Semester Project - Building a graph of skills from job offers

Pierre Schutz
{name}.{surname}@epfl.ch
Center for Digital Education
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

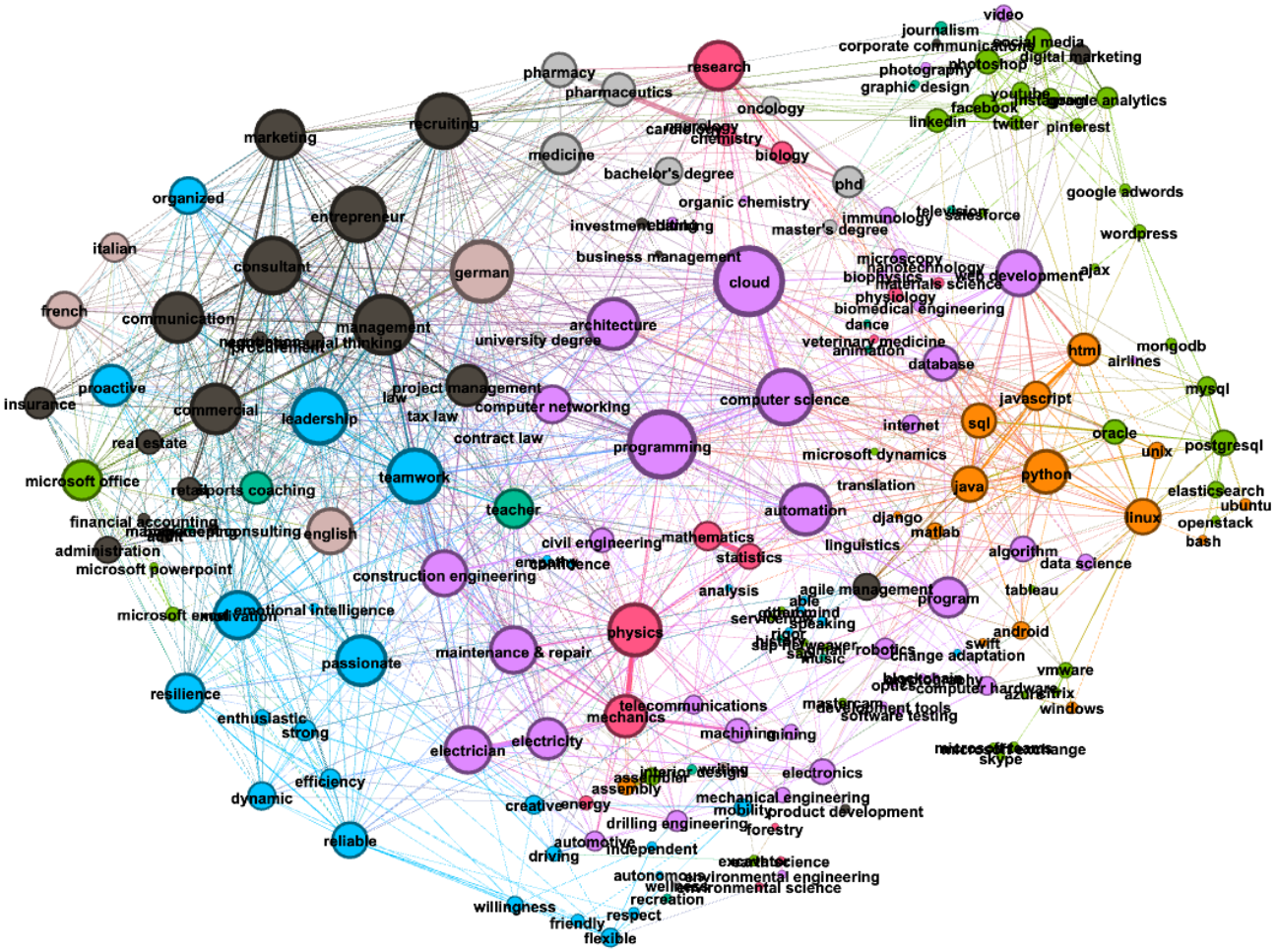

Figure 1: Top 10% of edges in normalized co-occurrences graph

ABSTRACT

A critical role of higher education is the ability to prepare students for the job market. Through their degree, the courses aim to help the student build a set of skills that will make them match the requirements of the market and be a valuable element for a company. From the school perspective, understanding the skills required from the job market is key to provide adapted courses and teachings for the students. This work aims to analyze the swiss job market by mining job offers data online and building a graph of skills from those offers. Then, we want to evaluate how different EPFL courses succeed in helping the student learn the appropriate skills. This work aims to provide metrics that help to select and tailor the courses to provide the students with adapted study programs.

