# Semantic-Based Top-k Retrieval for Competence Management

Umberto Straccia[1], Eufemia Tinelli[2], Simona Colucci[2], Tommaso Di Noia[2] and Eugenio Di Sciascio[2]

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

**Abstract.** We present a knowledge-based system, for skills and talent management, exploiting semantic technologies combined with top-k retrieval techniques. The system provides advanced distinguishing features, including the possibility to formulate queries by expressing both strict requirements and preferences in the requested profile and a semantic-based ranking of retrieved candidates. Based on the knowledge formalized within a domain ontology, the system implements an approach exploiting top-k based reasoning services to evaluate semantic similarity between the requested profile and retrieved ones. System performance is discussed through the presentation of experimental results.

## 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 matchmaking 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 [4] 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[1] 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[2], 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[3] 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 a candidate, preferably expert in Artificial Intelligence with an experience of at least two years and necessarily endowed with a doctoral degree"*. Let us suppose that there are three candidates **Sarah**, **Paul** and **Bill** skilled as presented in Figure 1 all having a doctoral degree fulfilling the strict constraint of the user request. Looking both 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 fully satisfy the requested experience (in years). On the other side, since ontology and semantic technologies relate to Artificial Intelligence Bill skills seems to

[1] `http://www.attract-hr.com/cm/about`, `http://www.oracle.com/applications/human_resources/irecruit.html`

[2] `http://www.hrojax.navy.mil/forms/selectguide.doc`

[3] `http://www.cpol.army.mil`

| 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

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 ([7], [22], [3], [16]) 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***) [2], 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*[4] and *Owlgres*[5]. They are DL-Lite reasoners providing consistency checking and conjunctive query services. Neither QuOnto nor OWLgres returns 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 [12] 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 [5] has been faced from two competing perspectives : *quantitative*– models, coping with preferences by means of utility functions [12, 15] and *qualitative*– models, using logical formulas [10, 5, 8]. Various approaches using numerical ranking in combination with either the top-k model [13, 9, 23], the Preference SQL [11] or the Preference XPath [10] have been also devised.

[4] `http://www.dis.uniroma1.it/\~quonto/`
[5] `http://pellet.owldl.com/owlgres/`

## 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 [4] Description Logic (DL) [1] without negation. DLR-Lite is different from usual DLs as it supports $n$-ary relations ($n \geqslant 1$), 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 built-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* [14, 17–20], (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.

Due to lack of space, we do not delve into details about the query and representation language at the basis of top-k retrieval problem (detailed in [19] for the interested reader) and just recall its definition in the following.

**Top-k Retrieval.** Given a knowledge base $\mathcal{K}$, and a union of conjunctive queries $\mathbf{q}$, 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

$$ans_k(\mathcal{K}, \mathbf{q}) = Top_k \; ans(\mathcal{K}, \mathbf{q}) \; .$$

A *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O} \rangle$ consists of a *facts component* $\mathcal{F}$ and an *Ontology component* $\mathcal{O}$. Informally, facts component is used to store data into a database and the ontology component is used to define the relevant abstract concepts and relations of the application domain.

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 [4, 17, 19]) and has been implemented also as part of the SoftFacts system [6].

## 4 System evaluation

The proposed system has been implemented by plugging the Top-K DLR-Lite retrieval into `I.M.P.A.K.T.` [21], a system for skills and knowledge management developed by *Data Over Ontological Models s.r.l.*[7] as a commercial solution implementing the skill matching framework designed in [6]. 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 2.

