Extending Knowledge-Based Profile Matching in the Human Resources Domain

Alejandra Lorena Paoletti, Jorge Martinez-Gil, Klaus-Dieter Schewe

Extending Knowledge-Based Profile Matching in the Human Resources Domain

Lorena Paoletti[1], Jorge Martinez-Gil[1], Klaus-Dieter Schewe[1,2]

[1]Software Competence Center Hagenberg, Austria
[2]Johannes-Kepler-University Linz, Austria
{Lorena.Paoletti, Jorge.Martinez-Gil, kd.schewe}@scch.at

Abstract. In the Human Resource domain the accurate matching between job positions and job applicants profiles is crucial for job seekers and recruiters. The use of recruitment taxonomies have proven to be of significant advantage in the area by enabling semantic matching and reasoning. Hence, the development of Knowledge Bases (KB) where curricula vitae and job offers can be uploaded and queried in order to obtain the best matches by both, applicants and recruiters is highly important. We introduce an approach to improve matching of profiles, starting by expressing jobs and applicants profiles by filters representing skills and competencies. Filters are used to calculate the similarity between concepts in the subsumption hierarchy of a KB. This is enhanced by adding weights and aggregates on filters. Moreover, we present an approach to evaluate over-qualification and introduce blow-up operators that transform certain role relations in a KB where matching of filters can be applied.

1 Introduction

In the Human Resources (HR) domain the accurate matching of job applicants to position descriptions and vice versa is of central importance for employers and job seekers. Therefore, the development of data or knowledge bases to which job descriptions and curricula vitae (CV) can be uploaded and, which can be queried effectively and efficiently by both, employers and job seekers to find best matching candidates for a given job profile and, best suitable job offers matching a given applicant skill set, respectively, is of high importance.

It seems appropriate to consider knowledge bases for the representation and thus the storage of the (job and CV) profiles, which in addition to pure storage would support the reasoning about profiles and their classification. It seems reasonable to exploit the underlying lattice structure of knowledge bases, i.e., the partial order on concepts representing skills. For instance, a skill such as "knowledge of C" is more detailed than "programming knowledge". Thus, defining profiles by filters, i.e., upward-closed sets of skills (e.g., if "knowledge of C" is in the profile, then "programming knowledge" is in there as well) and using measures on such filters as the basis for the matching seems adequate.

Concerning automatic matching of candidate profiles and job profiles, the commercial practice is largely dominated by Boolean matching, i.e. for a requested profile it is merely checked how many of

the requested terms are in the candidate profile [8][9] which amounts to simply counting the number of elements in difference sets. This largely ignores similarity between skills, e.g. programming skills in C++ or Java would be rated similar by a human expert.

Improving this primitive form of matching requires at least taking hierarchical dependencies between skill terms into account. For this various taxonomies have already been developed such as DISCO competences [12], ISCO [13] and ISCED [14]. Taxonomies can then be refined by using knowledge bases (ontologies) based on common description logics, which have been studied in depth for more than 20 years [1]. However, sophisticated knowledge bases in the HR domain are still rare, as building up a good, large knowledge base is a complex and time-consuming task, though in principle this can be done as proven by experiences in many other application domains [6].

Ontologies and more precisely description logics have been used as the main means for knowledge representation for a long time [5]. The approach is basically to take a fraction of first-order logic, for which implication is decidable. The common form adopted in description logics is to concentrate on unary and binary predicates known as concepts and roles, and to permit a limited set of constructors for concepts and roles. Then the terminological layer (TBox) is defined by axioms usually expressing implication between concepts. In addition, an assertional layer (ABox) is defined by instances of the TBox (or equivalently a ground theory) satisfying the axioms. The various description logics differ mainly by their expressiveness. A prominent representative of the family of description logics is $\mathcal{SROIQ}$-$\mathcal{D}$, which forms the formal basis of the web ontology language OWL-2 [4], which is one of the more expressive description logics. As the aim of this work is not focused on developing novel ideas for knowledge representation, but merely intends to use knowledge representation as grounding technology for the semantic representation of job offers and candidate CVs, it appears appropriate to fix $\mathcal{SROIQ}$-$\mathcal{D}$ as the description logics to be used in this work.

The lattice-like structure of concepts within a Knowledge Base provides basic characteristics to determine the semantic similarity between concepts included in both, job descriptions and curricula vitae. The matching algorithms implemented to determine the semantic similarity between concepts should allow to compare job descriptions and applicants profiles based on their semantics. By comparing the concepts contained within a particular job description against the applicants profile to that particular job through different categories, (i.e., competencies, education, skills) it is possible to rank the candidates and select the best matches for the job.

The two profiles (job descriptions and applicants) are defined by means of *filters*. If $\leq$ denotes the partial order of the lattice in the TBox, then a filter on the TBox is an *upward-closed*, non-empty set of concepts. Filter-based matching on grounds of partially ordered sets are the starting point of this work, this has been investigated previously [10]. The simple idea is that, for two filters $\mathcal{F}_1$ and $\mathcal{F}_2$ a matching value $m(\mathcal{F}_1, \mathcal{F}_2)$ is computed as $\#(\mathcal{F}_1, \mathcal{F}_2)/\ \#\mathcal{F}_2$, i.e. by counting numbers of elements in filters. Experiments based on DISCO already show that this simple filter-based measure significantly improves the matching accuracy [7].

The goal of our research is to provide solid techniques to improve the matching process of job and applicants profiles within the HR domain. We will show how adding weights on filters and categories can significantly improve the quality of the matching results based on filter-based matching on grounds of partially ordered sets. As part of the matching process, we also address the problem of over-qualification that clearly cannot be captured solely by means of filters. Finally, we introduce the novel concept of 'blow-up" operators in order to extend the matching by integrating roles in the TBox. The idea is to expand the TBox by using roles in order to define arbitrarily many sub-concepts so that the original matching measures could again be applied.