1 INTRODUCTION

In the goal of understanding the job market skills requirements, and how these skills relate to the teachings given at EPFL, we aim to build a graph that links the job market skills with the concepts taught at EPFL from the EPFL Graph [1].

Main job categories
Admin / HR / Consulting / CEO
Banking / Insurance
Catering / Food / Tourism
Chemical / Pharma / Biotechnology
Construction / Architecture / Engineer
Electronics / Engineering / Watches
Finance / Trusts / Real Estate
Graphic / Art / Typography / Printing
Information Technology / Telecom
Machine / Plant / Engine / Manufacturing
Medicine / Care / Therapy
Marketing / Communications / Editorial
Public / Administration / Education / Social
Purchasing / Logistics / Trading
Sales / Customer Service / Administration
Sport / Spas / Wellness / Culture
Surveillance / Police / Customs / Rescue
Vehicles / Craft / Warehouse / Transport

Table 1: Job offers website main categories

Through this project, we build a dataset of 64000 job offers. We also build a dataset of skills with equivalent Wikipedia concepts and related keywords. We then compute a graph of skills and iteratively improve and analyze it.

2 IMPLEMENTATION

This project was divided into 3 main milestones. The first was to explore the 'Jobs.ch' website and use web scrapping to build a job offers dataset. The second step was to define and build a skills dataset that is connected to Wikipedia articles (corresponding to EPFL graph concepts [1]). Finally, use both datasets to build a graph of skills and analyze how the job market skills correspond to EPFL courses related concepts.

The different components were all implemented using Python3 programming language [2] and Jupyter notebooks (IPython [3]) to separate the processing tasks, and visualize the data at different steps.

2.1 Job offers dataset

The first task of the project consisted of mining job offers description text. We then use the description to find the skills required by the job market. To get those jobs description, we perform web scraping, data cleaning, and translation as preprocessing steps for our job offers dataset.

2.1.1 Website structure. The first task to mine data from the job offer's website was to understand its structure. How are the job offers stored? How to access them using URLs or interactions? And how to extract the content from them?

The first observation we made was the job categories hierarchy. These categories are used in the website to search the jobs. The Table. 1 present the first level of categories.

Then, more specific categories can be accessed from the website's jobs filtering option or directly via an id in the URL. We observed that, for each job category page, the website has a list on the left with 24 job ids, and up to 100 list pages. We note that for all categories that contain more than 2400 jobs, those jobs ids are not accessible from the category page.

Finally, the website enables access to a job offer page given a job id (either a hash or an integer id). On the job offer page, we can access three tabs with job's information:

- **Job description** contains an HTML formatted job description. The is the part we are most interested in.
- **Info** contains the job offer's dates, related categories, position type, and time availability requirements.
- **Company** contains details about the company posting the job, such as rating, size, industry, website link, or location.

2.1.2 Tools and techniques. To mine the data from the website, we used multiple python tools that we describe in this section. Those tools helped us change dynamically our IP address, extract specific HTML blocks or text from an HTML page, and navigate on the website.

- **Tor** enables us to protect our IP address or avoid being blocked after too many requests, we used the Tor proxy [4]. This enables us to query the website through a circuit (list of tor nodes from which our network packet goes through between your computer and the website's server, to ensure anonymity), and change our IP address regularly. Our implementation followed this tutorial [5].
- **Selenium WebDriver** helped us interact with the website from a browser interface (e.g. navigate into the website, click on buttons/tabs) [6]. This helped us clicking on the job offer's page tabs in an automated manner to access the different information.
- **Requests** python package [7] provide easy-to-use methods to build HTTP requests. This was used to download the HTML pages for job categories, and job ids.
- **BeautifulSoup** python package [8] enables to parse HTML pages and retrieve HTML blocks to find page's content easily. It was used to extract the job description, ids, or categories from the website using the HTML class of containers with the searched information.

