Top-k matching queries for filter-based profile matching in knowledge bases*

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

Software Competence Center Hagenberg, Hagenberg, Austria
{Lorena.Paoletti, Jorge.Martinez-Gil, kd.schewe}@scch.at

Abstract. Finding the best matching job offers for a candidate profile or, the best candidates profiles for a particular job offer, respectively constitutes the most common and most relevant type of queries in the Human Resources sector. This technically requires to investigate top-k queries on top of knowledge bases and relational databases. We propose in this paper a top-k query algorithm on relational databases able to produce effective and efficient results. The approach is to consider the partial order of matching relations between jobs and candidates profiles together with an efficient design of the data involved. In particular, the focus on a single relation, the matching relation, is crucial to achieve the expectations.

1 Introduction

The accurate matching of job applicants to position descriptions and vice versa is of central importance in the Human Resources (HR) domain. The development of data or knowledge bases (KB) and databases 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 is of high importance. Finding the best matching job offers for a candidate profile or, the best candidate profiles to a particular job offer respectively, constitute the most common and most relevant type of query, which technically requires to investigate top-k queries on top of knowledge bases and relational databases.

A profile describes a set of skills either, a person posses detailed in form of a CV or, described in a job advertisement through the job description. Profile matching concerns to measure how well a *given* profile matches a *requested* profile. Although, profile matching is not only concerned to the Human Resources

* The research reported in this paper has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

The research reported in this paper was supported by the Austrian Forschungsförderungsgesellschaft (FFG) for the Bridge project "Accurate and Efficient Profile Matching in Knowledge Bases" (ACEPROM) under contract **841284**.

sector but a wide range of other application areas, real state domain, matching system configurations to requirements specifications, etc. The research in this paper is in line with a previous work [5] where an approach on improving profile matching in the HR sector is introduced. For this, the starting point is based on exact matching [6] that has been further investigated in [7].

With respect to querying knowledge bases in the HR domain, the commonly investigated approach is to find the best k (with $k \geq 1$) matches for a given profile, either a CV or a job offer [2]. This constitutes what is commonly known as top-k queries. Top-k queries have been thoroughly investigated in the field of databases, usually in the context of the relational data model [3, 4, 9]. The study of such queries in the context of knowledge bases has also been researched [8].

The most relevant queries in the human resources sector, are matching queries driven either by a CV (or a set of CVs) or by a job offer (or a set of job offers). These queries can characterized as top-k queries, skyline queries in case of partial orders on the matching measures or a combination of these. Top-k queries in relational databases are in general addressed by associating weights or aggregates acting as a ranking to the part of data relevant to the user's needs, a potential join of the relevant relations involved and, a ranking (or sorting) of the tuples that constitutes the expected result set. Computing all these steps at once can be a process able to consume many resources, depending on the design and nature of the data.

Our contribution in relation to top-k queries in relational databases and knowledge bases takes benefits of the partial order on matching measures and knowledge bases equipped with matching relations. The expectation is of course, that many of the results in the relational data model can be easily adopted to this case. In particular, the focus on a single relation, i.e. the matching, as the driver for the querying, is expected to ease the extension. This requires to investigate the supporting data structures. In view of the many results on efficient top-k queries in the context of the relational data model it is expected that these results can be largely achieved by adaptation to the case of knowledge bases, in which data structures for the support of hierarchies can be adopted from databases. The objective is to minimize the selection of tuples as well as eliminating the calculation of weighting (scoring) of tuples on the query itself, by making use of weighting on the partial order of concepts of knowledge bases by means of matching measures.

The paper is organized as follows: In Section 2 we cover the main aspects of our theory on profile matching introduced in a previous work [5]. The internal physical representation of profile matching is introduced in Section 3. In Section 3.1 we introduce our approach of a relational database schema to implement top-k queries and in Section 3.2 we show an algorithm implementing our approach of top-k queries.

2 Preliminaries

We have presented in a previous work [5] our representation of profile matching in the HR domain. It was shown in that work how we represent CVs and jobs profiles in a KB as well as the syntax and the semantic of the language used to represent the terminology of the KB. We also elaborated a matching theory to calculate the matching measures between two given profiles (CV and job offer) and the so called blow-up operators of a KB. We briefly refresh some of those concepts involved in the elaboration of queries.

Concepts C_i in a TBox of a KB define a *lattice* $(\mathcal{L}, \leq)$ with $\sqcap$ and $\sqcup$ as operators for the *meet* and the *join* respectively, and $\sqsubseteq$ for the partial order of elements of a KB, and the closure under $\sqcap$ and $\sqcup$ for concepts $C_i, \top, \bot$. In the following, we refer to concepts C_i in $\mathcal{L}$ to denote concepts C_i in a given KB. Thus, the terms TBox and lattice are used as synonyms from now on.

