

# A System for Retrieving Top-k Candidates to Job Positions

Umberto Straccia<sup>1</sup>, Eufemia Tinelli<sup>2</sup>, Simona Colucci<sup>2</sup>, Tommaso Di Noia<sup>2</sup> and Eugenio Di Sciascio<sup>2</sup>

ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy  
SisInfLab-Politecnico of Bari, via Re David 200, I - 70125 Bari, Italy  
straccia@isti.cnr.it

{e.tinelli,s.colucci, t.dinoia, disciascio}@poliba.it

## 1 Introduction

Nowadays more and more companies choose to employ e-recruiting systems to automatically assign vacant job positions. Such systems allow for electronically managing the whole recruitment process, reducing the related cost. E-recruiting systems efficiency is therefore significantly affected by the efficacy of the framework underlying the match between recruiters requests and candidates profiles stored. In available skill management systems, information about candidates employment and personal data as well as certifications and competence is usually modeled through relational databases with customized and structured templates. Nevertheless, even though a Data Base Management System (DBMS) is surely suitable for storage and retrieval, relational query languages do not allow for the flexibility needed to support a discovery process as complex as recruitment. The `order by` statement and the `min` and `max` aggregation operators are generally used to retrieve the *best tuples* but, in real scenarios, there are no candidates that are better than the others ones w.r.t. every selection criteria. Moreover, if exact matches are lacking, worse alternatives must be often accepted or the original requirements have to be negotiated for compromises.

Logic-based techniques and technologies permit to make more efficient and flexible the recruitment process. The system we present here automatically performs a match-making process between available candidate profiles and vacant job positions according to mandatory requirements and preferences provided by a recruiter. In order to perform it, we need a language suitable for data intensive applications with a good compromise between expressiveness and computational complexity. The system performs non-exact match through top-k retrieval techniques: it uses a match engine which performs top-k queries over a DLR-lite [1] Knowledge Base (KB) providing a ranked list of candidates.

In the remaining we proceed as follows: Section 2 motivates our proposal, also by comparing it with relevant related work; Section 3 shortly recalls language and algorithms we adopted. In Section 4 the proposed system is presented with particular reference to the evaluation of its performance. Finally, conclusions close the paper.

## 2 Why another system for HRM

Currently, several solutions for talent management<sup>1</sup> and e-recruitment are available on the market. Most of them are complete enterprise suites supporting human resource management, including solutions that, even though improving the recruitment process by means of innovative media and tools, do not bring a significant novelty charge with them. Available solutions in fact exploit databases to store candidate personal and employment information, and do not ground on a logic-based structure.

One of the few logic-based solutions to recruitment and referral process is, to the best of our knowledge, STAIRS<sup>2</sup>, a system in use at US Navy Department allowing to retrieve referral lists of best qualified candidates w.r.t. a specific mansion, according to the number of required skills they match. The commercial software supporting STAIRS is RESUMIX<sup>3</sup> an automated staffing tool making use of artificial intelligence techniques and adopted only as an internal tool. The system allows also to distinguish skills in *required* and *desired* ones in the query formulation: all required skills must be matched by the retrieved candidate, differently from *desired* ones.

We propose here a logic-based solution to recruitment process, allowing for distinguishing in required and preferred skills and exploiting a *Skills Ontology*, designed in (a subset of) OWL DL, to model experiences, education, certifications and abilities of candidates. The system translates a user request into a union of conjunctive queries for retrieving the best candidate to cover a given position. Hence, in order to perform a match both the user request and candidates CVs (which we generally call *profiles*) are defined w.r.t. the same Skills Ontology.

In order to understand the advantages of our system w.r.t. not logic-based solutions, we provide here a tiny example: imagine you are a recruiter, with the following request: "I'm looking for an expert in Artificial Intelligence with an experience of at least two years and he/she must have a doctorate". Let us suppose that there are three candidates **Sarah**, **Paul** and **Bill** skilled as presented in Figure 2 and that the three of them have a doctoral degree fulfilling the strict constraint of the user request. Looking both

Name	Knowledge
Sarah	Excellent experience in Business Intelligence (5 years) ...
Paul	1 years experienced in Knowledge Representation and Fuzzy Logic. Good knowledge of OWL, DLs, DL-lite family, ...
Bill	Skilled in ontology modeling with knowledge of semantic technologies ...

Fig. 1. Example of candidate skills

at the three profile descriptions and at the original request, we will rank the three candidates as **(1)** Paul; **(2)** Bill; **(3)** Sarah w.r.t. the preference expressed by the user. In fact, reasonably, the skills of Paul are very close to the requested ones even if he does not

<sup>1</sup> <http://www.attract-hr.com/cm/about>,  
[http://www.oracle.com/applications/human\\_resources/irecruit.html](http://www.oracle.com/applications/human_resources/irecruit.html)

<sup>2</sup> <http://www.hrojax.navy.mil/forms/selectguide.doc>

<sup>3</sup> <http://www.cpol.army.mil>

fully satisfy the requested experience (in years). On the other side, since ontology and semantic technologies relate to Artificial Intelligence Bill skills seems to be more useful than Sarah ones. It is easy to see that the only way to automatically perform such a ranking is exploiting a semantic-based approach, making use of a domain ontology modeling competence hierarchies and relations. Moreover, thanks to the information modeled in the ontology, the system is able to return all scores computed for each feature of the retrieved profiles.

