OpenSKIMR
Enabling a Fast Road to Digital Careers

Optimized Computing Time for Mobile Applications

Peter Mirski, Johannes Sappl, and Florian Dreier

Management Center Innsbruck (MCI), Innsbruck, Austria
peter.mirski@mci.edu, johannes.sappl@mci.edu, florian.dreier@mci.edu

Abstract

This paper is proposing an algorithm that is capable of constructing a learning route given restrictions on cost, duration, and venue. It is intended for users seeking job opportunities inside the ever-growing European labor market. We provide a detailed introduction of the mathematical framework behind it and compare our new approach to an existing algorithm. In respect of potential mobile applications, we lowered the computational complexity of the algorithm as effectively as possible without abandoning its strategic objective.

Keywords

OpenSKIMR, Job matching, ESCO, Europen Qualification framework, Employment, Skill Demand, Occupational Choice, Recommender Systems, eLearning

1. INTRODUCTION

OpenSKIMR, a pilot project, funded by the European Commission (EUROPEAN COMMISSION, Directorate General for Communications Networks, Content & Technology (DG CONNECT)) has the vision of setting up a prototype to showcase the impact of applying the ESCO catalogue identifying skills for job opportunities and best fitting learnings to underpin individual careers to efficiently upskill young talents with IT related competencies. The visualization of the matching results is realized like classical route planner interface with the option to adapt it to different desired specifications like price or duration. The phenomenon of skill mismatch involves underskilling as well as overskilling. Both can undermine the long-term potential of the workforce [1]. Suggesting an education route: The user receives recommendations for learnings which it might be interested in in order to better fit a job's requirements. This paper is a sequel to the work done in [2]. Since product owners demands have changed in the second project year due to a strong focus on mobile applications, the project was in need of a new algorithm that brings short computation time and closes essentially all skill gaps. This has been achieved by not constructing a graph and searching all of its branches for some almost perfect solutions as done in [2], but rather optimizing skills one by one in a greedy fashion. Speed has greatly improved at the expense of finding only a somewhat not as perfect solution to our problem.

2. NOTATION

OpenSKIMR is working with a newly created data set named *ESCO* (European Skills, Competences and Occupations) [3], who's first version went live just recently. This catalogue provides a standardized set of more than 12000 skills, knowledges, and about 3000 different occupations as well as the relationships between those two, see Figure 1. ESCO has been developed to help close the gap between educational institutions, and the European labor market, cf. [1], with the aid of said standard terminology.

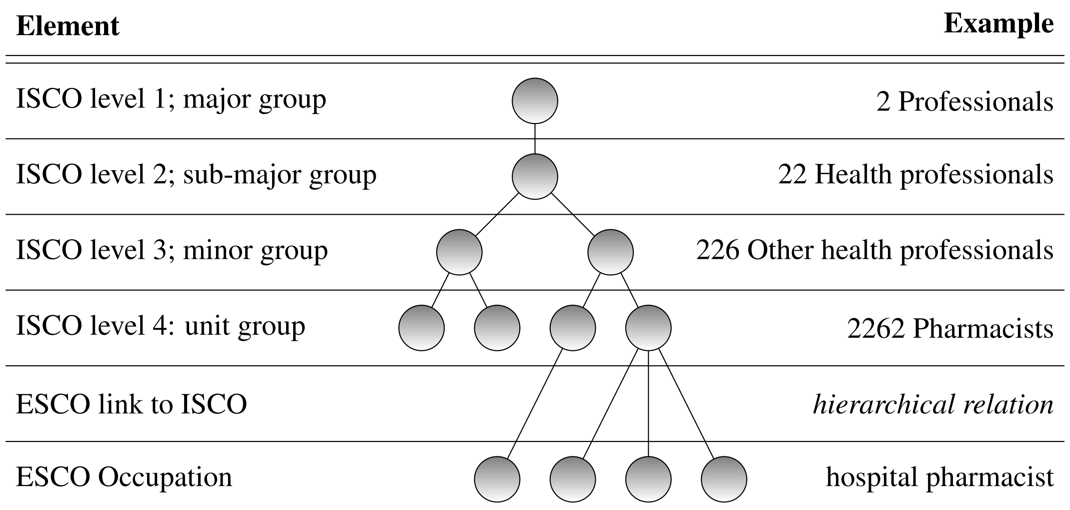

FIGURE 1 Hierarchy of the ESCO data set as defined by the ISCO/ESCO standard.

