \cdot)$  that satisfies constraints (1)-(4) a *semantic similarity* function.

Importantly, IC monotonicity requires that as more (specific) semantic information is added, *the similarity score can only increase*. This follows the *open world assumption* mentioned above, namely that user profiles are allowed to be highly incomplete. In this case, a similarity measure should rely on *known* similarities rather than unknown or uncertain dissimilarities. We would not penalize a potential partner of Isabella for specifying a profession, e.g. neuroscientist, she has not, since we cannot outrule that Isabella is also a neuroscientist but omitted to put it in her profile.

We next define a semantic similarity function that generalizes the standard measure of [28], for both facts and fact-sets. Given two semantic units, it computes a third semantic unit that represents the information *common* to the given units, i.e., that is subsumed by both. For instance, `{User hasProfession Scientific_Occupation}` is common to `{Isabella hasProfession Mathematician}` and `{Benjamin hasProfession Neuroscientist}` (assuming that every user is an instance of `User`). Intuitively, the more specific this common information is, the more similar the original two units are (e.g., two neuroscientists are more similar than two scientists). This intuition is quantified by the IC of the common unit, which serves as our similarity score. Since two semantic units may have several semantic units in common, their similarity is determined by the IC of the most specific common unit (following [28]).

**Definition 3.3** (semantic similarity). *Given a pair of semantic units  $X, Y$  which belong to the domain of the partial order  $\leq$ , and an IC function  $\text{ic}(\cdot)$  over this domain, we define the similarity between  $X, Y$  as  $\text{icsim}(X, Y) := \max_{Z|(Z \leq X) \wedge (Z \leq Y)} \text{ic}(Z)$ . We extend this definition to a pair of extended profile databases  $D = \langle \mathcal{A}, \mathcal{S} \rangle$ ,  $D' = \langle \mathcal{A}', \mathcal{S}' \rangle$  by  $\text{icsim}(D, D') := \max_{A \in \mathcal{A}, A' \in \mathcal{A}'} \text{icsim}(A, A')$ .*

The extension of the similarity function to databases computes the maximal IC similarity over all the pairs of fact-sets in these databases. The proposition below holds for  $\text{icsim}(\cdot, \cdot)$  (the proof is provided in Appendix A.2).

**Proposition 3.4.**  *$\text{icsim}(\cdot, \cdot)$  as formulated in Definition 3.3 is a semantic similarity function.*

**Example 3.5.** *Reconsider Table 1, and recall the fact-set  $\bar{A}$  specified in  $Q_{\text{Isabella}}$  in lines 11-13. The common ancestor with maximal IC of  $\bar{A}$  and Adam's (resp. Benjamin's) profile is  $\{\text{User practice Mind-body\_fitness. \_ at Park}\}$  (resp.  $\{\text{User practice Yoga. \_ at Park}\}$ ). The IC values of these common ancestors, 0.3 for Adam and 0.55 for Benjamin, are the semantic similarity scores.*

### 3.3 Support Similarity

Semantic similarity is a powerful means of comparing semantic units in our model, yet it does not account for support scores in extended user profiles. For instance, it does not reflect the fact that Benjamin practices Yoga at Jardin du Luxembourg very rarely, when seeking the user who best matches Isabella's query. As another example, two users are similar with respect to  $\{\text{User like Dogs}\}$  if they like dogs in a similar degree, regardless of whether this degree is high or low. Note, however, that support similarity is *independent* of semantic similarity: two extended profiles databases may contain semantically similar facts (e.g., two users that practice the same mind-body fitness) but assign different support scores, i.e., practice with different frequencies (e.g., every weekend or once a year). Hence, we provide a complementary, support-based similarity formulation, which we will combine with semantic similarity in the next section to obtain our final similarity function.

We formally state the connection between similar support and higher similarity via the following constraint on any similarity function  $\text{sim}(\cdot, \cdot)$ .

- (5) **Support Monotonicity.** For every three extended profile databases  $D = \langle \mathcal{A}, \mathcal{S} \rangle$ ,  $D' = \langle \mathcal{A}, \mathcal{S}' \rangle$ ,  $D'' = \langle \mathcal{A}, \mathcal{S}'' \rangle$ , if for every fact-set  $X \in \mathcal{A}$  it holds that  $|\mathcal{S}(X) - \mathcal{S}'(X)| \leq |\mathcal{S}(X) - \mathcal{S}''(X)|$ , then it should follow that  $\text{sim}(D, D') \geq \text{sim}(D, D'')$ .

We call a function that satisfies constraints (1)-(3) and (5) (i.e., every constraint mentioned above except IC monotonicity) a *support similarity* function. Intuitively, by constraint (5), when the support difference between two extended profiles databases  $D$  and  $D'$  is smaller than the support difference of  $D$  and  $D''$ ,  $D$  is at least as similar to  $D'$  as to  $D''$ . Note that the constraint is defined on databases containing the same fact-sets, where it is clear how to compare their support differences.

We next propose a support similarity function that satisfies the above constraints. First, we provide a formula for comparing the support of two semantic units (using, for uniformity, the same normalization as Def. 3.3). Based on this formula, we compare two extended profiles databases by considering the *intersection* between their fact-sets.

**Definition 3.6** (Support similarity). *Given a semantic unit  $X$  and two support functions  $\mathcal{S}, \mathcal{S}'$  with  $X$  in their domain, we define the similarity between the two support values assigned to  $X$  by these functions as  $\text{supsim}(\mathcal{S}(X), \mathcal{S}'(X)) := -\log\left(\frac{9|\mathcal{S}(X) - \mathcal{S}'(X)|}{10} + 0.1\right)$ . We extend this definition to extended profile databases  $D = \langle \mathcal{A}, \mathcal{S} \rangle$ ,  $D' = \langle \mathcal{A}', \mathcal{S}' \rangle$  by  $\text{supsim}(D, D') := 1$  if  $\sum_{A \in \mathcal{A} \cap \mathcal{A}'} \text{ic}(A) = 0$ , and otherwise*

$$\text{supsim}(D, D') := \frac{\sum_{A \in \mathcal{A} \cap \mathcal{A}'} \text{ic}(A) \cdot \text{supsim}(\mathcal{S}(A), \mathcal{S}'(A))}{\sum_{A \in \mathcal{A} \cap \mathcal{A}'} \text{ic}(A)}$$

*Finally, we extend the definition to enable comparing a fact-set  $A$  and an extended profile database  $D' = \langle \mathcal{A}', \mathcal{S}' \rangle$ , by  $\text{supsim}(A, D') := \text{supsim}(D_A, D')$ , where  $D_A = \langle \bigcup_{A' \leq A} A', \mathcal{S} \rangle$  and  $\mathcal{S}$  is the constant function  $A' \mapsto 1$ .*

Note that the similarity of two extended profile databases is computed as the average similarity of the fact-sets common to the two databases, weighted by IC to give higher weight to more specific fact-sets. For instance, similar support for practicing Yoga will have higher weight than similar support for practicing any kind of mind-body fitness. The last extension of the formula to single fact-sets enables comparing the extended profile of a user with a selected property, as in lines 11-13 of  $Q_{\text{Isabella}}$  from Figure 1.

**Proposition 3.7.**  *$\text{supsim}(\cdot, \cdot)$  as formulated in Def. 3.6 is a support similarity function.*

The proof is deferred to Appendix A.3.

**Example 3.8.** *Consider again Table 1, and the fact-set  $\bar{A}$  specified in  $Q_{\text{Isabella}}$  (lines 11-13). The only fact-set in the intersection of databases is  $\{\text{Adam practice Mind-body\_fitness. \_ at Park}\}$  for Adam (resp.  $\{\text{Benjamin practice Yoga. \_ at Park}\}$  for Benjamin). This fact-set has a score of 0.28 for Adam (resp. 0.01 for Benjamin), and by definition, a score 1 in  $D_{\bar{A}}$ . Then,  $\text{supsim}(0.28, 1) \approx 0.126$  (resp.  $\text{supsim}(0.01, 1) \approx 0.004$ ) and this is also the final support similarity score.*

### 3.4 Combined Similarity

Our combined similarity measure is the *product* of semantic and support similarity, i.e., for two extended profile databases  $D, D'$

$$\text{sim}(D, D') := \text{icsim}(D, D') \cdot \text{supsim}(D, D')$$

The use of product here addresses the case when combined similarity is applied over inputs without support scores or with the constant score 1 (specifically, the database  $D_A$  defined for a fact-set  $A$  in Def. 3.6): in such cases the obtained support similarity score will be 1, and consequently the  $\text{sim}(D, D')$  will coincide with  $\text{icsim}(D, D')$  (unlike, e.g., a weighted average of the semantic and support similarity scores). This is a desired feature of combined similarity since it can then be *uniformly used* in a SPARQ-U over any type of profile or selected part thereof. Moreover, this feature guarantees that our final similarity formula adheres to the constraints of both semantic and support similarity scores defined above, as we next state formally (proof is provided in Appendix A.4):

**Corollary 3.9.** *The combined similarity  $\text{sim}(\cdot, \cdot)$  is a semantic and support similarity function.*

The **SIMILAR** clause in SPARQ-U queries uses this combined similarity measure. A full illustration of the computation for  $Q_{\text{Isabella}}$  is provided in Appendix A.4.

Note that our proposed similarity measure (and its components) can in principle be replaced by any other similarity function,<sup>7</sup> as we demonstrate in our experimental study. Yet, as we explain in Section 6, our proposed measure makes a highly effective use of the available information in the considered setting (as it accounts for fact-set semantics, support scores, open world assumption, etc.) in comparison with previous metrics. This is also reflected in our experimental results, which show our similarity metric outperforms state-of-the-art alternatives.

## 4 Query Evaluation

In SPARQ-U, simple selections (within **RESTRICTED TO** and **WHERE** clauses) coincide with SPARQL selections. Optimizing these parts of the queries can thus be done using standard techniques and existing SPARQL engines. We therefore focus here on evaluating soft constraints. Specifically, we consider the dominant factor of computing semantic similarity, namely, finding the common ancestor of two semantic units that has the highest IC. The tractability of this task is non-trivial, since the number of ancestors may be exponential in the size of the term taxonomy [3, 4]. Our main contribution here is thus a novel efficient algorithm for computing our semantic similarity function *without materializing the fact/fact-set taxonomy*. Next, we leverage properties of our similarity function to develop a further optimization – a dedicated *caching mechanism* for sharing computations across different variable assignments.

**Computing IC Similarity** As will become clear in the sequel, calculating the semantic similarity of two semantic units according to Def. 3.3 is connected to the well-studied problem of finding the lowest common ancestor (LCA) in a general DAG – in this case, the DAG representing the subsumption partial order. For terms, LCAs can be computed using standard modules over the term taxonomy, given as a part of the ontology. As for facts and fact-sets, we wish to avoid a naïve computation that involves materializing their taxonomies (specifically, their size for fact-sets may be exponential in the number of terms [4]). Instead, our optimized algorithm computes the LCA for facts and fact-sets *directly*, in PTIME.