A relevant aspect of our work is the exploitation of classical relational database systems (RDBMS) and languages *i.e.*, SQL, for storing the reference ontology and candidate CVs and to perform reasoning tasks. Using the system, both recruiters and candidates refers to the same model of the domain knowledge. Several approaches ([2], [3], [4], [5]) have been presented in which databases allow users and applications to access both ontologies and other structured data in a seamless way. A possible optimization consists in caching the classification hierarchy in the database and to provide tables maintaining all the subsumption relationships between primitive concepts. Such an approach is taken in *Instance Store (iS)* [6], a system for reasoning over OWL KBs specifically adopted in bio and medical-informatics domains. *iS* is also able –by means of a hybrid reasoner/database approach– to reply to instance retrieval queries w.r.t. an ontology, given a set of axioms asserting class-instance relationships. Nevertheless, *iS* reduces instance retrieval to pure TBox reasoning and is able to return only exact matches (*i.e.*, instance retrieval) whilst we use an enriched relational schema storing only the Abox (*i.e.*, facts) in order to provide a logic-based ranked list of results and the not classified ontology. Other systems using RDBMS in order to deal with large amounts of data are *QuOnto*<sup>4</sup> and *Owlgres*<sup>5</sup>. They are DL-Lite reasoners providing consistency checking and conjunctive query services. Neither QuOnto nor Owlgres return a ranked list of results. SHER ([7],[8]) is a high-scalability OWL reasoner performing both membership and conjunctive query answering by querying large relational datasets using ontologies modeled in a subset of OWL-DL which excludes nominals. It relies on an indexing technique summarizing the instances data in the database into a compact representation used for reasoning and works by selectively uncompressing portions of the summarized representation relevant for the query, in a process called refinement. Internally, SHER uses Pellet to reason over the summarized data and obtain justifications for data inconsistency. SHER allows for getting fast incomplete answers to queries, but does not provide a ranked list of results.

As hinted before, our system also allows for formulating queries by distinguishing between preferred and required skills by exploiting top-k retrieval techniques. Top-k queries [9] ensure an efficient ranking support in RDBMSs letting the system to provide only a subset of query results, according to a user-specified ordering function (which generally aggregates multiple ranking criteria). The general problem of preference handling in RDBMS in information retrieval systems [10] has been faced from two competing perspectives : *quantitative*– models, coping with preferences by means of utility functions [9, 11] and *qualitative*– models, using logical formulas [12, 10, 13]. Various approaches using numerical ranking in combination with either the top-k model [14–16], the Preference SQL [17] or the Preference XPath [12] have been also devised.

---

<sup>4</sup> <http://www.dis.uniroma1.it/~quonto/>

<sup>5</sup> <http://pellet.owldl.com/owlgres/>

Profile										
profID	FirstName	LastName	Genre	BirthDate	CityOfBirth	Address	City	ZipCode	Country	...
2	Wayne	Hernandez	female	1979-10-04	Berlin	Via Volta	Terni	05100	Italy	...
34	Hillary 156	Gadducci	female	1978-01-27	Bangalore	Church ST	New York	10027	USA	...

Fig. 2. The profile table.

### 3 System background: Top-k Retrieval for DLR-Lite

For computational reasons the particular logic we adopt is based on an extension of the DLR-Lite [1] Description Logic (DL) [18] without negation. DLR-Lite is different from usual DLs as it supports  $n$ -ary relations ( $n \geq 1$ ), whereas DLs support usual unary relations (called *concepts*) and binary relations (called *roles*). The DL will be used in order to define the relevant abstract concepts and relations of the application, while data is stored into a database. On the other hand, conjunctive queries will be used to describe the information needs of a user and to rank the answers according to a scoring function. The logic extends DLR-Lite by enriching it with build-in predicates. Conjunctive queries are enriched with scoring functions that allow to rank and retrieve the top-k answers, that is, we support *Top-k Query Answering* [19–23], (find top-k scored tuples satisfying query), *e.g.*, “find candidates with excellent knowledge in DLR-Lite”, where EXCELLENT is a function of the years of experience.

#### 3.1 The query and representation language

A *knowledge base*  $\mathcal{K} = \langle \mathcal{F}, \mathcal{O} \rangle$  consists of a *facts component*  $\mathcal{F}$  and an *Ontology component* (also called, *DL component*)  $\mathcal{O}$ , which are defined below.

**Facts Component.**  $\mathcal{F}$  is a finite set of expressions of the form  $R(c_1, \dots, c_n)$ , where  $R$  is an  $n$ -ary relation, every  $c_i$  is a constant. For each  $R$ , we represent the facts  $R(c_1, \dots, c_n)$  in  $\mathcal{F}$  by means of a relational  $n$ -ary table  $T_R$ , containing the records  $\langle c_1, \dots, c_n \rangle$ . Fig. 2 shows an example of facts about Curricula Vitae stored into the *Profile* relation.

