Ontology-based skill matching algorithms

Conference Paper · September 2017

DOI: 10.1109/ICCP.2017.8117005

Marcel Antal
Technical University of Cluj-Napoca
70 PUBLICATIONS **1,876** CITATIONS
SEE PROFILE

Ioan Salomie
Technical University of Cluj-Napoca
210 PUBLICATIONS **2,897** CITATIONS
SEE PROFILE

Tudor Cioara
Technical University of Cluj-Napoca
167 PUBLICATIONS **2,648** CITATIONS
SEE PROFILE

Ionut Anghel
Technical University of Cluj-Napoca
160 PUBLICATIONS **2,626** CITATIONS
SEE PROFILE

Ontology-Based Skill Matching Algorithms

Teodor Petrican
Technical University of Cluj-Napoca
teodor.petrican@student.utcluj.ro

Ciprian Stan
Technical University of Cluj-Napoca
ciprian.stan@student.utcluj.ro

Marcel Antal
Technical University of Cluj-Napoca
marcel.antal@cs.utcluj.ro

Ioan Salomie
Technical University of Cluj-Napoca
ioan.salomie@cs.utcluj.ro

Tudor Cioara
Technical University of Cluj-Napoca
tudor.cioara@cs.utcluj.ro

Ionut Anghel
Technical University of Cluj-Napoca
ionut.anghel@cs.utcluj.ro

***Abstract*—Automatic recommendations based on skill matching techniques can prove to be an important component of an online recruitment platform, being able to lower the costs for employers, ease the process for candidates and increase the hiring quality overall. This is important nowadays, when online recruitment plays a major role in the hiring process. The main challenges in this area consist in providing relevant and computationally inexpensive results. In this paper we propose a semantic approach to the skill matching problem in the context of online recruitment. We present a metric of similarity based on a skills ontology and three algorithms on top of this metric, with the intent of obtaining a ranked skill-based matching between candidates and job offers. We also provide an analysis of those algorithms in terms of advantages, disadvantages and complexity.**

Keywords*—skill matching, ontology, semantic, similarity metric, taxonomy, algorithms

I. Introduction

Due to the globalization trend, the recruitment domain is in a continuous expansion. There is a large number of candidates in certain job sectors, which makes the hiring process a very difficult and consuming task for companies.

In order to facilitate this process, there is an ongoing trend to rely on automatic tools and platforms which can provide recommendations of job opportunities, or filter candidates based on their expertise. Skill matching is an important step in this process, i.e. determining a meaningful degree of matching between a candidate and a job based on the possessed expertise and the required skills for the job.

This domain is of particular interest, since online recruitment plays a large and ever growing role in the hiring process. As a testimony, half of the recent hires in Germany were done using online platforms [1].

There is a variety of approaches to compute this matching based on skills. The main challenges consist of providing relevant results in a computationally inexpensive manner, in order to make the tool feasible. In this paper we propose three approaches to this problem and, furthermore, we compare them based on their runtime efficiency, advantages and disadvantages. All of the three aproaches are based on the same underlying data model and metric of similarity, which are also presented in the paper. The first approach uses an exhaustive search in the input domain. The second approach improves upon the exhaustive search with the help of some observations, to achieve a better running time. Finally the third approach is based on another idea than the first two approaches, namely on the multi-source shortest path problem.

The remainder of this paper is organized as follows. Section 2 presents the work related to skill matching and other approaches in solving this problem. Section 3 presents a formalization of the problem and explains our approach to it. Section 4 presents the three algorithms that we propose, all based on the same underlying approach. Section 5 presents the evaluation results that we got on the algorithms, and, finally, section 6 presents the conclusions.

II. Related Work

There is a wide array of approaches in literature regarding the skills matching problem. There are solutions which rely on description logic [2], on machine learning [3], or semantic, ontology based approaches [1], [4], [5]. There are solutions which try to get the best of both worlds by merging the approaches and creating a hybrid solution, such as running machine learning algorithms on top of an ontology [6], or using a description logic model alongside an ontology framework [7].

It is important to note that, no matter the approach used, the success of the matchmaking process is largely dependent on how effective the participants' profile is modeled [8]. Thus, an important step in approaching this problem is the data modelling.