2.1.3 Web scrapping. After exploring the website structure and testing the different tools on the website, we started to mine the data. At the time of the scrapping (March 2021), the website contained 68000 job offers.

The scrapping of the job offers was completed in three steps. First, getting the job categories. These categories were then used to extract lists of job ids. Finally, the job ids were used to get the job offers data.

Job category. To find the job categories, we observe that they were represented using a three-digit index. Using a counter, we tried to query the website with id values ranging between 0 and 300 using the following URL: https://HOST/en/vacancies/?category=C_ID. We observed that the website contains category ids from 97 to 261 (165 categories). The main categories listed in Table. 1 correspond to the first 18 ids. These calls were performed using the Tor proxy to protect our local IP.

Job id. On the category page, we can find a list of corresponding jobs on the left. This list contains up to 24 jobs and continues up to 100 pages. These pages can be accessed directly using a URL: https://HOST/en/vacancies/?category=C_ID&page=P_ID (Using the URL enabled us to avoid needing to click on the "next page" button and lose time during loading).

From the list, we extracted the job ids of all elements and iterated over the pages to get all categories' jobs. Some categories contained more than 2400 elements and the last could not be accessed using this technique. Nevertheless, the job categories correspond to a three-level tree. Therefore, most jobs were associated with several categories and could be accessed through categories with fewer jobs.

Using this technique, we extracted 110k ids total and 64k jobs after removing duplicates. This corresponded to 94% of all available jobs.

Job offers. After collecting all job ids into a Pandas [9] dataframe, we needed to download each job offer HTML page. Using the URL: https://HOST/en/vacancies/detail/JOB_ID/, we accessed directly the job page given a job id.

At first, we used Selenium to navigate between the page's tabs by simulating mouse clicks. We then extracted the data from each HTML block. This process required to load three HTML pages for each job offer and wait between the click for the page to load in the Selenium web browser.

After looking more carefully into the job offer page's source code, we observed that all the job's data were stored into the `__INIT__` variable in a script block. This javascript variable contained all job offer's data displayed in the different tabs. This was a great improvement and enabled us to speed up the mining process by a factor of 3.

All these HTML files can be found in the project's `html.nosync` directory.

For each job id, we saved the corresponding HTML file in case it was needed for later use.

With an average time per request of 1.2 seconds, the total time to extract the HTML documents was $64000 \times 1.2s = 21.3h$.

2.1.4 Data cleaning . Once we downloaded the job offers' HTML file, we needed to process them to build a smaller and easy-to-use dataset. Also, Switzerland being a multi-languages country, we needed to translate the job offers into English.

Compute CSV dataset. Using BeautifulSoup, we extracted the `__INIT__` variable the javascript block in the job's HTML. We then parsed this object and added it as a row of our pandas dataframe (This is the `jobs_all.csv` dataset). We present in Table. 2 some of the interesting fields in this job offer object. The `text` field contains the HTML formatted job description that we will use later. We, therefore, build another dataset containing only job description and id (The `jobs_id_description.csv` dataset).

Translation. The job offers are written in numerous languages due to Switzerland's diversity. We show in Table. 3 the distribution of jobs per language. We need to translate the job offers into English to have a ready-to-use dataset.

First, we used the `langdetect` library [10] to get the language for each offer and build the table discussed above.

Key	Value
id	000005639743c5ce05c5ddfef0061cdef...
title	Regulatory Affairs Manager
text	Regulatory Affairs Manager ...
place	Rotkreuz
company	F. Hoffmann-La Roche AG
isActive	True
publicationDate	2020-12-18T20:35:05+01:00
categories	/97/100/127
expiredOn	2021-03-16
salaryFrom	NaN
salaryTo	NaN
popularKeywords	'Strategy', 'Legal', 'Supply Chain', ...

