Declarative User Selection with Soft Constraints

Yael Amsterdamer
Bar-Ilan University
Ramat Gan, Israel

Tova Milo
Tel Aviv University
Tel Aviv, Israel

Amit Somech
Tel Aviv University
Tel Aviv, Israel

Brit Youngmann
Tel Aviv University
Tel Aviv, Israel

ABSTRACT

In applications with large userbases such as crowdsourcing, social networks or recommender systems, selecting users is a common and challenging task. Different applications require different policies for selecting users, and implementing such policies is application-specific and laborious. To this end, we introduce a novel declarative framework that abstracts common components of the user selection problem, while allowing for domain-specific tuning. The framework is based on an ontology view of user profiles, with respect to which we define a query language for policy specification. Our language extends SPARQL with means for capturing soft constraints which are essential for worker selection. At the core of our query engine is then a novel efficient algorithm for handling these constraints. Our experimental study on real-life data indicates the effectiveness and flexibility of our approach, showing in particular that it outperforms existing task-specific solutions in prominent user selection tasks.

CCS CONCEPTS

• **Information systems → Resource Description Framework (RDF)**; **Web Ontology Language (OWL)**; *Similarity measures*; Query languages;

KEYWORDS

Semantic Similarity; SPARQL; User Selection

ACM Reference Format:
Yael Amsterdamer, Tova Milo, Amit Somech, and Brit Youngmann. 2019. Declarative User Selection with Soft Constraints. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358025

1 INTRODUCTION

Effectively selecting user profiles from a repository is a central component in many applications. Examples include crowd-based platforms that aim to choose relevant members to perform a given task, recommender systems that compute recommendations based on the preferences of previous users, and social networks that identify users who share similar properties. The criteria for user selection vary across applications, and include relevance of their profiles to a query/task (e.g. [1]); their (estimated) expertise (e.g. [2]); their similarity to the current end-users (e.g. [3]), and other considerations, usually combined in ways that are application or domain-specific. Lacking a *generic* solution, developers often have to invest non-trivial efforts in employing user selection in new applications.

```
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 Dogs} WITH SUPPORT >= 0.5
SIMILAR basic-profile(?u) TO basic-profile(Isabella)
    RESTRICTED TO {?y hasProfession ?h}
    WITH SIMILARITY >= 0.75
SIMILAR extended-profile(?u) TO
    {?u practice Yoga. _ at Park.}
    WITH SIMILARITY AS ?querySim >= 0
ORDER BY ?querySim LIMIT 5
```

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

Machine learning methods may account for some of the above mentioned facets and therefore be employed for user selection in some applications. However, adapting their results to different needs may require training data that is not always available for every possible selection criteria, 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 shareable and hard to maintain and optimize.

To this end, we develop *a declarative framework for customized user selection*, which is geared towards being used by developers of relevant applications, in their implementation of user selection modules. We next demonstrate types of user selection criteria that we support, then discuss the main components of the framework.

Running Example. Consider an online dating service supporting customized search for potential partners. Using a Web interface, our example user Isabella would like to specify her criteria for desired partners: she is interested in users who live in the vicinity of her neighborhood (Le Marais, Paris), like dogs, have a similar profession to hers, and who practice yoga in the park. These criteria can be categorized along two axes: *hard versus soft constraints*, and *client-dependent versus independent*. E.g., Isabella may insist, as a dog owner, on a partner who likes dogs (hard constraint). Other criteria may still be acceptable if partially satisfied (soft constraints): Isabella's ideal partner may be a person who practices yoga in the

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358025

park, but she may also consider, e.g., a person who practices Pilates (another kind of mind-body fitness), a person who enjoys spending time in parks, etc. Some criteria depend on Isabella's profile, and may be captured by (partial) profile similarity: in Isabella's case, she seeks partners with a similar profession to hers. Others may be independent of her profile: e.g., Isabella may have never practiced yoga, yet wishes for a partner who will help her acquire this habit. Generally, allowing customized selection of potential partners requires analyzing and comparing the semantically rich data within user profiles. The analysis must be supported by means of easily specifying and efficiently evaluating hard and soft constraints, both client dependent and independent, over this data.

We next overview the main components of our framework.

User Model. We support an RDF-based representation of user profiles, in two complementary repositories: an *RDF ontology* that captures application-specific and general knowledge (e.g., that yoga is a form of activity) and *user profiles* that capture the sets of properties describing each user (e.g., name, location, etc.) using the vocabulary defined by the ontology. Each property is composed of (RDF-style) facts, e.g., "Isabella likes dogs", which in turn are composed of terms/concepts such as "likes" and "dogs". To support relative or uncertain information we further allow associating user properties with a score called *support level*, that can captures degrees of user preferences, skills or activity (e.g. that Isabella likes dogs *a lot*, or reads books in the park *frequently*). Such user profiles are typically constructed from user input or derived by the hosting system, e.g., from past user activity [4] or social networks [5]. Support scores may be derived from user-provided ratings or by tools for user evaluation [2], (see details in Sections 2.1 and 6). To support the practically common scenario of incomplete profiles, we allow missing properties to be true or false (*open-world assumption*).

Query Language. Our query language SPARQ-U extends SPARQL with constructs that account for soft constraints, including support level in properties and similarity between profiles. This allows to formulate, customize and combine common user selection operations. Figure 1 shows an example "potential partners" query according to Isabella's input. Briefly, the first parts of the query (lines 2-7) define hard constraints: potential partners must live around Le Marais and like dogs. The other parts define soft/similarity constraints via the `SIMILAR` construct: potential partners should resemble Isabella in terms of profession (lines 8-10), and preferably practice yoga in the park (lines 11-13).

Query Semantics. The introduction of soft constraints on user profiles, requires extensions to the standard SPARQL semantics. There is a large body of work on semantics for SPARQL with constraints [3, 6]; however, in our context the constraints may be defined with respect to *entire user profiles*, each consisting of multiple RDF facts which are in turn associated with numerical scores. Our introduced semantics is then intuitively based on a "hypothetical" taxonomy whose individual nodes roughly correspond to *sets* of facts from the original taxonomy, and edges to a subsumption relation over such sets. This allows us to define similarity of two profiles or property sets, and in turn semantics for the soft constraints, based on the maximal *information content* of a profile subsumed by both. We provide some sanity checks for the introduced semantics and demonstrate via examples that it is fitting for the task.

Query Evaluation. The above mentioned taxonomy over fact-sets is hypothetical since if materialized, its size could be exponential in the size of the input taxonomy (since the number of fact-sets is exponential). In this sense, our semantics is non-operational. *Our main technical result is that, somewhat surprisingly, query evaluation may still be performed in polynomial time.* Briefly, based on the observation that our notion of semantic similarity is monotonic with respect to the subsumption taxonomies, we can restrict our search for the maximal information content to the lowest common ancestors (LCAs) of the input facts/fact-sets/profiles in these taxonomies. Using structural properties of the fact and fact-set taxonomies we gradually compute the LCAs of terms, facts, fact-sets and finally full profiles, using at each stage the LCAs of simpler units. We analyze the correctness and complexity of this algorithm.