Concerning the formalism for representing the knowledge, a subset of the description logic $\mathcal{SROIQ}$ is used in [5]. As for the semantics, concepts are given a set-theoretic interpretation where a concept is interpreted as a set of individuals and roles are interpreted as sets of pairs of individuals. The interpretation domain is arbitrary and can be infinite. Then, there is an interpretation $\mathcal{I}$ consisting of a non-empty set $\Delta^{\mathcal{I}}$ called the interpretation domain and, an interpretation function that associates specific concept names in a TBox to individuals of the universe. Then, it associates every atomic concept C_i to a set $\Delta(C_i) \subseteq \Delta^{\mathcal{I}}$ and, to every role R a binary relation $\Delta(R) \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

A *filter* in a lattice $(\mathcal{L}, \leq)$ is a non-empty subset $\mathcal{F} \subseteq \mathcal{L}$ such that for all C, C' with $C \leq C'$ whenever $C \in \mathcal{F}$ holds, then also $C' \in \mathcal{F}$ holds.

If $P \subseteq \mathcal{I}$ is a *profile*, P defines in a natural way a filter $\mathcal{F}$ of the lattice $\mathcal{L}$ of concepts:

$$\mathcal{F} = \{C \in \mathcal{L} | \exists p \in P \cdot p \in \Delta(C)\}$$

Therefore, for determining matching relations we can concentrate on filters $\mathcal{F}$ in a lattice.

2.1 Filter-Based Matching

Let $(\mathcal{L}, \leq)$ be a lattice, and let $\mathbb{F} \subseteq \mathcal{P}(\mathcal{L})$ denote the set of filters in this lattice. A *relative weight measure* on $\mathcal{L}$ is a function $m : \mathcal{P}(\mathcal{L}) \to [0, 1]$ satisfying

(a) $m(\mathcal{L}) = 1$,
(b) $m(\mathcal{L} - A) = 1 - m(A)$ for any $A \in \mathcal{P}(\mathcal{L})$,
(c) $m(\bigcup_{i \geq 1} A_i) = \sum_{i \geq 1} m(A_i)$ for pairwise disjoint $A_i \in \mathcal{P}(\mathcal{L})$.

A *matching measure* is a function $\mu : \mathbb{F} \times \mathbb{F} \to [0, 1]$ such that $\mu(\mathcal{F}_1, \mathcal{F}_2) = m(\mathcal{F}_1 \cap \mathcal{F}_2)/m(\mathcal{F}_2)$ holds for some relative weight measure m on $\mathcal{L}$ and any $\mathcal{F}_1, \mathcal{F}_2 \in \mathbb{F}$.
The matching measure μ defined in [6] uses cardinalities

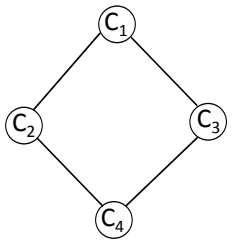
$$\mu(\mathcal{F}_1, \mathcal{F}_2) = \#(\mathcal{F}_1 \cap \mathcal{F}_2)/\#\mathcal{F}_2 \tag{1}$$

Thus, it is defined by the relative weight measure m on $\mathcal{L}$ with $m(A) = \#A/\#\mathcal{L}$.

Let w be a *weight* associated to every concept $C \in \mathcal{L}$ then, a matching measure μ is defined by weights $w(C) = m(\{C\}) \in [0, 1]$ such that

$$\mu(\mathcal{F}_1, \mathcal{F}_2) = \sum_{C \in \mathcal{F}_1 \cap \mathcal{F}_2} w(C) \cdot \left(\sum_{C \in \mathcal{F}_2} w(C) \right)^{-1} \tag{2}$$

Example 1. A simple lattice with four elements: $\mathcal{L} = \{C_1, C_2, C_3, C_4\}$ defines up to five filters $\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4, \mathcal{F}_5\}$, as shown in (a) and (b) Fig. 1, respectively.



(a) Lattice

$\mathcal{F}_1 = \{C_1\}$
$\mathcal{F}_2 = \{C_2, C_1\}$
$\mathcal{F}_3 = \{C_3, C_1\}$
$\mathcal{F}_4 = \{C_3, C_2, C_1\}$
$\mathcal{F}_5 = \{C_4, C_3, C_2, C_1\}$

(b) Filters

Fig. 1: A lattice, its filters and matching measures

If we give some weights to the elements of $\mathcal{L}$, for instance $w(C_1) = \frac{1}{10}, w(C_2) = \frac{2}{5}, w(C_3) = \frac{3}{10}$, and $w(C_4) = \frac{1}{2}$ and calculate the matching measure $\mu(\mathcal{F}_i, \mathcal{F}_j)$ and $\mu(\mathcal{F}_j, \mathcal{F}_i)$ (for $1 \leq i, j \leq 5$) with the formula in (2), we obtain the result shown in Fig. 2.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{F}_1$	1	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{9}$
$\mathcal{F}_2$	1	1	1	$\frac{5}{8}$	$\frac{5}{9}$
$\mathcal{F}_3$	1	$\frac{4}{5}$	1	$\frac{1}{2}$	$\frac{4}{9}$
$\mathcal{F}_4$	1	1	1	1	$\frac{8}{9}$
$\mathcal{F}_5$	1	1	1	1	1

Fig. 2: Matching Measures

It is easy to note at a glance on Fig. 2 that in general, the matching measures are not symmetric. If $\mu(\mathcal{F}_g, \mathcal{F}_r)$ expresses how well a given filter $\mathcal{F}_g$ matches a required filter $\mathcal{F}_r$, then $\mu(\mathcal{F}_r, \mathcal{F}_g)$ measures the excess of skills in the given filter

$\mathcal{F}_g$ that are not required in $\mathcal{F}_r$. And clearly, $\mu(\mathcal{F}_i, \mathcal{F}_j) = \mu(\mathcal{F}_j, \mathcal{F}_i) = 1$, when $i = j$ for $1 \leq i, j \leq 5$.