Algorithm 1 outlines the LCA computation for facts and fact-sets, which stands at the core of  $\text{icsim}(\cdot, \cdot)$  computation, assuming a black-box function for term LCA computations. While the structure of the functions is simple, it is the underlying analysis of the components that guarantees correctness and overall low complexity. We review here the main steps of semantic similarity computation, and defer the full algorithm, its formal analysis, and additional proofs to Appendix A.5. Recall that by Def. 3.3 the semantic similarity of two

---

<sup>7</sup>In case that the definition is not IC monotone, the replacement will require some adaptations in the caching mechanism (See Section 5).

```

Function FactLCA( $f, f'$ )
1 | Let  $f = \langle e_1, r, e_2 \rangle$  and  $f' = \langle e'_1, r', e'_2 \rangle$ ;
2 | return  $\text{LCA}(e_1, e'_1) \times \text{LCA}(r, r') \times \text{LCA}(e_2, e'_2)$ ;

Function FactSetLCA( $A, B$ )
3 |  $L \leftarrow \emptyset$ ;
4 | for  $f \in A, f' \in B$  do
5 |    $L \leftarrow L \cup \text{FactLCA}(f, f')$ ;
6 | for  $f = \langle e_1, r, e_2 \rangle, f' = \langle e'_1, r', e'_2 \rangle \in L$  do
7 |   if  $e_1 \leq e'_1 \wedge r \leq r' \wedge e_2 \leq e'_2$  then  $L \leftarrow L - \{f\}$ ;
8 | return  $L$ ;

```

**Algorithm 1:** icsim( $\cdot, \cdot$ ) computation

semantic units is the maximal IC of a common ancestor of these units. The following observation allows us to focus only on a small subset of the common ancestors.

**Observation 4.1.** *By IC monotonicity, it is sufficient to consider only the maximal (most specific) common ancestors of two semantic units  $X, Y$  in the computation of icsim( $X, Y$ ).*

We can thus compute IC similarity according to Definition 3.3 by finding every maximal common ancestor  $Z$  of  $X$  and  $Y$ , computing  $\text{ic}(Z)$  and returning the maximum. We now focus on LCA computation as means of finding these maximal common ancestors. Our Algorithm 1 assumes an efficient implementation of  $\text{LCA}(t, t')$ , namely, searching the LCAs of two terms  $t, t'$  in a general DAG representation of the term taxonomy (line 2).

The first function in Algorithm 1,  $\text{FactLCA}(f, f')$ , computes the LCA(s) of *two facts* using the LCA computation for terms. By the lifting of partial order from terms to facts, every LCA of two facts can be shown to consist of three terms that are LCAs for each of their respective terms, i.e.,  $\text{FactLCA}(e_1 \ r_1 \ e'_1, e_2 \ r_2 \ e'_2)$  is of the form  $e \ r \ e'$  where  $e \in \text{LCA}(e_1, e_2)$ ,  $r \in \text{LCA}(r_1, r_2)$  and  $e' \in \text{LCA}(e'_1, e'_2)$ . Thus,  $\text{FactLCA}(f, f')$  returns the set of facts that is the *Cartesian product* of the LCAs of each pair of corresponding terms in these facts (line 2 of  $\text{FactLCA}(f, f')$ ).

The second function,  $\text{FactSetLCA}(A, B)$  computes the LCA of two fact-sets. Importantly, by properties of the partial order over fact-sets, we can prove that unlike the cases of terms, facts, or general DAGs, the following lemma holds.

**Lemma 4.2.** *There exists exactly one LCA for every pair of fact-sets  $A, B$  in the partial order.*

Moreover, we can characterize this fact-set as the set of non-redundant facts that are ancestors of some pair of facts,  $f \in A$  and  $f' \in B$ . To compute this set, the function  $\text{FactSetLCA}(A, B)$  uses  $\text{FactLCA}(f, f')$  for each such pair of facts, takes the union  $L$  of the LCAs (lines 4-5) and then removes redundant facts (that are implied by other facts in this set, lines 6-7. Refer to Appendix A.5 for correctness).

The final icsim( $X, Y$ ) is computed according to the type of input (terms, facts, fact-sets, extended profile databases) according to Definition 3.3. The following proposition summarizes the complexity bounds we have achieved for IC similarity computation, according to the algorithm described above and known

complexity bounds for the base case, i.e., LCA computation for materialized taxonomies [8], and assuming that IC computation of a given semantic unit is done in  $O(1)$  (See Appendix A.5 for a full proof). In what follows,  $w[\Psi]$  denotes the *width of the term taxonomy*  $\Psi$ , namely, the maximum size of a set of incomparable terms.

**Proposition 4.3.** *After PTIME preprocessing of the partial order over terms, the complexity of computing the IC similarity of  $X, Y$  is:*

- $O(|\text{LCA}(X, Y)|)$ , if  $X, Y$  are terms or facts.
- $O(|X||Y|w[\Psi]^3 \log(|X||Y|w[\Psi]))$ , if  $X, Y$  are fact-sets (i.e., polynomial in fact-set sizes and term taxonomy width).
- $O(|X||Y|)$  times the complexity of computing the IC similarity of an fact-set pair, if  $X, Y$  are extended profile databases.

**Caching mechanism** We have so far discussed the similarity score of two *concrete* semantic units or extended profiles, i.e., for a given assignment of values to the SPARQ-U query variables. In some cases, we can improve the overall performance of query evaluation by leveraging the computations of one assignment to save computations for another. Using IC monotonicity and the work in [4] which defines a partial order over assignments such that more specific assignments yield equal or more specific facts and fact-sets, this enables us to make inferences across assignments, as follows.

**Observation 4.4.** *For SIMILAR ... TO clauses over fact-sets (with implicit support 1):*

- (1) *If some variable assignment  $\varphi$  is below the similarity threshold for one such clause, every more general assignment  $\psi$  will be below the threshold for the same clause.*
- (2) *If some variable assignment  $\varphi$  is above the similarity threshold for every such clause, every more specific assignment  $\psi$  will be above the threshold for all such clauses.*

Following this observation, by caching the results of maximally-specific (resp., -general) assignments that are below (resp., above) the threshold one can avoid many redundant similarity computations. This observation can then be used as follows. The results for each considered assignment will be stored in a cache. Whenever a new assignment is encountered, it will first be checked whether it falls within one of the items in the observation above. If it falls within the first item, the assignment can be discarded. If it falls within the second item, there is no need to compute the similarity for this assignment for clauses that involve fact-sets and whose results are not used in ordering the query output, since the assignment is guaranteed to satisfy these constraints. Note that we ignore clauses that are affected by support values, since their similarity is not IC monotone w.r.t. variable assignments. Our experimental study shows the importance of this optimization in practice.

## 5 Experimental Study

We developed a prototype engine for parsing and evaluating SPARQ-U queries. In particular, this engine uses preprocessing of the term taxonomy to speed

up LCA computation which is at the core of semantic similarity computation (see Section 4). A further speed-up is achieved by distributing over multiple cores (when available) the computation of similarity between different user pairs, which is virtually independent. Finally, the engine employs a dedicated caching mechanism to compactly store previous results and avoid unnecessary computations, based on Observation 4.4. The prototype is implemented in Java, using the Apache Jena library (<https://jena.apache.org/>) for handling RDF data and for a SPARQL engine. Using this engine we conducted an experimental study, designed to assess the flexibility, accuracy and efficiency of our approach for real user data, as described next.

## 5.1 Goals

Since user selection using a semantically-rich representation like ours has not been studied before, no standard benchmark over which one could test the full capabilities of SPARQ-U was available. We have thus constructed two benchmark datasets using real-world data from two sources: *Stack Overflow (SO)* (<http://stackoverflow.com/>), a large Q&A platform for computer programming and AMiner [33], an academic social network containing author profiles, collaboration and publication data. These datasets provide natural scenarios for user selection which we have captured via SPARQ-U queries, as detailed below.

Our first goal was to *evaluate the result quality* that our approach achieves, i.e., the adequacy of selected users. We compared the results of SPARQ-U to common alternative strategies from machine learning, graph processing, and collaborative filtering. To examine the contribution of components within our solution we implemented restricted variants of SPARQ-U that do not include some components, or use alternative similarity measures. Finally, we conducted a user study aiming to assess the relevance of results by real users. Our second goal was to *test the system’s running time and its scalability*. We examined the execution time of SPARQ-U queries for varying input sizes. As a complementary case study, we examined the *flexibility* of SPARQ-U and its ability to express specific user selection needs in typical contexts, including recommendations, predictions and expert finding.

Overall, the results of our experiments show that SPARQ-U is expressive and scalable enough to be practically used for selecting users in various real-life contexts. While our solution is declarative and generic, it still achieves high quality results, even when compared to task-dedicated, non-declarative alternatives. Interestingly, this is true even though we have only composed the queries manually and intuitively (as a real user would have done) rather than carefully tailored each query to outperform the competitors.

## 5.2 Experimental Setup

We next describe how data was extracted and converted into our representation, then explain the different baseline algorithms tested.

To enable using standard RDF tools over our data repositories, our system generates additional RDF data that explicitly capture information about facts and fact-sets, e.g., `{FactSet456 hasSupport 0.01}`. The system then converts the selection clauses within a given SPARQ-U query (`From...WHERE` and `RESTRICTED TO` clauses) into SPARQL selection queries over the resulting RDF

repository. All experiments were executed on a Linux server with 24 cores, a 2.1GHz CPU and 94GB memory.

**Stack Overflow dataset** In this popular Q&A platform, user questions are associated with a set of tags that reflect their topics. Among its answers, a question may have one designated answer chosen as the most accurate one. Each user has a profile with properties such as name, registration date, reputation score, etc.

We collected over 900K questions in 300 popular topics (tags), asked by over 175K users, using the SO API (<http://api.stackexchange.com>). We then gathered their personal profiles, and collected more than 2.3M previous answers of those users, to assemble a coherent subset. The retrieved data was naturally mapped into our model: we have constructed basic user profiles from the extracted profiles. The extended profile of each user was constructed from her tags to which she contributed with a support value that reflects the portion of her contribution to that tag among all tags. The ontology was constructed from DBpedia [10]. We aligned each tag to a matching concept and used the relevant part of DBpedia taxonomy. The resulting database consists of 20M entities.