**Ontology Component.** The ontology component is used to define the relevant abstract concepts and relations of the application domain by means of axioms. But, before we address the syntax of axioms, let us introduce the notion of concrete domains. In fact, *Top-k DLR-Lite* supports concrete domains with specific predicates on it. The allowed *concrete predicates* are relational predicates such as  $([i] \leq 1500)$  (*e.g.*, the value of the  $i$ -th column is less or equal than 1500) and  $([i] = \text{“Mayer”})$  (*e.g.*, the value of the  $i$ -th column is equal to the string “Mayer”). Formally, a *concrete domain* in  $\mathcal{D}$  is a pair  $\langle \Delta_{\mathcal{D}}, \Phi_{\mathcal{D}} \rangle$ , where  $\Delta_{\mathcal{D}}$  is an interpretation domain and  $\Phi_{\mathcal{D}}$  is the set of *domain predicates*  $d$  with a predefined arity  $n$  and an interpretation  $d^{\mathcal{D}}: \Delta_{\mathcal{D}}^n \rightarrow \{0, 1\}$ .

*Top-k DLR-Lite* allows to specify an ontology by relying on axioms. Consider an alphabet of  $n$ -ary relation symbols (denoted  $R$ ), *e.g.*, Profiles as described in Table 2, and an alphabet of unary relations, called *atomic concepts* (and denoted  $A$ ), *e.g.*, ItalianCity. Now, the DL component  $\mathcal{O}$  is a finite set of *axioms* having the form

$$(Rl_1 \sqcap \dots \sqcap Rl_m \sqsubseteq Rr)$$

where  $m \geq 1$ , all  $Rl_i$  and  $Rr$  have the same arity and where each  $Rl_i$  is a so-called *left-hand relation* and  $Rr$  is a *right-hand relation*. As illustrative purpose, a simple ontology axiom may be of the form

$$\text{AjaxProgrammer} \sqsubseteq \text{WebProgrammer}$$

with informal reading “any AJAX programmer is also a Web programmer”. In such an axiom,  $\text{AjaxProgrammer}$  and  $\text{WebProgrammer}$  are unary relations with signature  $\text{AjaxProgrammer}(\text{id})$  and  $\text{WebProgrammer}(\text{id})$ , respectively. But we may also involve  $n$ -ary realtions in ontology axioms. For instance, suppose that from the profiles records, we would like to extract just the profile ID and the last name, and call this new relation  $\text{HasLastName}$  with signature  $\text{HasLastName}(\text{profilD}, \text{LastName})$ . In database terminology this amounts in a projection of the  $\text{Profile}$  relation on the first and third column. In our language, the projection of an  $n$ -ary relation  $R$  on the columns  $i_1, \dots, i_k$  ( $1 \leq i_1, i_2, \dots, i_k \leq n, 1 \leq i \leq n$ ), will be indicated with  $\exists[i_1, \dots, i_k]R$ . Hence, e.g.,  $\exists[1, 3]\text{Profile}$  is the binary relation that is the projection on the first and third column of the  $\text{Profile}$  relation. So, for instance, the axiom

$$\exists[1, 3]\text{Profile} \sqsubseteq \exists[1, 2]\text{HasLastName}$$

states that the relation  $\text{HasLastName}$  contains the projection of the  $\text{Profile}$  relation on the first and third column. In case of a projection, we may further restrict it according to some conditions, using concrete predicates. For instance,  $\exists[1, 5]\text{Profile}.\left(\left([5] \geq 1979\right)\right)$  corresponds to the set of tuples  $\langle \text{profilD}, \text{BirthDate} \rangle$  such that the fifth column of the relation  $\text{Profile}$ , i.e., the person’s birth date, is equal or greater than 1979. The exact syntax of the relations appearing on the left-hand and right-hand side of ontology axioms is specified below (where  $h \geq 1$ ):

$$\begin{aligned} Rr &\longrightarrow A \mid \exists[i_1, \dots, i_k]R \\ Rl &\longrightarrow A \mid \exists[i_1, \dots, i_k]R \mid \\ &\quad \exists[i_1, \dots, i_k]R.(Cond_1 \sqcap \dots \sqcap Cond_h) \\ Cond &\longrightarrow ([i] \leq v) \mid ([i] < v) \mid ([i] \geq v) \mid ([i] > v) \mid \\ &\quad ([i] = v) \mid ([i] \neq v) \end{aligned}$$

where  $A$  is an atomic concept,  $R$  is an  $n$ -ary relation with  $1 \leq i_1, i_2, \dots, i_k \leq n$ ,  $1 \leq i \leq n$  and  $v$  is a value of the concrete interpretation domain of the appropriate type.

Here  $\exists[i_1, \dots, i_k]R$  is the projection of the relation  $R$  on the columns  $i_1, \dots, i_k$  (the order of the indexes matters). Hence,  $\exists[i_1, \dots, i_k]R$  has arity  $k$ .