Example 2. Take for instance, $\mathcal{F}_r = \mathcal{F}_3$ as a required profile and two given filters $\mathcal{F}_{g_1} = \mathcal{F}_3$ and $\mathcal{F}_{g_2} = \mathcal{F}_4$. They are both equally and highly qualified for the requirements in $\mathcal{F}_r$ given their matching measures:

$$\mu(\mathcal{F}_{g_1}, \mathcal{F}_r) = 1 \quad \text{and} \quad \mu(\mathcal{F}_{g_2}, \mathcal{F}_r) = 1$$

although, if we consider the measures $\mu(\mathcal{F}_r, \mathcal{F}_g)$:

$$\mu(\mathcal{F}_r, \mathcal{F}_{g_1}) = 1 \quad \text{and} \quad \mu(\mathcal{F}_r, \mathcal{F}_{g_2}) = \frac{1}{2}$$

$\mathcal{F}_3$ matches better than $\mathcal{F}_4$ as C_2 is not part of the required skill set.

3 Internal Structure of Profile Matching

In a modeled selection process where there is a set of profiles $\mathbb{P}$, i.e., job and applicants profiles, defined by filters in a lattice $\mathcal{L}$, we denote by φ the conditions to be met by profiles (either job or applicants profiles) in order to be selected, then $\mathbb{P}_\varphi$ denotes the set of profiles in $\mathbb{P}$ satisfying φ and $P_r \in \mathbb{P}$ is a *required* profile driving the selection by holding the conditions φ.

Note that, when referring to matching measures from now on, we refer on matching measures as in to formula (2) that includes weighting on the elements of the lattice $\mathcal{L}$, as shown in Example 1.

Definition 1. *For all $P \in \mathbb{P}_\varphi$ and $P' \in (\mathbb{P} - \mathbb{P}_\varphi)$, P is selected and P' is not selected if $\mu(P, P_r) > \mu(P', P_r)$ and no subset of $\mathbb{P}_\varphi$ satisfy this property.*

In order to obtain the best-k matching profiles (either job or applicant profiles) we first need to query for filters representing those profiles.

Consider $\mathcal{F}_r$ being a filter representing the required profile P_r, a requested job profile for instance. Then, consider l being a number of filters in $\mathbb{F}$ $(\mathcal{F}_{g_1}, \ldots, \mathcal{F}_{g_l})$ representing candidates profiles matching P_r in a certain degree, satisfying φ such that, their matching measures are above a threshold $t_i \in [0, 1]$ this is, $\mu(\mathcal{F}_{g_x}, \mathcal{F}_r) \geq t_i$ for $x = 1, \ldots, l$.

Then, every $\mathcal{F}_{g_x}$ represents a finite number j of profiles $(P_{g_1}, \ldots, P_{g_j})$, candidates profiles matching P_r, where $\mu(P_{g_y}, P_r) \geq t_i$ for $y = 1, \ldots, j$ and $j \leq k$.

Note that, the relation between filters in $\mathcal{L}$ and the number of *related* profiles represented by filters is defined by a function $\nu : \mathbb{N} \to \mathbb{N}$ where $\nu(x) = j$ and $\sum_{x=1}^{l} \nu(x) = k$. Then, any $\mathcal{F}_{g_{l+1}}$ *is not selected* as the matching value $\mu(\mathcal{F}_{g_{l+1}}, \mathcal{F}_r) < t_i$.

As for the second part of the definition, each filter $\mathcal{F} \in \mathbb{F}$ is uniquely determined by its minimal elements such that, we can write $\mathcal{F} = \{C_1, \ldots C_r\}$. Then, every profile represented by a filter is also uniquely determined by the elements in $\mathcal{F}$. Therefore, for any profile P'' in a subset of $\mathbb{P}_\varphi$ the matching value is $\mu(P'', P) < t_i$ then P'' does not satisfy the property.

Example 3. Assume to have a job offer profile P_a and four candidates profiles $\{P_b, P_c, P_d, P_e\}$ that meet the requirements in P_a. Let's also assume that the five profiles are represented by the filters in Example 1 such that:

$\mathcal{F}_4$ represents $\{P_a, P_b\}$
$\mathcal{F}_2$ represents $\{P_c\}$
$\mathcal{F}_3$ represents $\{P_d, P_e\}$

then, $\mathcal{F}_r = \mathcal{F}_4$ and $\mathcal{F}_g = \{\mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$. If $t_i = 0.5$, $l = 3$ and $k = 4$.