In this approach, research on the knowledge base will be based on a subset of the description logics $\mathcal{SROIQ}$-$\mathcal{D}$ that is introduced in Section 2. An example of a TBox and how to manipulate concepts in order to perform reasoning about it is presented in Section 3. We define the filter-based matching in Section 4. The introduction of weights on filters is presented in Section 4.1

while weighted aggregates on categories of profiles is introduced in Section 4.2. In Section 4.3 the problem of over-qualification is addressed. And finally, "blow-up" operators is introduced in Section 4.4.

2 Profile Matching in Description Logics

The representation of knowledge within taxonomies is used to represent the conceptual terminology of a problem domain in a structured way in order to perform reasoning about it. In this section, we introduce the syntax and the semantics of the language we use to represent the conceptual knowledge of the Human Resources domain within this work. This language is a subset of the Description Logics $\mathcal{SROIQ}$-$\mathcal{D}$.

The most elementary components of the logic are *atomic concepts* and *atomic roles*, denoted by the letters C and R respectively. Atomic concepts denote sets of objects and atomic roles denote binary relationships between atomic concepts. Note that the terms "concepts" and "sets" are not synonyms. While a set is a collection of arbitrary elements of the universe, a concept is an expression of the formal language of the description logics. *Nominal names* are names of individuals in the description language. Concept descriptions can be build using concept constructors as follows.

Definition 1. ***(Syntax of Concept Descriptions)***
Concept description are defined by the following syntax rules:

$$
\begin{array}{lcll}
C_1, C_2 & \longrightarrow & A \mid \top \mid \bot \mid & \\
 & & \neg C_1 \mid & \textit{negation of a concept } C_1 \textit{(or, complement of } C_1\textit{)} \\
 & & C_1 \sqcup C_2 \mid & \textit{union} \\
 & & C_1 \sqcap C_2 \mid & \textit{intersection} \\
 & & C_1 \sqsubseteq C_2 \mid & \textit{subsumption} \\
 & & \exists R.C_1 \mid & \textit{existential restriction} \\
 & & \forall R.C_1 \mid & \textit{value restriction} \\
 & & \leq nR.C_1 \mid & \textit{cardinality restriction } \leq \\
 & & \geq nR.C_1 \mid & \textit{cardinality restriction } \geq \\
 & & = nR.C_1 & \textit{cardinality restriction } =
\end{array}
$$

where A denotes an atomic concept (also known as concept name), $\top$ and $\bot$ denote the two reserved atomic concepts top and bottom which represent the universe and empty set, respectively, R denotes an atomic role (also known as role name), C_1 and C_2 denote concept descriptions and $n \in \mathbb{N}$.

The subsumption of the form $C_1 \sqsubseteq C_2$ denotes that C_1 is a subset of C_2. This is considered the basic reasoning service of the Knowledge Base. It is important to determine whether the concept C_2 is more general than the concept C_1 when using subsumption. This is, whether the elements of C_1 always denote a subset of the elements of C_2. Situations where, for instance, the programming language C is a subset of Programming Languages within a taxonomy can be expressed by using subsumption $\mathsf{C} \sqsubseteq \mathsf{Programming\ Languages}$.

Role descriptions are build from atomic roles, atomic concepts and nominal names as follows.

Definition 2. ***(Syntax of Role Descriptions)***
Let R_1, R_2 be role names and a nominal name a. Then every role name is a role description such that inverse role R_1^-*, roles involving individual names $\exists R_1.\{a\}$, and* role chain $R_1 \circ R_2$ *are also role descriptions.*

A role involving individuals of the form $\exists R.\{a\}$ denotes the set of all objects that have a as a "filler" of the role R. For example, $\exists$SpokenLanguage.{Russian} denotes that Russian is a spoken language. Inverse roles R_1^- are use to describe passive constructions, i.e., *a person owns something* (Owns.Person) can be expressed as *something is owned by a person* (Owns$^-$.Thing). Two binary relations can be composed to create a third relation. For instance, having a role R_1 that relates the element a_1 to element a_2 and role R_2 that relates a_2 with a_3, we can relate a_1 with a_3 by using role chain, this is $R_1 \circ R_2$. For example: by building a composition of the role hasSkill, that relates elements of concept Person with elements of a given Competency, with the role hasProficiencyLevel, that relates Competences with ProficiencyLevel, we have:

$$\mathsf{hasSkill} \circ \mathsf{hasProficiencyLevel}$$

that produces the proficiency level of individuals with experience in a particular competency. We can also define a role hasSkillExperience and express it as:

$$\mathsf{hasSkill} \circ \mathsf{hasProficiencyLevel} \sqsubseteq \mathsf{hasSkillExperience}$$

Note that property chain can only occur in the left hand side of the subsumption. This is given by the fact that hasSkillExperience is not a sufficient condition to say that someone hasProficiencyLevel in a particular Competency. In general terms, n roles can be chained to form a new role $R_1 \circ \cdots \circ R_n$.

We introduce the concept of an *interpretation* in order to define the formal semantics of the language. Concrete situations are modeled in logic through interpretations that associate specific concept names to individuals of the universe. An interpretation $\mathcal{I}$ is a non-empty set $\Delta^{\mathcal{I}}$ called the domain of the interpretation $\mathcal{I}$. We sometimes use also $\mathcal{D}$ to denote $\Delta^{\mathcal{I}}$. The interpretation function assigns, to every atomic concept C a set $\Delta(C) \subseteq \mathcal{D}$ and, to every role R a binary relation $\Delta(R) \subseteq \mathcal{D} \times \mathcal{D}$.

