Declarative User Selection (Full Version)

Yael Amsterdamer[†], Tova Milo[‡], Amit Somech[‡] and Brit Youngmann[‡]
[†]Bar Ilan University
Ramat Gan, Israel
first.last@biu.ac.il
[‡]Tel Aviv University
Tel Aviv, Israel
{milo,amitsome,brit}@post.tau.ac.il

Abstract

The selection of an adequate set of users for different purposes is a crucial component in many modern applications, including crowdsourcing platforms, social networks and user-based recommender systems. To assist application developers in the intricate task of employing user selection modules, we present SPARQ-U, a novel declarative query framework that supports flexible user selection. SPARQ-U allows specifying selection criteria that are personalized, high-level and context dependent via a dedicated SPARQL extension. In particular, the extended query language has embedded constructs for capturing the properties and similarity of (relevant parts of) user profiles/data, through which the appropriate users for a given context can be effectively identified. To capture user data with rich semantics and account for wide-ranging applications we use an RDF-based data representation. We then develop a new, generic definition of a similarity metric for complex, semantically-rich user data, along with efficient algorithms for computing similarity scores. Our experimental study on real-life data indicates the effectiveness and flexibility of our approach.

1 Introduction

Many current applications involve the selection of users out of a large user base as a central component. Prominent examples include crowd-based platforms that aim to choose relevant crowd members to perform a given task, recommender systems that compute their recommendations to end users based on the preferences of other users, and social networks that identify users who share similar properties. In spite of the common need of user selection modules and the proliferation of relevant research and tools, developers often have to invest non-trivial efforts in employing user selection in new applications.

Previous work has recognized the need to select users in a *context-dependent manner*, based on varying selection criteria or different tasks at hand. Yet, each study has focused on a specific facet of this problem. For example, the selection of users by the relevance of their profiles to a query (e.g., [6]); the estimation of user expertise (e.g. [10, 16, 24]); recommendations based on user similarity (e.g. [8, 20]); and so on. While these facets were studied individually, user selection is often a function of a few such facets *combined*. Given examples for scenarios and the ideal users to select per scenario, one may attempt to use machine learning methods to train a user selection model that implicitly

```
SELECT ?u
FROM ontology WHERE
{?x instanceOf Place. ?x near Le_Marais,Paris.}
FROM basic-profile(?u) WHERE
    {?u livesIn ?x}
FROM extended-profile(?u) WHERE
    {?u like Spicy_Food} WITH SUPPORT >= 0.5
SIMILAR basic-profile(?u) TO basic-profile(Isabella)
    RESTRICTED TO {?y hasHobby ?h.}
    WITH SIMILARITY >= 0.75
SIMILAR extended-profile(?u) TO
        {?u practice Yoga. _ on Weekends. }
    WITH SIMILARITY AS querySim >= 0
ORDER BY querySim LIMIT 5
```

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

incorporates a few different facets. However, accounting for every possible user selection criteria may require wide-ranging training data that is not always available, especially in new applications. Moreover, it is harder to explain and refine the results of such methods, due to their non-declarative nature. These shortcomings of existing user selection tools often lead developers to implement ad-hoc solutions, which are application specific, rarely sharable and hard to maintain and optimize.

To address these problems, this paper presents a novel declarative query framework that supports flexible user selection. It allows specifying selection criteria that are high-level and context-dependent via SPARQ-U, a dedicated extension of SPARQL [23] (a semantic query language).

SPARQ-U has embedded constructs that capture properties and similarity of user profiles (or relevant parts thereof), through which adequate users for a given context can be effectively identified. Additionally, we designed and implemented a query engine to allow for efficient query processing.

In the following example we illustrate the challenges of user selection that new applications may come across. Consider an online dating service that enables users to seek potential partners. The choice of best matches for a given user depends on her profile and her specific desires. Isabella, for instance, 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.

In other words, identifying the potential partners requires: (1) analyzing and comparing, in a *case-specific* manner, relevant *parts* of the user profiles, and (2) employing a metric of *similarity* that is applicable for such data and takes the *semantics* of user data into consideration.

To address this, we use an RDF-based data representation to capture complex, semantically-rich knowledge about users. Specifically, we use two comple-

mentary data repositories: an *ontology* that captures general knowledge about the world (e.g., that yoga and Pilates are forms of mind-body fitness) and *user profiles* that capture facts about individual users. The profile of a user consists of two parts: a *basic user profile* that records available properties of the user (e.g., name, location, hobbies, etc.) and an *extended user profile* where properties also have an associated score, called *support level*. The latter is used to model degrees and frequencies of user preferences and habits (e.g. that Isabella likes spicy food *a lot* or reads books in the park *frequently*). 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 [3]; tools that derive user profiles from social networks [9]; tools for worker performance assessment [1, 21]; etc.

In our example, SPARQ-U would allow the dating service to produce queries that compare the relevant parts of user profiles, and pose soft and hard constraints over them. For example, Isabella's preferences (specified by her, e.g., via a form within the dating website) may produce the query in Figure 1. 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). We explain the query syntax and semantics in Section 2. For now note that the language extends SPARQL with a dedicated `SIMILAR` construct, for specifying both user similarity and soft constraints, to which we develop a generic, semantically-aware definition (allowing, e.g., to identify and quantify the similarity of complex habit descriptions such as yoga on the weekends). The use of soft constraints and our semantically-aware similarity distinguish SPARQ-U from previous query languages for user selection, e.g., in social networks [18, 27] and our experiments show the flexibility and usefulness of SPARQ-U (see more in Sections 5 and 6).

The novel similarity metric that we employ in SPARQ-U lifts the notions of semantic similarity, traditionally defined only for *individual concepts* [8, 30], to sets of facts over these concepts that provide rich descriptions of users' habits, preferences, etc. It is then further extended to consider support values, as provided in the extended profiles. To ensure the soundness of our definitions, we first provide formal constraints on any similarity metric, then develop formulations that satisfy them. Importantly, while computing our formulation using traditional algorithms is intractable, we provide a novel efficient algorithm for computing it. In short, traditional settings only consider the semantic similarity of two *individual concepts*, which is computed based on a concept taxonomy [26]. However, our flexible representation captures complex user data as *sets of facts*, for which a taxonomy would be prohibitively large. The refined search that we perform relies on unique properties, which we prove for fact-set taxonomies, in order to compute the semantic similarity of two fact-sets. See full details in Section 4.

Our contribution may be summarized as follows:

1. SPARQ-U, a SPARQL 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.
2. A dedicated, semantically-aware, similarity metric that enables evaluating both soft constraints and user similarity over complex user data.

3. An efficient query evaluation algorithm for computing similarity scores and identifying relevant users.
4. An implementation of the above concepts 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.

Our framework was recently demonstrated in PVLDB [4]. The accompanying short paper provides only a high-level description of the system design and its usage scenario.

Paper Outline We present our data model and query language in Section 2. In Section 3 we formally define the similarity notions used in queries 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.

