Large Scale Skill Matching through Knowledge Compilation

Eufemia Tinelli[1], Simona Colucci[2], Silvia Giannini[1], Eugenio Di Sciascio[1], Francesco M. Donini[2]

[1] SisInfLab, Politecnico di Bari, Bari, Italy
[2] DISUCOM, Università della Tuscia, Viterbo, Italy

Abstract. We present a logic-based framework for automated skill matching, able to return a ranked referral list and the related ranking explanation. Thanks to a Knowledge Compilation approach, a knowledge base in Description Logics is translated into a relational database, without loss of information. Skill matching inference services are then efficiently executed via SQL queries. Experimental results for scalability and turnaround times on large scale data sets are reported, confirming the validity of the approach.

1 Introduction

We present a logic-based framework for automated skill matching, which combines the advantages of both semantic-based and database technologies through a Knowledge Compilation [2] approach. Coherently with it, our contribution makes computationally efficient the skill matching execution over the information contained in the Knowledge Base (KB) – modeling intellectual capital according to the formalism of Description Logics (DLs) – by splitting the reasoning process in two phases: **(i)** *off-line reasoning* - the KB is pre-processed and stored in a relational database; **(ii)** *on-line reasoning* - skill matching is performed by querying the data structure coming from the first phase. Other distinguishing features of the approach include the addition of a fully explained semantic-based comparison between the job request and the retrieved candidates as well as the possibility to express both strict requirements and preferences in the job request. Coherently with this perspective, our approach provides a two-steps matchmaking [5, 8] process: *Strict Match* retrieves candidates fully satisfying all the strict requirements; *Soft Match* implements an approximate match by retrieving candidates fully or partially satisfying at least one user preference.

The approach has been implemented in `I.M.P.A.K.T.`, an integrated system for automated HR management that provides team composition services [14, 6] and Core Competence extraction [7] (an embryonic `I.M.P.A.K.T.` version of the retrieval of candidates ranked referral lists has been presented in [13]).

Among the few semantic-based implemented solutions for HR management, one of the first ones is –to the best of our knowledge – STAIRS[3], a system still used at US Navy Department to retrieve referral lists of best qualified candidates w.r.t. a specific task. We

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

may also cite products offered by Sovren[4], which provide solutions for both CV and job requests parsing starting from several text formats to HR-XML schema. Recently, also *Monster.com*$^{(R)}$, the leading Web job-matching engine, introduced the *Monster Power Resume Search*^{TM} service [5]. The product relies on the semantic $6Sense^{(TM)}$ search technology, patented by Monster Worldwide, Inc. . All the previous solutions exploit the semantics of queries– and are able to distinguish between essential and nice-to-have skills– to perform the search process but no ranking explanation is returned. On the other hand, several approaches have been presented, where databases allow users and applications to access both ontologies and other structured data in a seamless way. Das et al. [4] developed a system that stores OWL-Lite and OWL-DL ontologies in Oracle RDBMSs, and provides a set of SQL operators for ontology-based matching. The most popular OWL storage is the recent OWLIM [11], a Sesame plug-in able to add a robust support for the semantics of RDFS, OWL Horst and OWL2 RL. Other systems using RDBMS to deal with large amounts of data are *QuOnto*[6] and *OWLgres*[7], both DL-Lite reasoners providing consistency checking and conjunctive query services. SHER [9] is a highly-scalable OWL reasoner performing both membership and conjunctive query answering over large relational datasets using ontologies modeled in a subset of OWL-DL without nominals. PelletDB[8] provides an OWL 2 reasoning system specifically built for enterprise semantic applications. Although all the previous approaches support languages more expressive than the one we use in our system, they are only able to return either exact matches (*i.e.*, instance retrieval) or general query answering. Instead, we use an enriched relational schema to deal with non-standard inferences and provide effective value-added services.

The rest of this paper is organized as follows. In the next section, the modeling approach translating the KB into the reference relational database is presented. Section 3 introduces the implemented services and Section 4 reports on an experimental evaluation using PostgreSQL 9.1 DBMS showing the effectiveness and the scalability of the proposal. Conclusions and future research directions close the paper.