Definition 3. ***(Semantic of the language)***
Given an interpretation $\mathcal{I}$, the atomic concepts top and bottom are interpreted as $\Delta(\top) = \mathcal{D}$ and $\Delta(\bot) = \varnothing$ and, the interpretation function can be extended to arbitrary concept and role descriptions as follows:

$$\begin{aligned}
\Delta(C_1 \sqcap C_2) &= \Delta(C_1) \cap \Delta(C_2),\\
\Delta(C_1 \sqcup C2) &= \Delta(C_1) \cup \Delta(C_2),\\
\Delta(\neg C) &= \mathcal{D} \backslash \Delta(C),\\
\Delta(C_1 \sqsubseteq C_2) &= \Delta(C_1) \subseteq \Delta(C_2),\\
\Delta(\forall R.C) &= \{a \in \mathcal{D} | \forall b.(a,b) \in \Delta(R) \rightarrow b \in \Delta(C)\},\\
\Delta(\exists R.C) &= \{a \in \mathcal{D} | \exists b.(a,b) \in \Delta(R)\},\\
\Delta(\leq nR.C) &= \{a \in \mathcal{D} | \#\{b \in \Delta(C) | (a,b) \in \Delta(R)\} \leq n\},\\
\Delta(\geq nR.C) &= \{a \in \mathcal{D} | \#\{b \in \Delta(C) | (a,b) \in \Delta(R)\} \geq n\},\\
\Delta(= nR.C) &= \{a \in \mathcal{D} | \#\{b \in \Delta(C) | (a,b) \in \Delta(R)\} = n\},\\
\Delta(R.\{a\}) &= \{b \in \mathcal{D} | (b,a) \in \Delta(R)\},\\
\Delta(R^-) = \Delta(R)^{-1} &= \{(b,a) \in \mathcal{D}^2 | (a,b) \in \Delta(R)\},\\
\Delta(R_1 \circ \cdots \circ R_n) \sqsubseteq \Delta(S) &\equiv \{(a_0,a_1) \in \Delta(R_1), \ldots, (a_{n-1},a_n) \in \Delta(R_n) | (a_0,a_n) \in \Delta(S)\}.
\end{aligned}$$

The *number restrictions*, $\leq nR.C$, $\geq nR.C$ and, $= nR.C$ denote, all elements that are related through the role R to at least n, at most n or, exactly n elements of the universe, respectively, where $n \in \mathbb{N}$ and $\#$ denotes the cardinality of the set.

New concepts can be introduced from previously defined concepts by using logical equivalence $C_1 \equiv C_2$. For instance, FunctionalProgrammer $\equiv$ Lisp $\sqcup$ Haskell introduce the concept Functional-Programmer denoting all individuals that have experience programming in Lisp or Haskell, or both

. In this context, a concept name occurring in the left hand side of a concept definition of the form $C_1 \equiv C_2$ is called a *defined concept*.

We have introduced in this section a subset of $\mathcal{SROIQ}$-$\mathcal{D}$ that is sufficient for this work. Although, for a comprehensive detail of description logics we recommend [2].

3 Representation of Profile Knowledge

Knowledge representation based on description logics is comprised by two main components, the Terminological layer or *TBox* for short, and the Assertional layer, or *ABox*. The TBox contains the terminology of the domain. This is the general knowledge description about the problem domain. The ABox contains knowledge in extensional form, describing characteristics of a particular domain by specifying it through individuals.

Within the TBox, it is possible to describe inclusion relation between concepts by using subsumption. Hence, we can specify, for instance that, Computing is part of Competences and, Programming is part of Computing and, different Programming Languages are included within Programming such that:

$$\begin{aligned} \mathsf{LISP} &\sqsubseteq \mathsf{Programming\ Languages} \sqsubseteq \mathsf{Programming} \sqsubseteq \mathsf{Computing} \sqsubseteq \mathsf{Competences} \\ \mathsf{Java} &\sqsubseteq \mathsf{Programming\ Languages} \sqsubseteq \mathsf{Programming} \sqsubseteq \mathsf{Computing} \sqsubseteq \mathsf{Competences} \\ &\vdots \end{aligned}$$

this gives rise to a partial order on the elements of the Knowledge Base. Given the nature of subsumption of concepts within Knowledge Bases, TBoxes are lattice-like structures. This is purely determined by the subsumption relationship between the concepts that determine a partially ordered set of elements. In this partially ordered set, the existence of the greatest lower bound (LISP, Java) is trivial which also implies the existence of the least upper bound (Competences).

In ABoxes, we specify properties about individuals characterized under a specific situation in terms of concepts and roles. Some of the concept and role atoms in the ABox may be defined names of the TBox. Thus, within an ABox, we introduce individuals by giving them names $(a_1, a_2, \dots)$, and we assert their properties trough concepts C and roles R. This is, *concept assertions* $C(a_1)$, denote that a_1 belongs to the interpretation of C and; *role assertions* $R(a_1, a_2)$, denote that a_1 is a filler of the role R for a_2.

As an example, we consider the TBox in Figure 1 corresponding to the Competences sub-lattice in Figure 2 that represent a small set of Programming Languages. Note that, we have refined the relation between the concepts in order to reflect the conceptual influence between the different programming languages. Note also that Programming Languages (PL) is not the least upper bound in Figure 2. For convenience, we have suppressed the upper part of the subsumption structure of the sub-lattice (Programming Languages $\sqsubseteq$ Programming $\sqsubseteq$ Computing $\sqsubseteq$ Competences).