As previously noted, there is an emerging trend to rely on a tree or graph structure to model the domain [9]. This structure is usually modeled with the help of an ontology, on which an algorithm operates to perform matching. It is generally accepted that such a representation is a good model for the information.

Lv et al [10] propose a system that has beneath an ontology. Job postings and candidates have a set of competences, which are the concepts stored in the ontology. The ontology modelled by the authors is a graph where nodes are skills and weighted edges are relationships among skills and is constructed from a series of sub-ontologies that map different domains. The approach of the authors for the matching between a job offer and a candidate is an exhaustive search which compares all the skills from the offer with all the skills of a candidate and computes the sum of the best matchings. In order to determine similarity between two skills in the same sub-ontology, the authors use a method that employs finding the shortest path in the graph. The similarity between skills is the sum of the weights of the edges in the shortest path.

A similar idea is also proposed in [4]. The advantage of such an approach is that it can provide highly relevant results and the algorithm can be computationally inexpensive if implemented right. The main drawback is that the ontology weights need to be manually assigned, possibly by domain experts, which can prove to be a laborious task.

Another idea is to make the ontology proposed take the form of a taxonomy for describing and classifying skills. The advantage of using a taxonomy is that it might be easier to develop than a graph-like ontology, while still maintaining useful information in the matching process.

Bizer et al. [1] take this approach further. They propose the enhancement of this taxonomy with the help of a similarity function. This similarity function is inversely proportional to the distance between two concepts in the taxonomy. The distance is then computed with the help of weights in the taxonomy, which are also inversely proportional to the depth of the skills in the ontology: $(Milestone\,(n) = \frac{1}{k^{l(n)}})$.

Another interesting observation is that, in order to have a more customizable matching process, one could allow the user to make a distinction between nice-to-have and must-have requirements [11]. In this way, not every matching will contribute with the same amount to the final matching result.

In our work, we use some of the observations presented above and enhance on them. We will use a tree structured ontology, with a weighting and similarity function similar to the one proposed by [1]. We enhance on this idea, by proposing a similarity metric which takes into account the direction of traversing the ontology when matching skills; we also associate an importance function to the required skills of an. On top of this ontology and metric, we propose three algorithms to effectively compute the matching. We then develop a test scenario to compare the proposed algorithms.

III. Skill-Matching Problem Formalization and Approach

Let us formalize the statement for the skill-matching problem. Throughout the following sections we will refer to a candidate's skill set with the term *CV* (as in the curriculum-vitae of the candidate).

Let S be the set of all skills. Let CV be the set of all the CVs in the search space and OF be the set of all offers in the search space. An individual CV, $c \to CV$, is a subset of S, consisting of a set of skills from S that the candidate has: $c = \{c_1, c_2, \dots c_n\} \to S$. Analogously, an individual offer, $o \to OF$, is a set of skills from S that are required by the job: $o = \{o_1, o_2, \dots o_n\} \to S$. Furthermore, each offer o has associated an importance function $imp_o : \{o_1, o_2, \dots o_n\} \to \{i_1, i_2, \dots i_n\}$, which maps an importance value to each skill from an offer. The problem is to compute a metric of similarity:

$$SF : CV * OF \to [0, 1] \tag{1}$$

which, applied to a pair of one CV and one offer, computes a similarity or matching degree. This function, applied from a candidate to all the offers, or from an offer to all the candidates can be used to provide a ranked list of recommendations.

A. Our approach

All of the algorithms that are presented in this paper are based on an ontology of skills. The ontology is a taxonomy which presents the relationship among all the skills in the set S in the form of a tree. Each node will represent a skill. There will be a top concept in the tree, and then the concepts will divide into domains and sub-doimains of activity, where higher nodes will define more general concepts, describing areas of expertise and lower nodes will define more specific and hard concepts/skills. Such an ontology is depicted in figure 1.