Implementation and Experiments. We have implemented the above components in a prototype system, including an optimized version of our query evaluation algorithm. Our optimizations are based, among others, on a *caching mechanism* that leverages properties of SPARQ-U to make inferences based on intermediate results, and thereby avoids a significant portion of the computations.

We have then conducted an extensive experimental study over real data from existing crowdsourcing and social network platforms, demonstrating the effectiveness and flexibility of our approach. We specifically study in isolation the effect of each component of our solution, showing each of them to contribute significantly to the overall output quality. We further compare our solution with non-declarative as well as task-specific algorithms, including machine learning solutions. The results show that interestingly, our framework is able to achieve *better performance in common user selection tasks* while also being generic and declarative.

Paper Outline. We present our data representation and query language in Section 2. In Section 3 we formally define the similarity notions that are used to capture soft constraints, and in Section 4 we propose an efficient algorithm SPARQ-U query evaluation. Our experimental study is described in Section 5. Related work is presented in Section 6 and we conclude in Section 7.

2 MODEL

We next describe our data representation, then present SPARQ-U, an extension of SPARQL for user selection.

2.1 Data Model

Data repositories are represented using RDF for their basic building blocks: sets of *facts* in the form of entity-relation-entity.

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

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 $l \in \Sigma^*$. Facts within a fact-set are concatenated using a dot. We use _ to denote a term placeholder in facts that use less than three terms. In what follows, we refer to terms, facts and fact-sets by the collective name *semantic units*.

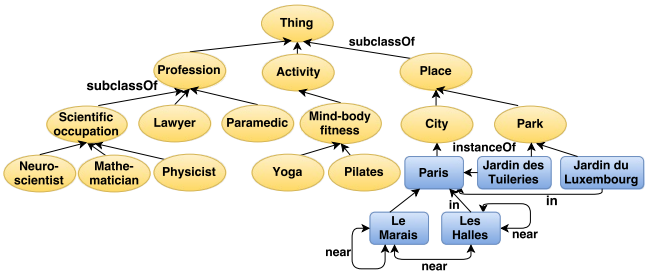


Figure 2: A sample ontology.

Ontology. An ontology is a named fact-set consisting of *general facts* that are not related to a specific user, e.g., {`Le_Marais in Paris`}. Figure 2 displays a partial ontology in the form of a graph, where nodes are entities and edges are relations (some edge labels are omitted). Ontologies typically contain a *hierarchical taxonomy* of concepts, instances and properties, e.g., that Paris is a city and that a city is a place. The ontology in Figure 2 includes a taxonomy of concepts (rounded nodes, connected by `subclassOf` relation) with instances (rectangular nodes, connected by `instanceOf` relation).

User Profiles. Each user profile includes two parts: a basic and an extended profile. The *basic profile* consists of absolute properties about users in the form of a fact-set. E.g., Isabella's profile include the facts {`Isabella livesIn Le_Marais,Paris. Isabella hasProfession Mathematician`}. The *extended profile* of a user describes each property by a separate fact-set that is further associated with a numeric score in $[0, 1]$ called its *support values*. This score may reflect a rating, habit frequency, likelihood and so on, and is useful in capturing user answers on a scale (such as ratings) or (normalized) numeric properties derived by the system [2, 4].

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

Table 1 shows examples of extended profiles. (Ignore, for now, the "Prevalence" and "IC" columns.) The support score associated with Isabella's liking of dogs (first row) may reflect a rating she provided to the website (normalized to $[0, 1]$). The support score of the second fact-set may reflect Isabella's reported frequency of reading books, e.g., every other day (normalized to $[0, 1]$).

2.2 User Selection Queries

We next overview the SPARQ-U query language. Our language extends SPARQL, using its native constructs to select relevant repository parts and apply hard constraints. We further extend SPARQL to support user selection queries. Our three main extensions are:

(e1) Variables allowing to dynamically choose from a pool of knowledge repositories (user profiles).
(e2) Querying fact-sets along with their associated support values
(e3) New soft constraint/similarity constructs

We will exemplify the general syntax and semantics of SPARQ-U, and specifically the above mentioned three main extensions through our running example query (Q_{Isabella} from the Introduction).

Syntactically, a SPARQ-U query is composed of two main types of (optional) clauses: `FROM...WHERE` and `SIMILAR...TO` clauses, corresponding, resp., to hard and soft constraints. Soft constraints form

Fact-set	Support	Prevalence	IC
`Isabella likes Dogs`	1.0	0.8	0.08
`Isabella reads Books`	0.5	0.25	0.48
`Isabella reads Books. __ at Jardin_des_Tuileries`	0.01	0.1	0.72
`Adam likes Dogs`	0.75	0.8	0.08
`Adam practices Mind-body_fitness. __ at Park`	0.28	0.45	0.3
`Adam practices Pilates. __ at Jardin_des_Tuileries`	0.27	0.2	0.55
`Benjamin likes Dogs`	0.5	0.8	0.08
`Benjamin practices Yoga. __ at Park`	0.01	0.2	0.55

Table 1: Extended user profiles.

a generic means of capturing useful notions 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 used throughout the query and as the output. Similarly to SPARQL queries, only users for which, for some assignment of the other variables, all of the following constraint clauses hold, i.e., return non-empty results, are selected.

The clauses standing for hard constraints use the `FROM` construct – extension (e1) over SPARQL – that allows specifying the knowledge repositories to be examined for each assignment of `?u` at each clause. The succeeding `WHERE` keyword is followed by a SPARQL selection over the chosen repository. In lines 2-3 the query selects places near Le Marais from the ontology, which are bound to the variable `?x`. In lines 4-5 the basic profile of `?u` is examined to ensure the candidate user lives near Le Marais. In lines 6-7 the extended profile is examined to ensure that candidates likes dogs.The latter clause demonstrates extension (e2) over SPARQL - allowing to provide a bound (lower, in this case) over support values via the `WITH SUPPORT` optional clause, which serves as a filter over the clause selection results.