On the other hand,  $\exists[i_1, \dots, i_k]R.(Cond_1 \sqcap \dots \sqcap Cond_l)$  further restricts the projection  $\exists[i_1, \dots, i_k]R$  according to the conditions specified in  $Cond_i$ . For instance,  $([i] \leq v)$  specifies that the values of the  $i$ -th column have to be less or equal than the value  $v$ .

**Query language.** Concerning queries, a *query* consists of a “conjunctive query”, with a scoring function to rank the answers. A *query* is of the form

$$q(\mathbf{x})[s] \leftarrow \exists \mathbf{y} R_1(\mathbf{z}_1), \dots, R_l(\mathbf{z}_l), \text{OrderBy}(s = f(p_1(\mathbf{z}'_1), \dots, p_h(\mathbf{z}'_h))) \quad (1)$$

where

1.  $q$  is an  $n$ -ary relation, every  $R_i$  is an  $n_i$ -ary relation,

2.  $\mathbf{x}$  are the  $n$  distinguished variables;
3.  $\mathbf{y}$  are so-called *non-distinguished variables* and are distinct from the variables in  $\mathbf{x}$ ;
4.  $\mathbf{z}_i, \mathbf{z}_j'$  are tuples of constants or variables in  $\mathbf{x}$  or  $\mathbf{y}$ . Any variable in  $\mathbf{x}$  occurs in some  $\mathbf{z}_i$ . Any variable in  $\mathbf{z}_j'$  occurs in some  $\mathbf{z}_i$ ;
5.  $p_j$  is an  $n_j$ -ary fuzzy predicate assigning to each  $n_j$ -ary tuple  $\mathbf{c}_j$  a score  $p_j(\mathbf{c}_j) \in [0, 1]_m$ . Such predicates are called *expensive predicates* in [24] as the score is not pre-computed off-line, but is computed on query execution. We require that an  $n$ -ary fuzzy predicate  $p$  is *safe*, that is, there is not an  $m$ -ary fuzzy predicate  $p'$  such that  $m < n$  and  $p = p'$ . Informally, all parameters are needed in the definition of  $p$ ;
6.  $f$  is a scoring function  $f: ([0, 1]_m)^h \rightarrow [0, 1]_m$ , which combines the scores of the  $h$  fuzzy predicates  $p_j(\mathbf{c}_j')$  into an overall score to be assigned to the rule head  $q(\mathbf{c})$ . We assume that  $f$  is *monotone*, that is, for each  $\mathbf{v}, \mathbf{v}' \in ([0, 1]_m)^h$  such that  $\mathbf{v} \leq \mathbf{v}'$ , it holds  $f(\mathbf{v}) \leq f(\mathbf{v}')$ , where  $(v_1, \dots, v_h) \leq (v'_1, \dots, v'_h)$  iff  $v_i \leq v'_i$  for all  $i$ .
7. We also assume that the computational cost of  $f$  and all fuzzy predicates  $p_i$  is bounded by a constant.

Finally, a *disjunctive query*  $\mathbf{q}$  is, as usual, a finite set of conjunctive queries in which all the rules have the same head. We omit to write  $\exists \mathbf{y}$  when  $\mathbf{y}$  is clear from the context.  $R_i(\mathbf{z}_i)$  may also be a concrete unary predicate of the form  $(z \leq v), (z < v), (z \geq v), (z > v), (z = v), (z \neq v)$ , where  $z$  is a variable,  $v$  is a value of the appropriate concrete domain. We call  $q(\mathbf{x})[s]$  its *head*,  $\exists \mathbf{y}. R_1(\mathbf{z}_1), \dots, R_l(\mathbf{z}_l), \text{OrderBy}(\dots)$  its *body* and  $\text{OrderBy}(\dots)$  the *scoring atom*.

**Top-k Retrieval.** Given a knowledge base  $\mathcal{K}$ , and a disjunctive query  $\mathbf{q}$ , retrieve  $k$  tuples  $\langle \mathbf{c}, s \rangle$  that instantiate the query relation  $q$  with maximal score (if  $k$  such tuples exist), and rank them in decreasing order relative to the score  $s$ , denoted

$$ans_k(\mathcal{K}, \mathbf{q}) = \text{Top}_k ans(\mathcal{K}, \mathbf{q}).$$

From a query answering point of view, it extends the DL-Lite/DLR-Lite reasoning method [1] to the fuzzy case. The algorithm is an extension of the one described in [1, 20, 22]). Roughly, given a query  $q(\mathbf{x})[s] \leftarrow \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ ,