In this TBox, atomic concepts are not defined as such but, they are used in concept descriptions and defined concepts. Concept descriptions describe mainly the subsumption structure of the atomic concepts while defined concepts describe the following characteristics of programming languages. The set of programming languages with a C-like structure, this is C-Family; the set of all programming languages but Java, NoJava; Programmer defines every individual that has experience programming with at least one programming language and Polyglot describes all individuals that have experience in programming in two or more programming languages. There is only one role here, hasSkill denoting all objects having some experience in certain domain.

Under a given interpretation $\mathcal{I}$ with individuals $a_1, a_2 \in \mathcal{D}$, we can for instance express the queries C_0 and C_1 below. C_0 expresses that the individual a_1 has some experience in programming in Haskell while C_1 states that a_1 is a programmer in at least one of the C-Family languages but

Concept Description
Imperative ⊔ Object Oriented ⊔ Unix Shell ⊔ Functional ⊑ Programming Languages
C# ⊑ C++ ⊑ C ⊑ Imperative
C++ ⊑ Object Oriented
C++ ⊑ FORTRAN ⊑ Imperative
Defined Concepts
C-Family ≡ C# ⊔ C++ ⊔ C ⊔ Java ⊔ Perl
NoJava ≡ ∀hasSkill.¬Java
Programmer ≡ ∃hasSkill.C_i
Polyglot ≡ ⩾2∃hasSkill.C_i
Roles
hasSkill

Fig. 1. Programming Languages TBox

Java:

$$C_0 := \{(a_1, a_2) \in \Delta(\mathsf{hasSkill}) \wedge a_2 \in \Delta(\mathsf{Haskell})\}$$
$$C_1 := \{(a_1, a_2) \in \Delta(\mathsf{hasSkill}) \wedge a_2 \in \Delta(\exists \mathsf{C\text{-}Family}) \wedge a_2 \in \Delta(\mathsf{NoJava})\}$$

If a_1 satisfies $C_0 \sqcup C_1$ and given that Δ(C-Family) is the set composed by {C#, C++, C, Java, Perl}, we can deduce other characteristics of a_1 in this ABox:

$a_1 \in \Delta(\mathsf{Programmer})$	$a_1$is a programmer
$a_1 \in \Delta(\mathsf{Polyglot})$	$a_1$is a polyglot programmer
$a_1 \in \Delta(\mathsf{Imperative})$	$a_1$has knowledge in Imperative Paradigm
$a_1 \in \Delta(\mathsf{Functional})$	$a_1$has knowledge in Functional Paradigm
$a_1 \in \Delta(\mathsf{Objec\ Oriented})$	$a_1$has knowledge in Object Oriented Paradigm

4 Matching Theory

In the Human Resource sector, the data exchange between employers and job applicants is based on a set of shared vocabularies or taxonomies describing relevant terms within the domain, i.e.: competencies, education, skills, etc. Knowledge Bases act as repository-like structures for the domain specific knowledge. The lattice-like structure of concepts within a Knowledge Base provides basic characteristics to determine the semantic similarity between concepts included within the two profiles: job descriptions and curricula vitae. We distinguish the two profiles involved by identifying them as, the *required competencies* to all characteristics included in a job description and, the *given competencies* to all characteristics of an applicant skill sets contained in a CV. The two profiles are defined by means of *filters*. If $\leq$ denotes the partial order of the lattice in the TBox, then a filter on the TBox is an *upward-closed*, non-empty set of concepts. More precisely, we can assume that each profile in the knowledge base representing either a candidate CV or a job offer, is defined by a set of (given or required) skills, each modelled as subconcepts of a concept "skill". Thus, it is possible to concentrate on filters on the sub-lattice of sub-concepts of "skill".

An example of filters taken from Figure 2 could be for instance, "someone with experience programming in C#". In this example, the upward-closed set of concepts is defined as:

$$\{\mathsf{C++} \sqsubseteq \mathsf{Object\ Oriented} \sqsubseteq \mathsf{PL} \sqsubseteq \mathsf{Programming} \sqsubseteq \mathsf{Computing} \sqsubseteq \mathsf{Competences}\}$$

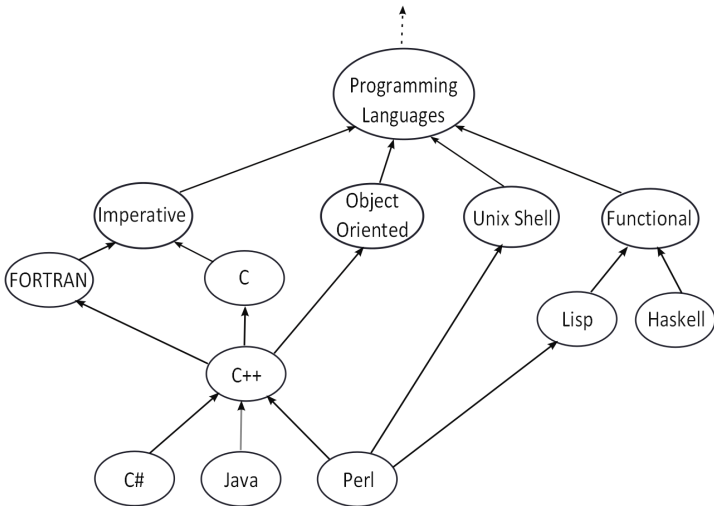

Fig. 2. Programming Languages sub-lattice

For a given job position (and applicant profile) it is expected to find many different filters that represent subsets of the applicant profiles and the job description. For instance:

$\mathcal{F}_1 =$ a required candidate that holds a Bachelor in Mathematics,

$\mathcal{F}_2 =$ a candidate with 5 years of experience teaching mathematics in primary schools,

$\mathcal{F}_3 =$ good level of English and German in both, speaking and writing,

$\vdots$

$\mathcal{F}_n =$ a resident of Berlin, Germany.