2 Knowledge Compilation

`I.M.P.A.K.T.` receives all the information needed to model and manage the domain of human resources from a specifically developed modular ontology $\mathcal{T} = \{M_i | 0 \leq i \leq 6\}$, currently including nearly 5000 concepts. Each ontology module M_i is modeled according to the formalism of $\mathcal{FL}_0(\mathrm{D})$ subset of DLs. In particular, every M_i may include the following items: **i)** a class hierarchy; **ii)** n optional properties $R^i_j, 1 \leq j \leq n$, defined over the classes specifying the module hierarchy; **iii)** optional concrete features p_i, either in the natural numbers or in the calendar dates domain.

Hereafter, we shortly describe the content modeled in each ontology module: **`Level`** models the hierarchy of candidate education and training levels; **`ComplementarySkill`**

[4] `http://www.sovren.com/default.aspx`

[5] `http://hiring.monster.com/recruitment/Resume-Search-Database.aspx`

[6] `http://www.dis.uniroma1.it/~quonto/`

[7] `http://pellet.owldl.com/owlgres/`

[8] `http://clarkparsia.com/pelletdb/`

models the class hierarchy about complementary attitudes; **`Industry`** models the hierarchy of company types a candidate may have worked for; **`Knowledge`** models the hierarchy of possible candidate competence and technical tools usage ability and the related experience role (*e.g.*, developer, administrator, and so on) exploiting the `type` property; **`JobTitle`** models the hierarchy of possible job positions; **`Language`** models the hierarchy of possible languages known by the candidate and provides three concrete features for expressing the related level (`verbalLevel`, `readingLevel` and `writingLevel`). Finally, modules `Industry`, `ComplementarySkill`, `Knowledge` and `JobTitle` provide also two predicates: `year`, to specify the experience level in years, and `lastdate`, which represents the last temporal update of work experience. M_0 is the main ontology module: it includes all the previous modules and models a property (called *entry point*) for each imported sub-module.

Thanks to the knowledge modeling outlined so far, it is possible to describe *CV Profiles* in the ABox. The CV classification approach we propose is based on a role-free ABox, which includes only concept assertions of the form $P(a)$, stating that the candidate a (*i.e.*, her CV description) offers profile features P (see Definition 1).

Definition 1 (Profile). *Given the skill ontology $\mathcal{T}$, a profile $P = \sqcap(\exists R_j^0.C)$ is a $\mathcal{ALE}(\mathrm{D})$ concept defined as a conjunction of existential quantifications, where R_j^0, $1 \leq j \leq 6$, is an **entry point** and C is a concept in $\mathcal{FL}_0(\mathrm{D})$ modeled in the ontology module M_j.*

As hinted before, our knowledge compilation approach aims at translating the skill knowledge base into a relational model, without loss of information and expressiveness, in order to reduce on-line reasoning time. Relational schema modeling is therefore the most crucial design issue and it is strongly dependent on both knowledge expressiveness to be stored and reasoning to be provided over such a knowledge base. We recall that $\mathcal{FL}_0(\mathrm{D})$ concepts can be normalized according to the *Concept-Centered Normal Form* (CCNF), [1, Ch.2]. The availability of a finite normal form turns out to be very useful and effective, since all non-standard reasoning services performed by `I.M.P.A.K.T.` process the atomic information making up the knowledge descriptions, rather than the concept as a whole. Thus, we map the KB to the database according to the following design rules:
1) a table CONCEPT is created to store all the atomic information managed by the system: i) concept and role names; ii) the CCNF atoms of all the $\mathcal{FL}_0(\mathrm{D})$ concepts defined in modules M_j, with $1 \leq j \leq 6$; **2)** two tables mapping recursive relationships over the table CONCEPT, namely PARENT and ANCESTOR; **3)** a table PROFILE including the profile identifier ($profileID$ attribute) and the so called *structured information*: extra-ontological content, such as personal data (*e.g.*, last and first name, birth date) and work-related information (*e.g.*, preferred working hours, car availability); **4)** a table $R_j(X)$ is created for each entry point R_j^0, $1 \leq j \leq 6$ where X is the set of attributes $X = \{profileID, groupID, conceptID, value, lastdate\}$. Once the $CCNF(P) = \sqcap(\exists R_j^0.CCNF(C))$ of a profile P (see Definition 1) has been computed, the assertion $P(a)$ is stored in the database. `I.M.P.A.K.T.` produces a unique identifier for candidate a and assigns it to attribute $profileID$ in table PROFILE. Then, for each conjunct $\exists R_j^0.C$ belonging to $P(a)$, it adds one tuple for each atom of the