[6] `http://gaia.isti.cnr.it/~straccia/software/SoftFacts/SoftFacts.html`

[7] `http://www.doom-srl.it/`

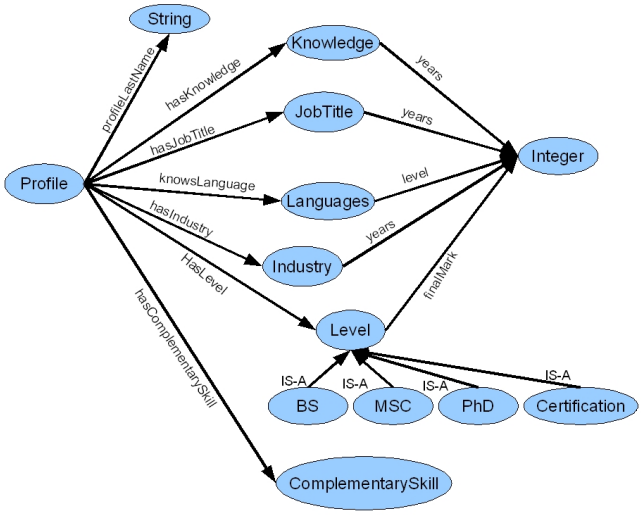


**Fig. 2.** A graphical representation of the ontology structure.

*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 3. Panels (a), (b) and (d) allow the recuiter 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 2. Once an item is selected in panel (d), the corresponding panel, representing the item itself, is dynamically filled 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.

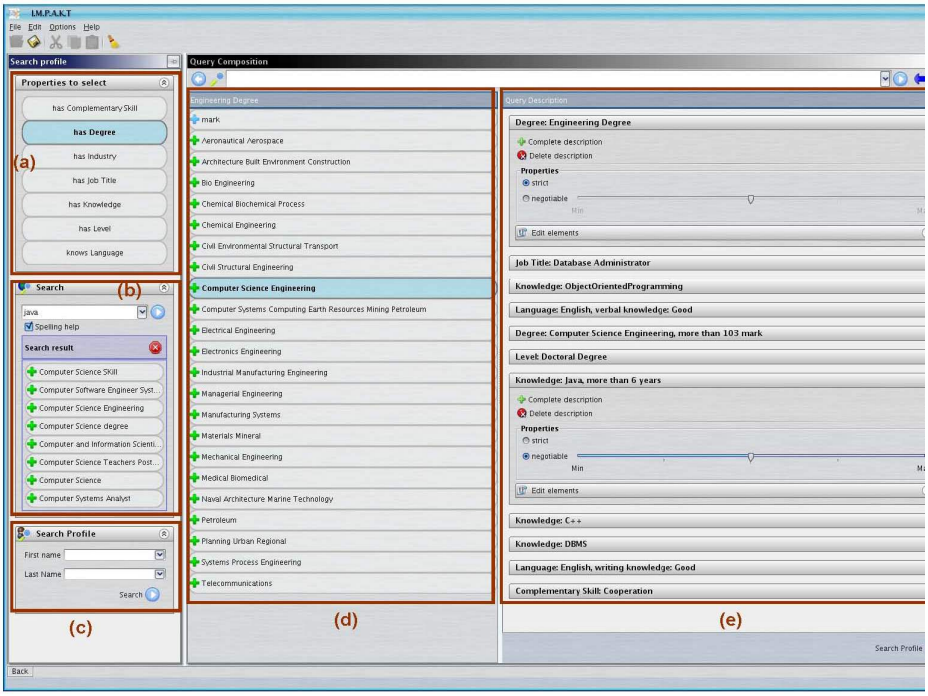


**Fig. 3.** Query composition GUI

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 used in the experiment queries is clarified in Figure 5. The queries at the basis of the experimentation are listed below, together with the corresponding encoding in *Top-K DLR-Lite*.

1. Retrieve CV's with knowledge in Engineering Technology

$$q(\mathsf{id}, \mathsf{lastName}, \mathsf{hasKnowledge}, \mathsf{Years}) \leftarrow \mathsf{profileLastName}(\mathsf{id}, \mathsf{lastName}), \mathsf{hasKnowledge}(\mathsf{id}, \mathsf{classID}, \mathsf{Years}, \mathsf{Type}, \mathsf{Level}), \mathsf{knowledgeName}(\mathsf{classID}, \mathsf{hasKnowledge}), \mathsf{Engineering\_and\_Technology}(\mathsf{classID})$$

2. Retrieve CV's referred to candidates with degree in Engineering

$$q(\mathsf{id}, \mathsf{lastName}, \mathsf{hasDegree}, \mathsf{mark}) \leftarrow \mathsf{profileLastName}(\mathsf{id}, \mathsf{lastName}), \mathsf{hasDegree}(\mathsf{id}, \mathsf{classID}, \mathsf{mark}), \mathsf{DegreeName}(\mathsf{classID}, \mathsf{hasDegree}), \mathsf{Engineering\_Degree}(\mathsf{classID})$$

```
(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. 4.** Excerpt of relational tables ontology mapping

```
(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. 5.** Excerpt of concepts and roles hierarchy

3. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence and degree 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), (mark ⩾ 100)

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.6 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. 7 (time is measured in seconds). Let us consider few comments about the results:

- 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.