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 allow sufficient expressive power and account for various applications, data in our framework is represented using RDF-style *facts* in the form of entity-relation-entity, e.g., {`Isabella hasHobby Dancing`}. We formally define facts and fact-sets, then describe the data repositories, using this representation, that we consider for user selection.

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 Σ^* denotes unrestricted literal terms over some alphabet. A fact-set is a set of facts.*

To express complex descriptions that cannot be captured by a single fact, e.g., "Isabella likes to read books at Jardin des Tuileries", or the co-occurrence of actions, e.g., "Isabella reads while she listens to music", a fact-set is used. In the following example and onward, _ denotes a term (entity or relation) placeholder, in facts that use less than three terms. 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 $l \in \Sigma^*$.[1] Facts within a fact-set are concatenated using a dot.[2] In what follows, we refer to terms in $\mathcal{E}, \mathcal{R}$, facts and fact-sets by the collective name *semantic units*, and use the variables X, Y, Z to denote any of these semantic units. This notation

[1] The curly brackets around facts are added for readability.

[2] Note that the relationships *between* facts are implicit RDF. Hence, it does not capture information such as fact temporal order. However, using RDF has other advantages that we describe throughout this paper.

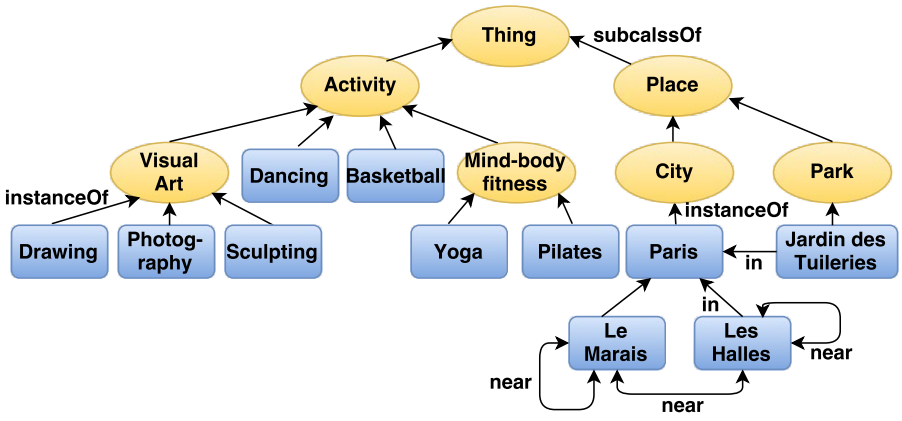


Figure 2: A sample ontology

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).

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 {`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. [7]) and different tools enable constructing domain-specific ontologies [29]. 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 "binary" data, namely, properties of this user that are asserted to be true, in the form of a fact-set. E.g., Isabella's profile may consist of the facts {`Isabella livesIn Le_Marais,Paris`. `Isabella hasHobby Photography`} and others. This information is often provided by the users themselves whenever they sign up to the system, and asked by the hosting system to insert some general details about herself. Alternately, the basic profiles can be constructed either from existing profiles in crowdsourcing platforms, or by using profile builder tools based on social networks such as [9].

Example 2.2. *Table 1 consists of example basic user profiles, with each fact in a separate row. The users Isabella, Adam and Benjamin have filled in their city of residence, their gender, some of their hobbies, etc. Some of the items that are used in the facts appear in the ontology in Figure 2, but others are omitted to simplify the figure.*

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

The *extended profile* of a user contains additional properties that are associated with a numeric score, called its *support value*. Support in our setting is a value in $[0, 1]$ that may reflect a rating, habits frequency or a level of agreement.[3] Formally,

Definition 2.3. *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*.*

Example 2.4. *Table 2 shows sample extended profiles for Isabella, Adam and Benjamin. (Ignore, for now, the "Prevalence" and "IC" columns which will be explained below.) The support score associated with Isabella's preference of spicy food (first fact-set from the top) may be derived from a rating of spicy food she provided to the dating website (normalized to* $[0, 1]$*). The support score of the second fact-set may reflect the frequency with which Isabella reads books, e.g., every other day (again, the frequency is normalized to* $[0, 1]$*).*

As mentioned in the Introduction, profile information may either be provided directly by the user or collected/derived by the hosting system [1, 3, 9, 21].

2.2 User Selection Queries

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 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

[3]The extended profile can clearly capture the basic profile by assigning a score 1 to the basic profile fact-set. The distinction of profile parts is done to ease the presentation here and in the sequel.

Fact-set	Support	Prevalence	IC
`Isabella like Spicy_Food`	1.0	0.8	0.08
`Isabella read Books`	0.5	0.25	0.48
`Isabella read Books.` `_ at Jardin_des_Tuileries`	0.01	0.1	0.72
`Adam like Spicy_Food`	0.75	0.8	0.08
`Adam practice Mind-body_fitness.` `_ on Weekends.`	0.28	0.45	0.3
`Adam practice Pilates.` `_ on Weekends.`	0.27	0.2	0.55
`Benjamin like Spicy_Food`	0.5	0.8	0.08
`Benjamin read Books`	0.25	0.25	0.48
`Benjamin practice Yoga.` `_ on Weekends.`	0.01	0.2	0.55

Table 2: Extended user profiles

(iii) explicitly declaring the knowledge repositories used.[4] The native constructs of SPARQL in this language are mostly used to select relevant repository parts and apply hard constraints.

Reconsider Isabella's query, Q_{Isabella}, from the Introduction (Figure 1). This query selects five users who (i) live near Isabella's neighborhood; (ii) like spicy food; (iii) have hobbies that are similar to Isabella's; and (iv) frequently practice yoga on weekends, or have a similar habit.

We next use Q_{Isabella} to explain the syntax and semantics of SPARQ-U. A SPARQ-U query is composed of two main types of clauses: `FROM...WHERE` and `SIMILAR...TO` clauses, corresponding, respectively, to hard and soft constraints that may be applied over different (parts of) repositories. The latter, soft constraints, form a generic means of capturing notions that are useful in the context of user selection, such as user similarity, expertise and relevance. 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. 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 of `?u` is examined to ensure they like spicy food. Here the clause further defines a lower bound on the support, to ensure selected users rated spicy food highly.[5]

[4]The syntax of SPARQL enables using *prefixes* for querying multiple sources/namespaces. However, since the profile and extended profile are user-dependent, our query constructs for repository selection admit variables.

[5]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 prefers users that gave spicy food a rating of at least 5 out of 10.

The following clauses define soft constraints. Lines 8-10 select users `?u` whose basic profile is similar to Isabella's. `RESTRICTED TO` defines a selection over the basic profiles, which specifies what facts are considered in the similarity computation. In our example, profiles are only compared in terms of hobbies. The selected users and Isabella should have similarity score ≥ 0.75 (similarity scores are defined in the next section). The clause in lines 11-13 considers the desired partner property "practices yoga on weekends" 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 and find 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 on weekends 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.