$CCNF(C)$ to the related table $R_j(X)$. Thus, all features modeled in profile descriptions according to Definition 1 are stored in tables $R_j(X)$ related to the involved entry points. Notice that, thanks to the fourth rule, our model can be easily extended. If the module M_0 in $\mathcal{T}$ is enhanced by a new entry point in order to capture a novel aspect of candidate CV, then the schema can be enriched by adding the corresponding table $R_j(X)$ to it.

3 Skill Matching Services

To evaluate the matching degree between a job request and a candidate profile, we need that both of them share the KB used for representation. Thus, the job requests submitted to `I.M.P.A.K.T.` have to be represented according to the syntax detailed in Definition 1. In particular, two groups of user requirements (preferences and strict constraints) compose a job request.

Formally, a *Job Request* $\mathcal{F}$ is defined as follows:

Definition 2 (Job Request). *A* Job Request $\mathcal{F}$ *is a Profile* $\mathcal{F} = \sqcap(\exists R_j^0.C)$ *(according to Definition 1), defined as a pair of feature sets* $\mathcal{F} = \langle \mathcal{FS}, \mathcal{FP} \rangle$ *such that:*

- $\mathcal{FS} = \{fs_i | 1 \leq i \leq s\}$ *is a set of* s *strictly required features* fs_i*, of the form* $\exists R_j^0.C_i$*;*
- $\mathcal{FP} = \{fp_k | 1 \leq k \leq p\}$ *is a set of* p *preferred features* fp_k *, of the form* $\exists R_j^0.C_k$*.*

`I.M.P.A.K.T.` provides two matchmaking processes, namely *Strict Match* and *Soft Match*, detailed in the following. More formally, *Strict Match* is defined as follows:

Definition 3 (Strict Match). *Given the ontology* $\mathcal{T}$*, a (part of) Job Request* $\mathcal{FS}$ *and a set* $\mathcal{P} = \{P(a_1), \ldots, P(a_n)\}$ *of* n *candidate profiles, modeled according to Definition 1 and stored in the DB according to the schema detailed in section 2, the* Strict Match *process returns all the candidate profiles* $P(a_j)$ *in* $\mathcal{P}$ *providing all the features* fs_i *in* $\mathcal{FS}$*.*

We notice that, thanks to the adoption of CCNF, the *Strict Match* can retrieve candidate profiles $P(a)$ also more specific than $\mathcal{FS}$. On the other hand, the *Soft Match* is devoted to implement the approach to approximate matching: the search has to revert also to candidates having some missing features and/or having features *slightly conflicting* w.r.t. $\mathcal{FP}$. We notice that, according to the formalism adopted, inconsistency may happen *e.g.*, when we have a preference $fp_k = \exists R_j^0.C_k$, with $C_k = D\sqcap \geq_n p$, and a candidate profile $P(a)$ with a specified feature $\exists R_j^0.C$, where $C = D\sqcap =_m p$, with $m < n$. In order to satisfy user preferences, candidate profiles modeling concrete features with values in an interval around the required value could represent a good result. We name such concrete features as *slightly conflicting* features (see Definition 4, $MC3$ class).

In order to search for possible approximate matches, *Soft Match* needs to investigate on single atoms of CCNF($\mathcal{FP}$) and compare them with candidate profiles features, which are stored in the DB in their CCNF. Thus, $\mathcal{FP}$ elements need to be further manipulated before the execution of *Soft Match* (notice that for *Strict Match* `I.M.P.A.K.T.` compares candidates features with the ones in $\mathcal{FS}$ without any preprocessing of $\mathcal{FS}$).