Note that, every job offer (and also applicants profiles) is comprised by a number of categories (Competences, Languages, Education, Skills, Social Skills, etc.). In turns, every category is expected to consist of at least one filter. For instance, for a given job advert it could be requested that candidates comply with $\mathcal{F}_j =$ knowledge of Java, $\mathcal{F}_l =$ knowledge of Linux, $\mathcal{F}_{db} =$ knowledge of database programming, etc. within the Competency category.

The filtered-based matching on partially ordered sets has been investigated in [10]. The basic idea is defined as follows:

Definition 4. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be filters in the given profile and in the required profile, respectively. The* matching value *$m(\mathcal{F}_1, \mathcal{F}_2)$ for $\mathcal{F}_1$ and $\mathcal{F}_2$ is computed as:*

$$m(\mathcal{F}_1, \mathcal{F}_2) = \frac{\#(\mathcal{F}_1 \cap \mathcal{F}_2)}{\#\mathcal{F}_2}$$

where $\#\mathcal{F}_2$ and $\#(\mathcal{F}_1 \cap \mathcal{F}_2)$ denote the cardinality of $\mathcal{F}_2$ and $\mathcal{F}_1 \cap \mathcal{F}_2$, respectively.

Note that the matching values are normalized in the range of $[0, 1]$ and satisfy the following Bayesian type rule:

$$m(\mathcal{F}_1, \mathcal{F}_2) \cdot \#\mathcal{F}_2 = m(\mathcal{F}_2, \mathcal{F}_1) \cdot \#\mathcal{F}_1.$$

An example taken from Figure 2 could be, a particular job description looking for applicants with experience in programming in C# and, a particular applicant profile claiming to have some experience programming in Java. The two filters are:

$\mathcal{F}_1$ = experience in Java $\quad\quad$ $\mathcal{F}_1 := \{(a_1, b_1) \in \Delta(\mathsf{hasSkill}) \wedge b_1 \in \Delta(\mathsf{Java})\}$

$\mathcal{F}_2$ = experience in C# $\quad\quad$ $\mathcal{F}_2 := \{(a_2, b_2) \in \Delta(\mathsf{hasSkill}) \wedge b_2 \in \Delta(\mathsf{C\#})\}$

The simplest algorithm would take the shortest distance between the two concepts from the least upper concept in the sub-lattice and calculate the distance between the two concepts (Java and C++) by counting cardinality of concepts.

$\mathcal{F}_1 = \mathsf{Java} \sqsubseteq \mathsf{C++} \sqsubseteq \mathsf{Object\ Oriented} \sqsubseteq \mathsf{PL} \sqsubseteq \mathsf{Programming} \sqsubseteq \mathsf{Computing} \sqsubseteq \mathsf{Competences}$

$\mathcal{F}_2 = \mathsf{C\#} \sqsubseteq \mathsf{C++} \sqsubseteq \mathsf{Object\ Oriented} \sqsubseteq \mathsf{PL} \sqsubseteq \mathsf{Programming} \sqsubseteq \mathsf{Computing} \sqsubseteq \mathsf{Competences}$

In this particular example, there is a measure of 7 for $\mathcal{F}_1$ and a measure of 7 for $\mathcal{F}_2$ as well, giving by the fact that the two elements (Java and C#) are siblings. By siblings we express the idea of a set of elements $\{i, j, x, y\}$ in a given lattice $\mathcal{L}$ where $i < x, y < j$ assuming $<$ is the partial ordering of the elements in $\mathcal{L}$. In this particular case, it is $\mathsf{C++} < \mathsf{Java}$ and $\mathsf{C++} < \mathsf{C\#}$. Although, it is the elements in common between the two filters that counts in here. Therefore, the matchability measurement of the two filters is $0,86$ calculated as:

$$m(\mathcal{F}_1, \mathcal{F}_2) = \frac{6}{7}$$

where, 6 is the number of common elements between $\mathcal{F}_1$ and $\mathcal{F}_2$, and 7 is the total number of elements in $\mathcal{F}_2$. In the context of the TBox in Figure 2 and, given the fact that matching on filters ranges between [0,1], we can say that having some experience in Java results in a relatively high score for the required experience in C#.

We introduce in the following sub-sections the main contribution of our research in this work. The main goal of this research is to provide an improvement on the matching process of job and applicants profiles within the HR domain. We will show how including weights can significantly improve the quality of the matching results based on filter-based matching on grounds of partially ordered sets. The introduction of a measure that improves matching on filters is detailed in Section 4.1. And aggregates on categories of profiles is introduced in Section 4.2 . We have also researched how to address over-qualification, as part of the matching process that, clearly cannot be captured solely by means of filters. This is introduced in Section 4.3. Finally, in Section 4.4 we introduce the novel concept of 'blow-up" operators. These operators allow us to extend the matching by integrating roles in the TBox. The idea is to expand the TBox by using roles to define arbitrarily many sub-concepts so that the original matching measures could again be applied.

4.1 Aggregates on Filters

A number of algorithms have been implemented to calculate the similarity of concepts between job descriptions and applicants profiles. It has already been shown in [10] that the idea of filter-based matching, as described in Section 4, significantly improves accuracy in comparison to simply taking differences of skill sets. A new matching measurement is introduced here, achieved by adding weights to the elements of the sub-lattice.