The soft constraints clauses (extension (e3)) are the most important new feature of SPARQ-U, whose semantics are explained in Section 3. The `SIMILAR` and `TO` keywords are each followed by a repository or fact-set definition to be compared. To compare only a subset of their facts, a SPARQL selection may be applied over both using the optional `RESTRICTED TO` construct. A bound on the computed similarity score can be defined to filter results through the `WITH SIMILARITY` construct. In the example query, lines 8-10 define a similarity constraint between the basic profiles of `?u` and Isabella, restricted to user professions. Line 10 requires that the similarity score for these subsets is ≥ 0.75. The clause in lines 11-13 considers the desired partner's property "practices yoga at some park" and identifies users whose profile contains a similar fact-set. The maximal score of such fact-set per user is assigned to the variable `?querySim`. This score is then used in line 14 (an optional `ORDER BY` clause) to rank the five most adequate users. I.e., among all users that match the other parts of the query, those who most frequently practice yoga at some park or have a highly similar habit, will be returned. SPARQ-U enables combining the similarity scores of multiple clauses using standard aggregate functions.

3 DEFINING SIMILARITY FOR SOFT CONSTRAINTS

Recall that we have described three main extensions of SPARQ-U w.r.t. SPARQL. The semantics of the first two extensions, (e1) and

(e2), was explained above, and the remaining challenge is to associate a proper semantics with the SIMILAR operator (extension (e3)).

We start by noting that any given taxonomy can be interpreted as a *subsumption relation* between elements or properties: e.g., every occurrence of Mathematician subsumes Scientific_Occupation. This relation may be lifted to subsumption between facts and fact-sets that contain these terms [4, 7] (see Example 3.1). Formally, given two terms t_1, t_2, let $t_1 \leq t_2$ denote the fact that t_1 is *subsumed* by t_2 (e.g., Scientific_Occupation≤Mathematician). The relation $\leq$ forms a partial order over the elements of $\mathcal{E}$ and relations of $\mathcal{R}$. Using the definition of [4], we can "lift" $\leq$ into subsumption partial order over the *compound* semantic units, 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 that is a composing unit of X_1 (e.g. facts in a fact-set, terms in a fact) there exists a corresponding unit P_2 in X_2 s.t. $P_1 \leq P_2$.

Example 3.1. Scientific_Occupation≤Mathematician *lifts to {*Isabella hasProfession Scientific_Occupation*} ≤ {*Isabella hasProfession Mathematician*} in the partial order over facts. Further lifting to fact-sets we have that: {*Isabella hasProfession Scientific_Occupation*}≤{*Isabella hasProfession Mathematician. Isabella hasProfession Paramedic.*}.*

The *prevalence* of a semantic unit X is defined as the fraction of users whose profiles subsume this unit (e.g., the prevalence of {User hasProfession Scientific_Occupation} will be measured as the fraction of profiles including this or more specific facts, e.g., {User hasProfession Neuroscientist}). The smaller the prevalence, the more informative the semantic unit is, as it helps identifying a more specific group of users.

We then define:

Definition 3.2 (Information content (IC)). *Let X be a semantic unit with the prevalence* $\mathrm{p}(X)$*. The IC of X is defined as:* $\mathrm{ic}(X) := -\log\left(\frac{9\,\mathrm{p}(X)}{10} + 0.1\right)$

Our notion of IC follows that of [6]. To ensure a value in $[0, 1]$, our IC formula performs an additional linear scaling on $\mathrm{p}(X)$ from $[0, 1]$ to $[0.1, 1]$. Importantly, this definition is *monotonic* w.r.t. the partial order on semantic units: $X \leq Y$ entails that $\mathrm{p}(X) \geq \mathrm{p}(Y)$, hence $\mathrm{ic}(X) \leq \mathrm{ic}(Y)$. This property will be useful in the sequel.

Example 3.3. *Assume that* 80% *of the profiles include a fact that subsumes a fact f. Then the IC of f is* $-\log(\frac{9}{10} \cdot 0.8 + 0.1) \approx 0.09$.

We are now ready to introduce *IC similarity*. Given two semantic units, we define this measure as the maximal IC of a semantic unit subsumed by both.

Definition 3.4 (IC similarity). *Given a pair of semantic units X, Y within a subsumption partial order $\leq$, we define the IC similarity of X and 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 pairs of fact-sets in these databases.

Example 3.5. *Reconsider Table 1, and denote the fact-set specified in $Q_{Isabella}$, lines* 11-13 *by $\bar{A}$. The common ancestor with maximal IC of $\bar{A}$ and Adam's (resp. Benjamin's) profile is {*User practices Mind-body_fitness. _ at Park*} (resp. {*User practices Yoga. _ at Park*}). The IC values of these ancestors,* 0.3 *and* 0.55 *for Adam and Benjamin, are the similarity scores. This makes sense, as Benjamin's habit is identical to Isabella's request, while Adam only resembles it.*

IC similarity satisfies some natural properties of a similarity function:

Proposition 3.6. $\mathrm{icsim}(\cdot, \cdot)$ *as formulated in Definition 3.4 satisfies the following properties.* $\forall X, Y, Z$*:*
(1) ***Maximum self-similarity.*** $\mathrm{icsim}(X, X) \geq \mathrm{icsim}(X, Y)$
(2) ***Symmetry.*** $\mathrm{icsim}(X, Y) = \mathrm{icsim}(Y, X)$
(3) ***Fixed value range.*** $\mathrm{icsim}(X, Y) \in [0, 1]$
(4) ***IC monotonicity.*** $X \leq Y \Rightarrow \mathrm{icsim}(X, Z) \leq \mathrm{icsim}(Y, Z)$

Accounting for support scores. The next step is accounting for support scores. We first provide a formula for comparing the support of two semantic units (using, for uniformity, the same normalization as in Def. 3.4), then use it to compare two extended profiles.

Definition 3.7 (Support similarity). *Given a semantic unit X, we compute the similarity of support values assigned to it by two support functions $\mathcal{S}, \mathcal{S}'$ as*

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

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

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

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 we weight the similarity of support score pairs by the IC of the semantic unit they annotate, to give higher weights to more specific fact-sets. E.g., 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 a 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.

The following proposition then holds:

Proposition 3.8. $\mathrm{supsim}(\cdot, \cdot)$ *as formulated in Def. 3.7 satisfies properties (1)-(3) of Prop. 3.6, and the following.*
(5) ***Support Monotonicity.*** *For every three extended profile databases $D = \langle \mathcal{A}, \mathcal{S} \rangle$, $D' = \langle \mathcal{A}, \mathcal{S}' \rangle$, $D'' = \langle \mathcal{A}, \mathcal{S}'' \rangle$, if for every fact-set $X \in \mathcal{A}$ it holds that $|\mathcal{S}(X) - \mathcal{S}'(X)| \leq |\mathcal{S}(X) - \mathcal{S}''(X)|$, then it should follow that* $\mathrm{supsim}(D, D') \geq \mathrm{supsim}(D, D'')$.