Table 2: Job offer content example

Language	Count
German	51205
English	7070
French	5953
Italian	299
other	98

Table 3: Number of job offers per language

Then, to translate the job offers, the initial plan was to use the lab's google translate API access. For the translation of our dataset (17.6 millions words, 139.9 millions characters), the API pricing [11] is 20$/million characters, corresponding to a total of $140 * 20\$ = 2800\$$. This was not affordable, so we searched for solutions to drastically reduce the number of characters to translate.

The initial solution we came up with was to translate sentences individually the sentences for each offer. By doing so, we could avoid translating the same sentence multiple times and reduce the total number of characters to translate. To do so, we computed a hash for each unique sentence, and associated each job id with a list of hash (See `description_hash.csv` dataset). We also build a dataset containing all sentences with their matching hash (See `id_hash_sent.csv` dataset). The resulting dataset then contained 441000 unique sentences and 105.9 millions of characters, corresponding to a reduction of about 24% of the total number of characters. The API cost was still 2120$ with standard pricing. These results were interesting but far from enough.

We then searched for alternatives such as offline translation using a pre-trained model T5 from Google [12], or corpus-based translation.

Our final solution was to use free python libraries that query the Google API to perform the translations (such as googletrans, translate-api, or `google_trans_new`). These libraries were using the free trial of the API and enable us to translate the job offers by chunks of 500000 characters. To bypass the threshold, we used a VPN to regularly update the IP each time the API was blocked with too many requests (we didn't use the Tor proxy because the Google API blocked the Tor IPs). This helped us build the final cleaned job offers dataset (See `jobs_id_description_translated.csv`).

With an average of 1.5 seconds per request, the total translation was computed in 26.7 hours.

2.2 Skills dataset

The second step for this project was to build a skills dataset. This dataset aims to define skills and connect them to Wikipedia concepts. Using this dataset, we will later be able to match job descriptions with skills and then with concepts taught at EPFL.

2.2.1 Skills definition. Building the skills dataset was an iterative process. We used multiple datasets found online and tested the results in the graph. We describe here the steps until our final approach.

Finding dataset online. Initially, we searched for already made and maintained dataset online. LinkedIn skills are very popular and widely used, so our initial goal was to find a dataset containing those skills. Here is a list of the dataset we found.

- LinkedIn & WorldBank 1
- LinkedIn 2
- LinkedIn 3
- LinkedIn 4

Nevertheless, datasets 2, 3, and 4 were noisy because they were built by users: numerous values were not corresponding to skills, others were acronyms without context. They also were of big size (more than 10000 items), so finding skills into them manually was hard.

Initial dataset. For the aforementioned reasons, we decided to proceed by creating manually the skills dataset. By doing so, we would be able to find the most frequent terms used in our data and select the skills in those. We used dataset 1 (LinkedIn and WorldBank) as a basis as it was clean and relatively small (around 100 skills). Building the skills dataset was an iterative process. We tried multiple datasets to build the graph until our final approach.

Add skills from keywords. The first iteration goal was to add skills that are specific to our job description dataset. Indeed, the WorldBank dataset was very general and did not describe well our dataset. To fix this, we used the keywords generated by the different NLP techniques (see 2.3.1) to find the most frequent terms that were referring to skills.

To perform this task, we faced a major challenge: defining the notion of skill. Indeed, choosing between what is a skill and what is not was a difficult task because of the ambiguous and large definition of skill. Wikipedia defines a skill as: "A skill is the learned ability to perform an action with determined results with good execution often within a given amount of time, energy, or both. Skills can often be divided into domain-general and domain-specific skills". From this definition and the different skills seen in the online datasets, we distinguished four types of skills:

- Human Abilities: Abilities applied to the work market. This is represented in our dataset by the "Soft" (any non-domain specific skill), and "Language" (multiple languages writing and speaking abilities) category of our dataset.
- Domain-Specific knowledge: This corresponds to knowledge in a domain (such as "Software development", or "Pharmaceutics") and is represented in our dataset by "Business", "Law", "Health", "Science", "Engineering", and "Humanities" categories.
- Technology usage skill: The ability to use specific software (e.g. Microsoft Excel) or programming language widely used in the industry. This is represented in the dataset by the "Software" and "Programming" categories.
- Diploma: Even though this is not specifically a skill, this is something that is often required by job offers. Therefore, we consider it as a skill as it discriminates against the candidates for a job ("Diploma" category in our dataset).