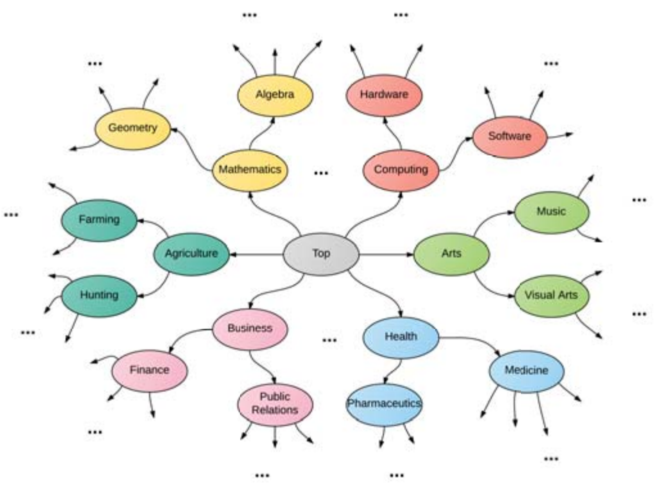

Fig. 1. Taxonomy of skills (first 3 levels)

An intuitive metric on this ontology to compute the similarity between two skills is the length of the path between two skills in the ontology. The problem with this direct approach is that it does not take into account specificity. In order to do this, the distance between nodes that are more specific (deeper in the ontological tree) should be smaller than the distance between nodes that are more generic (higher in the ontological tree). This is intuitive, as the distance between C++ and Java, for example, should be smaller than the distance between Programming and Agriculture. This can be achieved by setting appropriate weights on the relationships between skills. As previously mentioned, one way to achieve this is to manually assign the weights with the help of domain experts. Unfortunately, this is a laborious task which also needs to be repeated with subsequent changes to the ontology that are likely to happen. The other approach, which is preferred, is to try and automatically assign a relevant metric on the ontology.

As also presented in [1], we exploit the fact that the ontology is a taxonomy, and enhance it with a distance metric, $d\,(a, b) : T * T \to [0, 1]$, defined in equation 2. The distance function can then be used directly to conceive a metric of similarity, $SF\,(a, b) : CV * OF \to [0, 1]$, that is:

$$d(a,b) = \left\{ \begin{array}{ll} min(|milestone(b_1) - milestone(b)| + d(a,b_1), 1), & \text{if } a \in subtree(b) \land b_1 \in child(b) \land b_1 \in path(b,a) \\ min(\frac{children(a)}{depth(a)} * |milestone(a) - milestone(z)| + d(z,b), 1), & \text{if } a \in ancestor(b) \land z \in child(a) \land z \in path(a,b) \end{array} \right\} \tag{2}$$

$$s(a,b) = \begin{cases} 1, & \text{if } a = b \\ 1 - d(a,b), & \text{if } a \in ancestor(b) \lor b \in ancestor(a) \\ max(1 - (d(a,cpp) + d(cpp,b)), 0), & \text{if } siblings(a,b) \land cpp = LCA(a,b) \end{cases} \tag{3}$$

$$SF(cv,o) = S([c_1,...,c_m],[o_1,...,o_n]) = \frac{\sum_{i=1}^{n} imp_o(o_i) * \max_{j \in 1..m} s(c_j,o_i)}{\sum_{i=1}^{n} imp_o(o_i)} \tag{4}$$

In the previous formulae, the meaning of the defined functions are the following:

- $milestone(a) : T \rightarrow [0,1], milestone(a) = \frac{1}{2^{depth(a)}}$, the metric associated to each skill in the graph, used to compute distances between neighbors.
- $depth(a) : T \rightarrow N^*$ - the depth of the skill a in the tree of skills.
- $children(a) : T \rightarrow N^*$ - the number of children of node a in the ontology.
- $imp_o(a) : T \rightarrow N^*$ - importance of a skill in the offer, as explained in the problem definition.
- $LCA(a,b)$ - lowest common ancestor of a and b.

Let us briefly explain the defined formulae. At (2), we have defined the formula which computes the distance between 2 nodes in the tree, $d(a,b)$. The $milestone(a)$ is used; one can see that by doing an absolute difference between the milestones of 2 neighboring nodes we effectively get a weight associated for the edge between them. This results i a metric which has higher value if we are higher in the tree. As the $milestone(a)$ is an inversely proportional function in terms of depth in the tree, the distance between deeper edges (more specific skills) will be smaller than the distance between higher level edges (more general skills); this handles the specificity issue described previously. The d function is capped at 1 (hence the min function).

Also, the distance formula is not symmetric, i.e. the distance from skill a to b is different than the distance from b to a. This is because, when the user has a more general skill than the one required by the offer it should be penalized more than the other way around. For example, if a user knows Java and the required skill is OOP, it is reasonable to assume that the user knows OOP concepts too; but in the other way around, if the required skill is Java and the user knows OOP, we should penalize the matching more, since we cannot assume that the user also knows Java.