In order to obtain the best l filters satisfying φ, we first need to know the minimum matching value representing l filters. Thus, we start by selecting any t_i. If less than l solutions are found, we increase t_i (t_{i+1}). If more than l solutions are found, we decrease t_i (t_{i-1}). The search stops when the l filters satisfying $\mu(\mathcal{F}_{g_x}, \mathcal{F}_r) \geq t_i$ for $x = 1, \ldots, l$ are found.

With the optimum t_i, we query for the related k profiles where $\mu(P_g, P_r) \geq t_i$. This assumes to be given the matching measures between all filters in $\mathcal{L}$ and ultimately, between all profiles represented by filters.

As exposed in Example 1, matching measures between filters define a matrix (Fig 2) so do matching measures between profiles although, the number of filters is assumed to be smaller than the number of profiles ($l \leq k$). If we take for instance profiles P_a, P_c, P_d from example 3 we have the *minimum* number of profiles producing matching measures to define a matrix of profiles.

Definition 2. *A Matrix $\mathcal{M}$ is a matrix-like structure of matching measures between the minimum number of profiles that produce all possible measures.*

Fig. 3 shows a matrix $\mathcal{M}$ where columns represent the required profiles P_r and rows represent the given profiles P_g.

	P_1	P_2	$\ldots$	P_n
P_1	$\mu(P_1, P_1)$	$\mu(P_1, P_2)$	$\ldots$	$\mu(P_1, P_n)$
P_2	$\mu(P_2, P_1)$	$\mu(P_2, P_2)$	$\ldots$	$\mu(P_2, P_n)$
P_3	$\mu(P_3, P_1)$	$\mu(P_3, P_2)$	$\ldots$	$\mu(P_3, P_n)$
$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$
P_n	$\mu(P_n, P_1)$	$\mu(P_n, P_2)$	$\ldots$	$\mu(P_n, P_n)$

Fig. 3: Matrix $\mathcal{M}$ of profiles

Obtaining the k solutions in $\mathcal{M}$ involves refer either to one column or to one row. The the process is analogous if we focus either on rows or columns although, the perspective is different. While reading the measures from the columns perspective provides the so called *fitness* between profiles $\mu(P_g, P_r)$, the measures

read from the rows perspective are the inverted measure $\mu(P_r, P_g)$ denoted as *overqualification*. Overqualification may be considered as emphasized in Example 2 where profiles that equally (or almost equally) match the requirements, maybe subject to a second ranking with respect to the inverted measure.

If we focus on columns, when querying for a particular P_r representing the required skill set, there would be finitely many profiles P_g matching P_r. Although, we only focus on the k elements where $\mu(P_g, P_r) \geq t_i$. Ideally, all elements in the column are in total order according to the $\leq$ relation of $\mu(P_g, P_r)$. The advantage in here is that when searching for any given k and t_i we only need to point to the right element in the column and search for the next consecutive $k-1$ elements in descending order of matching measures.

If searching for the less overqualified, there would be finitely many P_r in every row of the *Matching Matrix* when querying for a particular P_g but we only need to point to the right t_i and search for the next $k-1$ elements in ascending order of $\mu(P_r, P_g)$.

Example 4. Profiles P_a, P_c, P_d are the minimum number of profiles from Example 3 regarding fitness

	P_c	P_d	P_a
P_c	-	-	$\frac{5}{8}$
P_d	-	-	$\frac{1}{2}$
P_a	-	-	1

Fig. 4: Matching Measures

We explain next how we organize profiles in order to provide an efficient search of these elements when querying for the best k-profile matching. We first assume an identification label for every row and column in *Matching Matrix* $\mathcal{M}$, where ρ_i represents a number i of rows and σ_i represents the number i of columns, for $i > 0$.

Definition 3. *A profile record of a required profile P_r in column σ_i in $\mathcal{M}$, is a finite number of elements*

$$(\mu_i, n_i^{>}, n_i^{=}, n_i^{<}, next, prev, p)$$

where μ_i denotes the matching measures $\mu(P_g, P_r)$ for every matching profile P_g and

$n_i^{>}$ *denotes the number of profiles P_g in σ_i where $\mu(P_g, P_r) > \mu_i$,*
$n_i^{=}$ *denotes the number of profiles P_g in σ_i where $\mu(P_g, P_r) = \mu_i$,*
$n_i^{<}$ *denotes the number of profiles P_g in σ_i where $\mu(P_g, P_r) < \mu_i$,*
next is a reference to the next matching value in σ_i where $\mu(P_{g+1}, P_r) \geq \mu_i$,

prev is a reference to the next matching value in σ_i where $\mu(P_{g-1}, P_r) \leq \mu_i$ and,
p is a reference to a linked-list of profiles matching P_r.

The numbers $n_i^{>}, n_i^{=}, n_i^{<}$ are significantly important when determining the number of profiles (either applicants or job profiles) represented by a filter without actually querying for them. It is easy to determine whether $(n_i^{>} + n_i^{=}) \geq k$ when querying for the pair (P_g, P_r). Then, search for the next pair (P_{g+1}, P_r) if that is not the case.