More formally, we define *Soft Match* as:

Definition 4 (Soft Match). *Given the skill ontology $\mathcal{T}$, a (part of) Job Request $\mathcal{FP}$ and a set $\mathcal{P} = \{P(a_1), \ldots, P(a_n)\}$ of candidate profiles, modeled according to Definition 1 and stored in the DB according to the schema detailed in Section 2, the Soft Match process returns a ranked list of candidate profiles $P(a_j)$ in $\mathcal{P}$ belonging to one of the following match classes:*

1. *$MC1$ is the set of profiles $P(a_j)$, such that each $P(a_j)$ provides at least one feature atom corresponding to a concept name in $fp_k \in CCNF(\mathcal{FP})$[9];*
2. *$MC2$ is the set of profiles $P(a_j)$, such that each $P(a_j)$ fully satisfies at least one feature $fp_k \in CCNF(\mathcal{FP})$ combining in C_k both a concept name and a concrete feature[10];*
3. *$MC3$ is the set of profiles $P(a_j)$, such that each $P(a_j)$ partially satisfies one feature $fp_k \in CCNF(\mathcal{FP})$ combining in C_k both a concept name and a concrete feature[11].*

Finally, in the most general case of job request $\mathcal{F}$ containing both $\mathcal{FS}$ and $\mathcal{FP}$, `I.M.P.A.K.T.` performs a two-step matchmaking approach, namely *Matchmaking*, which starts with *Strict Match* process, computing a set of profiles fully satisfying strict requirements, and then proceeds with *Soft Match* process, trying to approximately match preferences with profiles belonging to the set returned by *Strict Match*.

According to Definition 3, results retrieved by *Strict Match* have a 100% coverage level of the job request $\mathcal{F}$ and thus they do not need to be ranked after retrieval. On the contrary, a ranking process according to a unified measure is necessary for *Soft Match* resulting profiles w.r.t. $\mathcal{F}$. We remind that, among CCNF atoms deriving from features $fp_k \in \mathcal{FP}$, `I.M.P.A.K.T.` distinguishes between atomic concepts and value restrictions (*i.e.*, qualitative information) and concrete features (*i.e.*, quantitative information), since they need a different manipulation in the ranking process. `I.M.P.A.K.T.` computes a logic-based ranking by applying the following rules: **(1)** each conjunct in the retrieved candidate profile receives a score on the basis of the number and type (concept name or value restriction or concrete feature) of matched features $fp_k \in \mathcal{FP}$; **(2)** each conjunct ranked according to rule 1 is "re-weighted" based on the relevance of its related entry point R^0_j, $1 \leq j \leq 6$. In rule **(1)**, the score for qualitative information is computed by simply counting the retrieved atoms matching the requested ones. On the other hand, in order to assign a score to each feature specification involving p in a candidate profile, $\mathcal{FP}$ features in the form $\geq_n p$, $=_n p$ and $\leq_n p$ are managed by a different and specifically designed scoring function. Examining the second rule in our score computation strategy, it is easy to notice that a relevance order relation needs to be set among entry points (see the following formula (1) for our current implementation). Both *Strict Match* and *Soft Match*, regardless of their different behavior w.r.t. ranking, share the same *Explanation* process of match between a retrieved candidate $P(a)$ and a job request $\mathcal{F}$. Such a process classifies profile features w.r.t. each requirement in $\mathcal{F}$ in the following four groups: ***Fulfilled***: $P(a)$ features either perfectly matching or slightly conflicting those requested by $\mathcal{F}$; ***Conflicting***: $P(a)$ features slightly conflicting with $\mathcal{FP}$ requirements; ***Additional***: $P(a)$ features either more specific than the ones required in $\mathcal{F}$ or

[9] See queries $\mathcal{Q}(fp_k)$ and $Q_{NULL}(fp_k)$ in Section 3.1
[10] See query $Q_n(fp_k)$ in Section 3.1
[11] See query $Q_{n_{m\%}}(fp_k)$ in Section 3.1

not exposed in the user request and belonging to entry point $R_j^0 = hasKnowledge$; ***Underspecified***: $\mathcal{F}$ requirements which are not included in $P(a)$ features.