To achieve this, we have made 2 cases for the formula. In the first case, when skill of CV is a child of the skill of the offer ($a \in child(b)$), we don't add any penalization. We simply follow the path from b to a and sum the edge weights. In the second case, when going upwards in the tree, the branching factor is also multiplied with the distance (i.e. $\frac{children(a)}{depth(a)}$).

The function at (3), $s(a,b)$, simply computes the similarity between 2 skills, which is the inverse of the distance. The higher the value of this function is, the higher the similarity between the skills is.

Finally, the metric of similarity SF defined at (4), outputs the similarity between a full CV and a full offer. For each skill of the offer, it takes the best matching score with a skill of the CV (i.e. $\max_{j \in 1..m} s(c_j,o_i)$), and computes an average between those.

Because, as noted by [11], in a real job offer scenario not all skills have the same importance, we also add a discrete importance function to allow the possibility to provide customized offers, imp_o. Hence, the metric of similarity will assign as final matching degree between a candidate and an offer the *weighted* average of the smallest distances (best scores) between each skill from the offer and a skill of the candidate (4). In our experiments we have used a discrete, three-valued function for imp_o: 1 - *nice to have*, 3 - *should have*, 5 - *important to have*, but one can see that the way in which the metric of similarity is defined leaves freedom of choice for this function.

IV. Algorithms

The metrics that were presented in the previous section represent the foundation of the algorithms that were developed for computing the matching degrees between CVs of candidates and job offers of companies. Basically, all that remains to do is to find an efficient manner to compute the metric of similarity on a given ontological tree of skills, for a given set of CVs and offers.

A. Exhaustive search algorithm

In this subsection, two exhaustive approaches that are based one the same idea are going to be presented. As mentioned, both of them are exhaustive searches, because they compute the matching degree between all the skills from the offer against all the skills of a CV. The exhaustive approach procedure is presented in pseudocode 1.

The function of similarity is computed by the procedure call at line 9. It needs to compute the lowest common ancestors between the two given nodes (see formula (3) in previous section) and then walk through the whole path between the two nodes to compute the function (4). The code for

Algorithm 1 Exhaustive FIND-MATCH algorithm

Input:

cv - the CV of a candidate

$offer$ - the job offer of a company

Output:

$percentage$ - the matching percentage between CV and offer

```
procedure FIND-MATCH($cv, offer$)
    $skillsRequired \leftarrow offer.getSkills()$
    $skillsPossessed \leftarrow CV.getSkills()$
    $sum \leftarrow 0$
    $count \leftarrow 0$
    for each $skill\ o \in skillsRequired$ do
        $max \leftarrow 0$
        for each $skill\ c \in skillsPossessed$ do
            $score \leftarrow similarity(o, c)$
            if $score > max$ then
                $max \leftarrow score$
            end if
        end for
        $sum \leftarrow sum + max$
        $count \leftarrow count + 1$
    end for
    $percentage \leftarrow sum/count$
    return $percentage$
end procedure
```

$similarity(o, c)$ has been omitted, since it is straightforward to implement.

Then, given the above, the difference between the two approaches is the technique used for finding the lowest common ancestor that is needed for the similarity metric.

The first approach uses hierarchy lists for determining the lowest common ancestor between two skills, which take a linear time in terms of the height of the ontological tree. As it can be seen in the code, the LCA calculation (line 9) is executed for each combination of a skill of the candidate and a skill of the offers, for all offer, thus yielding a complexity of $\boldsymbol{O(O \cdot S_o \cdot S_{cv} \cdot h)}$, where O is number of available job offers, S_o is the number of skills of the job offer, S_{cv} is the number of skills from the candidate's CV and h is the height of the ontological tree of skills. The complexity is analogous in case we have one offer and multiple CVs.