Overview Different notions of similarity measures are considered in previous work (e.g., [8, 11, 14, 26]), yet none of them can fully account for the semantically-rich representation of user data in our setting, namely fact-sets, possibly accompanied by support scores. For instance, reconsider the soft constraint asking that Isabella's match should practice yoga on weekends, in Q_{Isabella} from Figure 1. According to Adam's profile (Table 2), he practices some other form of mind-body fitness, Pilates, on weekends, with a certain frequency. To compute Adam's relevance to the query, one should be able to *quantify* how similar this hobby and other hobbies of Adam are to Isabella's specified preference.

In comparison, Benjamin does practice yoga on weekends, but according to the support score he does so very rarely, much less frequently than Adam practices Pilates. In a sense, Adam should "gain points" for frequently practicing and Benjamin should "gain points" for being closer to the desired description. How do users who play chess on weekends, or practice yoga on weekdays, etc., compare to Adam and Benjamin?

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. By extending and adapting standard similarity functions to our data representation, we develop a formulation that (i) satisfy 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 in terms of performance and quality, as we show in Section 5.3.

Our definitions use two auxiliary notions: *semantic subsumption*, that allows identifying information common to two semantic units, and *information content*, that quantifies how informative the common information is based on its prevalence. The first factor of our similarity metric, *semantic similarity*, described in

Section 3.2, compares two semantic units based on their subsumption relationships and information content. The second factor, *support similarity*, described in Section 3.3, completes the semantic similarity by considering support values. Finally, we combine the semantic similarity and the support similarity to form a combined similarity measure, in Section 3.4.

3.1 Preliminaries

We start by formally defining the notions of semantic subsumption and information content.

Semantic Subsumption Ontologies often contain a taxonomy of concepts. 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). Such taxonomies define a subsumption relation between their elements: every occurrence of `Photography` subsumes `Visual_Art`, assuming photography is a (type of) visual art. [6]

This relation between terms extends to subsumption between facts and fact-sets that contain these terms [2, 3]: for instance, if Adam practices Pilates, he practices (some form of) mind-body fitness. We will use subsumption when finding similarities in knowledge about users, e.g., Adam resembles Isabella's query in practicing a mind-body fitness, although it is a different fitness than Isabella specified.

More formally, given two terms (elements or relations) 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 (for example, `Visual_Art`$\leq$`Photography`). The relation $\leq$ forms a partial order over the elements of $\mathcal{E}$ and relations of $\mathcal{R}$. Using the definition of [3] 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. *If* `Visual_Art`$\leq$`Photography`*, we obtain that {*`Isabella hasHobby Visual_Art`*}*$\leq$*{*`Isabella hasHobby Photography`*} in the lifted partial order over facts. As for fact-sets, we have that {*`Isabella hasHobby Photography`*}* $\leq$ *{*`Isabella hasHobby Photography. Isabella hasHobby Dancing.`*}.*

We conclude with a remark regarding literals. While literal values typically do not appear in a taxonomy of terms, one often wants to estimate the similarity of facts with literal values, e.g., {`Isabella bornAt "1987.04.29"`}. A simple solution (which we use 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, for instance, that two users are similar by being born in the same decade.

[6] In terms of concepts, visual arts includes photography. However, we are interested in *occurrences* of these concepts within facts, where every occurrence of photography implies an occurrence of (a type of) visual art.

Information Content Next, we define the notion of information content (IC) of a semantic unit, which measures how informative this unit is [26].

In the sequel we use a standard, statistical approach for IC, where a semantic unit that is *less prevalent* has a higher IC value [26]. approach is discussed in Appendix A, yet it is shown to be intractable.

We define the *prevalence* of a semantic unit $\mathrm{p}(X)$ as the fraction of users whose basic or extended profiles *subsume* this unit. For the sake of this computation we *ignore specific user names* and treat them as a generic `User` element. E.g., the prevalence of having visual art as a hobby will be measured as the ratio of all user profiles that include the fact {`User hasHobby Visual_Art`} or subsuming (more specific) facts, such as {`User hasHobby Photography`}.

Definition 3.2 ((Statistical) IC)**.** *Denote the prevalence of a semantic unit X by $\mathrm{p}(X)$. The IC of X is computed as* $\mathrm{ic}(X) := -\log\left(\frac{9\,\mathrm{p}(X)}{10} + 0.1\right)$

Our formula for IC extends the standard definition of [26] to reflect subsumption, with additional normalization to ensure its value lies within $[0, 1]$. Importantly, this definition is *monotonous* with respect to the partial order over semantic units. If $X \leq Y$ (i.e., X is subsumed by Y), then its prevalence can only be higher, hence its IC can only be lower. This property will be useful in the sequel.

Note that in practice 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-gram (`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 reasonable similarity metric $\mathrm{sim}(\cdot,\cdot)$ between pairs of semantic units should satisfy.

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

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

Specifically, IC monotonicity requires that as more semantic information is added, the similarity score can only increase. For example, knowing that Isabella and Adam engage in photography is more informative for the sake of similarity than knowing that both of them engage in a form of visual art, since (i) fewer people engage in photography, and (ii) we assume that visual art is subsumed by photography.

We next define a semantic similarity function that generalizes the standard metric of [26], 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 hasHobby`

`Visual_Art`} is information common to {`Isabella hasHobby Photography`} and {`Benjamin hasHobby Drawing`} (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., photography fans are more similar than fans of any type of visual art). Our function captures this by using the IC of the common unit as the similarity metric. Since two semantic units may have several semantic units in common, their similarity is determined by the IC of the most specific common unit.

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* $\mathrm{ic}(\cdot)$ *over this domain, we define the similarity between X, Y as*

$$\mathrm{icsim}(X, Y) := \max_{Z|(Z \leq X) \wedge (Z \leq Y)} \mathrm{ic}(Z)$$

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

$$\mathrm{icsim}(D, D') := \max_{A \in \mathcal{A}, A' \in \mathcal{A}'} \mathrm{icsim}(A, A')$$

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 $\mathrm{icsim}(\cdot, \cdot)$ (the proof is provided in Appendix B).

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

Example 3.5. *Consider the extended profiles of Adam and Benjamin in Table 2, including the two last columns: the prevalence given for each fact-set, and the IC computed using Def 3.2. Further recall the fact-set specified in $Q_{Isabella}$ in lines* 11-13*. Using* $\mathrm{icsim}(\cdot, \cdot)$ *over each pair of fact-sets produces the set of their common ancestors. One would obtain that {*`User practice Mind-body_fitness. _ on Weekends`*} is the common ancestor with the highest IC for Adam, and {*`User practice Yoga. _ on Weekends`*} is the common ancestor with the highest IC for Benjamin. The semantic similarity scores are then the IC values of these fact-sets,* 0.3 *for Adam and* 0.55 *for Benjamin.*

3.3 Support Similarity

Semantic similarity is a powerful means of comparing semantic units in our model. However, as previously explained, it cannot fully account for the similarity of two users' extended profiles since this function ignores the support scores they provide. For instance, it does not reflect the fact that Benjamin practices yoga on Weekends very rarely, when seeking the user who best matches Isabella's query. As another example, SPARQ-U queries often require comparing the extended profiles of users. In such a case, Isabella and Adam are similar with respect to {`User like Spicy_Food`} if they like it 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 month). Hence, we provide a complementary, support-based similarity formulation, which we will ultimately combine with semantic similarity to obtain our final, combined similarity function.