We notice that for a Job Request $\mathcal{F}$ such that $\mathcal{FP} = \emptyset$, that is the case of *Strict Match* only, the explanation of the match related to each returned $P(a)$, is characterized by empty sets of *Underspecified* and *Conflicting* features and by a set of *Fulfilled* features equivalent to $P(a)$ itself.

3.1 SQL-based Implementation

Coherently with the approach introduced and motivated so far, once our KB has been pre-processed and stored into the DB according to our relational schema, `I.M.P.A.K.T.` is able to perform all the reasoning services only through standard SQL queries. Notice that we do not use a specific preference language as in [10, 3, 12] but we exploit a set of standard SQL queries built on-the-fly according to both user requirements (*i.e.*, strict requirements and preferences) and required features (*i.e.*, atoms contained in each feature).

Let us consider a strict requirement fs_i of the form $\exists R_j^0.C_i$. We recall that C_i is a concept description in $\mathcal{FL}_0(\mathrm{D})$ which we can model as a conjunction of concepts defined according to the KB modeling: A – concept name, $\forall R.D$ – universal quantification, $\leq_n p (\geq_n p, =_n p)$ – concrete feature, *i.e.* $fs_i = \exists R_j^0.(A \sqcap \forall R.D \sqcap \geq_n p)$. From database querying point of view, fs_i has to be translated in a set of syntactic elements to search for in the proper $R_j(X)$ table. *Strict Match* asks for a profile to include all of the previous syntactic elements to be retrieved. Since each of these elements fills one tuple of a $R_j(X)$ table, the resulting query, $Q_s(fs_i)$, retrieves a results set by adopting the following conceptual schema: (*set of profiles in* $R_j(X)$ *containing* A) INTERSECT (*set of profiles in* $R_j(X)$ *containing* $R.D$) INTERSECT (*set of profiles in* $R_j(X)$ *containing* $\geq_n p$)[12].

According to such a schema and the required fs_i, the query $Q_s(fs_i)$ is automatically built on-the-fly considering a number of conditions in `WHERE` clause defined according to atoms in C_i. In particular, Fig. 1 presents an executable example for the query $fs_i = \exists hasknowledge.(Java \sqcap \forall skillType.Programming \sqcap \geq_3 years)$. We notice that $Q_s(fs_i)$ in Fig. 1 has three conditions in `WHERE` clause, as expected. On the other hand, *Soft Match* relies on a query schema involving each element $CCNF(fp_k), \forall fp_k \in \mathcal{FP}$. In particular, let $CCNF(fp_k) = \exists R_j^0.CCNF(C_k)$ be a normalized preference; a single query $\mathcal{Q}(fp_k)$ or a set of queries $\mathcal{Q}_p(fp_k)$ is built according to the following schema:

– if none of $\{\leq_n p, \geq_n p, =_n p\}$ elements occur in $CCNF(C_k)$, then a single query $\mathcal{Q}(fp_k)$ is built which retrieves the profiles containing – w.r.t. the related *entry point* R_j^0– at least one among syntactic element occurring in $CCNF(C_k)$;

[12] To improve engine performance, `I.M.P.A.K.T.` exploits, as far as possible, `EXISTS` operator instead of `INTERSECT`. Also for performance reasons, conditions in the form `conceptID=(SELECT conceptID FROM CONCEPT WHERE name='X')` are not executed at run-time but a lookup on a hash table directly assigns the proper `conceptID` value.

```
SELECT profileID
FROM hasKnowledge as R
WHERE conceptID = (SELECT conceptID
                   FROM concept WHERE name='Java')
AND EXISTS (SELECT *
            FROM hasKnowledge
            WHERE profileid=R.profileid AND groupid=R.groupid
            AND conceptid = (SELECT conceptID
                             FROM concept WHERE name='skillType.Programming'))
AND EXISTS (SELECT *
            FROM hasKnowledge
            WHERE conceptid=(SELECT conceptID
                             FROM concept WHERE name='years')
            AND value >= 3 AND profileid=R.profileid AND groupid=R.groupid)
```