The second approach comes with an improvement to the lowest common ancestor retrieval. It uses method that computes the LCA of two skills in $O(1)$ with the help of a dynamic programming approach that is also used to solve the range minimum query problem [12]. It requires $O(nlogn)$ additional memory for preprocessing the skills tree (n is the number of the skills in the tree). This preprocessing that is done only once before making any LCA query is used to compute the Eulerian tour of the skills tree, the level of each skill on the tree and the first appearance of each skill in the Eulerian tour. By having this information stored and using the idea from Range Minimum Query, sparse tables and dynamic programming, we can answer each LCA query in $O(1)$. As the skills ontology does not change, and having to do the preprocessing only once, we obtain an overall method complexity for answering a matching query of $\boldsymbol{O(O \cdot S_o \cdot S_{cv})}$. This could be further enhanced in terms of space complexity using the approach presented in [12] and later in [13].

Even though this approach is computationally expensive as search space increases, the great advantage of this method is that it is bidirectional, namely it can go from a CV to offers and from an offer to CVs, yielding the same result. We shall see that, unfortunately, not all of the methods proposed in this paper posses this feature.

B. Breadcrumbs algorithm

Instead of taking all the possible pairs of skills and computing the LCA between them, we could reduce the number of steps, by splitting the work in two stages and computing the LCA inherently as part of the process. The algorithm name is inspired by the technique used, namely that of leaving breadcrumbs along a way to mark a trace which can then be used later on. The approach presented in this paper works for one CV to multiple offers direction (i.e. ranking offers for a CV); the other direction is skipped since it is analogous.

Algorithm 2 Breadcrumbs algorithm

Input:

cv - the profile of one user

$\mathcal{O}$ - collection of offers

Output:

$\mathcal{M}$ - collection of Matching results used to compute the similarity percentage

```
procedure MAIN
    $\mathcal{T} \leftarrow createTreeFromOntology()$
    $BREADCRUMB - DROP(\mathcal{O}, \mathcal{T})$
    return $BREADCRUMB - SEARCH(cv, \mathcal{T})$
end procedure
```

Algorithm 3 BREADCRUMB-DROP procedure

Input:

$\mathcal{O}$ - collection of offers

$\mathcal{T}$ - tree of skills

Output:

$\mathcal{T}$ - tree, updated in-place

```
procedure BREADCRUMB-DROP($\mathcal{O}, \mathcal{T}$)
    for each $offer \in \mathcal{O}$ do
        $So \leftarrow offer.skills()$
        for each $skill \in So$ do
            $current \leftarrow \mathcal{T}.getNode(skill)$
            $distance \leftarrow 0$
            while $current \neq NIL$ do
                $current.addBreadcrumb(skill, distance, offer)$
                $distance \leftarrow updateDistance(distance, current)$
                $current \leftarrow current.parent$
            end while
        end for
    end for
end procedure
```

Algorithm 4 BREADCRUMB-SEARCH procedure

```
Input:
    $cv$ - the profile of one user
    $\mathcal{T}$ - tree of skills
Output:
    $\mathcal{M}$ - collection of Matching results

procedure BREADCRUMB-SEARCH($cv, \mathcal{T}$)
    $\mathcal{M} \leftarrow$ collection of Matching results indexed by offer

    for each $skill \in cv$ do
        $current \leftarrow \mathcal{T}.getNode(skill)$
        $distance \leftarrow 0$
        while $current \neq NIL$ do
            for each $b \in current.breadcrumbs$ do
                $newScore \leftarrow computeScore(distance + b.distance)$
                if $newScore > b.score$ then
                    $b.score \leftarrow newScore$
                    $b.skill \leftarrow current.skill$
                    $\mathcal{M}.updateMatching(offer, b)$
                end if
            end for
            $distance \leftarrow updateDistance(distance, current)$
            $current \leftarrow current.parent$
        end while
    end for

    return $\mathcal{M}$
end procedure
```

In the first stage, a walk is started from each skill of each offer in the ontological tree, upwards to the root, leaving "breadcrumbs" in each node on the way. A breadcrumb will have associated the origin of the path and the distance travelled so far (computed with the distance formula described in the previous chapter). This part of the algorithm is depicted in pseudocode 3; the main navigating loop which occurs for each skill of the offer is between the lines 7-11. The origin of the path (i.e. *skill*) will be used in the second stage of the algorithm to know which skill dropped the breadcrumb (i.e. the skill with which we have to compute the matching score). The distance travelled so far will also be used in the second stage to compute the final matching score between the two skills. Both in algorithm 3 and in algorithm 4, the distance travelled needs to reflect the formula (2) presented in the previous section. This is handles by the *updateDistance* (line 9 in 3) procedure which will increase the distance with a weight according to formula (2), depending on which of the two cases we are at. The code for this procedure was omitted.