Example 3.9. *Consider again Table 1, and the fact-set $\bar{A}$ specified in $Q_{Isabella}$ (lines 11-13). The only fact-sets in the intersection of the databases are {*Adam practices Mind-body_fitness. _ at Park*} for Adam and resp. {*Benjamin practices Yoga. _ at Park*} for Benjamin. These fact-sets have a score of* 0.28 *for Adam and resp.* 0.01 *for Benjamin, and by definition, a score 1 in $D_{\bar{A}}$. Then,*

supsim$(0.28, 1) \approx 0.126$ *and resp.* supsim$(0.01, 1) \approx 0.004$ *and these are also the final support similarity scores. Note that the much lower frequency of Benjamin's habit results here in a much lower support similarity in comparison with Adam's.*

Finally, the combined similarity – used for the SIMILAR clause of SPARQ-U – is the *product* of semantic and support similarities, i.e., for two extended profile databases D, D' : sim$(D, D') :=$ icsim$(D, D') \cdot$ supsim(D, D'). Using the product here ensures that for support scores fixed to 1, and specifically basic user profiles, the combined score coincides with IC similairty. Moreover, the next corollary follows from Propositions 3.6 and 3.8.

Corollary 3.10. *The combined similarity* sim$(\cdot, \cdot)$ *satisfies constraints (1)-(5) from Props. 3.6 and 3.8.*

Example 3.11. *Following Examples 3.5 and 3.9, the combined similarity scores for Adam and Benjamin are, resp.,* $0.126 \cdot 0.3 = 0.038$ *and* $0.004 \cdot 0.55 = 0.002$. *Intuitively, Adam's habit is only* slightly *less (semantically) similar to Isabella's specification, and so his* much *higher frequency of practicing it made him overall a more desirable partner.*

Other similarity metrics may be plugged into our framework as well, but we shall empirically show that the use of our measure outperforms state-of-the-art alternatives for user selection.

4 QUERY EVALUATION

Selections in SPARQ-U (within RESTRICTED TO and WHERE clauses) coincide by design with SPARQL selections. We assume henceforth that these selections are performed by a black-box SPARQL engine, that may employ any state-of-the-art indexing, query optimizations and RDF reasoning (see Section 7). The implementation of some of our extensions over SPARQL, namely, restrictions on support values and repository specification as well as the support similarity component of soft constraint evaluation, follow from their respective definitions, and are thus not detailed here.

We focus here on two crucial aspects of the SPARQ-U engine. The first is the computation of *semantic similarity* as a part of soft constraint evaluation. The second is a *cross-variable-assignment optimization*, which applies globally over the different query clauses including soft constraints in contrast with the SPARQL black-box that can only optimize SPARQL selections within the query locally.

4.1 Similarity Computation

Our contribution here is a novel PTIME algorithm for semantic similarity computation, and the analysis of its complexity. The tractability of this task is non-trivial, since it requires finding the highest IC of an ancestor (in a hypothetical taxonomy consisting of all fact-sets from the original taxonomy) common to two semantic units. The size of the hypothetical taxonomy and consequently the number of ancestors to consider may be exponential in the size of the input taxonomy. Indeed, some other fact-set–related tasks were shown to be intractable [4, 7]; yet surprisingly, we show here that semantic similarity can be efficiently computed while *avoiding the materialization* of the fact-set taxonomy.

Algorithm 1 outlines the functions required for efficiently computing the semantic similarity of each input type. Its main function, icsim$(\cdot, \cdot)$, employs by-case treatment to compute the similarity of

Function icsim(X, Y)
1 **if** $X = (\mathcal{A}, \mathcal{S}), Y = (\mathcal{A}', \mathcal{S}')$ *are extended profile databases* **then**
  return $\max_{A \in \mathcal{A}, A' \in \mathcal{A}'}$ ic(FactSetLCA(A, A'));
2 **else if** X, Y *are fact-sets* **then**
  return ic(FactSetLCA(X, Y))
3 **else if** X, Y *are facts* **then**
  return $\max_{f \in \text{FactLCA}(X, Y)}$ ic(f)
4 **else return** $\max_{t \in \text{LCA}(X, Y)}$ ic(t);
Function FactLCA(f, f')
5 Let $f = \langle e_1, r, e_2 \rangle$ and $f' = \langle e_1', r', e_2' \rangle$;
6 **return** LCA$(e_1, e_1') \times$ LCA$(r, r') \times$ LCA(e_2, e_2');
Function FactSetLCA(A, B)
7 $L \leftarrow \emptyset$;
8 **for** $f \in A, f' \in B$ **do**
9 $L \leftarrow L \cup$ FactLCA(f, f');
10 **for** $f = \langle e_1, r, e_2 \rangle, f' = \langle e_1', r', e_2' \rangle \in L$ **do**
11 **if** $e_1 \leq e_1' \wedge r \leq r' \wedge e_2 \leq e_2'$ **then** $L \leftarrow L - \{f\}$;
12 **return** L;

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

extended profiles, fact-sets, facts and terms, assuming an implementation of ic$(\cdot)$ for any semantic unit. While the flow of the algorithm is simple, it is the analysis of its components that guarantees an overall low complexity.

The connection to LCA computation. Recall that by Def 3.4, the IC similarity of two semantic units is the maximal IC of their common ancestors. The following observation allows us to focus only on a small subset of ancestors.

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

Similarity computation is thus connected to the well-studied problem of finding the lowest common ancestor (LCA) in a general DAG, representing the subsumption partial order. It is left to show that the LCA computation we employ for each type of semantic unit is correct. For terms, LCAs can be computed using standard modules over the term taxonomy, given as a part of the ontology. Thus, Algorithm 1 assumes an efficient implementation of LCA(t, t'), namely, searching the LCAs of two terms t, t' in a general DAG representation of the term taxonomy. As explained, to ensure efficiency/tractability, we wish to avoid materializing taxonomies for facts/fact-sets. This is achieved by our costume implementation of FactLCA and FactSetLCA, which uses LCA(t, t') as a black-box.

Facts LCA computation. 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, namely, LCA$(e_1\ r_1\ e_1', e_2\ r_2\ e_2')$ is of the form $e\ r\ e'$ where $e \in$ LCA(e_1, e_2), $r \in$ LCA(r_1, r_2) and $e' \in$ 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 6 of FactLCA(f, f')).

Lemma 4.2. *Given two facts f, f',* FactLCA(f, f') *computes all and only their LCAs.*

Example 4.3. *Assume that* LCA(`likes`, `practices`) = *{*`relatedTo`, `knows`*}, and that* LCA(`Pilates`, `Yoga`) = *{*`Mind-body_fitness`*}. The LCAs of the facts f =*{`User practices Pilates`*}, f' =*{`User likes Yoga`*} consist of all combinations of term LCAs, i.e., {*`User relatedTo Mind-body_fitness`*} and {*`User knows Mind-body_fitness`*}. Note that f, f' may have many other common ancestors, e.g., {*`User knows Thing`*}; but by Lemma 4.2, each such ancestor is subsumed by some LCA and thus can only have a lower IC.*

LCA computation for fact-sets. The FactSetLCA function computes the LCA of fact-sets. Importantly, using the partial order over fact-sets properties, we can prove that unlike the cases of terms or facts, the following lemma holds.