**AMiner dataset** We extracted the data of 1.7M Computer Science researchers, and considered their publications between 2005 – 2015 (over 2M papers). The basic profiles of the authors consist of their affiliation, h-index, fields of interest, and some publication data (e.g., title and year). The extended profile of each author records her co-authorships (with a support value reflecting the fraction of her publications with each co-author), and publication venues (with support value reflecting the fraction of her publications in each venue). The domain ontology was built in the same way as described above, aligning AMiner’s terms (affiliations, fields of interest, etc.) to concepts from DBpedia, and constructing the corresponding taxonomy. The resulting database consists of 16M entities.

**Alternative algorithms** To examine the adequacy of the users selected by our queries we considered several competing baselines. As mentioned above, our goal was to examine the contribution of each of the individual components of our solution as well as the performance of our framework as a whole.

**1.SPARQL.** User selection is done by a standard SPARQL query, that is, *only hard constraints are employed*. This allows evaluating the need of similarity-based soft constraints in user selection.

**2.No Support.** User selection is done by the same SPARQ-U queries, but the support values are ignored. This allows for evaluating the need of considering support values in user selection and the effect of our support-similarity measure.

**3.No Fact-sets.** User selection is done by the same SPARQ-U queries, but facts are considered individually and co-occurrence of facts is ignored. This allows evaluating the need for our rich data model and its corresponding semantic-similarity measure.

**4.SimRank.** We use SimRank [18], a structural similarity measure which is commonly used in social networks for link prediction problems, as an alternative similarity measure.

**5.SVM.** Users are selected using SVM, a common machine learning method for such a task. We used the scikit-learn (<http://scikit-learn.org/stable/modules/svm.html>) implementation. For each scenario separately, we employed



```

1 SELECT ?u
2 FROM basic-profile(?u) WHERE
3   {?u creationDate ?d. Filter (?d < 19.6.2015) }
4 SIMILAR basic-profile(?u) TO basic-profile(Basil_Bourque)
5   WITH SIMILARITY AS profSim > 0
6 SIMILAR extended-profile(?u) TO
7   {?u answeredOn Java. ?u answeredOn I/O }
8   WITH SIMILARITY AS topicSim > 0.2
9 ORDER BY AVG(profSim, topicSim) LIMIT 30

```

Figure 3: User selection for Stack Overflow question

an SVM classifier, where each user represents a class, and the algorithm computes the probabilities of each class to be chosen. This baseline is used to evaluate the ability of SPARQ-U to capture common user selection scenarios, which can alternatively be specified as machine learning problems.

Baselines 4 and 5 are used as alternative implementations to our similarity measure, that is, hard constraints are still used for initial data filtering. Additional similarity measures e.g. the cosine similarity and Jaccard measure used in collaborative filtering systems were examined, but consistently yielded inferior results, thus omitted throughout this section.

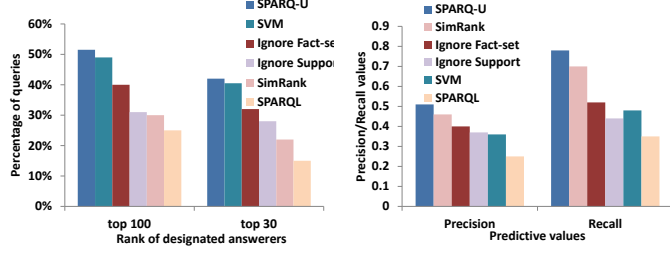
### 5.3 Experiments

We next present our experiments over the two datasets, along with an analysis of the results and performance. While SPARQ-U is able to express a wide range of user selection scenarios (see Section 5.4), to examine the adequacy of selected users we focused on specific cases for which there exists a *ground truth* that enables assessing the results quality. For each scenario we describe the used queries and compare our system’s output to the alternatives and ground truth. We then examine the scalability of our system.

**Stack Overflow experiment** Given a question in SO, we consider the task of finding adequate users to answer it. We considered several alternative SPARQ-U queries (including, e.g., the retrieval of top-ranked users for each of the question’s tags), but eventually chose a query that retrieves users whose past answers are relevant (similar) to the given question, and whose basic profile is similar to the user asking the question. The first requirement ensures the selected user has enough expertise in the relevant topics. The second, perhaps more interesting requirement, ensures that the user asking the question and the one answering it are interested in questions with similar difficulty level. It seems that a “peer” is more likely to provide a good answer than someone with significantly higher or lower capabilities. We constructed a template SPARQ-U query that captures this user selection approach and instantiated it for different pairs of questions and questioners, as exemplified next.

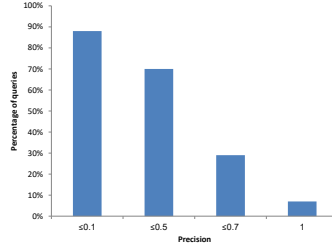
**Example 5.1.** Consider the question posted on 19.6.2015 by the user Basil Bourque: “How do I read from a file in Java?”, tagged by ‘Java’ and ‘I/O’. The corresponding SPARQ-U query, choosing users to answer this question, is depicted in Figure 3.

We randomly selected over 500 questions posed in 2015-2016 and examined the corresponding SPARQ-U queries, instantiated from our template for choosing



(a) Stack Overflow: Prediction in top-k.

(b) AMiner: Precision & recall.



(c) AMiner: Precision in user study.

Figure 4: Quality assessment

adequate responders to each question. Each of these queries was executed over data collected *before* the corresponding question was posted.

Recall that each question in SO has one designated answer chosen as its most accurate answer. To evaluate the correctness and quality of the results, we examined how the user that gave the true designated answer was ranked according to our query and the alternative baselines, and counted the percentage of queries where this user was among the top 30 or 100 users selected by our query. In the model we used for the SVM baseline each user represented a class and the questions are mapped into those classes using the question’s and the questioner’s features as an input. In the *SPARQL* baseline query, the users are ranked by their reputation score.

The results are depicted in Figure 4a. In all cases our solution performs better than the competitors. In particular, for 42% of the examined questions our *SPARQ-U* query ranked the designated answerer among the top 30 recommended users and for 51% of the questions, among the top 100 users. This is extremely positive given the task difficulty: there are many users that can potentially answer a single question, and SO system is currently implemented as a “passive” system, where users are not actively approached to answer questions, so many adequate users may not even see a given question. This indicates that *SPARQ-U* can be used to add a more active “answer request” module to SO.

Furthermore, observe that the three restricted variants of *SPARQ-U* (*SPARQL* only, no support, no fact-set) achieve inferior results, demonstrating the importance of soft constraints and of analyzing support scores and fact-sets. *SimRank* also shows significantly inferior results here, since unlike our similarity measure, it does not exploit semantic information. *SVM* is closer to *SPARQ-U* in terms of

quality, but unlike the customizable SPARQ-U the SVM classifier was specifically trained for this scenario and cannot be trivially adapted to other queries.

We carefully examined the cases where the designated answerers were ranked low or not selected at all by our query. We detected that this was typically due to missing data: in many cases, either the user posing the question or the designated answerer were new users with little or none past activities. As mentioned in Section 3, such a “cold start” problem may be addressed by actively asking users for missing information or by using information from external sources. We leave this for future research.

**AMiner experiment** We focus on a scenario that involves identifying potential collaborations for a given researcher – a typical *link prediction* problem in social networks. Our query selects, for a given researcher, the top-30 users in her “friends of friends” community (i.e., among those users that had previously collaborated with at least one of her co-authors) with the most similar profiles. Similarity here is measured w.r.t. both the basic profile (i.e., fields of interest, h-index, etc.) and the extended profile (i.e., past collaborations and conference contributions).

To examine the adequacy of the selected users, we conducted two experiments, one comparing the results to the actual collaboration data available in AMiner data, and the other via a user study.

For the first experiment we split the dataset into two parts: one contains the publications up to 2013, and the second contains the publications in later years. We focused on authors who have at least 10 publications/collaborations and whose profile contains at least one field of interest (to avoid cold start issues). We have randomly selected 150 such authors, and for each author we executed the corresponding SPARQ-U query on the  $\leq 2013$  data and compared the query results with actual collaborations in the following years (2014-2017). The precision and recall of these results (namely, how many of our predictions were fulfilled and how many of the true collaborations were predicted, respectively), averaged over the 150 users, are plotted in Figure 4b. The SPARQL baseline in Figure 4b orders researchers by their h-index. For the SVM classifier, each user represents a class and is modeled by a vector consisting of her fields of interest, past collaborations and publications.

For over 90% of the examined authors the SPARQ-U query has identified at least one true future collaborator, and for over 85% of them  $\geq 3$  of the predictions were correct. Furthermore, for over 60% of the authors,  $\geq 50\%$  of their true future collaborators were predicted by SPARQ-U.

Here again, our solution outperformed all of the alternatives, and in particular SimRank, although the latter is highly suitable to the context of collaboration networks.

To complete the picture, we conducted a user study in which we selected 27 known researchers (averaging h-index of 35) from a variety of countries. For each researcher, we executed a SPARQ-U query selecting 10 adequate collaborators, in the same manner as in the previous experiment, excluding their past co-authors,<sup>8</sup> and asked them to estimate how many of the selected users may be potential future collaborators (the precision). Figure 4c depicts the results. Note that in this study, we could not have estimated the recall, since it would require

---

<sup>8</sup>Assuming researchers would not object to collaborate with past co-authors.

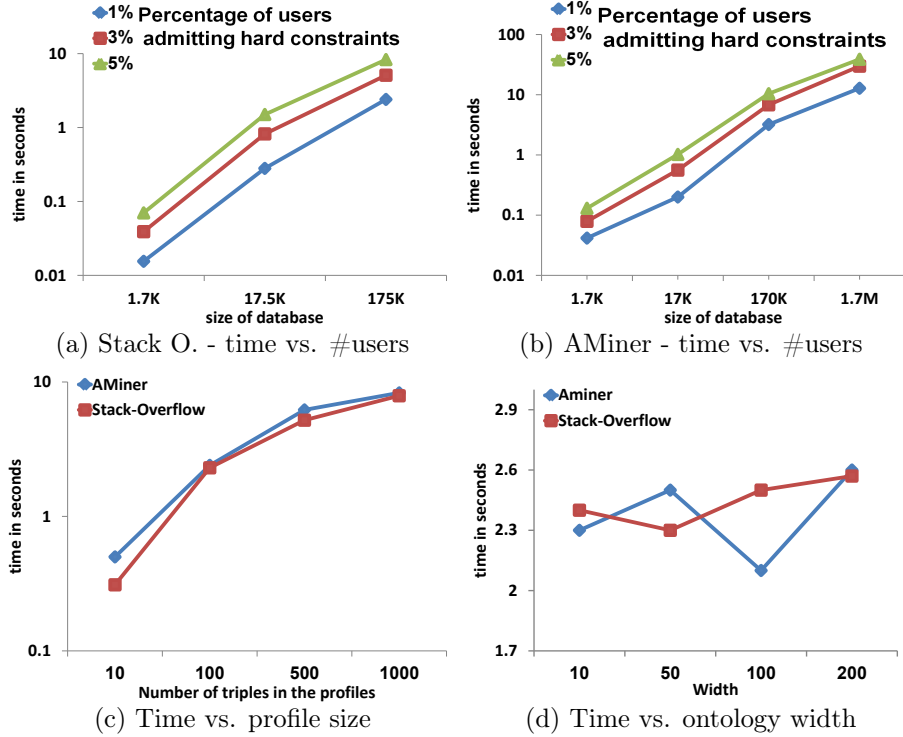


Figure 5: Execution times.

researchers to list all of their potential future co-authors, which is naturally unknown at a given moment.