In the second stage, a walk is started from each skill of the CV upwards to the root. This time, in each node, it iterates through all the breadcrumbs of it, computes a matching score based on the distance travelled so far and information from the breadcrumb, and checks if this matching score is better than any older matching for this skill and offer. Notice that a breadcrumb is first encountered in the node that is also the LCA between the skill of the CV and the breadcrumb's skill; a slight optimization could be done, such that after the first encounter of a skill's breadcrumb, the upcoming breadcrumbs of the same skill will be ignored, since they cannot possibly yield a better score. At the end of this stage the algorithm produces a collection of all the matches of the given CV with the given offers. All that remains to do is to sort this list and the problem is solved. The second stage algorithm is presented in pseudocode 4.

The worst case complexity of this algorithm is $\boldsymbol{O((S_{cv} \cdot S_o + S_o) \cdot O \cdot h)}$, where O is the size of the offers set, S_o is the average number of skills in an offer, S_{cv} is the average number of skills in the CV and h is the height of the ontological tree. Although the theoretical complexity is worse than the one of the exhaustive search, this is not a tight bound and the algorithm behaves much better in practice, as also presented in the evaluation section. Intuitively, this happens because it is improbable that in each node up to the root there will be a breadcrumb from each skill of each input. Actually, the number of encountered breadcrumbs in a node will be much smaller in the majority of cases, thus making the overall running time better.

C. Multi-source shortest path algorithm

The idea behind this algorithm is to view the ontology as a graph. In order to have this possibility, we need to convert it to a graph that is ready to be parsed by a shortest path algorithm. This operation needs to be done only once, and it involves creating two edges for each relationship in the ontology, one going upwards and one going downwards, each with different weights (one for each condition of the formula (2)). The algorithm, presented in pseudocode 5 assumes that this graph is already pre-processed and distances correctly set.

Now that we have this graph, we can start the algorithm. All the skills of a candidate are set as starting nodes for the algorithm. A reference is also added from each node in the ontology to all the offers that require the skill represented by that node. Then, a multi-source Dijkstra algorithm [14], [15] is employed on this setup, with a slight modification: the algorithm has no clear destination in mind, but it operates up to a certain distance threshold, from which skills are considered too distant to be relevant. At each step of the algorithm, the matching of the CV with all the offers that are referenced from the current node is updated. At the end, there will be a matching of the CV with all the offers in the input set, thus the problem is solved.

Using a Fibonacci heap for the implementation of the multi-source shortest path algorithm its complexity is $\boldsymbol{O(E + VlogV)}$. The algorithm is due to [16]. Since the graph is actually a tree (i.e. the number of edges is always V-1), the complexity becomes $\boldsymbol{O(V + VlogV) = O(VlogV)}$. Therefore, the final complexity of the algorithm is $\boldsymbol{O(VlogV + O \cdot S_o)}$, where V is the number of vertices visited, O is the number of offers and S_o is the average number of skills that an offer requires. This complexity looks very promising, since V (here, the number of vertices visited by the algorithm) is a function of THRESHOLD, therefore being constant with respect to input size.

The main disadvantage of this algorithm is that it only works in one direction of the problem, namely from a CV to more offers (ranking of offers for a candidate). The reason

Algorithm 5 Multi-source Shortest Path Algorithm

```
Input:
    $cv$ - the cv of a user
    $\mathcal{O}$ - collection of offers
    $\mathcal{G} = (V, E)$ - graph of skills described by vertices and edges
Output:
    $\mathcal{M}$ - collection of Matching results

 1: procedure SEARCH($cv, \mathcal{O}, \mathcal{G}$)                          ▷ executing the mssp
 2:     $priorityQueue \leftarrow emptyFibonacciHeap$
 3:     for each $skill \in cv$ do                                          ▷ add skills of CV as sources
 4:         $priorityQueue.insert(skill, 0)$
 5:     end for
 6:     $appendOffers(\mathcal{G})$                                             ▷ attach the offers to nodes

 7:     while $not(priorityQueue.empty())$ do                                ▷ execute the search
 8:         $current \leftarrow priorityQueue.dequeueMin()$
 9:         $current.setColor(BLACK)$
10:         for each $e \in current.edges()$ do
11:             $destination \leftarrow e.destination$
12:             if $destination.color \neq BLACK$ then
13:                 $dist \leftarrow current.distance + e.cost$
14:                 if $dist < destination.distance$ AND $dist < THRESHOLD$ then
15:                     if $destination.color = WHITE$ then
16:                         $destination.color = GRAY$
17:                         $priorityQueue.insert(destination, dist)$
18:                     else                                 ▷ if already in the queue, execute decreaseKey
19:                         $priorityQueue.decreaseKey(destination, dist)$
20:                     end if
21:                 end if
22:             end if
23:         end for

24:         for each $offer \in current.attachedOffers$ do                  ▷ update attached offers
25:             $offer.updateMatching(current)$
26:             $resultSet.add(offer)$
27:         end for
28:     end while

29:     return resultSet
30: end procedure
```