Lemma 4.4. *There exists* exactly one LCA *for every fact-sets-pair A, B.* FactSetLCA(A, B) *computes this LCA.*

Moreover, we can characterize this fact-set as the set of non-redundant facts that are the ancestors of some facts-pair, $f \in A$ and $f' \in B$. To compute this set, the function FactSetLCA(A, B) uses FactLCA(f, f') for each such pair, it takes their LCAs union (lines 8-9) and then removes redundant facts (that are implied by other facts, lines 10-11).

Example 4.5. *Consider the LCA computation for the fact-sets $A = \{f.f''.\}$ =*{`User practices Pilates._ at Park`*} and $B = \{f'\}$ =*{`User likes Yoga`*}. We compute the union of LCAs for the fact pairs f, f' and f'', f' (lines 8-9 of* FactSetLCA*). Example 4.3 has shown the LCAs of f, f'; and assume that* FactLCA(f'', f') =*{$\bot$ $\bot$ $\bot$}, where $\bot$ is a "root" term subsumed by any term. Namely,* FactLCA(f'', f') *is subsumed by* FactLCA(f, f') *and should be removed (lines 10-11). The resulting fact-set LCA is thus {*`User relatedTo Mind-body_fitness. User knows Mind-body_fitness`*}. It is unique, although the facts have multiple LCAs.*

Analysis. The following proposition summarizes the complexity bounds for IC similarity computation, according to known complexity bounds for LCA computation of materialized taxonomies [8], and assuming that IC computation of a given semantic unit is done in O(1) (proof omitted). In what follows, $w[\Psi]$ denotes the *width of the term taxonomy* Ψ, namely, the maximum size of a set of incomparable terms.

Proposition 4.6. *After PTIME processing of the term taxonomy, the complexity of computing IC similarity of X, Y is:*

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

4.2 Caching mechanism

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

Observation 4.7. *Let* $\mathbf{C}$ *be the set of* `SIMILAR ... TO` *clauses over fact-sets (with implicit support 1) in some query.*

- *If some variable assignment φ, for* some $\mathrm{C} \in \mathbf{C}$*, is below the similarity threshold Θ, then every more* general *assignment $\psi \leq \varphi$ will be below Θ for $\mathbf{C}$.*
- *If some variable assignment φ, for* every $\mathrm{C} \in \mathbf{C}$*, exceeds the similarity threshold Θ_{C}, every more* specific *assignment $\psi \geq \varphi$, for every $\mathrm{C} \in \mathbf{C}$, will exceed Θ_{C}.*

Thus, by caching the results of maximally-specific (resp., -general) assignments that are below (resp., above) the threshold, one can avoid many redundant computations. This observation can then be used as follows. The results for each considered assignment will be stored in a cache. Whenever a new assignment is encountered, it will first be checked whether it falls within one of the items in the observation above. If it falls within the first item, it can be discarded. If it falls within the second, there is no need to compute the similarity for this assignment for clauses that involve fact-sets and whose results are not used in ordering the query output, since the assignment is guaranteed to satisfy these constraints. We ignore clauses that are affected by support values, since their similarity is not IC monotone. We leave the theoretical analysis of this improvement for future research; yet, our experiments show the importance of this optimization in practice.

5 IMPLEMENTATION AND EXPERIMENTS

We developed a prototype engine for evaluating SPARQ-U queries. This engine uses preprocessing of the term taxonomy to speed up LCA computation. A further speed-up is achieved by distributing over multiple cores the similarity computation between different user pairs. Finally, it employs a dedicated caching mechanism to compactly store intermediate results and avoid unnecessary computations, based on Observation 4.7. The prototype is implemented in Java, using Apache Jena library (https://jena.apache.org/.) for a SPARQL engine.

Since general-purpose user selection framework has not been studied before, no standard benchmark over which one could test the capabilities of SPARQ-U was available. We thus constructed several benchmark datasets using real-world data including Q&A platforms such as Stack Overflow (SO) [13] and social networks such as AMiner [14], an academic social network. These datasets provide natural scenarios for user selection, which we captured via SPARQ-U queries as exemplified below.

5.1 Experimental Setup

As we describe next, we experimented with 5 different datasets from two domains. In all datasets, the domain ontology was constructed from DBpedia [15]. From space constraints, we present here only the results obtained for two representative datasets.

Q&A datasets. We examined 3 Q&A platforms: StackOverflow (SO), Mathematics and Ask-Ubuntu [13]. As the results over the latter two demonstrated similar trends to SO, we focus here on SO

Competitor	Description	Goal	Implementation
SPARQL	Standard SPARQL, i.e., only using hard constraints without support thresholds.	Evaluating the impact of soft constraints.	Ignore SIMILAR clauses and support thresholds in SPARQ-U queries.
No Support	The same SPARQ-U queries, but support values are ignored.	Evaluating the impact of support values and our support similarity measure.	Ignore hard constraints with support thresholds, use only semantic similarity in query evaluation.
No Fact-sets	The same SPARQ-U queries, but facts are considered individually and co-occurrence of facts is ignored.	Evaluating the impact of fact-sets, specifically in similarity computation.	Alter the database to include only facts.
Lin [9]	A commonly used node semantic similarity measure for ontologies.	An alternative similarity measure.	Use the taxonomy to compute similarity scores of relevant terms, return the average.
SimRank [10]	A node similarity measure, commonly used for link prediction in in social networks.	An alternative similarity measure.	Represent profiles as an (RDF) subgraph, compute the average similarity of relevant nodes.
PathSim [11]	A node similarity measure that further considers the labels on path edges.	An alternative similarity measure.	Represent profiles as an (RDF) subgraph, compute the similarity of relevant nodes.
LINE [12]	A representative *node embedding* technique for converting a subgraph to a vector, used in classification and link prediction.	An alternative similarity measure.	Relevant profile part are converted to vectors, over which cosine similarity is applied.
SVM	Users are selected using SVM, a common machine learning method for classification.	An alternative similarity measure.	Context-dependent translation of profiles to vectors, over which SVM is trained and applied.

Table 2: Baseline algorithms

as a representative example. SO is a popular Q&A platform, where users' questions are associated with a set of tags reflecting their topics. Each user has a profile with properties such as name, reputation score, etc. We collected over 900K questions in 300 popular topics, asked by over 175K users. We then gathered their personal profiles, and collected more than 2.3M of their previous answers, to assemble a coherent subset. We constructed the basic profiles from the extracted profiles. The extended profile of a user was constructed from the tags to which she contributed, with a support value reflecting the portion of her contribution to that tag. The resulting database consists of 20M entities.