Every user on the OpenSKIMR platform has a certain set of skills which can be represented by an element in an n-dimensional real vector space R^n, where $n \in \mathbb{N}$ is equal to the number of different skills in the ESCO data set and $R \in \mathbb{N}_{>0}$ is the number of skill levels. We denote a user by u and define the set of his skill indices like $I_u := \{i \in \{1, 2, \dots, n\} \mid u_i \neq 0\}$. In the following let us denote some given job the user u might be interested in by $v \in R^n$ and the set of skill indices which are required for the job by $I_v = \{i \in \{1, 2, \dots, n\} \mid v_i \neq 0\} = \{i_1, \dots, i_k\}$ with cardinality $k := |I_v|$.

Every learning L has a set of skills $\mathcal{S}_L \subset \{1, 2, \dots, n\}$ with an entry level $r_0(j), j \in S_L$ the user has to have, and an exit level $r_1(j) > r_0(j)$ which defines how much the learning improves the skill. Clearly, a user u can only attend the learning if $u(j) \geq r_0(j)$ for all $j \in \mathcal{S}_L$. Thus, we can define a learning function as a mapping $L \colon R^n \to R^n$ that either yields u itself, if there exists some $j \in S_L$ such that $u(j) < r_0(j)$, or a new user vector $\widetilde{u}$, where we set $u(j) = r_1(j)$ for all $j \in S_L$.

Having introduced all the relevant definitions for our algorithm, we can now begin to describe the route planning process in the following section.

3. ROUTEPLANNER

One of the main goals of the OpenSKIMR project is to offer the user a set of courses distributed in Europe which he has to attend to fulfill the requirements for a job he would like to have.

Based on the skill set of the user, he has in general the possibility to attend as many courses as he wants to improve his skills in order to gain the necessary skills for the desired job. Since a big amount of courses is available, the user would need a lot of time to collect the appropriate courses. Furthermore, there is large number of possibilities to attend different courses which improve the skill set of the user. Thus it is impossible for the user to pick the "best" learnings with respect to the job. These are two of many arguments why want to offer the user an automatically computed set of courses. The next section is concerning with the following questions:

- Does there exist a method, which computes *the* "best" learnings?
- If yes, is it practically applicable?

3.1. Mathematical Problem behind the Route Planner

In order to answer the above mentioned questions, we firstly have to clarify what we understand by *best learnings*.

3.1.1. What are the best learnings?

Generally, every learning $L \in \mathcal{L}$ has the following informations:

- name of the learning,
- country where the course will be held,
- required skill levels of the learning.
- granted skill levels.
- cost and duration.

With these informations, we could intuitively think that a "good learning" is a learning with small costs and a short duration. Under this agreement, we define a function which maps every learning to a real number between 0 and 1:

$$w : \mathcal{L} \longrightarrow [0,1] : L \longmapsto \gamma_1 \frac{C_L}{\max(\{C_{\tilde{L}} \mid \tilde{L} \in \mathcal{L}\})} + \gamma_2 \frac{D_L}{\max(\{D_{\tilde{L}} \mid \tilde{L} \in \mathcal{L}\})}$$

with $\gamma_1, \gamma_2 \in [0,1]$ and $\gamma_1 + \gamma_2 = 1$. Here, $C_{\tilde{L}}$ denotes the costs of a learning $\tilde{L}$ and $D_{\tilde{L}}$ the duration. The image $w(L)$ of a learning $L \in \mathcal{L}$ has to be interpreted as follows: The smaller the value $w(L)$ of a learning $L \in \mathcal{L}$, the smaller are the cost and duration of the learning L. If the cost and the duration of a learning should have the same weight, one has to choose $\gamma_1 = \gamma_2 = \frac{1}{2}$. If the cost should weigh more in balance than the duration, one has to choose $\gamma_1 > \gamma_2$. Otherwise, one has to choose $\gamma_1, \gamma_2 \in [0,1]$ such that $\gamma_2 > \gamma_1$. From now on, we can search for *good* learnings.