is that, during the running of the algorithm the paths from two source skills might intersect. At this point, only one of them will go further; thus, a skill appearing on the path from this point onward will only by matched by one of the two source skills. This is problematic for the offer to multiple candidates direction, since we want all the skills of the offer to be matched, but not necessarily all the skills of the candidate. This drawback could be eliminated with the use of a caching mechanism to save the matching degrees in the candidate to multiple offers direction and then use them to build results for the opposite direction too.

The advantage of this approach is that it can be used directly even on a differently structured ontology, which has more complex relationships among skills than a taxonomy can offer, since it is not tightly bound to how the similarity metric is defined. This is because of the fact that the algorithm treats the underlying data model as a graph, while the other approaches are relying on the tree-like structure.

V. Evaluation

For the purpose of evaluation, a taxonomy was manually created, which contains 485 concepts from various working domains. Of course, this is not an exhaustive ontology, but it served for the purpose of evaluation. The first levels of the ontology are the ones presented in figure 1.

A data generator has also been implemented and used in order to generate a large amount of candidates and offers in a semi-random manner. The data generator tries to simulate a real-world scenario where a candidate usually has the majority of skills clustered (closely related to each other) rather than scattered throughout the ontology. It does this by first randomly choosing a domain of activity, and then randomly selecting skills in that domain of activity. Then, it also adds a few other skills from different domains. All of those parameters (i.e. the number of skills from a domain, from different domains, number of users etc.) are configurable. The process is similar

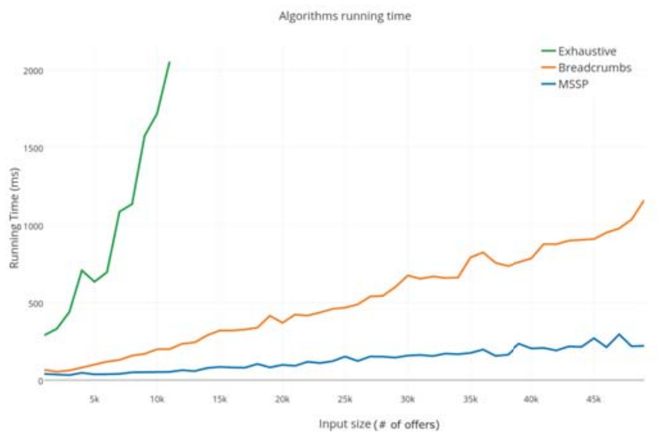

Fig. 2. Evaluations Result

for offer creation, only that importance values for the generated skills are also involved in the process.

To eliminate the randomness factor, the test was run 5 times for each data size on different input, and the average running time was taken as the final value. The tests were all executed on the same Linux machine, having an Intel Core i7-2670QM CPU @ 2.20 GHz and 8GB of DDR3 RAM. The results are presented in figure 2.

As it can be seen, the Multi-Source Shortest Path algorithm provides the best scalability. The differences are already notable from small input sizes: at 5000 offers the running times are 0.037 seconds for MSSP, 0.1 seconds for Breadcrumbs and 0.63 seconds for the exhaustive approach. The exhaustive approach quickly becomes unfeasible for real-time processing. At an input size of 40000 offers, the running time reaches 0.2 seconds, whereas the Breadcrumbs algorithm takes 0.75 seconds. The advantage of the Breadcrumbs approach is that it works on both directions of the problem without requiring additional mechanisms. Note that all of the three algorithms are based on a tree structured ontology and an impartial metric that is applied on all relationships, no matter their field of study. Of course, this approach involves a little bit of rigidity and can pose problems with the relevancy of the results; in our tests the results were relevant enough for recommendation systems, though not for automatic decision taking.