Social networks. We studied two social network datasets, AMiner, an academic social network and Pokec [16], a social network containing user profiles and friendship relations. For brevity, we give detailed results only for Aminer and mention briefly the results for Pokec, which exhibited similar trends. We extracted from AMiner the data of 1.7M computer science authors between $2005-2015$ (over 2M papers). The basic profile of an author consists of her affiliation, h-index, fields of interest, and publication data (e.g., title and year). The extended profile records her co-authorships (with support values reflecting the fraction of her publications with each co-author), and publication venues (with support values reflecting the fraction of her publications in each venue). The resulting database consists of 16M entities.

Alternative algorithms. We compare our results to multiple alternatives, as detailed in Table 2. The first three baselines are restricted variants of SPARQ-U, serving to examine the contribution of each of the framework's components. Particularly, these baselines help to asses the contribution of soft constraints, by ignoring all SIMILAR clauses; support scores, by using only semantic similarity; and the use of fact-sets, by altering the database to include only facts. As an example for the latter on Aminer, a fact-set describing the frequency of a researcher publishing with two co-authors in some venue, is decomposed to separate facts, each capturing one co-author or the venue. As a result, the similarity metric is unable to account for interdependencies in the profile data, e.g., that two authors published with the same co-author in a similar venue.

Further, we considered five existing similarity measures as alternative to our notion of soft constraints (for fairness, hard constraints are still used for initial filtering). These measures are all top preforming representative examples for different approaches for similarity assessment (see Section 6). Lin, SimRank and PathSim are all *graph node similarity measures*, and hence require an adaptation to our setting: we convert the profiles to a labeled graph form (similarly to Figure 2), ignoring fact-sets and where support scores serve as edge weights. Since similarity should be evaluated for subgraphs (i.e. profile parts) we added "dummy nodes" representing the basic or extended profile of a user. The SimRank and PathSim scores are computed on these additional nodes, and for Lin, we compute the average similarity between nodes pairs in the subgraphs (the node pairs are chosen to have the same relation to the user). In contrast, SVM and LINE use vector representations of the profiles, which better account for the interdependencies in profile data. These representations were achieved using LINE, also in the SVM baseline. To implement SimRank, we used an efficient approximation technique suggested in [17], and used the scikit-learn implementation for SVM. Additional similarity measures were examined, including the cosine and Jaccard measures used in collaborative filtering. Due to their inferior results, they are omitted from presentation.

All experiments were conducted on a Linux server with 24 cores and 96GB memory.

5.2 Qualitative Experiments

SPARQ-U, being a declarative framework, is capable of capturing a large variety of queries. In this set of experiments we focused on specific scenarios which (A) are typical user selection scenarios,

```
SELECT ?u
FROM basic-profile(?u) WHERE
    {?u creationDate ?d. Filter (?d < 19.6.2015) }
SIMILAR basic-profile(?u) TO basic-profile(J_Doe)
    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 30
```

Figure 3: User selection for Stack Overflow question.

```
SELECT ?u
FROM extended-profile(?u) WHERE
    {?u collaboratedWith ?v.}
FROM extended-profile(?v) WHERE
    {?v collaboratedWith J_Doe.}
SIMILAR basic-profile(?u) TO basic-profile(J_Doe)
    WITH SIMILARITY AS ?profSim > 0
SIMILAR extended-profile(?u) TO extended-profile(J_Doe)
    WITH SIMILARITY AS ?topicSim > 0
ORDER BY AVG(?profSim, ?topicSim) LIMIT 30
```

Figure 4: User selection for AMiner (for the user J Doe).

and (B) for which there exists a *ground truth* that enables evaluating the correctness of the results.

Context-based recommendation. In this experiment we selected users for a given task, as follows.

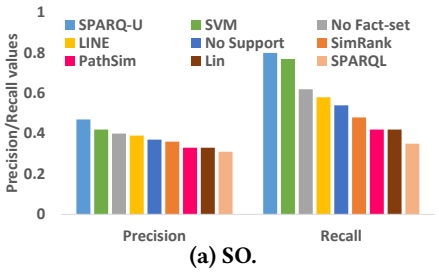
- **Task:** find users to answer a given question.
- **Dataset:** SO (Mathematics and Ask Ubuntu are omitted.)
- **Our query:** select the top-30 experts for the question's topics who are also similar to the asker.
- **Adjustments:** use only data generated before the question posting time.
- **Evaluation:** for 500 random questions, the precision and recall w.r.t. the ground truth.

The corresponding query (constructed automatically for one of the 500 questions) is demonstrated in Figure 3.

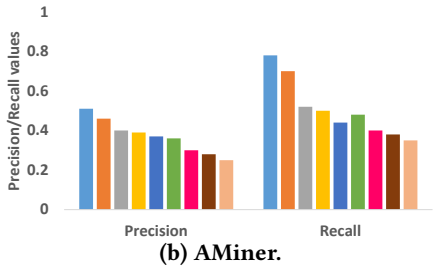
Figure 5(a) shows the average precision and recall w.r.t. the real answerers of each question on the SO dataset. Observe that the average recall achieved by SPARQ-U is 0.8, i.e., in most cases, the majority of answerers were successfully selected by our simple query. This is extremely positive given the task difficulty: the true answerers may be just a few out of many users that can potentially answer the question. The average precision of SPARQ-U is 0.48, the best among all alternatives. We note that here, precision values cannot be much higher, since the set of all answerers is often smaller than 30 (12 on average). Moreover, all alternatives may suffer from lower precision by selecting adequate users who have "missed" the question. Similar trends were obtained when varying the number of selected users, and also for the two other platforms, where again, SPARQ-U outperformed all competitors.

Among the alternative algorithms, the best results were obtained for ones that account for support scores and profile data dependencies, most notably SVM. However, none of these algorithms accounts for all the features of profile data, supported by SPARQ-U. For instance, SVM and LINE do not account explicitly for data semantics. Or, SimRank (which performs better in other tasks) is less adequate for assessing expertise areas. Moreover, unlike the customizable SPARQ-U, the SVM was specifically trained for this scenario and cannot be trivially adapted to other scenarios.

The cases where true answerers were not found by SPARQ-U typically occurred due to missing data: in over 70% of cases, either



(a) SO.



(b) AMiner.

Figure 5: Quality assessment

the asker or the answerers were new users with little past activities. Generally, we observed that the performance of *all algorithms* deteriorates as the profiles of asker and answerers contain less information, but the differences between algorithms are preserved (SPARQ-U achieves the best performance). An application owner may address such "cold start" problems by actively asking new users for missing data, or by using data from external sources.