Definition 5. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be filters in the given profile and in the required profile, respectively. Let $\mathcal{F}_1 \cap \mathcal{F}_2$ denote the set of concepts that appear in both $\mathcal{F}_1$ and $\mathcal{F}_2$. Let w_i and w_j be the weight associated to every concept C_i in $\mathcal{F}_1 \cap \mathcal{F}_2$ and C_j in $\mathcal{F}_2$, respectively. Then the aggregate on filters $m_w(\mathcal{F}_1, \mathcal{F}_2)$ is defined:*

$$m_w(\mathcal{F}_1, \mathcal{F}_2) = \frac{\sum_{C_i \in \mathcal{F}_1 \cap \mathcal{F}_2} w_i}{\sum_{C_j \in \mathcal{F}_2} w_j}$$

where $i, j \in \mathbb{N}$.

By adding weights to the concepts of the sub-lattice structure, we are not only improving the matching but also, providing the possibility of adding a ranking of importance to every element in the underlying sub-lattice for a required aspect within a job profile. In this way, one could emphasize the search on the generic areas of required competencies. For instance, if searching for someone with experience in Object Oriented is more important than someone with experience in a specific Programming Language, then aggregates could be distributed as follows:

$$\mathsf{C{+}{+}}_{[0,10]}, \mathsf{Object\ Oriented}_{[0,60]}, \mathsf{PL}_{[0,10]}, \mathsf{Programming}_{[0,10]}, \mathsf{Computing}_{[0,05]}, \mathsf{Competences}_{[0,05]}$$

Or one could instead emphasize the search in more specific required competencies. For instance, if an experienced person in C++ is absolutely relevant to the position, we could write,

$$\mathsf{C{+}{+}}_{[0,60]}, \mathsf{Object\ Oriented}_{[0,10]}, \mathsf{PL}_{[0,10]}, \mathsf{Programming}_{[0,10]}, \mathsf{Computing}_{[0,05]}, \mathsf{Competences}_{[0,05]}$$

Either way, the distribution of aggregates through the elements of the underlying sub-lattice is normalized to be within the range [0,1].

4.2 Aggregates on Categories

Ranking the top candidates for a particular job position can be challenging in situations where, for example, a number of the top candidates result with the same matching measurement. The Fitness column in Table 1 shows an example where three different candidates have a final score of 0,5 as a result of the 4 analyzed categories (Competences, Languages, Education, Skills). To overcome these situations, adding aggregates on the categories of the evaluated job profile has been considered. This is a normalized adjustable measure defined within $[0, 1]$, intends to provide a more granular set of results.

Consider the aggregates on filters $m_w(\mathcal{F}_1, \mathcal{F}_2)$ as in Definition 5 and, consider as well the set of filters a given category is composed by, i.e., $Category = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_x\}$. The score of a given category within a given profile, is calculated by $\bar{m}_w = \sum_{l=1}^{x} \frac{m_{w_l}(\mathcal{F}_1, \mathcal{F}_2)}{x}$. Then, the aggregates on categories is defined as follows.

Definition 6. *Let $\bar{m}_{wj}$ be the score per category j and, w_j be the weights associated to every category within a profile. We define the aggregates on categories m_c as follows:*

$$m_c = \sum_{j=1}^{m} \bar{m}_{wj} * w_j$$

where $m \in \mathbb{N}$ denotes the total number of categories.

Table 1. matching measures with same fitness

	Competences	Languages	Education	Skills	Fitness
Candidate 1	1	0	1	0	0,5
Candidate 2	0,5	0,5	0,5	0,5	0,5
Candidate 3	0,8	0,2	0,2	0,8	0,5

As an example, we apply different weights to the categories in Table 1, for instance: 0,5 to Competences, 0,2 to Languages, 0,1 to Education and 0,2 to Skills. This results in a completely different set of scores, that is clearly less problematic to evaluate during the ranking of the candidates, as shown in Table 2.

Table 2. ranking candidates with aggregates on categories

	Competences (50%)	Languages (20%)	Education (10%)	Skills (20%)	Fitness
Candidate 1	1	0	1	0	0,6
Candidate 2	0,5	0,5	0,5	0,5	0,5
Candidate 3	0,8	0,2	0,2	0,8	0,62

This approach provides the experts in the HR domain, with the option of ranking the different categories in order of importance. One may guess that Skills and Competences could almost always be the most important categories although, this is subject to every particular case where advise from the experts in the area is required.

4.3 Over-qualification in Profile Matching

An over-qualified applicant for a given job offer is defined as a person that is skilled or educated beyond what it is required to conform to a given advertised position. It appears natural to approach profile matching by the dual notion of [fitness, over-qualification] as part of a person skill set such that, fitness is calculated by means of aggregates on filters, as in Definition 4.1 and over-qualification is calculated by means of cardinality of filters as defined in Definition 4. Therefore, we would look for the maximality of the fitness scores and the minimality of the over-qualification scores of candidates during the ranking process.

It seems obvious to note that over-qualification is only evaluated in cases where fitness > 0, this is, if $m(\mathcal{F}_1, \mathcal{F}_2) > 0$, as there is no point on doing so otherwise.

Definition 7. *Given two filters $\mathcal{F}_1$ representing a candidate profile and $\mathcal{F}_2$, representing a job profile, over-qualification is calculated as follows:*

$$O(\mathcal{F}_2, \mathcal{F}_1) = \frac{\#(\mathcal{F}_2 \cap \mathcal{F}_1)}{\#\mathcal{F}_1}$$

where $O(\mathcal{F}_2, \mathcal{F}_1) \in [0, 1]$.

Note that, over-qualification is calculated by counting the number of elements on filters as in Definition 4 although, filters are taken in the opposite direction than on matching. Such that, it measures the characteristics of the job profile against the characteristics of the applicants profile. This is, skills, academic records, competencies present in the applicants profile that are not required in the job description.