In a real-life usage of the system, users may further personalize the query, adding individual filtering conditions on location, language, etc. Yet, even our simple, generic query obtained some interesting results. Indeed, 89% of the participants found at least one recommendation relevant, and 85% of the participants found at least 3 recommendations out of 10 relevant.

We further analyzed the false positives for this study. In particular, only in 3 out of the 27 cases, the authors found the recommendation irrelevant. These cases appear to be a consequence of incomplete data (e.g., missing facts in both the basic profile and past collaborations) or unclear data (e.g., authors with basic profiles that do not reflect their fields of interest). As mentioned previously, such problems may be addressed by enriching the data, e.g., using tools for building profiles from social networks [12].

**Performance evaluation** Our quality assessment queries (previously described), had an average execution time of less than 3 seconds. To further analyze the performance and scalability of our system, we modified these queries by e.g. relaxing the hard constraints to control the number of users admitting them, to examine the effect of different parameters on the execution time: (1) the number of users in the database, and in particular the percentage of users admitting the hard constraints; (2) the size of the examined profiles and (3) the shape of the ontology in terms of width and depth. For comparison, in the previous set of experiments, the average percentage of users that admitted the hard

constraint was  $\sim 1\%$  in the SO experiment (and lower in Aminer), each profile contained  $\sim 100$  RDF triples on average and the average width and depth of the ontology part used in each query was  $\sim 50$  and  $\sim 7$ , respectively. In the following experiments, when varying the value of some parameter we use these average numbers as a default value for the other, fixed parameters.

We do not report here the running times of the SimRank and SVM baselines, since they were significantly inferior. Unlike SPARQ-U, which is dynamically evaluated over the queried profile data, these alternatives require a preprocessing phase per query (sampling random walks for SimRank computation [34], and training a model for the SVM classifier). Re-executing this phase alone required above one minute per query in our experimental setting, and overall could not compare with the scalability of SPARQ-U.

Figures 5(a) and 5(b) depict the average running time for SO and AMiner queries, respectively, w.r.t. the database size and for different ratios of users admitting the hard constraints. Observe the sublinear (roughly square root) increase in execution time, which depends on the number of users admitting the hard constraints. This sublinearity demonstrates the effectiveness of our caching mechanism, which as explained above, leverages previous results to save computations and whose effect is increased as the number of users, hence the overlap between their profiles, increases. In comparison, the execution time we observed without caching (omitted from the graph) increases linearly with the number of users, and was at least 5 times slower for any of the settings we have tested.

Next, to examine the effect of the profile size on query execution times, we altered the hard constraints of the SPARQ-U queries, to include each time a fixed profile size between 10 and 1000 RDF triples. Figure 5(c) depicts the effect of the profile size on execution time, again exhibiting a sublinear dependency achieved by our caching mechanism.

Last, we examined the effect of ontology *shape* on query execution times. Since in each query only a restricted subgraph is effectively used in similarity computations (which form the computational bottleneck), we obtained the desired ontology shape by dividing the queries into sub-buckets according to the shape of the ontology part they use in these computations. Figure 5 (d) shows the execution time as a function of the subgraph width. Interestingly, the results demonstrate that our polynomial bound from Prop. 4.3 is far from being met in practice, and the preprocessing of term taxonomy (to support LCA computation) effectively guarantees that execution times are oblivious to the taxonomy width. We omitted the analog plots for varying subgraphs depth since they demonstrate similar trends.

Overall, the performance experiments show that SPARQ-U as a framework is indeed efficient in evaluating user selection queries, and in particular online computation of our similarity measure. Our dedicated caching mechanism makes the solution highly scalable.

## 5.4 Flexibility: A Case Study

We now look into the flexibility of SPARQ-U. Since to our knowledge no standard benchmark of user selection queries exists, we collected prominent user selection scenarios in real-life applications, exemplified them by queries over the SO And Aminer datasets and analyzed the use of SPARQ-U constructs in each scenario.

| Use Case                             | Example   | Required SPARQ-U features   |
|--------------------------------------|---|---|
| 1. Expert finding                    | Select users highly linked to the key term "Database" and who frequently publish in a top DB conference. See Figure 6.  | Combined similarity, order by selected values.                          |
| 2. Link prediction                   | Select co-authors of co-authors of a given researcher ranked by their (extended) profile similarity, as her potential future collaborators. See AMiner experiment in Section 5.3. | Hard constraints, combined similarity, order by similarity.             |
| 3. Profile-based user recommendation | Select users with similar (extended) profile, see, e.g., Figure 1.  | Combined similarity, order by similarity, restricted parts of profiles. |
| 4. Context-based user recommendation | Select users that are highly relevant to the tags "Java" and "I/O" based on their answers since 2015. See Figure 3.   | Hard constraints, combined similarity, order by similarity.             |
| 5. Community Extension               | Select users who frequently interact with as many members of a given community.   | Combined similarity, order by similarity.                               |

Table 2: Use cases of SPARQ-U

Table 2 lists several high-level categories of common user selection scenarios. Each scenario is provided with a concrete example demonstrating a typical “information need”, and the features required for its formulation in SPARQ-U. In particular, note that all of the example scenarios entail the use of soft constraints, indicating their importance in SPARQ-U.

1. **Expert finding.** As described in Section 6, much effort is made to determine a user’s level of expertise and thereby identify the experts in some topic. For example, consider the selection of database experts from the AMiner dataset. An expert database researcher may be defined e.g., as a person who is (a) specialized in databases (or related topics), (b) frequently publishes at a top-tier conferences in the field and (c) has a high h-index. Figure 6 depict the corresponding SPARQ-U query, that retrieves researchers having at least one subject similar to “Databases” and published in some top DB conference, ordered by their h-index. To verify that this query indeed retrieves database experts, we presented the top-10 results to 10 database researchers. For comparison, we attached a list of top-10 database experts generated by the AMiner platform, omitting the origin of either list (AMiner/SPARQ-U). 10 out of 10 participants stated that the SPARQ-U ranking is more accurate.<sup>9</sup>

2. **Link prediction.** This problem is mostly known from the study of links that are likely to form in a (social) network (See Section 6). The SPARQ-U query we have used on the AMiner dataset in Section 5.3 is one example such query, which uses hard constraints to select collaborators of collaborators, then employs combined similarity to find the researchers most similar to a given researcher in terms of their (basic and extended) profiles.

3. **Profile-based user recommendation** and 4. **Context-based user recommendation** aim to provide a given user with recommendations of other *relevant users* from some repository<sup>10</sup>. E.g. the query in Figure 1 selects appropriate matches to a user on an online dating platform, and in Figure 3 the query selects users

<sup>9</sup>We do not provide the lists here to avoid any offence to the involved researchers. However, the lists can be easily recomputed from the AMiner data.

<sup>10</sup>Do not confuse with similar scenarios in recommending *items* to a given user.

```

1 SELECT ?u
2 FROM basic-profile(?u) WHERE
3     { ?u h-index ?h. }
4 SIMILAR basic-profile(?u) TO {?u keyTerm Databases .}
5     WITH SIMILARITY > 0.5
6 SIMILAR extended-profile(?u) TO
7     {?u published_at Top_DB_Conference. }
8     WITH SIMILARITY > 0.2
9 ORDER BY DESC(?h)

```

Figure 6: SPARQ-U query selecting database experts

adequate for answering a professional question on a given topic in Stack Overflow. The former uses *similarity to a target profile* (i.e. the profile of Isabella’s “ideal” match in Figure 1, and the profile of Basil Bourque, the user who posted the question in Figure 3). The latter query also takes the *context* into account, by ranking higher candidates that frequently answered other questions in similar topics to “Java” and “I/O” (Lines 6-8 in Figure 3).

5. **Community Extension.** Given a Community (i.e. a set of highly interlinked users) in a social network, SPARQ-U may be used to detect additional (new or potential) community members. For example, given a seed collaboration community in AMiner comprising of three current members (‘X’, ‘Y’, and ‘Z’), a SPARQ-U query that detect more members would compare the extend-profile of a user ?u to:

{?u collaboratedWith X. ?u collaboratedWith Y. ?u collaboratedWith Z.} and order the retrieved users by their similarity score.

To conclude, we demonstrated that queries, which can be easily and intuitively expressed using SPARQ-U, capture various common scenarios of selecting users by their profiles.

## 6 Related Work

The selection of candidate users from a repository has been studied in various contexts and for different needs.

*Recommender systems* leverage similarities between users (and items) in order to recommend users (or items) to end users. These recommendations are based on similar item ratings and/or similar user profiles, possibly exploiting semantic knowledge (e.g., [11, 23]). In the context of social networks, friend recommendations are computed by techniques of *link prediction*, which rely on analysis of the network structure (e.g., [15, 18, 1]). However, none of these works considers a similarity measure that can fully account for the rich semantics of our model, since our lifting of IC similarity to general facts and fact-sets is novel.

The field of *expert finding* is concerned with selecting users (individuals or a team) by an assessment of their level/ areas of expertise (e.g., [9, 13, 19, 26]). Such an assessment may then assist in assigning multiple tasks to users with relevant expertise [22, 30, 7, 35]. A related problem is *quality control* in crowdsourcing, where users are selected based on (an estimation of) their performance in previous tasks (e.g., [2, 13, 6]). Some contributions of these works are orthogonal to ours: we can store derived expertise or quality scores as the support of fact-sets in extended user profiles, and use these along with our similarity measure to compute e.g. the *relevance of a user to a new context*, provided by a query. In turn, our ranking of users can then be used in task assignment. Our solution further differs from these studies by providing a generic and declarative

framework for user selection rather than targeting a specific context or facet of the problem.

Other query languages have been developed for user selection, mostly in the context of *social network analysis* (e.g., [21, 29]). Another language that focuses on querying user preferences is presented in [16], and [20] introduced user-defined path searching for mining relationships between users. We note that SPARQ-U only uses graph/RDF selection constructs that are available in SPARQL, so some of the dedicated constructs of the aforementioned languages (e.g., group identification constructs) may be used to further enrich our language. However, these other languages do not incorporate semantic knowledge or soft constraints as SPARQ-U does, features which our experiments indicate to be vital in capturing various scenarios such as expert finding.