Fig. 1. SQL definition of query $Q_s(fs_i)$ w.r.t. a single feature $fs_i = \exists hasKnowledge.(Java \sqcap \forall skillType.Programming \sqcap \geq_3 years)$

– otherwise a set of queries $\mathcal{Q}_p(fp_k) = \{Q_n(fp_k), Q_{NULL}(fp_k), Q_{n_{m\%}}(fp_k)\}$ is built retrieving candidate profiles belonging to a different match class –*i.e.*, either profiles fulfill fp_k ($Q_n(fp_k)$) or profiles do not fulfill it ($Q_{n_{m\%}}(fp_k)$) or profiles do not specify p ($Q_{NULL}(fp_k)$). The resulting set of candidate profiles is made up by the `UNION` of all the tuples retrieved by each of the query in $\mathcal{Q}_p(fp_k)$.

As for *Strict Match*, for each $CCNF(fp_k)$ the previous queries are automatically built on-the-fly according to syntactic elements occurring in $CCNF(C_k)$. Here, due to the lack of space, we do not report the SQL definition of both $\mathcal{Q}(fp_k)$ query and the set of queries $\mathcal{Q}_p(fp_k)$. We only notice that the score for each retrieved atom (*i.e.*, tuple) of candidate feature is computed directly in the SELECT clause of each query implementing the *Soft Match*. In particular, for qualitative information (*i.e.*, atomic concepts and value restrictions) score is equal to 1, whereas for concrete feature p score is an expression computed according to scoring functions aforementioned strategy. Moreover, we notice that, by construction, *Soft Match* retrieves candidate profiles belonging to one of the match classes in Definition 4 for each feature fp_k. Thus, such candidates profiles have to be properly rearranged for defining the final results set. Each retrieved profile is finally ranked according to a linear combination of scores:

$$rank = \sum_{i=1}^{N} w_i * score_{l^i} \tag{1}$$

where w_i are heuristic coefficients belonging to the $(0, 1)$ interval, N is the number of relevance levels defined for the domain ontology and $score_{l^i}$ represents the global score computed summing the score of tuples related to entry points falling in the same relevance level l^i. `I.M.P.A.K.T.` defines a number $N = 3$ of ontology levels represented by $Level = \{l^1, l^2, l^3\}$ (l^1 is the most relevant one), with *hasKnowledge* set to l^1; *hasIndustry*, *hasComplementarySkill* and *hasJobTitle* set to l^2 and the remaining entry points set to l^3. Moreover, the following values are assigned to w_i coefficients: $w_1 := 1$, $w_2 := 0.75$ and $w_3 := 0.45$.

4 System Performances

In this section we focus on the evaluation of *data complexity* and *expressiveness complexity* of our knowledge compilation approach and present obtained results.`I.M.P.A.K.T.` is a client-server application developed in Java. Our current implementation exploits the open source PostgreSQL 9.1 DBMS. In order to prove the effectiveness and efficacy of the proposed approach, we initially created a real dataset by collecting approximately 180 CVs on ICT domain, originated from three different employment agencies. The dataset has been exploited for an iterative refinement phase of both the Skill Ontology development and the setting of the Skill Matching parameters (*i.e.*, entry points levels and weights in scoring strategy). We implemented a synthetic KB instances generator able to automatically build satisfiable profiles according to a given format (*i.e.*, number of features for each relevance level, number of numeric restrictions, etc.). In this way, we generated datasets having different size, ranging from 500 to 5500 profiles, with bigger datasets including the smaller ones. We point out that for the datasets construction we considered a number of features for each candidate comparable to the average value of candidate profiles in the previous mentioned real dataset. In particular, each generated profile has at least: 30 features for *hasKnowledge* entry point, 2 features for *hasLevel* and *knowsLanguage* entry points, and 3 features for *hasJobTitle*, *hasIndustry* and *complementarySkill* entry points. Tests refer to `I.M.P.A.K.T.` running on an Intel Dual Core server, equipped with a 2.26 GHz processor and 4 GB RAM and measure the retrieval time calculated as average time over ten iterations. Here we report retrieval times of 9 significant queries, selected among several test queries with a different expressiveness divided into 3 groups: **(A)** only strict requirements represented by either generic concepts (Q_4) or features with an higher specificity (Q_5); **(B)** only preferences again represented by either generic concepts (Q_2) or features with an higher specificity (Q_3); **(C)** a combination of all A) and B) groups features (Q_1,Q_6,Q_7,Q_8,Q_9). We notice that: 1) Q_1 query is a translation in our formalism of a real job request available on `http://jobview.monster.co.uk` titled *"SQL Developer (Business Intelligence)"* and containing 2 strict and 12 soft requirements for entry point *hasKnowledge* and only one soft request for entry point *complementarySkill*; 2) queries from Q_2 to Q_7 are composed by one feature for each entry point; 3) $Q_6 = Q_2 \cup Q_4$ and $Q_7 = Q_3 \cup Q_5$; 4) Q_8 involves only three entry points, *i.e.*, *hasKnowledge*, *knowsLanguage* and *hasLevel*; 5) Q_9 involves several features for each entry point.