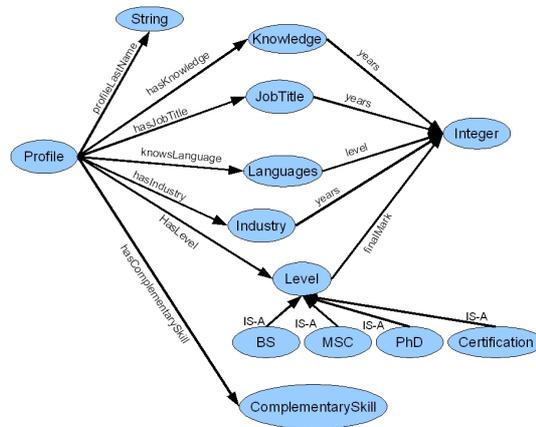
1. by considering  $\mathcal{O}$ , the user query  $\mathbf{q}$  is *reformulated* into a set of conjunctive queries  $r(\mathbf{q}, \mathcal{O})$ . Informally, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each axiom is seen as a logic programming rule.
2. from the set of reformulated queries  $r(\mathbf{q}, \mathcal{O})$  we remove redundant queries;
3. the reformulated queries  $q' \in r(\mathbf{q}, \mathcal{O})$  are translated to ranked SQL queries and evaluated. The query evaluation of each ranked SQL query returns the top- $k$  answer set for that query. Specifically, we use the RankSQL [14] system for this purpose;
4. all the  $n = |r(\mathbf{q}, \mathcal{O})|$  top- $k$  answer sets have to be merged into the unique top- $k$  answer set  $ans_k(\mathcal{K}, \mathbf{q})$ . As  $k \cdot n$  may be large, we apply a *Disjunctive Threshold Algorithm* (DTA, see e.g., [22]) to merge all the answer sets.

The detailed description of the algorithm embedded in our system to solve top-k retrieval problem is beyond the scope of this work. The algorithm is an extension of the one described in [1, 20, 22]) and has been implemented as part of the SoftFacts system<sup>6</sup>.

<sup>6</sup> <http://gaia.isti.cnr.it/~straccia/software/SoftFacts/SoftFacts.html>

## 4 System evaluation

The proposed system has been implemented by plugging the Top-K DLR-Lite retrieval approach into I.M.P.A.K.T. [25], a system for skills and knowledge management developed by *Data Over Ontological Models s.r.l.*<sup>7</sup> as a commercial solution implementing to some extent the skill matching framework designed in [26]. The efficiency and scalability of the approach has been tested using the skill ontology underlying I.M.P.A.K.T.. Both requests and candidate profiles have been modeled w.r.t. to this ontology containing 2594 relations, both unary (classes) and n-ary ones, and 5119 axioms. The main structure of the ontology is depicted in Figure 3.



**Fig. 3.** A graphical representation of the ontology structure.

*Level* represents profile education such as certifications, masters, doctorate, etc.; *Knowledge* represents technical skill and specific competences of the candidate; *ComplementarySkill* represents abilities and hobbies of the candidate; *JobTitle* represents work experiences of the candidate; *Industry* representing sectors (institutes, research laboratories, companies, etc.) in which candidate works/worked; *Language* represents the knowledge of foreign languages. Data properties, have been used to represent years of experience, degree final mark and knowledge level of foreign languages.

The system exploits the user interface of I.M.P.A.K.T., shown in Figure 4. Panels (a), (b) and (d) allow the recruiter to compose her semantic-based request. In fact, in menu (a) all the entry points are listed whilst panel (b) allows to search for ontology concepts according to their meaning and section (d) enables the user to explore both taxonomy and properties of a selected concept. Entry points in menu (a) represent, to some extent, the main classes and relations represented in Figure 3. Once an item is selected in panel (d), the corresponding panel, representing the item itself, is dynamically filled