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

(5) **Support Monotonicity.** For every three answer 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 $\mathrm{sim}(D, D') \geq \mathrm{sim}(D, D'')$.

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 normalized formula for comparing the support of two semantic units. 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*

$$\mathrm{supsim}(\mathcal{S}(X), \mathcal{S}'(X)) := -\log\left(\frac{9\,|\mathcal{S}(X) - \mathcal{S}'(X)|}{10} + 0.1\right)$$

We extend this definition to extended profile databases $D = \langle \mathcal{A}, \mathcal{S} \rangle$, $D' = \langle \mathcal{A}', \mathcal{S}' \rangle$ by $\mathrm{supsim}(D, D') := 1$ *if* $\sum_{A \in \mathcal{A} \cap \mathcal{A}'} \mathrm{ic}(A) = 0$*, and otherwise*

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

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
$\mathrm{supsim}(A, D') := \mathrm{supsim}(D_A, D')$*, 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_{Isabella} from Figure 1.

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

See Appendix C for the full proof. The following example demonstrates the evaluation process of the support similarity function for extended profiles and a given fact-set.

Example 3.8. *Reconsider the extended profiles in Table 2, and the fact-set specified in $Q_{Isabella}$ (lines 11-13). Even though the fact-set in the query refers to practicing yoga on weekends, by Def. 3.6 the computation includes any fact-set subsumed by it, e.g., practicing (any type of) mind-body_fitness on weekends (since* `Mind-body_fitness`$\leq$ `Yoga`*).*

We next compute the support similarity of Adam and Benjamin, considering the support differences, to the required fact-set. For Adam, we get that the only fact-set in the intersection is {`Adam practice Mind-body_fitness. _ on Weekends`*}, where Adam's support value is* 0.28 *and by definition the query's support value is* 1*. Then,* $\text{supsim}(0.28, 1) \approx 0.126$ *and this is also the final support similarity of Adam. Benjamin also has only one fact-set in the intersection,* {`Benjamin practice Yoga. _ on Weekends`}*, hence his support similarity is* $\text{supsim}(0.01, 1) \approx 0.004$*.*

3.4 Combined Similarity

To combine the semantic and support similarity defined above, we simply multiply them, i.e., for two extended profile databases D, D'

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

By our definition, when semantic units are not assigned support values (e.g., user's basic profiles), their support is always 1, hence the support similarity of such units X, Y is always 1, and by using an equivalent formula for $\text{sim}(X, Y)$ the result coincides with $\text{icsim}(X, Y)$.

The proof of the following is in Appendix D.

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_{Isabella} is provided in Appendix D.

4 Query Execution

In SPARQ-U, simple selections (within `RESTRICTED TO` and `WHERE` clauses) coincide with SPARQL selections. Optimizing this part of the queries can thus be done using standard techniques and existing SPARQL engines. We therefore focus here on evaluating soft constraints, given assignments to the query variables. 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 [2, 3]. Nonetheless, our contribution here is an efficient algorithm for computing semantic similarity for terms, facts and fact-sets and an analysis of its complexity bounds. Then, we consider optimizations.

Computing IC Similarity We start from the basic computation of the similarity of two terms, and then show how to lift it to facts, fact-sets and extended profile databases.

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)

1 $L \leftarrow \emptyset$;

2 **for** $f \in A, f' \in B$ **do**

3 $\quad L \leftarrow L \cup \text{FactLCA}(f, f')$;

4 **for** $f = \langle e_1, r, e_2 \rangle, f' = \langle e'_1, r', e'_2 \rangle \in L$ **do**

5 $\quad$ **if** $e_1 \leq e'_1 \wedge r \leq r' \wedge e_2 \leq e'_2$ **then** $L \leftarrow L - \{f\}$;

6 **return** L;

Function icsim(X, Y)

1 **if** $X = (\mathcal{A}, \mathcal{S}), Y = (\mathcal{A}', \mathcal{S}')$ *are extended profile databases* **then**

2 $\quad$ **return** $\max_{A \in \mathcal{A}, A' \in \mathcal{A}'} \text{ic}(\text{FactSetLCA}(A, A'))$;

3 **else if** X, Y *are fact-sets* **then**

$\quad$ **return** $\text{ic}(\text{FactSetLCA}(X, Y))$

4 **else if** X, Y *are facts* **then**

$\quad$ **return** $\text{ic}(\text{FactLCA}(X, Y))$

5 **else return** $\text{ic}(\text{LCA}(X, Y))$;

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

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 partial orders (as in particular, the number of fact-sets may be exponential in the number of terms [3]). Instead, our optimized algorithm computes the LCA for facts and fact-sets *directly*, in PTIME.

Algorithm 1 outlines the implementation of three functions required for efficiently computing the semantic similarity of each type of input. While the structure of the functions is simple, it is the underlying analysis of the components that guarantees overall low complexity. We describe here the flow of the algorithm, and defer the proofs of its correctness and complexity analysis to Appendix E. Recall that by Def. 3.3 the semantic similarity of two 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).

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). Additionally, the algorithm assumes an implementation of ic$(\cdot)$ for any semantic unit, according to Def. 3.2.

The first function in Algorithm 1, 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., the output of 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, 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 FactLCA(f, f')).

The second function, 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 case of terms and 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 FactSetLCA(A, B) uses FactLCA(f, f') for each such pair of facts, takes the union L of the LCAs (lines 2-3) and then removes redundant facts (that are implied by other facts in this set, lines 4-5. Refer to Appendix E for correctness).

Finally, the function icsim(X, Y) gets two inputs and computes their similarity according to their type (extended or basic profiles databases, fact-sets, facts or terms) using the relevant LCA function and the IC function. The following proposition summarizes the complexity of computing IC similarity, according to the algorithm described above and known complexity bounds for computing the LCA of terms [5], and assuming that IC computation of a given semantic unit is done in O(1). The full proof is deferred to Appendix E. In what follows, $w[\Psi]$ denotes the *width of the term taxonomy* Ψ, namely, the maximum size of a set of incomparable terms.

Proposition 4.3. *After PTIME preprocessing of the partial order $\leq$ 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.*
- $O\big(|X|\,|Y|\,w[\Psi]^3 \log\left(|X|\,|Y|\,w[\Psi]\right)\big)$*, if X, Y are fact-sets (i.e., polynomial in the fact-set sizes and the width of the term taxonomy).*
- $O(|X|\,|Y|)$ *times the complexity of computing the IC similarity for each pair of fact-sets, if X, Y are extended profile databases.*

Leveraging computations across assignments The output of Algorithm 1 is the similarity score of two *concrete* semantic units or extended profiles databases, 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 [3] 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 φ is below the similarity threshold for one *such clause, every more general assignment ψ will be below the threshold for the same clause.*

(2) If some variable assignment φ is above the similarity threshold for every *such clause, every more specific assignment ψ will be above the threshold for all such clauses.*

This observation can then be used as follows. The results for each considered assignment will be kept in a cache during the SPARQ-U query evaluation. 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 semantic similarities 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.