Consider the following example where:

$$\mathcal{F}_1 := \{(a_1, b_1) \in \Delta(\mathsf{hasSkill}) \wedge b_1 \in \Delta(\mathsf{Java}) \wedge (a_1, c_1) \in \Delta(\mathsf{hasSkill}) \wedge c_1 \in \Delta(\mathsf{Haskell})\}$$
$$\mathcal{F}_2 := \{(a_2, b_2) \in \Delta(\mathsf{hasSkill}) \wedge b_2 \in \Delta(\mathsf{C\#})\}$$

We have calculated the matching measure between Java and C# in Section 4 already where, $m(\mathcal{F}_1, \mathcal{F}_2) = 0,86$. Given that Haskell is clearly not part of the job profile filter, we calculate over-qualification of Haskell in terms of C#. This results in a measure of $0,86$ as well, given that $O(\mathcal{F}_2, \mathcal{F}_1) = \frac{6}{7}$. Then, the measurement [fitness, over-qualification] is [0,86; 0,86]. We could say that this individual is equally (and highly) competent and over-qualified for the position as a programmer in C#.

4.4 Blow-up Operators

In order to take full benefit of the knowledge based representation of profiles, we define TBox transformations by means of operators that remove roles and instead, enlarge the knowledge base by many new sub-concepts. We call such operators *blow-up operators*, because the removal of a single rule may lead to many new concepts, even infinitely many. However, for the sake of computability, the filters should remain finite. The idea in here is to extend the approach of matching by integrating new roles in the TBox. This is basically to define operators that expand the TBox by using roles in order to define arbitrarily many new sub-concepts so that the original measures of matching, as defined in Section 4, could again be applied. We define a blow-up operator as follows:

Definition 8. *Let C and $C^{'}$ be two concept names and let R be a role name that relates elements of C with elements of $C^{'}$. Let I be an interpretation such that for a given two nominal names a and b, $b \in \Delta(C^{'})$ and $\{a \in \Delta(C) | (a,b) \in \Delta(R)\}$. Then, Blow-up operator is defined as:*

$$\textit{blow-up}(C, \{R, C^{'}\}) \equiv C \sqcap \exists R.C^{'}$$

where an instance b of $C^{'}$ defines sub-concepts of C such that, the R-values are restricted to b, then $R(a,b) \in \Delta(R)$.

By extending this definition, it is easier to note the new created concepts, denoted by $\bar{C}_i$:

$$\begin{aligned} \bar{C}_1 &:= \text{blow-up}(C, \{R, C_1^{'}\}) \\ \bar{C}_2 &:= \text{blow-up}(C, \{R, C_2^{'}\}) \\ &\vdots \\ \bar{C}_n &:= \text{blow-up}(C, \{R, C_n^{'}\}) \end{aligned}$$

Note that, an order on the instances of $C^{'}$ defines a subsumption on the elements $\bar{C}_i$. This is, for every $i, j \leq n$ it is $\bar{C}_i \sqsubseteq \bar{C}_j$ if and only if $i \leq j$. The generic definition is:

$$\text{Blow-up}_n(C, \{(R, C_1), \ldots, (R, C_n)\}) \equiv \bigsqcap_{1 \leq i \leq n} \text{blow-up}(C, \{R, C_i\})$$

Consider an example where people have certain years of experience working with programming languages in general. This can be thought as a relation between two concepts: Years of Experience and Programming Languages, where the two concepts are related through the role hasExperience, as shown in Figure 3.

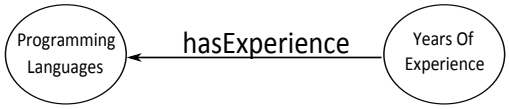

Fig. 3. Role hasExperience between Programming Languages and Years of Experience

By applying blow-up operators, the role hasExperience is removed and instead, creates many sub-concepts of Programming Languages with every instance of Years of Experience. It is important to note that, this applies only to every instance of Programming Languages where the value in hasExperience is restricted to Years of Experience. Thereby, after applying blow-up operators, the role in Figure 3 is expanded, as shown in Figure 4, where C++ is an instance of Programming

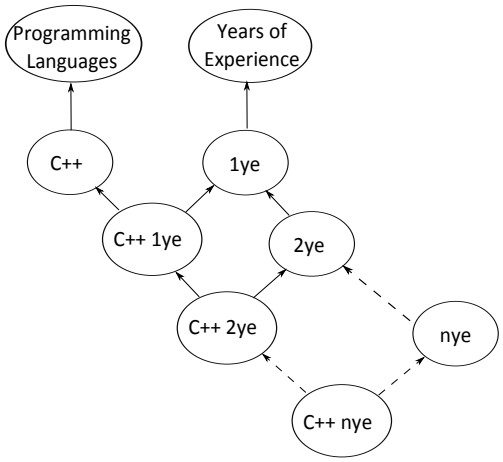

Fig. 4. Role hasExperience blowed-up with C++ and Years of Experience

Languages, and $\{1ye, 2ye, \dots\}$ are instances of Years of Experience. Where 1ye expresses "1 year of experience", 2ye = "2 years of experience", etc.