<sup>7</sup> <http://www.doom-srl.it/>

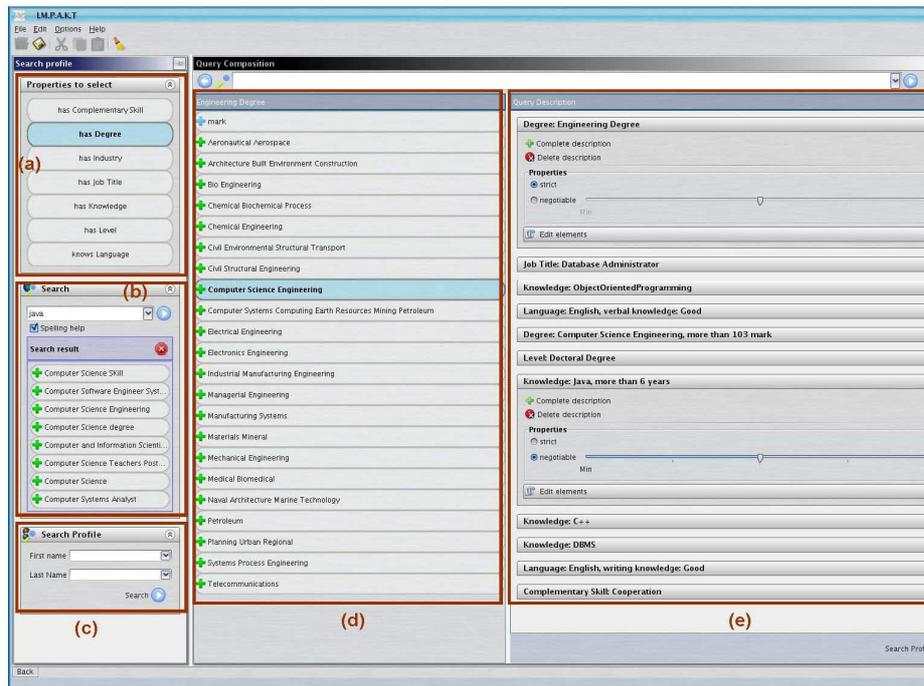


Fig. 4. Query composition GUI

and added to panel (e). This latter enumerates all the requested features in the query. For each of them, the GUI of I.M.P.A.K.T. allows: (1) to define if the feature is strict (crisp) or negotiable (fuzzy); (2) to delete the whole feature; (3) to complete the description showing all the elements (concepts, object properties and data properties) that could be added to the selected feature; (4) to edit each feature piece as well as existing data properties. Finally, panel (c) enables searches like “*I’m searching a candidate like John Doe*” i.e., it is useful to model all those situations where you are looking for a candidate whose skills and knowledge are similar to the ones of *John Doe*. In this case, the job-seeker fills first and/or last name field of the known candidate and the system consider her/his profile as starting request. The user can view the query –automatically generated– and eventually she can edit it before starting a new search.

In the experiments we carried out, we considered 100.000 automatically generated CVs and stored them into a database having 17 relational tables. In Figure 4 we show the ontology axioms mapping the relational tables involved in the proposed queries, in order to provide the reader with the alphabet of the query language. Each axiom renames with the role name given as first parameter the table defined as second parameter with all its fields. We build several queries, with/without scoring atom and submitted them to the system, with different values for  $k$  in case of top-k retrieval ( $k \in \{1, 10\}$ ). We run the experiments using the top-k retrieval SoftFacts system as back-end. No indexes have been used for the facts in the relational database. The concept and role hierarchy

```
(MAP-ROLE Profile Profile (profID, FirstName, LastName, Genre,
BirthDate, CityOfBirth, Address, City, ZipCode, Country, IdentityCode,
PhoneNumber, Email, WebPage, Nationality, ResidentIn,
SuddenJobAvailability, JobLocation, FlexibleWorkHours,
TravelingAvailability, CertificationInstitute, Salary, CarAvailability))
(MAP-ROLE degreeName Degree (degID, Name))
(MAP-ROLE knowledgeName Knowledge (knowID, Name))
(MAP-ROLE knowledgeLevelName KnowledgeLevel (knowLevelID, Name))
(MAP-ROLE knowledgeTypeName KnowledgeType (knowTypeID, Name))
(MAP-ROLE knowledgeLevelName KnowledgeLevel (knowLevelID, Name))
(MAP-ROLE knowledgeTypeName KnowledgeType (knowTypeID, Name))
(MAP-ROLE hasDegree HasDegree (profID, classID, Mark))
(MAP-ROLE hasKnowledge HasKnowledge (profID, classID, Years, Type, Level))
```

**Fig. 5.** Excerpt of relational tables ontology mapping

used in the experiment queries is clarified in Figure 4. The queries at the basis of the

```
(IMPLIES Engineering_and_Technology Knowledge)
(IMPLIES Artificial_Intelligence Computer_Science_Skill)
(IMPLIES Information_Systems Computer_Science_Skill)
(IMPLIES Computer_Science_Skill Engineering_and_Technology)
(IMPLIES Engineering_Degree Degree)
(IMPLIES Fuzzy_Artificial_Intelligence)
(IMPLIES Data_Mining Artificial_Intelligence)
(IMPLIES Machine_Learning Artificial_Intelligence)
(IMPLIES Knowledge_Rappresentation Artificial_Intelligence)
(IMPLIES Natural_Language Artificial_Intelligence)
(MAP-ROLE profileLastName Profile (profID, LastName))
(IMPLIES (SOME[1] profileLastName) Profile)
```

**Fig. 6.** Excerpt of concepts and roles hierarchy

experimentation are listed below, together with the corresponding encoding in *Top-K DLR-Lite* for the most significant ones.

1. Retrieve CV's with knowledge in Engineering Technology
2. Retrieve CV's referred to candidates with degree in Engineering
3. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence and degree final mark not less than 100/110
4. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence, degree in Engineering with final mark not less than 100/110

```
q(id, lastName, hasKnowledge, Years, degreeName, mark)
← profileLastName(id, lastName), hasKnowledge(id, classID, Years, Type, Level),
knowledgeName(classID, hasKnowledge), Artificial_Intelligence(classID),
hasDegree(id, degreeID, mark), DegreeName(degreeID, hasDegree), Engineering_Degree(degreeID),
(mark ≥ 100)
```

5. Retrieve CV's referred to candidates experienced in Information Systems (not less than 15 years) , with degree final mark not less than 100

```
q(id, lastName, hasKnowledge, Years, degreeName, mark)
← profileLastName(id, lastName), hasKnowledge(id, classID, Years, Type, Level),
knowledgeName(classID, hasKnowledge), Information_Systems(classID), (Years ≥ 15)
hasDegree(id, degreeID, mark), DegreeName(degreeID, hasDegree),
(mark ≥ 100)
```