With that definition, we can consider the question concerning the existence of a method that computes the best learnings.

3.1.2. Problem Statement

The task of finding the best learnings for user u regarding to the job v can formulated as follows: *Find a sequence of learnings $L_1, L_2, \dots, L_n$ with minimal costs and durations such that the user u has the best updated profile regarding to the job v after attending the learnings $L_1, L_2, \dots, L_n$.*

Or in mathematical terms:

Find a finite sequence of learnings $(L_1, L_2, \dots, L_n) \in \mathcal{L}^n$ such that

$$r_{v,\lambda_1,\lambda_2}((\ell_{L_n} \circ \cdots \circ \ell_{L_1})(u)) = \underset{N \in \{1,\dots,|\mathcal{L}|\}}{\arg\max} \left(\{r_{v,\lambda_1,\lambda_2}((\ell_{\tilde{L_N}} \circ \cdots \circ \ell_{\tilde{L_1}})(u)) \mid (\tilde{L_1}, \dots, \tilde{L_N}) \in \mathcal{L}^N\}\right)$$

and $\sum_{i=1}^{n} w(L_i) \to \min$.

This problem often occurs in graph theory: Finding a shortest path between two nodes in a graph. It is known that this problem can be solved via Dijkstra's algorithm [4], for instance. So in our case, it is theoretical possible to compute the best learnings, i. e. to compute the best learning route for the user.

3.1.3. Graph Construction

Now, we want to construct the corresponding graph (V, E) in order to find the best learnings. Before we are constructing the graph, we firstly reduce the number of learnings which the use can attend. Namely, we consider only those learnings which bring the user nearer to given job v concerning those skills which the job requires *and* the user currently has:

$$\mathcal{L}_{\alpha_1} := \{L \in \mathcal{L} \mid 1 - p_v(L) \geq \alpha_2\}, \quad \alpha_1 \in (0,1).$$

Moreover, the learnings should improve those skills of the user which he doesn't have but the job, namely, as many as possible:

$$\mathcal{L}_{\alpha_1} := \left\{L \in \mathcal{L} \mid c\Big(\big(H_0(L_i)\big)_{i \in I_v}, \big(H_0(v_i - u_i)\big)_{i \in I_v}\Big) \geq \alpha_1\right\}, \quad \alpha_2 \in (0,1).$$

Our learnings set is defined as follows:

$$\mathcal{L}_{\alpha_1,\alpha_2} := \mathcal{L}_{\alpha_1} \cap \mathcal{L}_{\alpha_2}.$$

1. Our start node is the user u himself. Define

$$u_{\text{end}} := \underset{N \in \{1,\dots,|\mathcal{L}_{\alpha_1,\alpha_2}|\}}{\arg\max} \left(\{r_{v,\lambda_1,\lambda_2}((\ell_{\tilde{L_N}} \circ \cdots \circ \ell_{\tilde{L_1}})(u)) \mid (\tilde{L_1},\dots,\tilde{L_N}) \in \mathcal{L}^N_{\alpha_1,\alpha_2}\}\right)$$

2. Let $L \in \mathcal{L}_{\alpha_1,\alpha_2}$ be a learning which the u can attend. If $r_{v,\lambda_1,\lambda_2}(\ell_L(u)) < r_{v,\lambda_1,\lambda_2}(u_{\text{end}})$, then apply learning L on the user u. Denote all applied learnings by $\mathcal{L}_{\alpha_1,\alpha_2,p}$. Compute the corresponding weights of the edges $w(L)$ for all $L \in \mathcal{L}_{\alpha_1,\alpha_2,p}$.
3. Repeat the process from step 2 by replacing u with $\ell_L(u)$ for all $L \in \mathcal{L}_{\alpha_1,\alpha_2,p}$. If no learnings are applied on the new user, then the graph (V,E) is finally constructed.