To conclude this section we note that in some cases the computation of support similarity may suffer from a *sparsity* problem: when comparing users extended profiles, it is possible that their intersection, required for support similarity computation, is small or empty. To overcome this, previous works have suggested several techniques to extend the relevant user profiles, such as crowd mining [3], or extracting past activities from social networks [6, 9].

5 Experimental Study

We developed a prototype engine for parsing and evaluating SPARQ-U queries, and implemented a caching mechanism based on Observation 4.4, which stores previous calculations (e.g., previously computed LCAs of semantic units). The prototype is implemented in Java, using the Apache Jena library (`https://jena.apache.org/`) for handling RDF data and for a SPARQL engine, and the JGraphT library (`http://jgrapht.org`) for graph processing operations on the ontologies. To enable using standard RDF tools over our user profile 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.

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

We set three goals for our experimental study. Our first goal was to *examine the flexibility* of SPARQ-U and its ability to express specific user selection needs in different contexts, including recommendations, predictions and expert finding. 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* (`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,[7] as detailed below.

Our second goal was to *evaluate the result quality* that our approach achieves, namely, the adequacy of selected users. We examined the SPARQ-U framework as a whole, as well as the contribution of each of its components. We focused on several typical scenarios for user selection over the datasets, where the quality of the results could be evaluated with respect to an actual ground truth. For each of the scenarios, we compared the results of the SPARQ-U framework to common alternative strategies from machine learning (SVM) and graph processing (Simrank [15]). To examine the contribution of components within our framework, and in particular the ingredients of our dedicated similarity metric, we examined restricted variants of SPARQ-U that do not include some components, or use alternative similarity functions. Finally, as an alternative quality metric we conducted a user study to let real users assess the relevance of the result set.

Our third goal was to *test the system's running time and its scalability.* We examined the execution time for SPARQ-U queries, and in particular the improvement achieved by our cashing mechanism, for varying data and ontology sizes.

Overall, the results of our experimental study show that SPARQ-U is expressive enough to capture practical ways of selecting users in different 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 the data was extracted and converted into our data model, then explain the different baseline algorithms used in our experiments. All experiments were executed on a Dell PC with 4 cores and 8GB memory.

Stack Overflow dataset In this popular Q&A platform, each question is posed by a specific user, and associated with a set of tags that reflects its topic. Among its answers, a question may have one designated answer chosen as the most accurate one. Each user in the system has a profile, consisting of the user's name, registration date, badges awarded to the user for answering questions, and a reputation score, a rank computed by the system that represents the quality of the user's previous questions/answers.

We collected 800k questions in 200 popular topics (tags), asked by over 1.6k users, using the API of Stack Overflow (`http://api.stackexhange.com`). We then gathered their personal profiles, and collected more than 700k 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

[7] In fact, AMiner uses data from DBLP, which is often used in link prediction [15], one of our experimental scenarios.

the Stack Overflow user profiles. The extended profile of each user was constructed from her topics (tags) to which she contributed with a support value that reflects the portion of her contribution to that tag among all tags. The ontology for this domain was constructed from DBpedia [7]. We aligned each tag to a matching DBpedia concept and used the relevant part of the DBPedia taxonomy. The resulting ontology consists of 40k entities.

AMiner dataset We extracted the data of 1.8k Computer Science researchers, and considered their publications between 2005 and 2015 (over 5.4k papers). The basic user profiles we constructed from this data consist of facts capturing affiliation, h-index, fields of interest, and publications data (e.g., year and venue). The extended profile of each user records her co-authorships (with a support value reflecting the fraction of her publications with a given co-author among all of her publications), and publication venues (with support value reflecting the fraction of her publications among all publications in this 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 ontology consists of 35k 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 solution as a whole.
1.SPARQL. User selection is done by a standard SPARQL query, namely only the hard constraints are employed. This is used to evaluate the need of considering similarity-based soft constraints in user selection.
2.No Support. User selection is done by the same SPARQ-U queries, but the support values in the extended profile are ignored. This is used to evaluate 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 each considered individually and co-occurrence of facts is ignored. This is used to evaluate the need for our rich data model and our semantic similarity metric that is defined over this model.
4.Simrank. We use Simrank [15], a structural similarity measurement which is commonly used in social networks for link prediction problems, as an alternative similarity metric in the query.
5.SVM. Users are selected using SVM, a common machine learning method for such a task. We experimented with regression and classification-based variants and describe here the classification variant that performed better. We used the scikit-learn (`http://scikit-learn.org/stable/modules/svm.html`) implementation. For each scenario separately, we employed an SVM classifier, where each user represents a class, and the algorithm computes the probabilities of each class to be chosen. Then, we used the obtained probability as similarity metric in the query. This baseline is used to evaluate the ability of SPARQ-U to capture common user selection scenarios, which can alternatively be specified as a machine learning problem.

```
SELECT ?u
FROM basic-profile(?u) WHERE
    {?u creationDate ?d. Filter (?d < 19.6.2015) }
SIMILAR basic-profile(?u) TO basic-profile(Basil_Bourque)
    WITH SIMILARITY AS profSim > 0
    SIMILAR extended-profile(?u) TO {?u answeredOn Java .
                ?u answeredOn I/O . }
    WITH SIMILARITY AS topicSim > 0.2
ORDER BY AVG(profSim, topicSim) LIMIT 50
```

Figure 3: User selection for Stack Overflow question

Baselines 4 and 5 are used as alternative implementations to our similarity metric, namely, hard constraints are still used for initial data filtering. The results of these tools without initial filtering were inferior in all the experiments. Additional alternative similarity metrics that we examined were the standard cosine and Jaccard similarity metric, which also yielded inferior results and we thus omit them for space constraints.

5.3 Experiments

We next present our experiments in both domains, along with an analysis of the results and performance. Following typical user selection scenarios in previous work, we constructed SPARQ-U queries that capture each scenario over our Stack Overflow and Aminer datasets, and analyzed the query constructs they require. Notably, the use of soft constraints in all of these scenarios was very natural and often necessary.

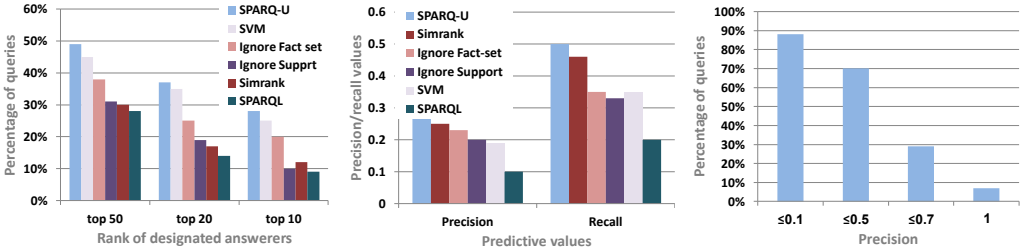
We defer the discussion of these scenarios to Appendix F and focus below on specific scenarios for which there exists *ground truth* that enables assessing the user selection quality. For each scenario we describe the corresponding SPARQ-U queries and analyze their performance compared to the alternative algorithms and the ground truth.