References *Next* and *Prev* make possible to track the following greater or smaller matching value of profiles by following the references. Every μ_i contains additionally a reference p to the related profiles (jobs or applicants) in σ_i column. All the related profiles are organized in a linked-list like structure of profiles, ordered by the smaller-than-or-equal elements of matching values.

Example 5. Consider the profiles $\{P_a, P_b, P_c, P_d, P_e\}$ as in Example 3 with matching measures: $\mu(P_b, P_a) = 1, \mu(P_c, P_a) = 0.63, \mu(P_d, P_a) = \mu(P_e, P_a) = 0.5$. Consider also the graphic in Fig. 5, representing the profile records of column σ_i in $\mathcal{M}$ corresponding to P_a

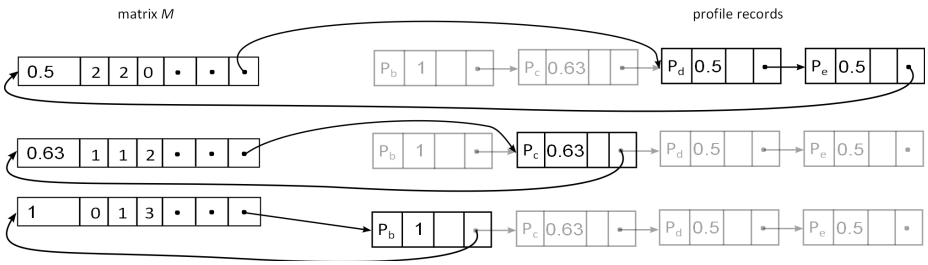


Fig. 5: Linked list of matching measures

Organizing data in a structure like $\mathcal{M}$ with profile records implies a fast and efficient search of the k matching profiles where the main objective is to fetch the corresponding columns σ_i and together with $n_i^{>}$,$n_i^{=}$ and $n_i^{<}$ calculate how many of profile records we need in order to get k profiles, then follow the linked-list of profiles until the k elements are found. If we need $k = 3$ with $t_i \geq 0.5$ we could get to profile record 1 in Example 5 where there are in total 4 $(2 + 2 + 0)$ matching profiles where:

2 profiles have matching measures ≥ 0.5,
2 profiles match P_a with $= 0.5$ and,
none matches P_a with measures ≤ 0.5.

then, we know we have to visit 2 profile records to get the total number of profiles. Of course, one can claim that starting on row 3 of Example 3 is the best

approach. Therefore, an ordering on the elements of columns in $\mathcal{M}$ seems to be essential.

The definition of matching records on rows of *Matching Matrix* $\mathcal{M}$ is analogous to Definition 3

Definition 4. *A matching record of a given profile P_g in row ρ_i in $\mathcal{M}$, is a finite number of elements*

$$(\mu_i, n_i^{>}, n_i^{=}, n_i^{<}, next, prev, p)$$

where μ_i denotes the matching value $\mu(P_r, P_g)$ for every matching profile P_r and

$n_i^{>}$ *denotes the number of profiles P_r in ρ_i where $\mu(P_r, P_g) > \mu_i$,*
$n_i^{=}$ *denotes the number of profiles P_r in ρ_i where $\mu(P_r, P_g) = \mu_i$,*
$n_i^{<}$ *denotes the number of profiles P_r in ρ_i where $\mu(P_r, P_g) < \mu_i$,*
next is a reference to a matching values in ρ_i where $\mu(P_{r_+1}, P_g) \geq \mu_i$,
prev is a reference to a matching values in ρ_i where $\mu(P_{r_-1}, P_g) \leq \mu_i$ and,
p is a reference to a linked-list of profiles matching P_g.

The consideration regarding ordering on elements of rows in $\mathcal{M}$ in Definition 4 is also to be consider in order to achieve an efficient retrieval of matching overqualified profiles for a given P_g.

The following section shows an implementation of the Matrix $\mathcal{M}$ and profile records in a relational database schema. We define the database structure supporting our definition of top-k queries in Section 3.1 while in Section 3.2 we show an algorithm that implements our definition of top-k queries in profile matching in relational databases.

3.1 Implementation of Top-K Profile Matching

Our implementation approach of top-k queries as described in Section 3 is designed on a relational database schema. The schema **HR** shown in Fig. 6 is designed to store and maintain filters of a given lattice $\mathcal{L}$, as well as profiles and matching measures of an instance of $\mathcal{L}$ for the implementation of top-k queries. Note that for the definition of the database schema we use the notation of the *unnamed and logic programing perspective* as defined in [1]. Where, under the unnamed perspective, a *tuple* $\langle a_1, \ldots, a_n \rangle$ is an ordered n-tuple ($n > 0$) of constants of a Cartesian product $\mathbf{dom}^n$ (where **dom** is the underlying set of constants, the *domain*). As for the programming perspective, a relation R with arity n is an expression of the form $R(a_1, \ldots, a_n)$ where $a_i \in \mathbf{dom}$ for $i \in [1, n]$ are the attributes of the relation.