Table 1 shows the retrieval times together with the number of retrieved profiles ($\#p$) for each dataset and request. In particular, in order to better evaluate matching performances, we differentiate among the request normalization process times (see t_n in Table 1), which is dataset-independent, *Strict Match* retrieval times (see t_{st} in Table 1) and *Soft Match* retrieval times (see t_{sf} in Table 1) including also the ranking calculation times.

As we expected, retrieval times of both match procedures linearly increase with datasets size (*e.g.* see Q_5). In particular, *Strict Match* times are also dramatically affected by $\#p$ (see results for DS_5 in Table 1), whereas the *Soft Match* times seem to grow more slowly with $\#p$. We therefore observe that the number of retrieved profiles, though affecting the whole matchmaking process, mostly impacts *Strict Match*, since it involves the SQL intersection of several queries by construction. In particular, profiles

Table 1. Retrieval times in milliseconds and number of retrieved profiles ($\#p$) for datasets DS_1, DS_2, DS_3, DS_4 and DS_5 of, respectively, 500, 1000, 2000, 3500 and 5500 profiles.

		DS_1			DS_2			DS_3			DS_4			DS_5		
	t_n	t_{st}	t_{sf}	$\#p$	t_{st}	t_{sf}	$\#p$	t_{st}	t_{sf}	$\#p$	t_{st}	t_{sf}	$\#p$	t_{st}	t_{sf}	$\#p$
Q_1	724.4	124.2	210.6	4	246.6	240.8	10	545.6	382.5	20	784.5	402.2	28	2334.8	551.8	144
Q_2	335.8	0	305.7	461	0	456.8	927	0	563.6	1829	0	756.2	3202	0	1158.8	5029
Q_3	474.8	0	440.5	396	0	578.2	740	0	782.2	1560	0	1624.8	2729	0	2775	4270
Q_4	225.9	71.2	0	10	110.4	0	13	212.8	0	23	336.4	0	35	423	0	52
Q_5	224.4	74.1	0	1	115.2	0	1	218	0	1	342	0	1	441.4	0	1
Q_6	240.6	96.7	103.8	10	147.4	128.4	13	227.4	139.4	23	344.4	173	35	485.5	180.4	52
Q_7	538.8	84.8	97.8	1	119.8	133.4	1	219.2	179.6	1	343.8	193.8	1	473.2	208	1
Q_8	347	228.6	96.6	17	456.6	113	44	927	125.2	79	1277.4	132.4	131	2593.4	196.8	226
Q_9	317.8	136.8	163	3	244.2	166.5	3	385.6	168.6	4	671.2	180	5	1245.8	252	7