Finally, our work is related to other problems that involve semantic similarity analysis, such as *entity matching* in ontology alignment (e.g., [14, 17, 27, 32]) and *similarity search*, which is used in evaluating imprecise or relaxed queries (e.g., [36]). While some of these methods resemble ours in measuring the similarity of RDF subgraphs and using taxonomical data, their goal is different: they aim to identify different representations of the *same data* (entity or query answer). Thus, in contrast to our work, quantifying the similarity of distinct entities (e.g., French mathematicians and Italian physicists) is not targeted. Moreover, our data representation of user profiles, involving fact-sets and support is not accounted for by the measures of these works. Specifically, comparing large user profiles requires an efficient similarity computation, hence, e.g., the NP-hard similarity metric of [36] is not suitable for our setting.

## 7 CONCLUSION AND FUTURE WORK

This work presents SPARQ-U, a declarative framework that allows specification of customized user selection criteria. Its SPARQL-based query language has embedded constructs for capturing the properties and similarity of (relevant parts of) user profiles, via a semantically-aware similarity measure. Dedicated novel algorithm and optimizations allow for efficient query processing. Our experimental study on real-life data indicate the effectiveness and usefulness of our approach.

Interesting directions for future work include *diversification*, *clustering* and *classification* constructs, which may be built on top of our semantic notions of similarity. The cold start problem of an initially small profile repository can be further considered, possibly by actively asking users for missing information or by using external sources. Finally, it would be interesting to adapt our novel similarity measure to other applications such as entity matching.

## References

- [1] C. G. Akcora, B. Carminati, and E. Ferrari. User similarities on social networks. *Social Network Analysis and Mining*, 2013.
- [2] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. Motahari-Nezhad, E. Bertino, and S. Dustdar. Quality control in crowdsourcing systems. *IEEE*, 2013.



- [3] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2014.
- [4] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *SIGMOD*, 2014.
- [5] Y. Amsterdamer, T. Milo, A. Somech, and B. Youngmann. December: A declarative tool for crowd member selection. *VLDB*, 2016.
- [6] J. Anderton, M. Bashir, V. Pavlu, and J. A. Aslam. An analysis of crowd workers mistakes for specific and complex relevance assessment task. In *CIKM*, 2013.
- [7] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Information Processing & Management*, 2009.
- [8] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin. Finding least common ancestors in directed acyclic graphs. In *SODA*, 2001.
- [9] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: expert finding in social networks. In *EDBT*, 2013.
- [10] About DBpedia, 2017. <http://wiki.dbpedia.org/about>.
- [11] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. In *I-SEMANTICS*, 2012.
- [12] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell me what you like, and i’ll tell you what to do. In *WWW*, 2013.
- [13] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. iCrowd: An adaptive crowdsourcing framework. In *SIGMOD*, 2015.
- [14] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *DKE*, 2008.
- [15] Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. *JCDL*, 2005.
- [16] M. Jacob, B. Kimelfeld, and J. Stoyanovich. A system for management and analysis of preference data. *PVLDB*, 2014.
- [17] Y. Jean-Mary and M. Kabuka. Asmov: Ontology alignment with semantic validation. In *SWDB-ODBS*, 2007.
- [18] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *SIGKDD*, 2002.
- [19] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *SIGKDD*, 2009.
- [20] J. Liang, D. Ajwani, P. K. Nicholson, A. Sala, and S. Parthasarathy. What links alice and bob?: Matching and ranking semantic patterns in heterogeneous networks. 2016.

- [21] M. Martín, C. Gutierrez, and P. Wood. SNQL: A social networks query and transformation language. In *AMW*, 2011.
- [22] P. Mavridis, D. Gross-Amblard, and Z. Miklós. Skill-aware task assignment in crowdsourcing applications. In *Int. Sym. Web Algo.*, 2015.
- [23] B. Mobasher, X. Jin, and Y. Zhou. Semantically enhanced collaborative filtering on the web. In *EMWF*, 2003.
- [24] J. Provan and M. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM*, 1983.
- [25] J. S. Pudipeddi, L. Akoglu, and H. Tong. User churn in focused question answering sites: characterizations and prediction. In *WWW*, 2014.
- [26] H. Rahman, S. Thirumuruganathan, S. B. Roy, S. Amer-Yahia, and G. Das. Worker skill estimation in team-based tasks. *PVLDB*, 2015.
- [27] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 2011.
- [28] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, 1995.
- [29] R. Ronen and O. Shmueli. SoQL: A language for querying and creating data in social networks. *ICDE*, 2009.
- [30] S. B. Roy, I. Lykourantzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das. Task assignment optimization in knowledge-intensive crowdsourcing. *PVLDB*, 2015.
- [31] C. Sarasua, E. Simperl, and N. F. Noy. Crowdmap: Crowdsourcing ontology alignment with microtasks. In *ISWC*, 2012.
- [32] F. M. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 2011.
- [33] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, 2008.
- [34] B. Tian and X. Xiao. Sling: A near-optimal index structure for simrank. 2016.
- [35] J. Yao, B. Cui, Q. Han, C. Zhang, and Y. Zhou. Modeling user expertise in folksonomies by fusing multi-type features. In *DASFAA*, pages 53–67. Springer, 2011.
- [36] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao. Semantic SPARQL similarity search over RDF knowledge graphs. *PVLDB*, 2016.

## A Additional Proofs and Examples

### A.1 Alternative IC Definition

In Section 3.1 we provide a definition of IC which is based on a *statistical approach*. Previous work [23] also mentions a *structural approach* which is based on the structure of the partial order of semantic units. More concretely, the IC of a semantic unit is computed as a function of the *number of descendants* it has in the subsumption partial order. More specific concepts are expected to have less sub-concepts and instances. Unfortunately, in the case of fact-sets we can show that a structural similarity function is intractable to compute, since in particular, computing the number of descendants of a fact-set is intractable.

**Proposition A.1.** *Given a subsumption partial order  $\leq$  over  $\mathcal{E}, \mathcal{R}$  and a fact-set  $A$  over  $\mathcal{E}, \mathcal{R}$ , counting the descendants of  $A$  in the lifted partial order  $\leq$  over fact-sets is  $\#P$ -hard.*

*Proof.* Counting the *antichains* in a partial order is known to be a  $\#P$ -hard problem [24]. Intuitively, the definition of an antichain is related to our requirement of non-redundancy in fact-sets. Formally, the definition of the lifted partial order over fact-sets leads to a bijective mapping between fact-sets and antichains of the subsumption partial order of facts. Thus, to show a reduction from the antichain counting problem to our problem, it is only left to prove that every partial order corresponds to some partial order over facts. Assume that the partial order input to the antichain counting problem is  $\leq_i$  over the domain of elements  $\mathcal{D}$ . Then define  $\mathcal{E} = \{\_\}$  and  $\mathcal{R} = \mathcal{D}$ , and take the partial order over  $\mathcal{R}$  to be  $\leq_i$ . Then there is a bijective mapping between every  $d \in \mathcal{D}$  and the single fact that contains it,  $\{\_\ d \_\}$ . Denote this single fact by  $f(d)$ . Then  $f(d) \leq f(d')$  iff  $d \leq_i d'$  (since the other parts of the triple are identical). This means that the partial order over facts that we obtained is isomorphic to the input partial order, and thus its fact-sets correspond to the antichains of the input partial order. If we can count the descendants of the root of the antichain partial order (the most specific fact-set) in PTIME, then we can also count the antichains in the original partial order.  $\square$

### A.2 Semantic Similarity

In Section 3.2 we listed desiderata for a *semantic similarity function*. We next show that  $\text{icsim}(\cdot, \cdot)$  (see Definition 3.3), fulfills these desiderata and is therefore a semantic similarity function.

(*Prop. 3.4*). We prove that desiderata (1)-(4) described in Section 3.2 hold for  $\text{icsim}(\cdot, \cdot)$ .

- (1) *Maximum self-similarity.* By definition,  $\text{icsim}(X, X) = \text{ic}(X)$ . By the observation above that IC is monotonous w.r.t. the subsumption partial order, since the common ancestor of  $X$  and some other unit  $Y$  is in particular an ancestor of  $X$ , whose IC can only be smaller or equal.
- (2) *Symmetry.* This follows directly from the symmetry of the formula for the common ancestor.
- (3) *Fixed value range.* This follows directly from the range of values of IC.

- (4) *IC monotonicity.* Let  $X' \leq X, Z$  be a common ancestor of  $X$  and  $Z$  with maximal IC value. If  $X \leq Y$  then  $X'$  is also a common ancestor of  $Y$  and  $Z$ , hence  $\text{icsim}(X, Z) \leq \text{icsim}(X, Y)$ .  $\square$

### A.3 Support Similarity

We prove that  $\text{supsim}(\cdot, \cdot)$  (articulated in Definition 3.6) is a *support similarity function* as the desiderata described in Section 3.3 hold.

(Prop. 3.7). We prove that desiderata (1)-(3),(5) described in Sections 3.2, 3.3 hold for  $\text{supsim}(\cdot, \cdot)$ .

- (1) *Maximum self-similarity.* For a fact-set  $A$ ,  $\text{supsim}(\mathcal{S}(A))$ ,  $\mathcal{S}(A) = 1$  by definition. For a database  $D$  (or  $D_A$  derived from a fact-set  $A$ ), either all of its fact-sets have IC 0, in which case its similarity is defined to be 1, and otherwise it is a weighted average of 1-s, which also equals 1.
- (2) *Symmetry.* This follows directly from the symmetry of the formula.
- (3) *Fixed value range.* This is guaranteed by the normalization of  $\text{supsim}(\mathcal{S}(X), \mathcal{S}'(X))$  to be between 0 and 1. A weighted average of values in  $[0, 1]$  is also in  $[0, 1]$ .
- (5) *Support monotonicity.*  $\text{supsim}(\mathcal{S}(X), \mathcal{S}'(X))$  is monotonous with respect to  $|\mathcal{S}(X) - \mathcal{S}'(X)|$ . Then if these values for a pair of databases  $D, D'$  are greater or equal than the values of  $D, D''$ , the same holds for the weighted average of these values with the same weights.  $\square$

### A.4 Combined Similarity

In Section 3.4, we state that the combined similarity  $\text{sim}(\cdot, \cdot)$ , as defined there, is a semantic and support similarity function. The proof for desiderata (1)-(3) follows from that of Prop. 3.4 and 3.7. Desideratum (5) also holds because it refers to extended profile databases whose common fact-sets are the same, and hence also their support similarity. The support monotonicity of definition 3.6 is proven in Prop. 3.7. Finally, desideratum (4) is only defined for fact-sets, and since  $\text{sim}(\cdot, \cdot)$  coincides with  $\text{icsim}(\cdot, \cdot)$  for fact-sets, the desideratum holds by Prop. 3.4. However, note that in extended profile databases such monotonicity may not hold: if we replace some fact by another fact with higher IC, we will obtain higher semantic similarity, but the support similarity may decrease (depending on the support difference of this fact-set), leading to a lower combined similarity. This is, however, a necessary trade-off when balancing semantic and support similarity. To complete the picture we illustrate the computation of the results of  $Q_{\text{Isabella}}$ .