Using these categories, we added all skills terms found in the 1000 most frequent keywords using multiple keyword extraction techniques (keyBERT, TF-IDF, and NLTK tokenizer).

Final approach. The above approach was not satisfactory for multiple reasons. First, some skills terms were too broad and were matching with too many job offers (e.g. "Engineering", "Science", "Software", or "Innovative", see Figure. 2). We removed all those skills from the final dataset. Then, the second issue was the lack of "Software" and "Programming" categories skills. Indeed, those skills constitute some of the most important skills for engineers, and many are used in EPFL courses. To improve this, we searched for Technology related skills and found an adapted dataset in the O*NET database. Matching job offers keywords with the dataset helped add 40 more skills. Our final skills dataset contains 313 skills.

2.2.2 Wikipedia concepts. To match the job skills with the EPFL courses, but also to have a standard representation of the skills, we used Wikipedia articles and their ids to identify the skills. Indeed, the EPFL graph uses Wikipedia articles as concepts and associates courses with them. Therefore, while building our skills dataset, we wanted to match those skills with Wikipedia articles.

To do so, we implemented a semi-automated program using the wikipedia-api [13] python package. This package enables to query of the Wikipedia website pages and search functionalities.

Using our skills dataset, we used the wikipedia-api package to search for an article using the skill name. This search leads to 4 possible outcomes:

- Page found, right Wikipedia page: This is the best case, where nothing needs to be done and the associated Wikipedia concept is automatically found for a given skill.
- Page found, wrong Wikipedia page: We need manually check that the page is wrong, and then search for a more appropriate page on Wikipedia and add it manually.
- Disambiguation page: In this case, our program detects it and proposes to manually select in the list of proposed pages the most adapted one. If none match, the program tag handles it as "Page not found".
- Page not found: We have to manually search on Wikipedia using similar terms and concepts.

Using this process enabled us to match most of the pages automatically. Nevertheless, due to disambiguation, invalid page, or page not found cases, a manual review and errors handling was required to associate all skills.

Generally, there might not be a one-to-one mapping between skills and Wikipedia articles. For instance, numerous human qualities do not have an associated Wikipedia article but only a dictionary definition (e.g. Willingness or Friendliness). In these cases, we manually associated a Wikipedia article that is close to the skill. Also, some skills can have more than one concept associated with them. Due to time constraints, we remained with a one-to-one mapping in this project, but this could be implemented as an improvement.

2.2.3 Skills' keywords. For each skill, we manually added a list of keywords with synonyms to enable matching the skill with a bigger set of job description keywords. We later refer to those as `skill_keywords`, and will discuss their use in more detail in the section 2.3.2.

2.3 Graph of skills

Once the job offer and skills datasets were ready, the final goal was to build a graph of skills. This graph aims to represent the relationships between different skills using the job market information. Therefore, we must use the job offers dataset to associate the different skills defined in our skills dataset. Once we associated the skills together in a graph, we want to match them with the EPFL courses using the `"Edges_N_Course_N_Concept_T_Semiauto"` collection in ArangoDB.

The different steps to build the graph are the following:

- 1. Extract important keywords from job descriptions (later called `description_keywords`).
- 2. Build a skills dataset, map to Wikipedia concepts, and define a set of keywords associated with each skill (later called `skill_keywords`).
- 3. Match the `description_keywords` and the `skill_keywords` to compute a list of skills for each job offer.
- 4. Compute a similarity metric using the job offers to build the edges of the graph.

2.3.1 Keywords extraction. To extract meaningful keywords from the jobs' descriptions, multiple tools and algorithms were tested. We discuss here the different methods, and the chosen approach.