The database schema **HR** is composed by eight relation names:
Filter, describes every filter in a given lattice $\mathcal{L}$. If we consider the lattice $\mathcal{L}$ in Example 1, *Filter* contains 5 tuples: $\langle \mathcal{F}_1 \rangle, \langle \mathcal{F}_2 \rangle, \langle \mathcal{F}_3 \rangle, \langle \mathcal{F}_4 \rangle, \langle \mathcal{F}_5 \rangle$.
Concept describes all concepts in a Knowledge Base. An instance of *Concept* based on Example 1 is $\langle C_1 \rangle, \langle C_2 \rangle, \langle C_3 \rangle, \langle C_4 \rangle$.
FilterComposition represents the relation between concepts and filters within a

HR = {*Filter, Concept, FilterComposition, Profile, ProfileByFilters, FilterWeight, MatchingFilters, MatchingProfiles*}

where:

Filter = (FilterName)
Concept = (ConceptName)
FilterComposition = (FilterName, ConceptName)
Profile = (ProfileName, ProfileType)
ProfileByFilters = (ProfileName, FilterName)
FilterWeight = (Filtername, ProfileName, ConceptName, Weight)
MatchingFilters = (FID,RequiredFilter, GivenFilter, BMatch, EMatch, SMatch, MValue, BOverq, EOverq, SOverq, OValue, NextMV, NextOV)
MatchingProfiles = (PID, RequiredFilter, RequiredProfile, GivenFilter, GivenProfile, Fitness, Overq, NextF, NextO)

key constraints:

$Filter_{(key)}$ (FilterName)
$Concept_{(key)}$ (ConceptName)
$FilterComposition_{(key)}$ (FilterName, ConceptName)
$Profile_{(key)}$ (ProfileName)
$ProfileByFilters_{(key)}$ (ProfileName, FilterName)
$FilterWeight_{(key)}$ (Filtername, ProfileName, ConceptName, Weigth)
$MatchingFilters_{(key)}$ (GivenFilter, RequiredFilter)
$MatchingProfiles_{(key)}$ (PID)

and referential constraints:

FilterComposition(FilterName) $\subseteq$ *Filter*(FilterName)
FilterComposition(ConceptName) $\subseteq$ *Concept*(ConceptName)
FilterWeight(FilterName, ConceptName) $\subseteq$ *FilterComposition*(FilterName, ConceptName)
FilterWeight(ProfileName, FilterName) $\subseteq$ *ProfileByFilter*(ProfileName, FilterName)
ProfileByFilter(ProfileName) $\subseteq$ *Profile*(ProfileName)
ProfileByFilter(FilterName) $\subseteq$ *Filter*(FilterName)
MatchingFilters(RequiredFilter) $\subseteq$ *Filter*(FilterName)
MatchingFilters(GivenFilter) $\subseteq$ *Filter*(FilterName)
MatchingProfiles(RequiredProfile) $\subseteq$ *Profile*(ProfileName)
MatchingProfiles(GivenProfile) $\subseteq$ *Profile*(ProfileName)

Fig. 6: Database Schema **HR**

lattice. For instance, in order to specify the composition of filter $\mathcal{F}_2$ in Example 1, tuples $\langle \mathcal{F}_2, C_1 \rangle$, $\langle \mathcal{F}_2, C_2 \rangle$ have to be present in the relation.
In relation *Profile*, *ProfileName* identifies an instance of profile P and *Profile-Type* describes either a given profile P_g or a required profile P_r.
For every filter $\mathcal{F}$ in $\mathcal{L}$, *ProfileByFilter* details profiles names in a given instance of $\mathcal{L}$ represented by $\mathcal{F}$.
FilterWeight, represents records of the associated weights assigned to concepts C in an instance of $\mathcal{L}$, to be used in formula (2). For instance, in order to describe the weights assigned to a given profile P, being an instance of filter $\mathcal{F}_3$ in Example 1, the tuples $\langle \mathcal{F}_3, P, C_3, \frac{3}{10} \rangle$, $\langle \mathcal{F}_3, P, C_1, \frac{1}{10} \rangle$ have to be present in the relation.
MatchingFilters represents the minimum matching measures between profiles

as in matrix $\mathcal{M}$ (Definition 2) such that for two profiles, P_g and $P_r \in \mathbb{P}$, the attribute *MValue* represents the measure $\mu(P_g, P_r)$ and *OValue* represents the measure $\mu(P_r, P_g)$. The attributes BMatch, EMatch and SMatch represent the numbers $\mu_i^{>}, \mu_i^{=}, \mu_i^{<}$ respectively, of the profile record as in Definition 3. And, the attributes BOverq, EOverq, SOverq represent the numbers $\mu_i^{>}, \mu_i^{=}, \mu_i^{<}$ respectively, as described in Definition 4.
MatchingProfiles represents the linked-list of profiles per matching measure as shown in Example 5 such that for two profiles, P_g and P_r in $\mathbb{P}$, the attribute *Fitness* represents the measure $\mu(P_g, P_r)$ while the attribute *Overq* represents the measure $\mu(P_r, P_g)$. Note that *MatchingProfiles* intents to be a representation of elements in Definitions 3 and 4 where the attributes Attributes *NextF* and *NextO* are references to other tuples in the relation (to the tuple with smaller *Fitness* and, to the tuple with the smaller *Overq*, respectively). The attribute PID is intended as an unique identifier of tuples within the relation where *NextF* and *NextO* reference to. We describe in detail the use of attributes *NextF*, *NextO* and ID in the following section.