By applying Dijkstra's algorithm we can find the shortest path between u and some arbitrary $\tilde{u} \in \{\tilde{u} \in V \mid r_{v,\lambda_1,\lambda_2}(\tilde{u}) = r_{v,\lambda_1,\lambda_2}(u_{\text{end}})\}$. Finally, we get the best learning route for the user u with respect to the job v. Algorithm 1 shows the graph construction in slightly changed form given in [2]. It turned out that this algorithm could not be applied on the learning data, since the graph construction alone takes a lot of calculation time. In next section we present a more faster algorithm which is applicable. This algorithm is currently used for the route planner on the OpenSKIMR platform.

3.2. New Algorithm

In the following, we will introduce a new algorithm that is much faster than the previous algorithm. Again, let u be the user and v the chosen job of the user. We denote the set of skill indices of the user as $I_u := \{i \in \{1,2,\dots,n\} \mid u_i \neq 0\}$ and $I_v = \{i \in \{1,2,\dots,n\} \mid v_i \neq 0\} = \{i_1,\dots,i_k\}$ as the set of skill indices of the job, respectively. The main goal is to compute a learning route which is fast to compute and closes all skill gaps of the user u with respect to job v. We therefore have to assume that such learnings exist and proceed as follows:

1. Let i_ℓ, $1 \leq \ell \leq k$ be a skill index of the job where the user has a skill gap. Search for all learnings in $\mathcal{L}$ that the user can attend and improve the skill s_{i_ℓ}. Denote the new learning set by $\mathcal{L}_s$ and remove all learnings where the granted skill level of s_{i_ℓ} is higher than the skill level of the job.
2. Sort $\mathcal{L}_s$ by
   (i) skill level of s_{i_ℓ} (descending)
   (ii) number of job skills that are improved according to the rating function defined in Algorithm 2 (descending)
   (iii) weight of cost and duration by the function w (ascending)
3. Choose first learning L of the new sorted list $\mathcal{L}_s$, append it to the learning route and apply it on the user u.

Repeat step 1, 2 and 3 by replacing skill s_{i_ℓ} with another job skill and the user u with $\ell_L(u)$ until the user fulfills the requirements of the job v. Algorithm 3 shows this process in detail.