**Example A.2.** For both Adam and Benjamin the **WHERE** clauses holds (i.e., both of them live near Le Marais and like dogs with support values above the threshold). The first **SIMILAR** clause is computed by finding the information common to Isabella and each of the others, based on their profession as listed in their basic profiles. Assume that Benjamin is a Neuro-scientist and Adam is a Physicist, thus the professions of all three are subsumed by the singleton fact-set  $\{\text{User hasProfession Scientific-Occupation}\}$ . Further assuming that IC of this fact-set exceeds the desired threshold, then both users comply with the first **SIMILAR** clause.

Finally, we compute the combined similarity of both users to the fact-set in the second **SIMILAR** clause. In this case two factors are affecting the final similarity scores: (a) Benjamin's habit is semantically more similar to the fact-set in the query than Adam's, yet (b) Adam's support for his most relevant habit is significantly higher. Using the scores computed in the previous examples, we get that the support similarity of Adam is 0.126 and the semantic similarity is 0.3, therefore the combined similarity is  $0.126 \cdot 0.3 = 0.0378$ . As for Benjamin, the support similarity is 0.004, and the semantic similarity is 0.55, yielding a combined similarity of 0.0022. Intuitively, Adam is ranked higher than Benjamin in the query result, since his habit is not too far from Isabella's specified preference, and his support for this habit is much higher. In this case, a Pilates enthusiast may be a better match than someone who practices Yoga rarely (otherwise, Isabella could have specified this as a hard constraint).

## A.5 Computing IC Similarity

We next prove the correctness of Algorithm 1 for computing the IC similarity of two semantic units, according to Def. 3.3, and analyze the complexity of the algorithm, proving Prop. 4.3.

**Lemma A.3.** *The complexity of computing the IC similarity of two terms  $t, t'$  is  $\Theta(|\text{LCA}(t, t')|)$ , after PTIME preprocessing of the partial order  $\leq$  over terms.*

*Proof.* By the monotonicity of the IC predicate, it is sufficient (and necessary) to consider the LCAs of  $t$  and  $t'$  when seeking the maximal IC of any common ancestor of  $t, t'$ . For each term in  $\text{LCA}(t, t')$  we can compute its IC and return the maximal IC value. Note that this is also necessary since the LCAs are incomparable by  $\leq$  and thus their ICs are also incomparable. It is left to consider the complexity of finding the LCAs. By the result of [8], after PTIME processing of the partial order/DAG one can find the LCA of every two elements in it in constant time.  $\square$

The basic complexity result for computing terms' IC can then be used to compute the similarity of two facts, again, by first computing their LCAs.

**Lemma A.4.** *The complexity of computing the IC similarity of two facts  $f, f'$  is  $\Theta(|\text{LCA}(f, f')|)$ , after PTIME preprocessing of the partial order  $\leq$  over terms.*

*Proof.* We note that in the partial order over facts, the LCAs of  $f, f'$  is the cartesian product of the LCAs of their terms. E.g., if  $f = \{e_1 \ r \ e_2\}$  and  $f' = \{e'_1 \ r' \ e'_2\}$  then the set of triplets  $\text{LCA}(e_1, e'_1) \times \text{LCA}(r, r') \times \text{LCA}(e_2, e'_2)$  forms the LCAs of  $f, f'$ : first, every fact in this set  $f''$  precedes both  $f$  and  $f'$  since each of  $f''$ -s terms precedes the corresponding term of  $f$  or  $f'$ . Second, if there exists  $\bar{f}$  such that  $f'' < \bar{f} < f$ , then it cannot be the case that  $\bar{f} \leq f'$  (and vice versa for  $f'' < \bar{f} < f'$ ). By  $f'' < \bar{f}$  at least one term of  $\bar{f}$  is a descendant of a term in  $f''$ , but since all the terms of  $f''$  are LCAs of the terms of  $f, f'$  this means that the replaced term in  $\bar{f}$  cannot be an ancestor of both of the corresponding terms in  $f, f'$ . Last, all of the facts in the aforementioned set are incomparable by  $\leq$ , since they are composed of incomparable terms (LCAs).

Then, by Lemma A.3 we can compute in constant time each of the LCAs of the terms composing  $f, f'$ . Computing their cartesian product is linear in the number of LCAs.  $\square$

Finally, we show the complexity of computing the similarity of two fact-sets. For that, we first prove Lemma 4.2, which states that for any pair of fact-sets there exists exactly one LCA. The proof further characterizes which fact-set is the LCA (as explained, intuitively, in Section 4), thus enabling its efficient computation.

(Lemma 4.2). Prop. 3.6 of [3] highlights the bijective correspondence between non-redundant sets, such as the fact-sets in our setting, and *order ideals*, namely, each fact-set  $A$  can be uniquely characterized by the set of facts  $\{f \mid \exists f' \in A, f' \leq f\}$  (the facts of  $A$  and all of their ancestors). The LCA of two fact-sets  $A, B$  is then the unique fact-set  $C$  that corresponds to the *intersection of their order ideals*. The intersection can be expressed as  $\text{OD}_C = \{f \mid \exists f' \in A, f'' \in B, f' \leq f \wedge f'' \leq f\}$ , and it is an order ideal since by definition, if fact  $f$  is in  $\text{OD}_C$ , then so are also all of its ancestors.  $\square$

In what follows,  $w[\Psi]$  denotes the *width of the term taxonomy*, i.e., the maximum size of a set of incomparable terms.

**Lemma A.5.** *The complexity of computing the similarity of two fact-sets  $A, B$  is  $O(|A| |B| w[\Psi]^3 \log(|A| |B| w[\Psi]))$ , after PTIME preprocessing of the partial order  $\leq$  over terms.*

*Proof.* Assume we have preformed preprocessing on the partial order  $\leq$  over terms (as in Lemma A.3). To compute the single LCA (by Lemma 4.2 above) of  $A, B$  we can first compute the LCA of every pair of facts  $f \in A, f' \in B$ . We claim that the set of all such facts  $L$  corresponds to the LCA of  $A, B$ , but may contain redundant facts: clearly,  $L$  is contained in  $\text{OD}_C$ , the intersection of order ideals corresponding to  $A, B$ . This is since every fact in it is an ancestor of both a fact in  $A$  and a fact in  $B$ . Next, we need to show that  $\text{LCA}(A, B) \subseteq L$ . Assume by contradiction that this is not the case. Then there exists a fact  $f \in \text{LCA}(A, B)$  that does not belong to  $L$ .  $f$  cannot be an ancestor of a fact in  $L$ , because then  $f$  is subsumed by other fact(s) in its order ideal and would not appear in a fact-set. This means that  $f$  is an ancestor of some fact in  $A$  and some fact in  $B$  which is maximal by  $\leq$  – therefore it is an LCA and belongs to  $L$ , in contradiction with our initial assumption.

It is left to remove redundant facts from  $L$  to obtain  $\text{LCA}(A, B)$ . This can be done, e.g., by sorting them topologically. The comparison between two facts can be done by checking whether all the terms of one fact subsume the terms of the other. Since the number of terms is fixed, and if we assume that subsumption check can be done in constant time, every comparison can be done in constant time.

Let us now analyze the complexity of this algorithm. Computing the LCA of every pair of facts (computing  $L$ ) can be done in time  $O(|A| |B| \text{maxLCA})$ , where  $\text{maxLCA}$  is the maximum number of LCA facts for a pair of facts from  $A$  and  $B$ . This number can be bounded from above by  $w[\Psi]^3$ : the width of the term taxonomy,  $w[\Psi]$  is the maximal number of LCAs two terms may have. To compute the LCA facts we at most need to compute all the combinations of the LCAs of the 3 terms, hence  $w[\Psi]^3$  is the maximal number of LCAs per any two facts. Now, it holds that  $|L| \leq |A| |B| \text{maxLCA}$ . To remove the redundant facts by ordering them, we need at most  $O(|L| \log |L|)$  constant-time comparisons between facts. It is then left to compute the IC of the resulting fact-set. Summing these numbers gives the complexity result stated above.  $\square$

## B SPARQ-U Language Manual

### Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                                   | <b>2</b> |
| <b>2</b> | <b>Basic RDF and SPARQL selection</b>                 | <b>4</b> |
| 2.1      | RDF Ontologies . . . . .                              | 4        |
| 2.2      | Basic SPARQL Query . . . . .                          | 5        |
| <b>3</b> | <b>User Selection with SPARQ-U</b>                    | <b>5</b> |
| 3.1      | Applying hard constraints . . . . .                   | 6        |
| 3.1.1    | Special Issues . . . . .                              | 7        |
| 3.2      | Applying soft constraints . . . . .                   | 8        |
| 3.2.1    | Using Fact-set . . . . .                              | 9        |
| 3.2.2    | Order by similarity scores . . . . .                  | 10       |
| 3.2.3    | Considering only relevant parts of profiles . . . . . | 10       |
| 3.2.4    | Special Issues . . . . .                              | 11       |

# 1 Introduction

SPARQ-U is a SPARQL<sup>1</sup> extension that allows for high-level, flexible user selection. In particular, SPARQ-U enhances SPARQL with embedded constructs for expressing soft constraints on user profiles (or relevant parts thereof) and for capturing user similarity.

Consider the following scenario: Isabella is seeking for a potential partner in an online dating service. In particular, she is interested in users who live in the vicinity of her neighborhood, like spicy food, resemble her in terms of hobbies, and who frequently practice yoga on weekends. Some of these requirements may be specified as hard constraints, e.g., the neighborhood, and others, specified as soft constraints, may still be acceptable if partially satisfied, e.g., a user who practices Pilates on weekends, which, similarly to yoga, is a kind of mind-body fitness.

Formulating such query in SPARQ-U requires the following RDF knowledge repositories:

- *An ontology* that captures general knowledge about the world (e.g., that yoga and Pilates are forms of mind-body fitness)
- *user profiles repository* that contains basic facts about individual users (e.g., name, location, hobbies, etc.), and information regarding their habits and preferences (e.g. that Isabella likes spicy food *a lot* or reads books in the park *frequently*)

A Multitude of general-purpose ontologies are available online, e.g. YAGO<sup>2</sup>, and DBPedia<sup>3</sup>. Moreover, domain-specific ontology (e.g., for music, geography, etc.) may be found online<sup>4</sup> or constructed via designated tools<sup>5</sup>

As for the user data repository, we distinguish between the *basic user profile*, that records available properties of the user, often provided by the users themselves such as their place of birth, age, and education, and the *extended user profile* containing additional information e.g. user preferences and habits. Each property in the extended profile also has an associated score in  $[0, 1]$ , called *support level*. The support level is used to model degrees and frequencies. For example, the support value for the fact “Adam practices yoga twice a week” is  $2/7 \approx 0.286$ . Such profile data may either be provided by the user or collected/derived by the hosting system.