3.2 Querying the Top-k Candidate Profiles

Filters from a lattice $\mathcal{L}$ represent the properties of profiles via the hierarchical dependency of concepts in $\mathcal{L}$. Therefore, for every *required* profile P_r in $\mathbb{P}$ there is a *required* filter $\mathcal{F}_r \in \mathcal{L}$ representing the profile. Thus, retrieving the top-k candidate profiles for a required filter from the database schema **HR** is mainly performed by querying on relations *MatchingFilters* and *MatchingProfiles*.

For every *RequiredFilter* in *MatchingFilters* there is a number of *GivenFilter* that satisfy the requirements with a specific matching measure. This can be seen as the minimum number of profiles producing all possible measures as in Definition 2. Then if we focus on column σ_i of $\mathcal{M}$ there is matching measure for every *GivenFilter* satisfying the requirements (in every row of $\mathcal{M}$). The attribute *NextMV* in *MatchingFilters* is a reference to another tuple in the relation defining the sequence of *GivenFilter* by their *smaller-than-or-equal* relation of elements of *MValue*.

In turns, for every *RequiredFilter* in *MatchingFilters* there is a *RequiredProfile* in *MatchingProfiles* where the attribute *NextF* is a reference to another tuple in the relation (representing the linked list of profiles as in Example 5). These references define an order on the tuples of *MatchingProfiles* given by the *smaller-than-or-equal* relation of elements of *Fitness*. Thus, if *RequiredFilter* is provided, retrieving the top-k profiles out of the set $\mathbb{P}_\varphi$ as in Definition 1 can be done by pointing to the tuple with the greatest value of Fitness and following the references on *NextF* until the k tuples are reached and $\mu(P_g, P_r) < t_i$.

The algorithm in Fig. 7 shows how to retrieve an ordered list of top-k profiles for a given filter in schema **HR**. Note that we use the notation of relational algebra with subscripts as in [1] thus, σ, π and $\bowtie$ are the *selection*, *projection* and *natural join* operators respectively. We also use numeric subscripts to denote relation attributes. For instance, $\pi_1(\textit{MatchingProfiles})$ is the projection of

Input

Required filter: $\mathcal{F}_r$,
maximum number of matching profiles: k

Output

top-k matching profiles $P_{g1}, \ldots, P_{gk}$,
matching measures $\mu_1, \ldots, \mu_k$ and,
over-qualification measures $o_1, \ldots, o_k$.

Begin

1 $SumP := \pi_{(\mathrm{SUM}_{(\mathrm{EMatch})})} \left(\sigma_{2=\mathcal{F}_r} \left(MatchingFilters \right) \right)$
2 IF $SumP < k$ THEN
3 PRINT (*"There are less than k solutions for filter $\mathcal{F}_r$"*)
4 ENDIF

5 DELETE relation *Results*
6 CREATE relation *Results* = (GivenProfile, Fitness, Overq, NextF)

7 count := 1
8 $P_r := \pi_1 \left(\sigma_{2=\mathcal{F}_r} \left(ProfileByFilters \right) \right)$

9 $MaxMV := \pi_{(\mathrm{MAX}_{(\mathrm{MValue})})} \big(\sigma_{2=\mathcal{F}_r} (MatchingFilters) \big)$
10 $GF := \pi_{(3)} \big(\sigma_{\substack{(\ 2=\mathcal{F}_r,\ \ ) \\ 7=MaxMV}} (MatchingFilters) \big)$

11 WHILE (count $\leq k$) OR ($GF = 0$) DO
12 $Results := \pi_{(5,6,7,8)} \big(\sigma_{\substack{4=GF, \\ 3=P_r}} (MatchingProfiles) \big)$
13 $NextP := \pi_{(4)} (Results)$

14 WHILE *NextP* <> 0 DO
15 $Results := + \ \pi_{5,6,7,8} \big(\sigma_{\substack{4=GF, \\ 3=P_r}} (MatchingProfiles) \underset{1=4}{\bowtie} Results \big)$
16 $NextP := \pi_{(4)} (Results)$
17 count := count + 1
18 END WHILE

19 $GF := \pi_{(12)} \big(\sigma_{3=GF} (MatchingFilters) \big)$

20 END WHILE
21 RETURN ($\pi_{1,2,3}(Results)$)

End

Fig. 7: Retrieving the top-k profiles from **HR**

attribute *GivenFilter* of relation *MatchingProfiles*.