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. 6.** 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. 7.** Retrieval times.

## 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 conduced 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 PS_092 and PS_121 .

## References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

2. S. Bechhofer, I. Horrocks, and D. Turi. The OWL Instance Store: System Description. In *Proc. of CADE '05*, pages 177–181, 2005.
3. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proc. of ISWC '02*, pages 54–68. Springer, 2002.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR'06*, pages 260–270, 2006.
5. J. Chomicki. Querying with Intrinsic Preferences. In *Proc. of Advances in Database Technology - EDBT 2002*, pages 34–51. Springer, 2002.
6. S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and A. Ragone. Semantic-based skill management for automated task assignment and courseware composition. *J. Univ. Comp. Sci.*, 13(9):1184–1212, 2007.
7. S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting ontology-based semantic matching in RDBMS. In *Proc. of VLDB'04*, pages 1054–1065. VLDB Endowment, 2004.
8. B. Hafenrichter and W. Kießling. Optimization of relational preference queries. In *Proc. of ADC '05*, pages 175–184, Darlinghurst, Australia, 2005. Australian Computer Society, Inc.
9. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *Proc. of ACM SIGMOD*, pages 259–270. ACM, 2001.
10. W. Kießling. Foundations of preferences in database systems. In *Proc. of VLDB-02*, pages 311–322. Morgan Kaufmann, Los Altos, 2002.
11. W. Kießling and G. Köstler. Preference SQL - design, implementation, experiences. In *Proc. of VLDB-02*, pages 990–1001. Morgan Kaufmann, Los Altos, 2002.
12. C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: query algebra and optimization for relational top-k queries. In *Proc. of ACM SIGMOD'05*, New York, NY, USA, 2005. ACM Press.
13. C. Li, M. A. Soliman, K. C.-C. Chang, and I. F. Ilyas. RankSQL: supporting ranking queries in relational database management systems. In *Proc. of VLDB-05*, pages 1342–1345. VLDB Endowment, 2005.
14. T. Lukasiewicz and U. Straccia. 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.
15. P. Bosc and O. Pivert. SQLf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 3(1):1–17, Feb. 1995.
16. Z. Pan and J. Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Proc. of PSSS1*, volume 89, pages 109–113. CEUR-WS.org, 2003.
17. U. Straccia. Answering vague queries in fuzzy DL-Lite. In *Proc. of IPMU-06*, pages 2238–2245. E.D.K., Paris, 2006.
18. U. Straccia. Towards top-k query answering in deductive databases. In *Proc. of SMC-06*, pages 4873–4879. IEEE, 2006.
19. U. Straccia. Towards top-k query answering in description logics: the case of DL-Lite. In *Proc. of JELIA-06*, number 4160 in LNCS, pages 439–451. Springer Verlag, 2006.
20. U. Straccia. Towards vague query answering in logic programming for logic-based information retrieval. In *Proc. of IFSA-07*, number 4529 in LNCS, pages 125–134. Springer Verlag, 2007.
21. E. Tinelli, A. Cascone, M. Ruta, T. D. Noia, E. D. Sciascio, and F. M. Donini. I.M.P.A.K.T.: an innovative, semantic-based skill management system exploiting standard SQL. In *Proc. of ICEIS'09*, volume AIDSS, pages 224–229.
22. K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proc. of SWDB'03*, pages 131–150, 2003.
23. H. Yu, S.-W. Hwang, and K. C.-C. Chang. RankFP: A Framework for Supporting Rank Formulation and Processing. In *Proc. of ICDE-2005*, pages 514–515. IEEE Comp. Soc. Press, 2005.