Stack Overflow experiment Given a user question in Stack Overflow, we consider the task of finding adequate users to answer it. We considered several alternative SPARQ-U queries that may be used this task (including, e.g., the retrieval of top-ranked users for each of the question's tags), but eventually chose a query that retrieves users who have previously answered questions on topics relevant to the question, and whose basic profile is similar to that of the user that posed the question. The first requirement ensures the 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. Apparently, a "peer" is more likely to provide a good answer to one's question rather than someone with significantly higher or lower capabilities. Following this approach, we constructed a template SPARQ-U query to identify potentially good responders to a given Stack Overflow question, 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 and write out to another file, in Java?", tagged by 'Java' and 'I/O'. The corresponding* SPARQ-U *query instantiated from our template for this question is depicted in Figure 3.*

We randomly selected over 500 Stack Overflow questions posed in 2015-2016 and examined the corresponding SPARQ-U queries, instantiated from our template for choosing adequate responders to each question. Each of these queries was only executed over data that was collected in Stack Overflow before the question corresponding to the query was posted.

Now, recall that each question in Stack Overflow 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 counted the percentage of queries where this user was among the top 10, 20, and 50 users selected by our query. We also measured the same for the alternative baselines. 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.



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

(b) AMiner: Precision & recall.

(c) AMiner: Precision in user study.

Figure 4: Quality assessment

The results are depicted in Figure 4a. Observe that in all cases our query performs better than the competitors. In particular, we can see that for 28% of the examined questions, our SPARQ-U query ranked the designated answerer among the top 10 recommended users and for 49% of the questions, among the top 50 recommended users. This is extremely positive given the fact that Stack Overflow 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 Stack Overflow.

Furthermore, observe that the three restricted variants of SPARQ-U (SPARQL only, no support, no fact-set) achieve inferior results, demonstrating respectively the importance of soft constraints and the dedicated components of our similarity measure. Simrank also shows significantly inferior results here, since unlike our similarity measure, it does not use semantic information. SVM is closer to SPARQ-U in terms of quality, but unlike our generic similarity measure, the SVM classifier was specifically trained for this scenario and cannot be trivially used for other scenarios.

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 particular, in 60% of the cases, either the user posing the question or the designated answerer were new users with few or no past activities

(i.e., these users have not posted many questions nor answers). As mentioned in Section 3, such a "cold start" problem may be resolved 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. (see Appendix F for other examples.) Our query selects, for a given researcher, the top-10 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 most similar profiles. Similarity here is measured both w.r.t. 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 DBLP (`http://dblp.uni-trier.de`), and the other via a user study.

For the first experiment we split the dataset into two parts: one contains the publications up to 2010, 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), and randomly selected over 150 such authors. We then executed a corresponding SPARQ-U query on data containing only the earlier years (i.e., the first part of the dataset), for each of these authors, and compared the query results with actual collaborations in the following years (2011-2016). To measure how well the queries preformed, we then computed the precision and recall scores for each author (namely, how many of her predictions were fulfilled and how many of her true collaborations were predicted, respectively), and plotted the average precision and recall measurements in Figure 4b.

The SPARQL baseline 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 query has identified at least one true future collaborator, namely the precision was ≥ 0.1. Over 85% of the examined authors collaborated with at least 3 of the predicted co-authors. Furthermore, for over 60% of the authors, more than 50% of their true future collaborators were predicted by SPARQ-U, i.e., the recall was ≥ 0.5. In contrast, the restricted variants of SPARQ-U achieved at most 0.2 precision and at most 0.36 recall in average. Unlike the Stack Overflow experiment, here Simrank significantly outperforms SVM. Indeed, Simrank is designed for the social network context and leverages the similarity of user connectivity in the collaboration graph. Interestingly, SPARQ-U still provides superior results and competes successfully even with techniques that are specifically tailored to the given context.

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 designated SPARQ-U query, excluding their past co-authors,[8] and asked them to estimate how many of the selected users

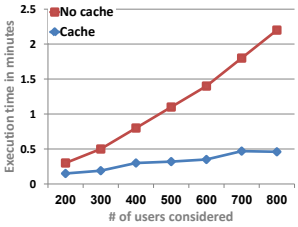
[8] Assuming most researchers would not object to collaborate again with past co-authors.

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

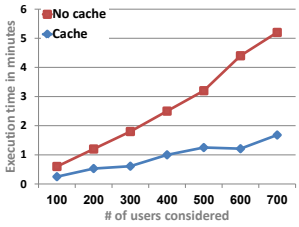
In a real-life usage of the system, users may further personalize the SPARQ-U query, adding individual filtering conditions on location, language, etc. Yet, even our simple, generic query obtained some interesting results. Indeed, 89% of the participants found at least one recommendation relevant (precision ≥ 0.1), and 85% of the participants found at least 3 recommendations out of 10 relevant (precision ≥ 0.3).

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 unclean data (e.g., authors with basic profiles that do not reflect their fields of interest). As mentioned previously, tools for building profiles based on social networks such as [9] may be used to enrich the data and consequently the recommendations.

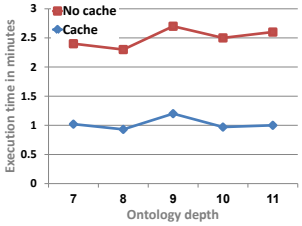
Query execution experiment First, note that the hard constraints in our queries are evaluated by the selected SPARQL engine. Since we use an existing state-of-the-art engine, we measure the runtime of our dedicated extensions only. We consider the effect of two factors on the execution time: (1) The number of users admitting the hard constraints, namely the users on which our extension operates in practice; and (2) the shape of the ontology (i.e., its width and depth). In both cases we examined the execution time with and without our dedicated cache mechanism (which, as described in the beginning of this section, stores recently computed results and avoids unnecessary computations).



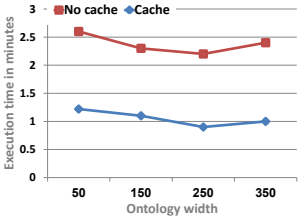
(a) Stack O. - time vs. #users



(b) AMiner - time vs. #users



(c) AMiner - time vs. depth



(d) AMiner- time vs. width

Figure 5: Execution times.

We grouped the AMiner and Stack Overflow queries based on the number of users admitting the hard constraints in each query (100-200, 201-300, ...) and computed the average execution time in each group (the variance was less than 5%). Figures 5a and 5b depict the average running time (in minutes) for AMiner and Stack Overflow queries as a function of the number of admitted users. Note that our caching mechanism significantly affects both the absolute running times and the scalability of our computations. Further observe that our example AMiner queries take approximately twice the time that our example Stack Overflow queries take, since only the former queries compute the similarity of full extended profiles. In all cases, the running time was below 2 minutes, which is reasonable for the type of applications we target (crowdsourcing, social networks, recommended systems, etc.) where user rankings and similarity scores are typically computed offline.