Algorithm 1: Route Planner [2]

```
Data: $\mathbf{u}$, $\mathbf{v}$, $r_v, \mathcal{L}, \ell_L, d_{\max}, \alpha, \beta$
Result: graph $G = (V, E)$, shortest path and $r_{\mathbf{v}}(\mathbf{u}^*_{\mathbf{v},d_{\max}})$
1  $m \leftarrow 0$;                                                      // initializing
2  $V_{\text{seen}} \leftarrow \varnothing$;
3  $V^{(m)} \leftarrow \varnothing$;
4  $V^{(m+1)} \leftarrow \varnothing$;
5  $E \leftarrow \varnothing$;
6  $\mathcal{L}_{\alpha_1} \leftarrow \{L \in \mathcal{L} \mid c((\mathbf{x}_L, H_0(\mathbf{v}-\mathbf{u})) \geq \alpha_1\}$;    // preprocessing
7  $\mathcal{L}_{\alpha_2} \leftarrow \{L \in \mathcal{L} \mid 1 - p_{\mathbf{x}_{R_{L,2}}}(\mathbf{v}) \geq \alpha_2\}$;
8  $\mathcal{L} \leftarrow \mathcal{L}_{\alpha_1} \cap \mathcal{L}_{\alpha_2}$
9  while $V^{(n+1)} \setminus V^{(n)} \neq \varnothing \wedge n < d_{\max}$ do
10     $V^{(n)} \leftarrow V^{(n+1)}$;
11     $V_{\text{temporary}} \leftarrow V^{(n+1)}$;
12     for $\tilde{u} \in V^{(n+1)} \setminus V_{seen}$ do
13         for $L \in \mathcal{L}$ do
14             $\tilde{u}_L \leftarrow \ell_L(\tilde{u})$;
15             if $r_{v,\lambda_1,\lambda_2}(\tilde{u}_L)/r_{v,\lambda_1,\lambda_2}(\tilde{u}) > 1 + \beta$ then
16                 $V^{(n+1)} \leftarrow V^{(n+1)} \cup \{\tilde{u}_L\}$;
17                 $V_{\text{temporary}} \leftarrow V_{\text{temporary}} \cup \{\tilde{u}_L\}$;
18             end
19         end
20         $V_{\text{seen}} \leftarrow V_{\text{seen}} \cup \{\tilde{u}_L\}$
21     end
22     $V^{(n+1)} \leftarrow V_{\text{temporary}}$;
23     $n \leftarrow n + 1$;
24 end
25 $V \leftarrow V^{(n)}$;
26 $G \leftarrow (V, E)$;
27 $\mathbf{u}^*_{\mathbf{v},d_{\max}} \leftarrow \operatorname{argmax}_{\tilde{u} \in V} r_{\mathbf{v}}(\tilde{u})$;
28 Apply Dijkstra's Algorithm with initial node $u$ and destination node $u^*_{v,d_{\max}}$;
29 return $G = (V, E)$, shortest path and score $r_{\boldsymbol{v},\lambda_1,\lambda_2}(\boldsymbol{u}^*_{\boldsymbol{v},d_{\max}})$
```

Algorithm 2: Skills Improved

```
Data: learning ID $\ell \in \mathbb{N}$, job ID $j \in \mathbb{N}$
Result: score $\sigma \in \mathbb{R}$
$\sigma \leftarrow 0$;                                   // initialize score
for $ns$ in job $j$ skills do
    if learning $\ell$ improves skills $ns$ then
        $\ell\ell \leftarrow$ skill $ns$ learning level granted;
        $j\ell \leftarrow$ skill $ns$ job level required;
        if $\ell\ell < j\ell$ then
            $score \leftarrow score + (j\ell - \ell\ell)/j\ell$
        end
    end
    else
        $score \leftarrow score + 1$
    end
end
return $score/(\#\, job\ j\ skills)$
```

Algorithm 3: Route Planner 2.0

```
Data: user skill vector $\mathbf{u} \in \mathbb{Z}^n$, job ID $j \in \mathbb{N}$, max. iterations $maxIt \in \mathbb{N}$
Result: learning Route $\mathbf{L} \in \mathbb{Z}^m$, improved skill vector $\hat{\mathbf{u}} \in \mathbb{Z}^n$
$\hat{u} \leftarrow u$;                                  // copy start vector
$\mathbf{L} \leftarrow [\,]$;                            // initialize learning route
$\ell \leftarrow [\,]$;                                  // initialize possible learnings
$count \leftarrow 0$;
while learnings applied and $count$ < $maxIt$ + $1$ do
    for $ns$ in job $j$ skills do
        if user has skill gap then
            for $n\ell$ in learnings do
                if user can attend $n\ell$ and $n\ell$ improves $ns$ then
                    $\ell \leftarrow \ell + n\ell$
                end
            end
            if $\ell \neq \varnothing$ then
                maxLevel $\leftarrow$ maximum of granted skill levels of ns in $\ell$;
                if $maxLevel >$ level of ns in job then
                    maxLevel $\leftarrow$ level of ns in job;
                end
                $\tilde{\ell} \leftarrow \ell$ with granted level of ns less or equal maxLevel;
                sort $\tilde{\ell}$ by granted level of ns, most skills improved and weight;
                $\hat{u} \leftarrow$ first learning $\ell_0$ of $\tilde{\ell}$ applied to $\hat{u}$;
                $\mathbf{L} \leftarrow \mathbf{L} + \ell_0$
            end
        end
    end
    $count \leftarrow count + 1$
end
return $L$, $\hat{u}$
```

4. Comparison of Algorithms

In this section we list the differences between the first version, namely Algorithm 1, and Algorithm 3, the one we are currently working with on the OpenSKIMR platform.