returned by Q_5 are dataset-independent, as the *Strict Match* procedure always returns the same profile (*i.e.*, no other profile satisfying strict requirement exists in the datasets). In order to verify the approach expressiveness complexity, we evaluated retrieval times of different test queries on one dataset at a time. It has to be observed that: (i) for queries only expressing preferences (Q_2,Q_3) or only strict requirements (Q_4,Q_5), the retrieval time increases with the query expressiveness; (ii) for the other queries, thanks to preliminary execution of *Strict Match*, the *Soft Match* times are always notably reduced, so confirming the theoretical complexity results. In particular, we notice that for larger data sets and a number of retrieved profiles larger than 3000 (see t_{sf} in Q_2 and Q_3 on DS_4, DS_5), expressiveness of soft requirements has a more relevant impact on retrieval times. Moreover, for each dataset, the real-data query Q_1 has retrieval times comparable to all queries belonging to C group considering also the $\#p$ value. Thus, in the whole matchmaking process, involving both strict requirements and preferences, the query expressiveness does not significantly affect retrieval times.

Summing up, we can claim that `I.M.P.A.K.T.` is able – with time performances encouraging its application in real-world scenarios – to provide crucial value-added information with respect to typical HR management tasks, even on large datasets.

5 Discussion and Future Work

Motivated by the need to efficiently cope with real-life datasets in semantic-enhanced skill matching, we presented a knowledge compilation approach able to translate a KB into a relational database while retaining the expressiveness of the logical representation. The obtained model allows to perform reasoning services through standard-SQL queries, in the framework of `I.M.P.A.K.T.`. Performance evaluations on various datasets show an efficient behavior although several optimization techniques have not been implemented yet. Future work aims at testing further devised strategies for score calculation, including the possibility for the user to assign a weight to each preference, along with a full optimization of the database.

6 Acknowledgments

We gratefully acknowledge support of projects UE ETCP ”G.A.I.A.” and Italian PON ERMES ”Enhance Risk Management through Extended Sensors”.

References

1. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook – 2nd edition. Cambridge University Press (2007)
2. Cadoli, M., Donini, F.M.: A survey on knowledge compilation. AI Commun. 10(3-4), 137–150 (1997)
3. Chomicki, J.: Querying with Intrinsic Preferences. In: Proc. of EDBT 2002. pp. 34–51. Springer (2002)
4. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme. In: Proc. of VLDB 2005. pp. 1216–1227. VLDB Endowment (2005)
5. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F., Mongiello, M.: Concept Abduction and Contraction for Semantic-based Discovery of Matches and Negotiation Spaces in an E-Marketplace. In: Proceedings of the 6th Int. Conf. on Electronic Commerce, ICEC’04. pp. 41–50 (2004)
6. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F., Piscitelli, G., Coppi, S.: Knowledge Based Approach to Semantic Composition of Teams in an Organization. In: Proceedings of the 20th Annual ACM (SIGAPP) Symposium on Applied Computing SAC-05. pp. 1314–1319. ACM (2005)
7. Colucci, S., Tinelli, E., Sciascio, E.D., Donini, F.M.: Automating competence management through non-standard reasoning. Engineering Applications of Artificial Intelligence 24(8), 1368–1384 (2011)
8. Di Noia, T., Di Sciascio, E., Donini, F.: Extending Semantic-Based Matchmaking via Concept Abduction and Contraction. In: Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004). LNAI, Springer (2004)
9. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: Proc. of AAAI 2007 (2007)
10. Kießling, W.: Foundations of Preferences in Database Systems. In: Proc. of VLDB 2002. pp. 311–322. Morgan Kaufmann, Los Altos (2002)
11. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - A Pragmatic Semantic Repository for OWL. In: Proc. of WISE’05. vol. 3807, pp. 182–192. Springer (2005)
12. P. Bosc and O. Pivert: SQLf: a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems 3(1), 1–17 (Feb 1995)
13. Tinelli, E., Cascone, A., Ruta, M., Di Noia, T., Di Sciascio, E., Donini, F.M.: I.M.P.A.K.T.: An Innovative Semantic-based Skill Management System Exploiting Standard SQL. In: Proc. of ICEIS 2009. pp. 224–229 (2009)
14. Tinelli, E., Colucci, S., Di Sciascio, E., Donini, F.M.: Knowledge compilation for automated team composition exploiting standard SQL. In: proc. of ACM SAC ’12. pp. 1680–1685 (2012)