We next examined how the *shape* of the restricted subgraph that is relevant to the similarity evaluation (our main computational bottleneck) affects query execution time. We consider queries admitting 301-400 users (other buckets demonstrated similar trends), and further divide them to sub-buckets by the shape of the ontology part they use. Figure 5c shows execution times as a function of the subgraph depth for width around 150, and Figure 5d shows execution times as a function of the subgraph width for depth 7. Interestingly (and positively) the results demonstrate that our preprocessing of the ontologies, as described in Section 4, is extremely effective and the shape of the considered ontology subgraphs does not affect the execution time. As before, using our caching mechanism achieves better performance.

6 Related Work

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

Recommender systems leverage similarities between user (and item) data in order to recommend users (or items) to end users. These recommendations are based on similar item ratings and/or similar user profiles,sometimes using semantic knowledge (e.g., [8, 20]) In the context of social networks, friend recommendations are computed by techniques of *link predication*, which rely on analysis of the network structure (e.g., [12, 15]). However, none of these works considers a similarity metric 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., [6, 10, 16, 24]). Such an assessment may then assist in assigning multiple tasks to users with relevant expertise [19, 28]. A related problem is *quality control* in crowdsourcing, where users are selected based on (an estimation of) their performance in previous tasks (e.g., [1, 10, 21]). Some contributions of these works are orthogonal to ours: we can append derived expertise or quality scores to user profiles, and use our similarity metric to compute the *relevance of a user to a new context*, provided by a query. 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 focusing on a specific facet of the problem.

Other query languages have been developed for user selection, mostly in the context of *social network analysis* (e.g., [18, 27]). Another language that focuses on querying user preferences is presented in [13], and [17] 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., [11, 14, 25, 31]) and *similarity search*, which is used in evaluating imprecise or relaxed queries (e.g., [34]). 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 *equivalent* representations of the same data (entity or query answer). Thus, in contrast to our work, quantifying the similarity of distinct entities (e.g., French photographers and Italian artists) is not targeted. Moreover, our data representation of user profiles, involving fact-sets and support is not accounted for by the similarity metrics of these works. In future work, it would be interesting to test our similarity metric as a means of evaluating general RDF queries with soft constraints.

7 CONCLUSION AND FUTURE WORK

This work presents SPARQ-U, a framework that allows the declarative specification of personalized, context dependent, user selection criteria. Its query language, a SPARQL extension, has embedded constructs for capturing the properties and similarity of (parts of) user profiles, via a semantically-aware similarity metric. Dedicated novel algorithms 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 would be to include *diversification*, *clustering* and *classification* constructs, which may be built on top of our semantic notions of similarity. The cold start problem 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 to other applications such as entity matching.

References

[1] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. Motahari-Nezhad, E. Bertino, and S. Dustdar. Quality control in crowdsourcing systems. *IEEE*, 2013.

[2] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2014.

[3] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *SIGMOD*, 2014.

[4] Y. Amsterdamer, T. Milo, A. Somech, and B. Youngmann. December: A declarative tool for crowd member selection. *VLDB*, 2016.

[5] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin. Finding least common ancestors in directed acyclic graphs. In *SODA*, 2001.

[6] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: expert finding in social networks. In *EDBT*, 2013.

[7] DBpedia. `http://dbpedia.org/About`.

[8] 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.

[9] 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.

[10] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. iCrowd: An adaptive crowdsourcing framework. In *SIGMOD*, 2015.

[11] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *DKE*, 2008.

[12] Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. JCDL, 2005.

[13] M. Jacob, B. Kimelfeld, and J. Stoyanovich. A system for management and analysis of preference data. *PVLDB*, 2014.

[14] Y. Jean-Mary and M. Kabuka. Asmov: Ontology alignment with semantic validation. In *SWDB-ODBIS*, 2007.

[15] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *SIGKDD*, 2002.

[16] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *SIGKDD*, 2009.

[17] 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. In *WWW*, 2016.

[18] M. Martın, C. Gutierrez, and P. Wood. SNQL: A social networks query and transformation language. In *AMW*, 2011.

[19] P. Mavridis, D. Gross-Amblard, and Z. Miklós. Skill-aware task assignment in crowdsourcing applications. In *Int. Sym. Web Algo.*, 2015.

[20] B. Mobasher, X. Jin, and Y. Zhou. Semantically enhanced collaborative filtering on the web. In *EMWF*, 2003.

[21] Q. Ni and E. Bertino. Credibility-enhanced curated database: Improving the value of curated databases. In *ICDE*, 2010.

[22] J. Provan and M. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM*, 1983.

[23] E. Prud'Hommeaux, A. Seaborne, et al. SPARQL query language for RDF. *W3C rec.*, 15, 2008.

[24] H. Rahman, S. Thirumuruganathan, S. B. Roy, S. Amer-Yahia, and G. Das. Worker skill estimation in team-based tasks. *PVLDB*, 2015.

[25] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 2011.

[26] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, 1995.

[27] R. Ronen and O. Shmueli. SoQL: A language for querying and creating data in social networks. ICDE, 2009.

[28] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das. Task assignment optimization in knowledge-intensive crowdsourcing. *PVLDB*, 2015.

[29] C. Sarasua, E. Simperl, and N. F. Noy. Crowdmap: Crowdsourcing ontology alignment with microtasks. In *ISWC*. Springer, 2012.

[30] N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in WordNet. In *ECAI*, 2004.

[31] F. M. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 2011.

[32] J. Tang, J. Zhang, R. Jin, Z. Yang, K. Cai, L. Zhang, and Z. Su. Topic level expertise search over heterogeneous networks. *Mach. Learn.*, 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] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao. Semantic SPARQL similarity search over RDF knowledge graphs. *PVLDB*, 2016.

Appendix

A Structural Similarity

In Section 3.1 we provide a definition of IC which is based on a *statistical approach*. Previous work [20] 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 [22]. 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 $\mathrm{f}(d)$. Then $\mathrm{f}(d) \leq \mathrm{f}(d')$ 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. □

B Semantic Similarity

In Section 3.2 we listed constraints on a *semantic similarity function*. We next show that $\mathrm{icsim}(\cdot,\cdot)$ (see Definition 3.3), satisfies these constraints and is therefore a semantic similarity function.

(Prop. 3.4). We prove that constraints (1)-(4) described in Section 3.2 are satisfied by $\mathrm{icsim}(\cdot,\cdot)$.

(1) *Maximum self-similarity.* By definition, $\mathrm{icsim}(X, X) = \mathrm{ic}(X)$. 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 $\text{icsim}(X, Z) \leq \text{icsim}(X, Y)$. □

C Support Similarity

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

(Prop. 3.7). We prove that constraints (1)-(3),(5) described in Sections 3.2, 3.3 are satisfied by $\text{supsim}(\cdot, \cdot)$.

(1) *Maximum self-similarity.* For a fact-set A, $supsim(\mathcal{S}(A), \mathcal{S}(A)) = 1$ by definition. For a database D (or D_A derived from an 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.

□

D 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 constraints (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, constraint (4) is only defined for fact-sets, and since $\text{sim}(\cdot, \cdot)$ coincides with $\text{icsim}(\cdot, \cdot)$ for fact-sets, the constraint is satisfied 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_{Isabella}.