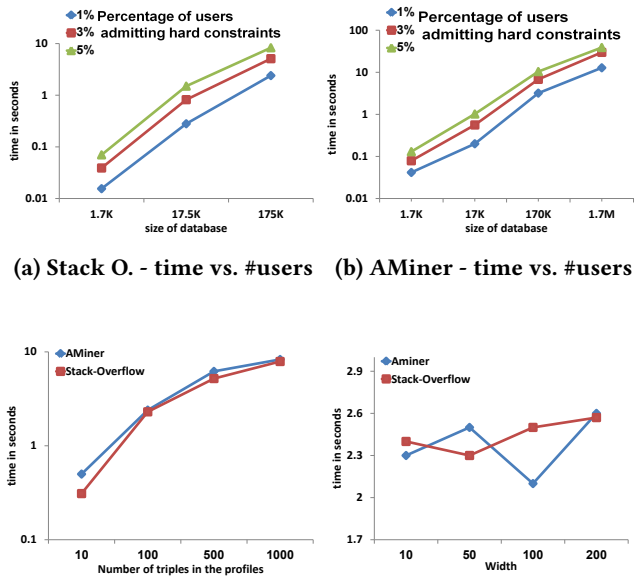
Link prediction. Next, we focus on predicting links in social networks, as follows.

- **Task:** identify potential collaborators/friends.
- **Datasets:** AMiner and Pokec.
- **Our query:** collaborators of collaborators (resp. friends of friends) of a user that also resemble her profile.
- **Adjustments:** use data up to 2013 for querying, and later as ground truth to be predicted.
- **Evaluation:** (i) average precision and recall of top-30 potential collaborators/friends, and (ii) for 27 authors, the precision of top-10 selected authors in a user study.

See Figure 4 for an example query on AMiner dataset. To obtain the ground truth, we split the data into two parts: one contains publications up to 2013, and the second contains publications in later years (ground truth). To mitigate cold start issues, we focused on authors with ≥ 10 publications. In AMiner, the SPARQL baseline orders users by their h-index. Similar adjustments were made for Pokec, e.g., we selected 1K users, randomly removed $\frac{1}{3}$ of their friends and used those relations as the ground truth.

The precision and recall of all baselines, averaged over 150 AMiner authors, are plotted in Figure 5(b). Note that the SPARQ-U query achieved the highest precision and recall values (of 0.51 and 0.78). Similarly, SPARQ-U achieved best results on the Pokec baseline as well, although the marginal advantage of SPARQ-U in the Aminer setting is larger than in Pokec, due to richer semantics of the data (e.g., a deeper taxonomy). Moreover, all approaches perform better on Pokec than on AMiner. This stems from the availability of more activity data in the former dataset.

Among all baselines, SimRank was the closest one to SPARQ-U, yet as demonstrated, it does not perform as well in other scenarios.



(a) Stack O. - time vs. #users **(b) AMiner - time vs. #users**

(c) Time vs. profile size **(d) Time vs. ontology width**

Figure 6: Execution times.

SVM, which previously performed well, exhibits less impressive results and is outperformed by LINE – intuitively, since LINE is able to capture latent semantic relations. However, measures such as PathSim and Lin that do consider semantic relations, achieved poor performance here. This is mostly due to the fact that they are designed to capture *node similarity* rather than full profiles.

AMiner user study. We recruited 27 researchers and for each of them, executed the SPARQ-U query selecting 10 possible collaborators, excluding past co-authors (assuming researchers would not object to collaborate with past co-authors), and asked them to estimate how many of the selected users may be potential future collaborators (precision). Note that in this study we could not estimate the recall (this would require all potential future co-authors). Yet our simple, generic query obtained some interesting results: On average, the precision was 0.53. In particular, 89% of the participants found at least one recommendation relevant, and 85% of the them found at least 3 out of 10 relevant.

We further analyzed the false positives for this study. In particular, only in 3 out of the 27 cases, the authors found the recommendations irrelevant. These cases appear to be a consequence of incomplete data (e.g., missing facts in their profiles) or unclean data (e.g., authors with profiles not reflecting their current fields of interest). There exists previous work on dealing with such issues, e.g., by enriching the data using tools for building profiles from social networks [5].

5.3 Scalability Evaluation

The queries in our quality assessment experiments had an average execution time of less than 3 seconds. We further synthetically tune different parameters of our settings by modifying: (1) the percentage of users admitting hard constraints; (2) the profiles size and (3) the ontology shape. For comparison, in the SO setting, the average percentage of users admitted the hard constraints was ∼1%, each profile contained 100 RDF triples on average, and the average width/depth of the ontology part used was 50 and 7, resp.

We briefly report the running times of alternative algorithms. These can be divided into two classes: approaches that require a preprocessing (SVM, LINE, SimRank) and ones that can directly be computed on the relevant subgraph induced by the hard constraints (PathSim, Lin). The performance of the former were significantly inferior in comparison with SPARQ-U. The preprocessing per query (sampling walks for SimRank [17], training a model for SVM and embedding the subgraphs for LINE) alone required above one minute per query, and overall could not compare with SPARQ-U. Execution times for the second class were similar to SPARQ-U for the tested settings (differences smaller than 0.2 seconds). However, these alternatives exhibit a linear time growth in the profile size, as opposed to SPARQ-U, which, as we show, facilitates a scalable computation thanks to our caching. Figures 6(a) and 6(b) depict the average running time for SO and AMiner queries, w.r.t. the database size, and for different ratios of users admitting the hard constraints. Figure 6(c) depicts the average running times for these datasets and fixed profile sizes between 10 and 1000 facts. The sublinear growth in execution times demonstrates the effectiveness of our caching mechanism. Since this mechanism leverages intermediate results to save computations, its effect is increased when the number of users or their profile size grows, and hence the overlap between their profiles, increases. In comparison, the execution time without caching (omitted from the graphs) increases linearly with the number of users, and was approx. 5 times slower for all tested settings.

Last, we examined the dependency of execution times on the ontology shape. The results show no significant increase (see Figure 6(d) for width; similar results were observed for height), even though our theoretical analysis predicts a polynomial upper bound. The reason is that our preprocessing guarantees a constant time computation of term LCAs, and the upper bound is met only in cases where the number of LCAs is large, which is evidently rare in practice.

6 RELATED WORK

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

Recommender systems leverage similarities between users (and items) in order to recommend users (or items) to users [3]. In social networks, recommendations are computed by techniques of *link prediction*, which rely on analysis of the network structure [10]. However, none of these works consider a similarity measure that can fully account for the rich semantics of our model, since our lifting of IC similarity to general facts and fact-sets is novel.