Figure 1 displays a small partial ontology in the form of a graph, where nodes are entities and edges are relations (some edge labels are omitted). Table 1 and Table 2 depict (respectively) basic user profiles and extended profiles repositories.

We next explain the syntax and semantics of different components of SPARQ-U, accompanied by examples.

---

<sup>1</sup>SPARQL: <http://www.w3.org/TR/rdf-sparql-query>

<sup>2</sup>YAGO2s. Max Planck Institute for Informatics. <http://www.mpi-inf.mpg.de/yago-naga/yago/>

<sup>3</sup>DBPedia. <http://dbpedia.org>

<sup>4</sup>SemanticWeb. <http://http://semanticweb.org/>

<sup>5</sup>E.g. Vitro. <https://github.com/vivo-project/Vitro>



| Fact-sets |               |                    |
|-----------|---------------|--------------------|
| Isabella  | livesIn       | Le_Marais, Paris.  |
| Isabella  | hasGender     | Female.            |
| Isabella  | hasHobby      | Photography.       |
| Isabella  | hasHobby      | Dancing.           |
| Isabella  | graduatedFrom | NYU.               |
| Adam      | livesIn       | Les_Halles, Paris. |
| Adam      | hasGender     | Male.              |
| Adam      | hasHobby      | Photography.       |
| Adam      | hasHobby      | Sculpting          |
| Benjamin  | livesIn       | Le_Marais, Paris.  |
| Benjamin  | hasGender     | Male.              |
| Benjamin  | hasHobby      | Drawing            |
| Benjamin  | hasHobby      | Basketball         |

Table 1: Basic user profiles

| Fact-set  | Support |
|---|---------|
| Isabella like Spicy_Food                          | 1.0     |
| Isabella read Books                               | 0.5     |
| Isabella read Books.<br>_ at Jardin_des_Tuileries | 0.01    |
| Adam like Spicy_Food                              | 0.75    |
| Adam practice Mind-body_fitness.<br>_ on Weekends | 0.28    |
| Adam practice Pilates.<br>_ on Weekends           | 0.27    |
| Benjamin like Spicy_Food                          | 0.5     |
| Benjamin read Books                               | 0.25    |
| Benjamin practice Yoga.<br>_ on Weekends          | 0.01    |

Table 2: Extended user profiles

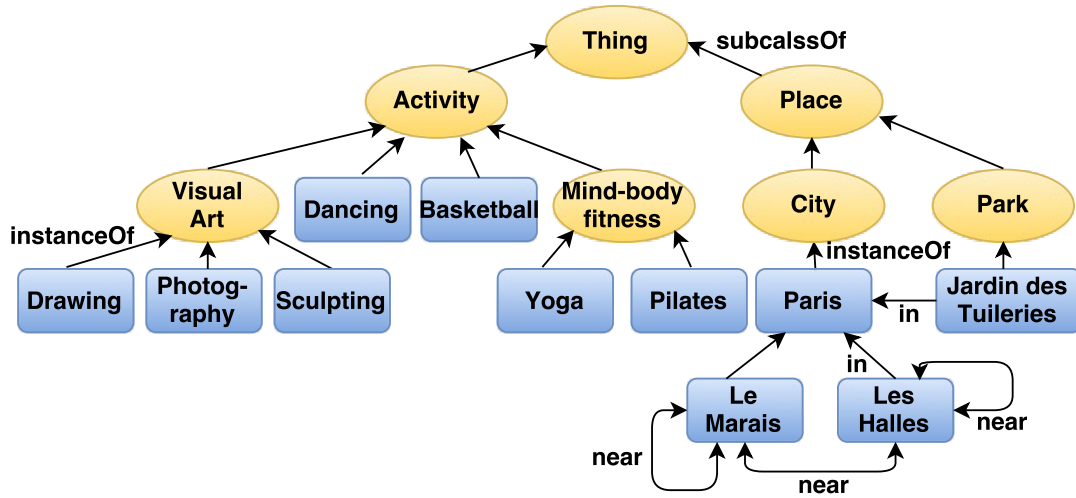


Figure 1: Sample ontology

## 2 Basic RDF and SPARQL selection

We start by explaining the basic syntax of SPARQL over RDF. This syntax can be used to express, e.g., “find all parks in Paris that are near Le Marais”, over an ontology which contains the relevant data. Note that in order to write SPARQL queries one must be familiar with the relevant ontology contents: for instance “Visual arts” can be represented by, e.g., `<http://dbpedia.org/resource/Visual_arts>`.

**Tip:** Publicly available ontologies usually provide documentation about their structure.<sup>6</sup> In addition, there exist user-friendly SPARQL editors with auto-completion features, which can assist the user in formulating the query.<sup>7</sup>

### 2.1 RDF Ontologies

RDF is a declarative representation language, which is commonly used in ontologies. In particular, the OWL<sup>8</sup> language on top of RDF is dedicated to ontology specification. In this context, we greatly simplify matters, and treat an RDF as a set of facts, as follows.

A *fact* is a triplet of the form `elem1 rel elem2`, which states that the relationship `rel` holds between two elements. For example, `Le Marais (is) in Paris` may be used to convey the fact that Le Marais district is in Paris, France.

An RDF ontology can be viewed as a set of such facts, that conveys general knowledge (encyclopedic knowledge), on different domains.

<sup>6</sup>For example, see the DBPedia access guide: [wiki.dbpedia.org/OnlineAccess](http://wiki.dbpedia.org/OnlineAccess)

<sup>7</sup>E.g., YASGUI. <http://laurensrietveld.nl/yasgui>

<sup>8</sup>OWL Web ontology language. <http://www.w3.org/TR/owl-ref/>

```

1 SELECT ?x WHERE
2   {?x instanceOf Visual_Art}

```

Figure 2: Example SPARQL query.

```

1 SELECT ?x WHERE
2   {?x instanceOf Park. ?x in Paris.}

```

Figure 3: Example SPARQL query.

## 2.2 Basic SPARQL Query

The very basic form of a SPARQL selection query, is specified as a fact or set of facts (in which case they are concatenated by dots), where some of the elements or relations are replaced by variables. For example, consider the simple SPARQL query presented in Figure 2. This query uses the ontology relation `instanceOf`, which is used to denote a particular instance of a category, e.g., `Sculpting instanceOf Visual Art`. `?x` is a variable, and this selection query returns all that can be assigned to `?x`, such that the resulting fact exists in the ontology. In other words, it returns all types of visual arts in the ontology. In the example ontology in Figure 1, the values `$x` can get are as follows.

| <code>?x</code> |
|-----------------|
| Drawing         |
| Photography     |
| Sculpting       |

Consider the slightly more complicated example presented in Figure 3. This query uses, in addition to `instanceOf`, the relation `in` which indicates, for two geographic locations, that one resides in the other. This query finds all the assignments to `?x` such that `?x` is a park and `?x` is in Paris. For example, according to the ontology in Figure 1, the only match would be:

| <code>?x</code>      |
|----------------------|
| Jardin des Tuileries |

## 3 User Selection with SPARQ-U

Given the data representation described above, we next define the SPARQ-U query language, which enables specifying the criteria by which users are selected. This language supports user selection queries by (i) new soft constraint/similarity constructs, (ii) querying/restricting support values along with their associated fact-sets and (iii) explicitly declaring the knowledge repositories used. The native constructs of SPARQL in this language are mostly used to select relevant repository parts and apply hard constraints.

We use Isabella’s example from the Introduction to illustrate the syntax and semantics of SPARQ-U. Recall that Isabella is interested in users who live in the vicinity of her neighborhood, like spicy food, resemble her in terms of hobbies,

```

1 SELECT ?u
2 FROM ontology WHERE
3     {?x near Le_Marais,Paris.}
4 FROM basic-profile(?u) WHERE
5     {?u livesIn ?x.}
6 FROM extended-profile(?u) WHERE
7     {?u like Spicy_Food.} WITH SUPPORT >= 0.5

```

Figure 4: Example query using hard constraint on different repositories.

and who frequently practice Yoga on weekends. Some of these requirements may be specified as hard constraints, e.g., the neighborhood, and others, specified as soft constraints, may still be acceptable if partially satisfied, e.g., a user who practices Pilates on weekends, which, similarly to Yoga, is a kind of mind-body fitness. We next explain how these constraints can be expressed using SPARQ-U.

### 3.1 Applying hard constraints

Here we show the basic “hard constraints” of SPARQ-U, i.e how to query from different repositories and filter out irrelevant users. In our example, Isabella is interested only in users who live in the vicinity of her neighborhood and like spicy food. That is, we wish to select only users who admit these constraints.

Figure 4 depicts a simple SPARQ-U query that demonstrate how these hard constraints can be expressed.

Reserved keywords are given in capital letters. First, the **SELECT** clause defines what type of output the query should return. In this case, the **SELECT** statement (line 1) defines a variable `?u`, which will be assigned names of candidate users and serve as the query output. The **FROM**. . .**WHERE** clauses serve to apply SPARQL selection over chosen repositories, as described in the previous section. In lines 2-3 the query selects places near **Le Marais**, which are bound to the variable `?x`. In lines 4-5 the *basic profile* of `?u` is examined to ensure the considered users live near **Le Marais**, and in lines 6-7 the *extended profile* is examined to ensure they like spicy food. The clause further defines a lower bound on the support, to ensure selected users rated spicy food highly.

The support in our setting may reflect a rating, habit frequency, likelihood and so on, and is useful in capturing user facts on a scale (such as ratings). In this example, the support threshold (line 7) represents the minimum preference of spicy food given by the users to the hosting system (the dating website), normalized to  $[0, 1]$ . To obtain more results one should lower the threshold, and to obtain fewer results one should increase the threshold.

As in SPARQL, any other binary operator may be used to filter using the support values. For example, the following example selects only users that do not like spicy food

**Evaluation** To conclude, we briefly explain how the example query above is evaluated. For the example query above, let us start with the assignment of the variable `?x`. According to the ontology, the possible assignments to `?x` are either

```

1 SELECT ?u
2 FROM basic-profile(?u) WHERE
3     {?u like Spicy_Food.} WITH SUPPORT <= 0.25

```

Figure 5: Example usage of support values.

```

1 SELECT ?u
2 FROM extended-profile(?u) WHERE
3     {?u hashobby ?x. ?x subClassOf* Activity.}

```

Figure 6: Example path query.