The algorithm accepts as inputs:

- the required filter $\mathcal{F}_r$ and,
- the number $k \in \mathbb{N}$ representing the number of profiles to be retrieved

The output is composed by:

- the k given profiles $P_{g1}, \ldots, P_{gk}$,
- the matching measures $\mu_1, \ldots, \mu_k$ and,
- the over-qualification measures denoted $o_1, \ldots, o_k$.

The algorithm starts by calculating the number of profile instances satisfying the conditions φ in $\mathcal{F}_r$ and allocating it to variable *SumP*. This is calculated by adding up the values of EMatch in *MatchingFilters* where *RequiredFilter* $= \mathcal{F}_r$. EMatch, as well as BMatch and SMatch in *MatchingFilters* represent the number of given profiles P_g matching a required profile P_r in *MatchingProfiles*, ($\mu_i^{>}, \mu_i^{=}, \mu_i^{<}$ respectively, as in Definitions 3). If *SumP* is less than k, it is notified in line 3.

We make use of a temporary relation *Results*, deleted in line 5 and re-created in line 6. *Results* is used to recursively query for the ordered list of matching measures later in the algorithm.

The variable count where **dom**(count)$\in \mathbb{N}$ in line 7, counts the number of profile found in relation *MatchingFilters*.

In line 8, the variable P_r contains the required profile defined by the required filter $\mathcal{F}_r$. For this, we query *ProfileByFilters*.

Between line 11 and 20, we will literally go through every μ_i in the column corresponding to $\mathcal{F}_r$ in matrix $\mathcal{M}$, which is basically done by searching in the linked-list structure in *MatchingFilters*. And, Between line 11 and 20, we get all related profiles as shown in Example 5 until the k elements are found, which is basically to search on the linked-list structure of *MatchingProfiles*.

To achieve this, *GP* maintains record of the *GivenFilter* in *MatchingFilters* to query for in *MatchingProfiles* until, either count $> k$ (k profiles are found) or $GP = 0$ (there is no more μ_i elements per $\mathcal{F}_r$). Also, *NextP* maintains record of the next PID tuple to query for within *MatchingProfiles* until there is no more elements (*NextP*=0) in the linked-list.

Between lines 14 and 18, a recursive search on tuples of *MatchingProfiles* is performed and appended to relation *Results*. For this, the algorithm searches for all tuples in the linked-list of profiles in *MatchingProfiles* where the value of *RequiredProfile* is P_r and the *GivenFilter* $= GF$. In line 15 the algorithm follows the references in *Results* by querying for the next element *NextF* in relation *Results* that is an instance of PID in *MatchingProfiles*. Every queried tuple in *MatchingProfiles* is appended to the tuples in *Results*.

In line 19, *GF* is recalculated in order to get the next *GivenFilter* to be searched for.

In line 17, the algorithm finishes by returning the *GivenProfile*, *Fitness* and *Overq* values of tuples of *Results*.

Note that for simplicity, we use in here SUM, MAX, MIN as the aggregate operators from SQL.

The algorithm in Fig. 7 is intended to search for the best-k profiles matching a required filter $\mathcal{F}_r$ where overqualification has not been considered there. The reason behind it being that overqualification is to be queried the same way as fitness so we need an analogous algorithm as in Fig. 7 where the algorithm to retrieve the best Overq. measure in *MatchingProfiles* should follow the references on *NextOV* in *MatchingFilters* and *NextO* in *MatchingProfiles*.

4 Conclusion

In this paper we presented an algorithm to address top-k queries of matching measures that produce ranking. The approach is intended as an alternative to the sorting and merging of large portions of data in order to produce only very few elements as result, the best k ranked elements. For the implementation of the algorithm we made use of linked-list data structure on top of a relational database schema to store and maintain the ranked elements. We still have to take into account the identification of missing requirements on applicants profiles that are essential on the selection of the best candidates. This implies an investigation of gap queries on grounds of matching measures that is the focus of our future research.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. Kaushik Chakrabarti, Michael Ortega-Binderberger, Sharad Mehrotra, and Kriengkrai Porkaew. Evaluating refined queries in top-k retrieval systems. *IEEE Trans. Knowl. Data Eng.*, 16(2):256–270, 2004.
3. Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.
4. Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
5. Alejandra Lorena Paoletti, Jorge Martínez Gil, and Klaus-Dieter Schewe. Extending knowledge-based profile matching in the human resources domain. In *Database and Expert Systems Applications - 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part II*, pages 21–35, 2015.
6. Nikolaj Popov and Tudor Jebelean. Semantic matching for job search engines: A logical approach. Technical Report 13-02, RISC Report Series, University of Linz, Austria, 2013.
7. Gábor Rácz, Attila Sali, and Klaus-Dieter Schewe. Semantic matching strategies for job recruitment: A comparison of new and known approaches. In *Foundations of Information and Knowledge Systems - 9th International Symposium, FoIKS 2016, Linz, Austria, March 7-11, 2016. Proceedings*, pages 149–168, 2016.
8. Umberto Straccia and Nicolás Madrid. A top-k query answering procedure for fuzzy logic programming. *Fuzzy Sets and Systems*, 205:1–29, 2012.

9. Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 648–659, 2004.