The field of *expert finding* is concerned with selecting users (individuals or a team) by an assessment of their level/areas of expertise (e.g., [1, 18]). A related problem is *quality control*, where users are selected based on (an estimation of) their performance in previous tasks (e.g., [2]). Some contributions of these works are orthogonal to ours: we can store derived expertise scores as the support of fact-sets in user profiles, and use them to compute the *relevance of a user to a new context*. In turn, our ranking of users can then be used in task assignment. Our solution further differs from these studies by providing a generic, declarative framework for user selection.

Other query languages have been developed for user selection, mostly in the context of *social network analysis* (e.g., [19, 20]). Another language that focuses on querying user preferences is presented in [21], and [22] introduced user-defined path searching for relationships between users. We note that SPARQ-U only uses selection constructs that are available in SPARQL, so some of the dedicated constructs of the aforementioned languages (e.g., group identification) may be used to further enrich our language. However, these languages do not incorporate semantic knowledge or soft constraints, features which our experiments indicate to be vital in various scenarios such as expert finding.

Our similarity measure is related to other similarity measures for graph-like or semi-structured data (e.g., information network, XML and RDF). Existing approaches can be divided into three main classes: (i) structural similarity measures, where object similarity is often based on the graph structure [10, 11] or other edit distance-based methods (e.g., [23]); (ii) semantic measures, typically involving ontological knowledge [6, 9, 24, 25], which also serves as the basis of our metric, and (iii) Measures that combine structural and semantic properties. These include techniques such as edge matching and path similarity, often dedicated to a specific task (e.g., [26] for XML documents similarity), and, more recent, state-of-the-art approaches based on *representation learning*, such as node embedding [12]. The latter converts subgraphs to representative vectors over which similarity can be computed. In our experimental study we examined state-of-the-art representatives of all of these approaches, showing that the competitors were outperformed by SPARQ-U in terms of quality, scalability and flexibility. In particular, as mentioned in Section 5.1, the first two approaches require non-trivial adaptations to allow comparing complex structures like fact-sets with support values (e.g., as we apply in our experiments). Moreover, many practical semantic measures are not domain-independent, but rather tailored to a specific context such as bioinformatics research [24]. Representation learning can account implicitly for graph substructures, but does not use explicit semantic information such as taxonomies, and requires a time-consuming preprocessing step.

Other studies that considered semantic similarity include tasks like *entity matching* in ontology alignment [27, 28] and *similarity search*, which is used in evaluating imprecise or relaxed queries [29]. While some of these methods resemble ours in measuring similarity of RDF subgraphs and using taxonomical data, their goal is different: they aim to identify different representations of the *same* entity/query answer. Thus, in contrast to our work, quantifying similarity of *distinct* entities (e.g., French mathematicians and Italian physicists) is not targeted. Here too, user profiles that involve fact-sets and support are not supported. Specifically, comparing large profiles requires an efficient similarity computation, hence, e.g., the NP-hard metric of [29] is not suitable for our setting.

7 CONCLUSION

This work presents a declarative framework that allows specification of customized user selection criteria. Its SPARQL-based query language has embedded constructs for capturing hard and soft constraints over (relevant parts of) user profile. Dedicated algorithms and optimizations allow for efficient query processing. Our experiments on real-life data indicate the effectiveness and usefulness of our approach. Interesting directions for future work include extending the language to e.g. include *diversification*, *clustering* or *classification* constructs, as well as further optimizations.

Acknowledgment. This work has been partially funded by the Israel Innovation Authority, the Binational US-Israel Science foundation, Len Blavatnik, the Blavatnik Family foundation, and by the Israel Science Foundation (grants No. 1157/16 and 639/17).

REFERENCES

[1] H. Rahman, S. Thirumuruganathan, S. B. Roy, S. Amer-Yahia, and G. Das, "Worker skill estimation in team-based tasks," *PVLDB*, 2015.
[2] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng, "iCrowd: An adaptive crowdsourcing framework," in *SIGMOD*, 2015.
[3] 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.
[4] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech, "OASSIS: query driven crowd mining," in *SIGMOD*, 2014.
[5] 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.
[6] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *IJCAI*, 1995.
[7] A. Amarilli, Y. Amsterdamer, and T. Milo, "On the complexity of mining itemsets from the crowd using taxonomies," in *ICDT*, 2014.
[8] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin, "Finding least common ancestors in directed acyclic graphs," in *SODA*, 2001.
[9] D. Lin *et al.*, "An information-theoretic definition of similarity." in *Icml*. Citeseer, 1998.
[10] G. Jeh and J. Widom, "Simrank: A measure of structural-context similarity," in *SIGKDD*, 2002.
[11] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *PVLDB Endowment*, 2011.
[12] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*. International World Wide Web Conferences Steering Committee.
[13] "Stack exchange website," 2017, https://stackexchange.com/.
[14] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *SIGKDD*, 2008.
[15] "About DBpedia," 2017, http://wiki.dbpedia.org/about.
[16] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.
[17] B. Tian and X. Xiao, "Sling: A near-optimal index structure for simrank," 2016.
[18] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das, "Task assignment optimization in knowledge-intensive crowdsourcing," *PVLDB*, 2015.
[19] M. Martın, C. Gutierrez, and P. Wood, "SNQL: A social networks query and transformation language," in *AMW*, 2011.
[20] R. Ronen and O. Shmueli, "SoQL: A language for querying and creating data in social networks," ser. ICDE, 2009.
[21] M. Jacob, B. Kimelfeld, and J. Stoyanovich, "A system for management and analysis of preference data," *PVLDB*, 2014.
[22] J. Liang, D. Ajwani, P. K. Nicholson, A. Sala, and S. Parthasarathy, "What links alice and bob?: Matching and ranking semantic patterns in heterogeneous networks," 2016.
[23] J. Tekli, R. Chbeir, and K. Yetongnon, "Structural similarity evaluation between xml documents and dtds," in *WISE*. Springer, 2007.
[24] G. K. Mazandu, E. R. Chimusa, and N. J. Mulder, "Gene ontology semantic similarity tools: survey on features and challenges for biological knowledge discovery," *Briefings in bioinformatics*, 2016.
[25] N. Seco, T. Veale, and J. Hayes, "An intrinsic information content metric for semantic similarity in WordNet," in *ECAI*, 2004.
[26] S. Amer-Yahia, L. V. Lakshmanan, and S. Pandit, "Flexpath: flexible structure and full-text querying for xml," in *SIGMOD*. ACM, 2004.
[27] V. Rastogi, N. Dalvi, and M. Garofalakis, "Large-scale collective entity matching," *PVLDB*, 2011.
[28] F. M. Suchanek, S. Abiteboul, and P. Senellart, "Paris: Probabilistic alignment of relations, instances, and schema," *PVLDB*, 2011.
[29] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao, "Semantic SPARQL similarity search over RDF knowledge graphs," *PVLDB*, 2016.