Les Halles or Le Marais. Next, according to the users profile as depicted in Table 1, the possible assignments to the variable ?u are Isabella, Adam and Benjamin, as all three lives in either Les Halles or Le Marais. Finally, when applying the last hard constraint on the extended profile (lines 6-7), according to Table 2, we obtain that all three candidates admit this constraint, so their all serve as the query output.

| ?x         | ?u       |
|------------|----------|
| Les Halles | Adam     |
| Le Marais  | Benjamin |
| Le Marais  | Isabella |

### 3.1.1 Special Issues

- **Basic-profile support values.** Note that the support values are only associated to the extended profiles and not to the user basic profiles. Clearly, by assigning 1 to the basic profile’s fact-set we can merge it with the extended profile. The distinction of profile parts is done to ease the presentation and for logical separation between.
- **Path queries.** Among the various features of SPARQL selection queries, we note in particular the following feature, which is extremely useful when working with ontologies. In our example dating query, we may be interested in users who has some kind of a hobby, not necessarily a specific one. I.e., we would be interested in considering instances of all the sub-classes of Activity. This is expressed by the example query in Figure 6. This query selects only users who have in their basic profile some fact that indicates the user have some kind of a hobby (e.g., dancing, drawing, Yoga and etc.).
- **Fact set.** We use `_` to denote a term (entity or relation) placeholder, in facts that use less than three terms. For example, in Table 2 we use the placeholder to describe the fact the Isabella read books at jardin des Tuileries.

```

1 SELECT ?u
2 SIMILAR basic-profile(?u) TO basic-profile(Isabella)
3     WITH SIMILARITY >= 0.75
4 SIMILAR extended-profile(?u) TO extended-profile(Isabella)
5     WITH SIMILARITY >= 0.5

```

Figure 7: Basic use of similarity statement.

### 3.2 Applying soft constraints

Soft constraints form a generic means of capturing useful notions in the context of user selection, such as user similarity, expertise and relevance.

Recall that Isabella is interested in users who resemble her in terms of hobbies, and who frequently practice Yoga on weekends.

For a start, let us assume that Isabella is interested in users who similar to her. In this case, we can either compare the users basic-profile to Isabella’s basic profile, their extended-profiles against Isabella’s, or both. Figure 7 demonstrate how SPARQ-U embedded constructs can capture properties and similarity of user profiles, through which adequate users for a given context can be effectively identified. In particular, the similarity statements define soft constraints. Lines 2-3 define a similarity constraint on the basic profiles of ?u and lines 4-5 define a similarity constraint on the extended profiles of ?u. Similar to the support values, the obtained similarity scores may be used to filter out irrelevant candidates. E.g, in line 3, we ensure that only users who their basic-profile is highly similar (similarity score  $\geq 0.75$ ) to Isabella’s basic profile will be consider.

In general, the similarity statements have the following form:

SIMILAR \$X(?u) TO \$Y WITH SIMILARITY op val

where \$X is either extended-profile or basic-profile,  $op \in \{>, <, >=, <=, =, <>\}$  and val is the obtained similarity score  $\in [0, 1]$ . As for \$Y, we can either compare between two users profiles, as in Figure 7, or we can compare between one user profile and a desire property.

In our example, Isabella is interested in users who practice Yoga, although according to her profile, she does not practice Yoga. In this case, we wish to compare the users profile against some desire property (practicing Yoga). Figure 8 demonstrate how this can be done using SPARQ-U. More concretely, the extended profiles of the users is compare with the property “practicing Yoga”, where the support value of the desire property is 1. That is, users who practice Yoga more frequently (e.g., twice a week) would get higher similarity scores than one who rarely practice Yoga (e.g., once a month).

Note that in this case, practicing Yoga is defined as a soft constraint, i.e., users who do not practicing Yoga, yet practicing some other mind-body fitness (e.g., Pilates), not necessarily will be filtered out. That is, even-though according to Adam’s profile he do not practice Yoga yet practices Pilates, as Yoga and Pilates are both mind-body fitness activities, his obtained similarity score is  $> 0$ .

```

1 SELECT ?u
2 SIMILAR extended-profile(?u) TO {?u practice Yoga.}
3 WITH SIMILARITY >= 0.5

```

Figure 8: Example similarity statement (similarity assessment between user profile and desire property).

```

1 SELECT ?u
2 SIMILAR extended-profile(?u) TO {?u practice Yoga. _ on Weekends. }
3 WITH SIMILARITY >= 0.5

```

Figure 9: Example similarity statement (similarity assessment between user profile and desire property).

To conclude, one may use similarity statements in SPARQ-U queries to assess the similarity between users (basic or extended) profiles, or between candidate users and a desire set of properties. In the latter case, whenever extended profile of a user is compared against a set of properties, the support value of the desire set of properties is set to 1. Furthermore, the obtained similarity scores may be used as a minimal threshold to filter out irrelevant users.

### 3.2.1 Using Fact-set

Recall that the profile of a user consists of different facts and fact-sets. For example, according to Adam's extended profile he practices Yoga on weekends. This fact-set has its own support score that may be different from the one assigned to the fact that Adam practices Yoga. I.e., he might practice Yoga once a weekend yet twice a week.

Now, assume Isabella is searching for a mate who practices Yoga on weekends *a lot*. In this case, the corresponding SPARQ-U query is presented in Figure 9. Note that in this case the desire set of properties is practicing Yoga on weekends, and the support value of this set is set to 1.

Importantly, all facts that appear on the same similarity statement would get one support value and would be treated as a single fact set. In cases where we wish to consider each fact-set separately, one can use multiple similarity statements. For example, in Figure 10, we use two similarity statements, each evaluating the similarity of the extended profiles of the candidate users to a different desire property.

```

1 SELECT ?u
2 SIMILAR extended-profile(?u) TO {?u practice Yoga. _ on Weekends. }
3 WITH SIMILARITY >= 0.5
4 SIMILAR extended-profile(?u) TO {?u read Books. }
5 WITH SIMILARITY >= 0.2

```

Figure 10: Example multiple similarity statement.

```

1 SELECT ?u
2 FROM basic-profile(?u) WHERE
3     {?u like Spicy_Food.} WITH SUPPORT <= 0.25
4 SIMILAR extended-profile(?u) TO {?u practice Yoga.}
5     WITH SIMILARITY AS querySim >= 0.5
6 ORDER BY querySim

```

Figure 11: Order query output using similarity scores.

```

1 SELECT ?u
2 SIMILAR basic-profile(?u) TO basic-profile(Isabella)
3     WITH SIMILARITY AS basicSim >= 0.5
4 SIMILAR extended-profile(?u) TO {?u practice Yoga.}
5     WITH SIMILARITY AS extendedSim >= 0.5
6 ORDER BY AVG(basicSim, extendedSim)

```

Figure 12: Aggregating similarity scores.

### 3.2.2 Order by similarity scores

Assume Isabella wishes to sort all relevant candidates by their similarity scores. As in SPARQL queries, numeric values may be used to sort the query output.

Consider Figure 11. In this simple example we used the obtained similarity scores in order to sort the query output. The maximal score of the fact-set `?u practice Yoga.` per user is computed and given the alias name `querySim` (line 5). This score is then used in line 6 (an optional `ORDER BY` clause) to rank the adequate users, sorted by their `querySim` score. I.e., among the users that match the other parts of the query, those who most frequently practice Yoga or have a highly similar habit will be returned.

SPARQ-U enables combining the similarity scores of multiple clauses using standard aggregate functions. For example, consider Figure 12. In this simple example, an average of the two obtained similarity scores is used to order the relevant users.

### 3.2.3 Considering only relevant parts of profiles

Recall that Isabelle is interested in users who resemble her only in terms of hobbies, i.e., she does not mind if other users' properties are different than hers (i.e., education, age, etc.), yet she wishes her match would share similar hobbies to hers.

An example query that demonstrates how to express Isabella's wishes is presented in Figure 13. Lines 2-3 define a similarity constraint on the basic profiles of `?u`. The `RESTRICTED TO` construct (line 3) restricts the similarity evaluation to a subset of the profiles - in this example, to user hobbies.

Note that the assignment for variables `?y` include all candidate users assigned to the variable `?u` and Isabella herself, as we compare between Isabella's hobbies and the candidate users' hobbies.



```

1 SELECT ?u
2 SIMILAR basic-profile(?u) TO basic-profile(Isabella)
3   RESTRICTED TO {?y hasHobby ?h.}
4   WITH SIMILARITY >= 0.5

```

Figure 13: Comparing only relevant parts of users' profiles.

```

1 SELECT ?u
2 FROM ontology WHERE
3   {?x instanceOf Place. ?x near Le_Marais,Paris}
4 FROM basic-profile(?u) WHERE
5   {?u livesIn ?x}
6 FROM extended-profile(?u) WHERE
7   {?u like Spicy_Food} WITH SUPPORT >= 0.5
8 SIMILAR basic-profile(?u) TO basic-profile(Isabella)
9   RESTRICTED TO {?y hasHobby ?h}
10  WITH SIMILARITY >= 0.75
11 SIMILAR extended-profile(?u) TO {?u practice Yoga. _ on Weekends}
12   WITH SIMILARITY AS querySim >= 0
13 ORDER BY querySim LIMIT 5

```

Figure 14: SPARQ-U corresponding to Isabella's preferences.

### 3.2.4 Special Issues

- **Similarity scores.** SPARQ-U has embedded constructs for capturing the properties and similarity of (relevant parts of) user profiles, via a semantically-aware similarity measure. For more details on the similarity measure and the evaluation algorithm we refer interested readers to [http://cs.tau.ac.il/~amitsome/pdf/sparqu\\_full.pdf](http://cs.tau.ac.il/~amitsome/pdf/sparqu_full.pdf).
- **Excluding the user asking the query from the output set.** Consider again Figure 13. In this example, the user who have the maximal similarity score is Isabelle herself. However, since Isabelle is the one searching for a mate, it is irrelevant to return her as a possible candidate. On the other hand, assume that Isabelle is looking for a spouse that is similar to her dream man, Brad Pitt (that in this case, also have a profile in the system). Here again, the user with the maximal similarity score is Pitt himself, however, in this case, we do want to return Pitt as part of the output set. Therefore, SPARQ-U does not automatically exclude any user from the output set, as we do not know who is the user asking the query. Hence, the hosting system is in charge of excluding the user asking the query.

**Conclusion** Concluding, Isabella's preferences (specified by her, e.g., via a form within the dating website) may produce the query in Figure 14. Intuitively, the query selects five users that live in the vicinity of Les Marais, Paris (lines 2-5), like spicy food (lines 6-7), their hobbies resemble those of Isabella's (lines 8-10), and who practice Yoga on weekends or have a similar habit (lines 11-13).