Note that, an order on the instances of Years of Experience gives rise to subsumption between the so-defined sub-concepts of Programming Languages, i.e., "C++ with n year of experience" $\sqsubseteq$ "C++ with n-1 years of experience" $\sqsubseteq \cdots \sqsubseteq$ "C++ with 2 years of experience" $\sqsubseteq$ "C++ with 1 year of experience". In consequence, integrating matching measurements as explained in Sections 4 with blow-up operators seems plausible. Consider an example taken from Figure 2 where, $\mathcal{F}_1$ = a given competency in C++ with 2ye and $\mathcal{F}_2$ = a required competency in C++ with 3ye expressed as follows:

$$\mathcal{F}_1 := \{(a_1, b_1) \in \Delta(\mathsf{hasSkill}) \wedge b_1 \in \Delta(\mathsf{C{+}{+}}) \wedge (b_1, c_1) \in \Delta(\mathsf{hasExperience}) \wedge c_1 \in \Delta(\mathsf{2ye})\}$$
$$\mathcal{F}_2 := \{(a_2, b_2) \in \Delta(\mathsf{hasSkill}) \wedge b_2 \in \Delta(\mathsf{C{+}{+}}) \wedge (b_2, c_2) \in \Delta(\mathsf{hasExperience}) \wedge c_2 \in \Delta(\mathsf{3ye})\}$$

The basic matching measurement is $m(\mathcal{F}_1, \mathcal{F}_2) = 1$ given that $m(\mathcal{F}_1, \mathcal{F}_2) = \frac{6}{6}$. While, the basic matching measurement applied to the blow-up of years of experience results as follows:

$$\frac{\#(\mathsf{blow\text{-}up}(\mathsf{C{+}{+}}, \{\mathsf{hasExperience}, \mathsf{2ye}\}) \cap \mathsf{blow\text{-}up}(\mathsf{C{+}{+}}, \{\mathsf{hasExperience}, \mathsf{3ye}\}))}{\#\mathsf{blow\text{-}up}(\mathsf{C{+}{+}}, \{\mathsf{hasExperience}, \mathsf{3ye}\})} = \frac{4}{5}$$

It seems clear how the use of blow-up operators to the role hasExperience in Figure 3 influences the result of the original matching measure of 1. We believe blow-up operators is a relevant improvement on how matching of profiles is performed nowadays in relation to this complex roles such as "years of experience" on skills and competencies.

Note that a blow-up is performed up-to the level of the required profile. Therefore blow-up operator is defined within the range $[0, 1]$.

Blow-up operators allow us to expand the knowledge base, and more precisely the TBox, in such a way, that allows the filter-based matching measures to remain unchanged. Blow-up operators are not intended to be part of the TBox itself. Nevertheless, they extend concepts within the TBox in a lattice-like structure by blowing-up the role R. Blow-up operators are intended to be applied to a wider range of circumstances, i.e., years of experience in competences, proficiency level of skills, versioning, etc. Therefore, blowing-up all these possibilities and possibly combining them all together within a given TBox would be rather slow. Then, calculating blow-up operators out of the TBox is a better and more efficient approach.

4.5 Conclusion and Further Work

In this approach we have shown a number of improvements to the matching process of profiles between job offers and applicants profiles within the HR domain. Starting from a previous work (Popov-Jebelean, 2013), we have introduced an improvement to the matching of filters by adding aggregates on filters and also on the categories of profiles. We have also addressed the problem of over-qualification with the dual measure of [fitness, over-qualification] and introduced the novel idea of blow-up operators by integrating roles in the TBox.

Further research to this approach will be focus on the investigation of methods for the learning of matching relations from large ABoxes. With respect to the enrichment of knowledge bases several approaches for ontology learning have been investigated [3], [11]. Also attempts to exploit Formal Concept Analysis have been undertaken [7].

References

1. F. Baader, H.-J. Bürckert, J. Heinsohn, B. Hollunder, J. Müller, B. Nebel, W. Nutt, and H.-J. Profitlich. Terminological knowledge representation: A proposal for a terminological logic. In Description Logics, pages 120-128, (1991).
2. Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P., The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press 2003.
3. P. Cimiano, A. Hotho, G. Stumme, and J. Tane. Conceptual knowledge processing with formal concept analysis and ontologies. In ICFCA, pages 189-207, 2004.
4. B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler. Owl 2: The next step for owl. J. Web Sem., 6(4):309-322, 2008.
5. T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? Int. J. Hum.-Comput. Stud., 43(5-6):907-928, 1995.
6. M. C. A. Klein, J. Broekstra, D. Fensel, F. van Harmelen, and I. Horrocks. Ontologies and schema languages on the web. In Spinning the Semantic Web, pages 95-139, 2003.
7. D. Looser, H. Ma, and K.-D. Schewe. Using formal concept analysis for ontology maintenance in human resource recruitment. In Proc. 9th Asia-Pasific Conference on Conceptual Modelling (APCCM), pages 61-68, 2013.
8. M. Mochol, L. J. B. Nixon, and H.Wache. Improving the recruitment process through ontology-based querying. In SEBIZ, (2006).
9. M. Mochol, L. J. B. Nixon, and H.Wache. Improving the Accuracy of Job Search with Semantic Techniques. 10th International Conference on Business Information Systems (BIS2007), Poznan, Poland 25-27 April 2007.
10. N. Popov and T. Jebelean. Semantic matching for job search engines: A logical approach. Technical report no. 13-02 in RISC Report Series, University of Linz, Austria, 2013.
11. E. Zavitsanos, G. Paliouras, and G. A. Vouros. Gold standard evaluation of ontology learning methods through ontology transformation and alignment. IEEE Trans. Knowl. Data Eng., 23(11):1635-1648, 2011
12. European Dictionary of Skills and Competences, http://www.disco-tools.eu
13. International Standard Classification of Occupations, http://www.ilo.org/public/english/bureau/stat/isco/isco08/index.htm
14. International Standard Classification of Education, http://www.uis.unesco.org/Education/Pages/international-standard-classification-of-education.aspx