6. Retrieve top- $k$  CV's referred to candidates with knowledge in Artificial Intelligence and degree final mark scored according to  $rs(mark; 100, 110)$

```
q(id, lastName, degreeName, mark, hasKnowledge, years)
← profileLastName(id, lastName), hasDegree(id, degreeId, mark), degreeName(degreeId, degreeName),
hasKnowledge(id, classID, years, type, level), knowledgeName(classID, hasKnowledge),
Artificial_Intelligence(classID), OrderBy(s = rs(mark; 100, 110))
```

7. Retrieve CV's referred to candidates with degree in Engineering and final mark scored according to  $rs(mark; 100, 110)$

```
q(id, lastName, hasDegree, mark)
← profileLastName(id, lastName), hasDegree(id, classID, mark), DegreeName(classID, hasDegree),
Engineering_Degree(classID), OrderBy(s = rs(mark; 100, 110))
```

8. Retrieve top- $k$  CV's referred to candidates with knowledge in Artificial Intelligence, degree in Engineering with final mark scored according to  $rs(mark; 100, 110)$

```
q(id, lastName, hasKnowledge, Years, degreeName, mark)
← profileLastName(id, lastName), hasKnowledge(id, classID, Years, Type, Level),
knowledgeName(classID, hasKnowledge), Artificial_Intelligence(classID),
hasDegree(id, degreeID, mark), DegreeName(degreeID, hasDegree), Engineering_Degree(degreeID),
OrderBy(s = rs(mark; 100, 110))
```

9. Retrieve CV's referred to candidates with knowledge in Information Systems and with degree final mark and years of experience both scored according to  $rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6$ ;

```
q(id, lastName, hasKnowledge, Years, degreeName, mark)
← profileLastName(id, lastName), hasKnowledge(id, classID, Years, Type, Level),
knowledgeName(classID, hasKnowledge), Information_Systems(classID), (Years ≥ 15)
hasDegree(id, degreeID, mark), DegreeName(degreeID, hasDegree),
OrderBy(s = rs(mark; 100, 110) · 0.4 + rs(years; 15, 25) · 0.6)
```

10. Retrieve CV's referred to candidates with good knowledge in Artificial Intelligence, and with degree final mark, years and level of experience scored according to  $rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot pref(level; Good/0.6, Excellent/1.0) \cdot 0.6$ ;

```
q(id, lastName, degreeName, mark, hasKnowledge, years, kType)
← profileLastName(id, lastName), hasDegree(id, degreeId, mark), degreeName(degreeId, degreeName),
hasKnowledge(id, classID, years, type, level), knowledgeLevelName(level, kType), Good(level),
knowledgeName(classID, hasKnowledge), Artificial_Intelligence(classID),
OrderBy(s = rs(mark; 100, 110) · 0.4 + rs(years; 15, 25) · pref(level; Good/0.6, Excellent/1.0) · 0.6)
```

Queries 1-5 are crisp queries. There is no preference expressed and no actual ranking. As each answer has score 1.0, we would like to verify whether there is a retrieval time difference between retrieving all records, or just the  $k$  answers. The other queries are top- $k$  queries. In query 9, we show an example of score combination, with a preference on the number of years of experience over the degree's mark, but scores are summed up. In query 10, we use the preference scoring function

$$pref(level; Good/0.6, Excellent/1.0)$$

that returns 0.6 if the level is good, while returns 1.0 if the level is excellent. In this way we want to privilege those with an excellent knowledge level over those with a good level of knowledge. In Fig.7 we report the output of query 10.

The tests have been performed on a MacPro machine with Mac OS X 10.5.5, 2 x 3 GHz Dual-Core processor and 9 GB or RAM and the results are shown in Fig. 8 (time is measured in seconds). Let us consider few comments about the results:

10 Results found (Top-10):							
Score	id	lastName	degreeName	mark	hasKnowledge	Years	KType
0.42	299	Jensen	Animal_Science	109.0	Fuzzy	16.0	Excellent
0.36	938	Young	Civil_Structural_Engineering	109.0	Machine_Learning	11.0	Excellent
0.28	360	Taylor	Animal_Health	107.0	Knowledge_Rappresentation	15.0	Excellent
0.08	956	Cook	Physiotherapy	102.0	Natural_Language	7.0	Good
0.08	187	Allan	Ecology	102.0	Artificial_Intelligence	2.0	Excellent
0.04	1109	Graham	African_Sub_Saharan	101.0	Machine_Learning	4.0	Excellent
0.0	1081	James	Humanities	75.0	Artificial_Intelligence	5.0	Good
0.0	682	Scott	Film_Studies_Television	82.0	Data_Mining	7.0	Good
0.0	538	Cox	Development_Studies	95.0	Machine_Learning	6.0	Good
0.0	604	Scott	English_Literature	61.0	Knowledge_Rappresentation	1.0	Good

Fig. 7. Retrieval output of query 10.