4.3. Runtime

The biggest advantage of the revised version in comparison with the first algorithm is its greatly reduced runtime. Since this one is constructing a graph of learning routes that is taking into account all possible permutation of learnings, runtime is increasing like the factorial $n!$, see Figure 2. The problem is that quite often two learnings ℓ_1, ℓ_2 are not independent from another meaning that

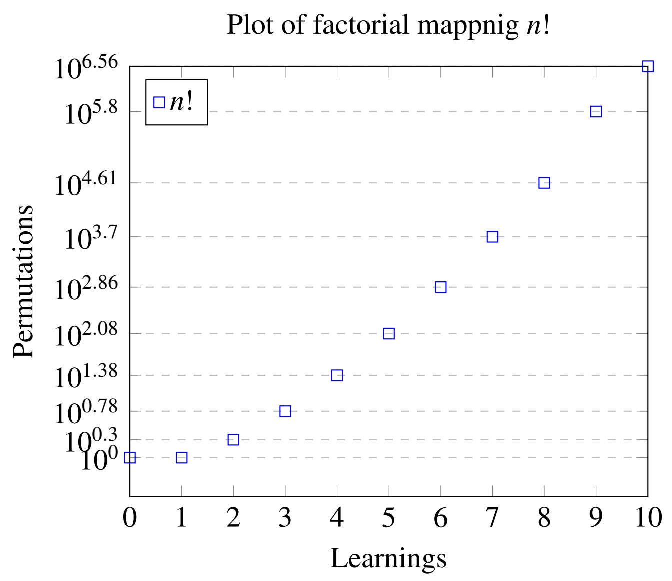

FIGURE 2 The reason why the first algorithm is taking so much time is visible in this plot. While the number of learnings in a route is increasing linearly, the number of possible combinations plotted on a logarithmic y-axis is increasing at an immense rate.

you need to complete ℓ_1 to fulfill all the entry skill levels needed for ℓ_2. That is why all permutations of learnings in some set need to be considered. The improved version of the algorithm is improving skills in a greedy fashion. We iterate over all the skills required by some job $v \in \mathcal{V}$ and immediately partake of a learning if the user needs to improve this skill accordingly. Thus, we save ourselves the trouble of considering all permutations since we always attach a learning as soon as it closes any skill gap.

4.4. Closing of Skill Gaps

One request of the European Union was to construct an algorithm that can guarantee a user of the OpenSKIMR platform a route to his job of choice. Thus, we needed to construct a route planner that is closing all skill gaps with respect to some user $\mathbf{u}$ and job offer $\mathbf{v}$, cf. the section above where we stated the problem. The novel algorithm is living up to this expectation as it does not cancel as long as either the maximum number of iterations is not reached or new learnings are still available. This constitutes another advantage as opposed to Algorithm 1, which is merely constructing a learning path that is similar to the job offer with respect to the cosine similarity. However, we have to note that the much improved runtime is coming to the disfavor of optimization regarding cost and duration owed to the greedy approach.

5. References

[1] E. Commission, “European Classification of Skills/Competences, Qualifications and Occupations.” Booklet, Publications Office of the European Union: Luxembourg, 2013.

[2] A. Kofler and M. Prast, “OpenSKIMR A Job and Learning Platform,” *CS & IT-CSCP*, pp. 95–106, Jan. 2017.

[3] E. Commission, “ESCO Strategic framework - European Skills, Competences, Qualifications and Occupations,” July 2017.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, “Dijkstra’s algorithm,” in *Introduction to Algorithms 2nd edition*, ch. 24, pp. 595–599, MIT Press, 2009.