Example D.1. *For both Adam and Benjamin the* `WHERE` *clauses holds (i.e., both of them live near Le Marais and like spicy food 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 the hobbies in their basic profiles. E.g., in the case of Benjamin, this would be the singleton fact-set {*`User hasHobby Visual_Art`*}, as he is interested in drawing and Isabella in photography, and every other common information (both of them also implicitly have* `Activity` *and* `Thing` *as hobbies) is subsumed by this fact-set. Assume that*

the semantic similarity score, i.e., the IC of this fact-set, exceeds the threshold for both users. 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 then 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).*

E 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 E.1. *The complexity of computing the IC similarity of* two terms t, t' *is* $\Theta(|\mathrm{LCA}(t, t')|)$*, 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 $\mathrm{LCA}(t, t')$ 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 [5], after PTIME processing of the partial order/DAG one can find the LCA of every two elements in it in constant time. □

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 E.2. *The complexity of computing the IC similarity of* two facts f, f' *is* $\Theta(|\mathrm{LCA}(f, f')|)$*, 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 $\mathrm{LCA}(e_1, e_1') \times \mathrm{LCA}(r, r') \times \mathrm{LCA}(e_2, e_2')$ 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).

Use Case	Example	Required SPARQ-U features
1. Expert finding	Select users who are highly linked to the key term "Database" and who frequently publish in SIGMOD and VLDB, ranked by their H-index. 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 to the researcher, 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 and Figure 3.	Hard constraints, combined similarity, order by similarity
4. Context-based user recommendation	Select users that are highly relevant to the tags "Java" and/or "Python" based on their answers since 2015. See Figure 7.	Hard constraints, combined similarity, order by similarity
5. Community detection	Select users who frequently collaborate with many of the community members.	Combined similarity, order by similarity

Table 3: Use cases of SPARQ-U

Then, by Lemma E.1 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. □

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 [2] 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 $\mathrm{OD}_C = \{f \mid \exists f' \in A, f'' \in B, f' \leq f \wedge f'' \leq f\}$, and it is an order ideal since by definition, if fact f is in OD_C, then so are also all of its ancestors. □

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

Lemma E.3. *The complexity of computing the similarity of* two fact-sets A, B *is* $\mathrm{O}\big(|A|\,|B|\,w[\Psi]^3 \log\left(|A|\,|B|\,w[\Psi]\right)\big)$*, 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 E.1). 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 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 $\mathrm{LCA}(A, B) \subseteq L$. Assume by contradiction that this is not the case. Then there exists a fact $f \in \mathrm{LCA}(A, B)$ 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 $\mathrm{LCA}(A, B)$. 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 $\mathrm{O}(|A|\,|B|\,\mathrm{maxLCA})$, where 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 $|L| \leq |A|\,|B|\,\mathrm{maxLCA}$. To remove the redundant facts by ordering them, we need at most $\mathrm{O}(|L| \log |L|)$ 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. □

F Flexibility of SPARQ-U

We next demonstrate the flexibility of SPARQ-U and its ability to capture common user selection scenarios, via (additional) example queries over the Stack Overflow and AMiner datasets. Table 3 lists user selection scenarios, and for each such scenario it describes an example query and the features of SPARQ-U that are used in this query. In particular, all the example queries use soft constraints and hence implicitly require our semantic similarity function.

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. Our goal is not to devise a new definition but rather enable expressing a wide range of such definitions via SPARQ-U. For instance, Stack Overflow ranks experts in tags by the scores that other users gave to their answers related to that tag (i.e., reputation score). Obtaining the same ranking in SPARQ-U can be done via a simple query that selects the scores of users in a give tag and orders them

```
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 SIGMOD . ?u published_at PVLDB .}
8     WITH SIMILARITY > 0.2
9 ORDER BY DESC(?h)
```

Figure 6: SPARQ-U query selecting database experts

```
1 SELECT ?u
2 FROM basic-profile(?u) WHERE
3         { ?u lastAccessedDate ?d. Filter(?d > 2015) .}
4 SIMILAR extended-profile(?u) TO
5         {?u answeredOn Java . ?u answeredOn Python .}
6     WITH SIMILARITY AS querySim > 0.2
7 ORDER BY querySim
```

Figure 7: SPARQ-U query selecting all the Stack Overflow users who have frequently answered Java- and/or Python-related questions since 2015.

accordingly. When such scores are not available, experts can be identified by their *similarity to selected properties* that are related to the topic in question. For example, consider selecting database experts based on the AMiner dataset. The SPARQ-U query in Figure 6 selects users who are specialized in databases (or have a similar skill, see the first `SIMILAR` clause), and who frequently publish at two top conferences in the field, SIGMOD and PVLDB (see the second `SIMILAR` clause). Among the users who match these criteria, the query returns those with the top-10 H-index (see the `ORDER BY` clause, where `?h` is defined in line 3). It turns out that this query is quite useful: AMiner uses a machine learning approach to determine the level of expertise [32]. We have sent the two lists of top-10 experts by our query and by AMiner's ranking to 10 researchers from the field, without explaining how they were created (the overlap of the two lists, ignoring the order, was 60%). All of the replies preferred our rankings as more accurate.[9]

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 example AMiner query we have used in Section 5 is one example such query, which uses hard constraints to select collaborators of collaborators, then uses combined similarity to find the researchers most similar to a given researcher in terms of their (basic and extended) profiles, and finally returns the most similar such researchers.

3. Profile-based user recommendation and 4. Context-based user recommendation are two general strategies of providing, to a given user, recommendations of other relevant users from some community.[10] These recommendations are based, respectively, on *similarity to a target profile*, such as the profile of the given user or of her "ideal" match, and *context* such as a set of keywords or properties provided by the user. Context-based recommendations typically also take into account the profile of the given user [8, 19]. For example, the queries

[9] We do not provide the lists here, since the personal judgments that we obtained may offend some of the readers. However, the lists can be easily recomputed.

[10] Do not confuse with similar scenarios in recommending *items* to a given user.

in Figure 1 and in Figure 3 include profile-based recommendations since they retrieve users whose profile resembles a target user (Isabella/Basil Bourque). Figure 7 presents a query that selects Stack Overflow users that are most relevant to the context of the tags "Java" and "Python", considering only data from 2015 and on.

5. Community detection. This is the problem of identifying a highly interlinked community in a network, possibly with respect to some context, e.g., the members of one family in a social network. In some cases, detecting a community from scratch requires repeatedly evaluating the relevance of each member to the others, and hence cannot be done via a single SPARQ-U query, since its current semantics allows considering each candidate user only once. However, SPARQ-U can be used to detect a community of users who are relevant to some topic (as in context-based recommendation) or to detect additional members of a seed community. For example, given a community in AMiner, one can use SPARQ-U to query researchers that have frequently collaborated with many members of this community, by posing a soft constraint of the form `?u collaboratedWith X. ?u collaboratedWith Y...` where `X`, `Y` and so on are the members of the seed community.