Size 100000				
Query	All	top-1	top-10	$ ans(\mathcal{K}, q) $
1	12.344	3.596	6.182	3985
2	0.375	0.116	0.125	445
3	0.366	0.117	0.118	19
4	4.263	3.877	3.897	8
5	0.397	0.325	0.357	19
6	0.104	0.103	0.099	40
7	0.209	0.178	0.189	128
8	4.086	3.895	3.998	20
9	0.471	0.422	0.395	201
10	0.391	0.357	0.373	19
Average	2.301	1.295	1.573	488
Median	0.394	0.341	0.365	30

Fig. 8. Retrieval times.

- overall, the response time is quite good (almost fraction of second) taking into account the non negligible size of the ontology, the number of CVs and that we did not consider any index for the relational tables;
- if the answer set is large, e.g., query 1, then there is a significant drop in response time, for the top-k case;
- for each query, the response time is increasing while we increase the number of retrieved records.

## 5 CONCLUSION AND FUTURE WORK

We presented an innovative and scalable logic-based system for efficiently managing skills and experiences of candidates in the e-recruitment field. The system grounds on a Skill Ontology in order to return a ranked list of profiles and on scoring functions in order to weight each feature of the retrieved profiles. Differently from existing recruitment systems, our approach allows to express a user request as the composition of both mandatory requirements and preferences, by means of top-k retrieval techniques. The implemented retrieval framework was embedded into an existing system for skill management and experiments conducted on a preliminary profiles dataset show a satisfiable behavior. Future work aims at evaluating system performance on several datasets and at providing user-friendly explanation facilities to better clarify scores of obtained results.

## Acknowledgments

We wish to acknowledge partial support of Projects PS\_092 and PS\_121 .

## References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR'06. (2006) 260–270
2. Das, S., Chong, E.I., Eadon, G., Srinivasan, J.: Supporting ontology-based semantic matching in RDBMS. In: Proc. of VLDB'04, VLDB Endowment (2004) 1054–1065
3. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Proc. of SWDB'03. (2003) 131–150
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Proc. of ISWC '02. (2002) 54–68
5. Pan, Z., Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: Proc. of PSSS1. Volume 89., CEUR-WS.org (2003) 109–113
6. Bechhofer, S., Horrocks, I., Turi, D.: The OWL Instance Store: System Description. In: Proc. of CADE '05. (2005) 177–181
7. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The summary abox: Cutting ontologies down to size. In: Proceedings of 5th International Semantic Web Conference (ISWC 2006). (2006) 343–356
8. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007). (2007)
- 9.
10. Chomicki, J.: Querying with Intrinsic Preferences. In: Proc. of Advances in Database Technology - EDBT 2002. (2002) 34–51
11. P. Bosc and O. Pivert: SQLf: a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems 3(1) (Feb. 1995) 1–17
12. Kießling, W.: Foundations of preferences in database systems. In: Proc. of VLDB-02. (2002) 311–322
13. Hafenrichter, B., Kießling, W.: Optimization of relational preference queries. In: Proc. of ADC '05, Darlinghurst, Australia, Australian Computer Society, Inc. (2005) 175–184
14. Li, C., Soliman, M.A., Chang, K.C.C., Ilyas, I.F.: RankSQL: supporting ranking queries in relational database management systems. In: Proc. of VLDB-05, VLDB Endowment (2005) 1342–1345
15. Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A system for the efficient execution of multi-parametric ranked queries, ACM (2001) 259–270
16. Yu, H., Hwang, S.W., Chang, K.C.C.: RankFP: A Framework for Supporting Rank Formulation and Processing. In: Proc. of ICDE-2005. (2005) 514–515
17. Kießling, W., Köstler, G.: Preference SQL - design, implementation, experiences. In: Proc. of VLDB-02. (2002) 990–1001
18. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
19. Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the semantic web. In: Proc. of SUM-07. Number 4772 in LNCS, Springer Verlag (2007)
20. Straccia, U.: Answering vague queries in fuzzy DL-Lite. In: Proc. of IPMU-06, E.D.K., Paris (2006) 2238–2245
21. Straccia, U.: Towards top-k query answering in deductive databases. In: Proc. of SMC-06, IEEE (2006) 4873–4879

22. Straccia, U.: Towards top-k query answering in description logics: the case of DL-Lite. In: Proc. of JELIA-06. Number 4160 in LNCS, Springer Verlag (2006) 439–451
23. Straccia, U.: Towards vague query answering in logic programming for logic-based information retrieval. In: Proc. of IFSA-07. Number 4529 in LNCS, Springer Verlag (2007) 125–134
24. Chang, K.C.C., won Hwang, S.: Minimal probing: Supporting expensive predicates for top-k queries. In: SIGMOD Conference. (2002)
25. Tinelli, E., Cascone, A., Ruta, M., Noia, T.D., Sciascio, E.D., Donini, F.M.: I.M.P.A.K.T.: an innovative, semantic-based skill management system exploiting standard SQL. In: Proc. of ICEIS'09. Volume AIDSS. 224–229
26. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F.M., Ragone, A.: Semantic-based skill management for automated task assignment and courseware composition. *J. Univ. Comp. Sci.* **13**(9) (2007) 1184–1212