The relevancy could be improved with the use of a more complex ontology structure, where there are more relationships among all skills from a domain, and they have custom weights. This would require more manual work initially, as some domain experts would have to carefully assign the weights. Work can be done in developing an underlying mechanism/technique that allows altering the weights automatically based on user feedback and other input.

VI. Conclusions

In this paper, a metric that can automatically enhance a skills ontology has been presented. Based on this, three ontology-based algorithms were developed and evaluated in terms of their runtime efficiency. The most efficient approach is the multi-source shortest path one, having the disadvantage that it only works in one direction of the problem, although this can be bypassed with the help of a caching mechanism.

These algorithms, along with the skills ontology, can form the basis for a recruitment platform with automatic matching features, which can be used both by companies for easier selection of candidates, as well as by job seekers as a recommendation tool well searching for a job. The skills ontology can also help in other auxiliary processes on such a platform, such as automatic CV parsing.

Also, the matching techniques presented in this paper could also be used in larger contexts for matching problems where the individuals can be reasonably described with the help of a similar ontology, such as in e-commerce or web-services.

References

[1] C. Bizer, R. Heese, M. Mochol, R. Oldakowski, R. Tolksdorf, and R. Eckstein, *The Impact of Semantic Web Technologies on Job Recruitment Processes*. Heidelberg: Physica-Verlag HD, 2005, pp. 1367–1381.

[2] A. Cali, D. Calvanese, S. Colucci, T. D. Noia, T. Di, N. Francesco, F. M. Donini, and U. D. Tuscia, “A description logic based approach for matching user profiles,” in *In Proc. of the 8th Int. Conf. on KnowledgeBased Intelligent Information & Engineering Systems (KES 2004), volume 3215 of Lecture Notes in Artificial Intelligence*, 2004, pp. 187–195.

[3] E. Faliagka, K. Ramantas, A. Tsakalidis, and G. Tzimas, “Application of machine learning algorithms to an online recruitment system,” *The Seventh International Conference on Internet and Web Applications and Services*, 2012.

[4] S. Mohaghegh and R. R. Mohammad, “An ontology driven matchmaking process,” *TSI PRESS SERIES*, vol. 16, pp. 248–253, 2004.

[5] F. M. Hassan, I. Ghani, M. Faheem, and A. A. Hajji, “Ontology matching approaches for erecruitment,” *International Journal of Computer Applications*, vol. 51, no. 2, pp. 39–45, August 2012.

[6] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, “Ontology matching: A machine learning approach,” in *Handbook on ontologies*. Springer, 2004, pp. 385–403.

[7] M. Fazel-Zarandi and M. S. Fox, “Semantic matchmaking for job recruitment: an ontology-based hybrid approach,” *Proceedings of the 8th International Semantic Web Conference*, 2009.

[8] M. Joshi, V. C. Bhavsar, and H. Boley, “Knowledge representation in matchmaking applications,” in *Advanced Knowledge Based Systems Model Applications & Reasearch*.

[9] F. M. Hassan, I. Ghani, M. Faheem, and A. A. Hajji, “Article: Ontology matching approaches for erecruitment,” *International Journal of Computer Applications*, vol. 51, no. 2, pp. 39–45, August 2012, full text available.

[10] H. Lv and B. Zhu, “Skill ontology-based semantic model and its matching algorithm,” in *2006 7th International Conference on Computer-Aided Industrial Design and Conceptual Design*, Nov 2006, pp. 1–4.

[11] M. Fazel-Zarandi and M. S. Fox, *Reasoning about Skills and Competencies*. Springer Berlin Heidelberg, 2010, pp. 372–379.

[12] M. A. Bender and M. Farach-Colton, “The lca problem revisited,” in *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, ser. LATIN ’00. London, UK, UK: Springer-Verlag, 2000, pp. 88–94.

[13] J. Fischer and V. Heun, “Theoretical and practical improvements on the rmq-problem, with applications to lca and lce,” in *Proceedings of the 17th Annual Conference on Combinatorial Pattern Matching*, ser. CPM’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 36–48.

[14] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[16] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987.