Wikify. The original implementation idea was to used the lab's Wikify API [14] to associate directly the jobs' description with Wikipedia concepts and then select a subset of those concepts that corresponds to skills. This approach provided interesting results as we can see in Figure. 3, where numerous of the most frequent terms are skills. Nevertheless, we faced issues using the API. First, for some job descriptions with a short text, the API didn't return any results (this was later fixed by adding a boost parameter to the query). Also, the concepts returned were something far from the keyword the associations were wrong (requiring to filter the results using the graph score parameter). Then, the extraction process was slow as the API require up to 7 seconds per request. Finally, due to a technical issue with the API though the course of this project making it unavailable, this option was abandoned.

NLTK. A naive approach used the Natural Language Toolkit (NLTK) [15] to extract words from the text using the NLTK tokenizer. It then computes the term frequencies and inverse document frequency (TF-IDF). We built a histogram of keywords by tokenizing the job descriptions using NLTK, and removing stop words using NLTK's 'english' stop words dataset. The resulting histograms show the most frequent keywords in Figure. 4. This solution was interesting by its simplicity and computing efficiency but did not make use of the terms meaning to compute keywords. It and also gives a very simplistic notion of importance within a document (frequency), and returns one-word keywords.

SpaCy. A slightly more advanced approach used the Spacy python NLP library [16] to extract keywords from the text. We used Spacy's default NLP model `"en_core_web_sm"` to parse a subset of the job descriptions. Spacy 'nlp' object enables to extract entities (or ents) corresponding to concepts from the text. We used those extracted entities to build a histogram of concepts from the text (see Figure. 5). This technique provides a better interpretation of words in the text and return keywords that correspond to concepts, but do not provide a metric to rank those keywords by relevance.

KeyBERT. The final approach we chose makes use of a recent algorithm working on top of BERT (a state-of-the-art transformers model [17]) called keyBERT. Bert model advantage is to analyze words within sentences and compute word embeddings that are dependant on the sentence structure. The keyword extraction algorithm named keyBERT [18] is a basic keyword extraction algorithm implementation that aims to outperform Rake by taking advantage of the BERT embedding and provide a list of keywords of length k with the associated score. Here is a short description of the process from the project's GitHub: "First, document embeddings are extracted with BERT to get a document-level representation. Then, word embeddings are extracted for N-gram words/phrases. Finally, we use cosine similarity to find the words/phrases that are the most similar to the document. The most similar words could then be identified as the words that best describe the entire document.".

In our implementation of the job description keywords extraction, we chose to select a length of both 1 and 2 words keywords and extracted the 30 most important for each. This leads to getting constantly 60 keywords for each job offer.

The keyword extraction required on average 2.4 seconds per job offer, for a total of 42.5 hours for the full dataset. The `job_id_description_bertify.csv` dataset contains the extract keywords along with the job id and descriptions.

2.3.2 Keywords matching: mapping description to skills. After extracting the `description_keywords` and building the skills dataset with `skill_keywords`, we wanted to use those keywords to associate a list of skills to a job description.

GloVe. The initial goal was to take advantage of a machine learning model and build a spatial representation of keywords using some word embeddings. Indeed, using this representation, we could use space metrics such as euclidean distance to match keywords together. Nevertheless, after the first test with GloVe model (100 dimensions, 50 million words) [19] that did not give satisfying results (too many keywords with no direct meaning were matched with skills, and it was slow to load and compute the embeddings), we decided to go with a simpler method.

Exact matching. The baseline approach we implemented consisted into a direct matching between `description_keywords` and `skill_keywords`. This means that a skill was matched and added to a job offer if at least one of the `skill_keywords` matched at least one of the `description_keywords`. This approach was simple but failed to match together different forms of the same word (conjugation, plural). The next method fixes this issue.

Porter Stemming. To fix the aforementioned issue, we added the Porter Stemming algorithm [20] from the NLTK library to compute the root of keywords before matching them. This method is still less generic than comparing to word embedding and requires to define precisely the set of keywords for a given skill, but is fast and to not require a model. The stemming enabled to match all keyword forms (example: entrepreneur, entrepreneurs, entrepreneurship, entrepreneurial). The Figure. 8 shows the improvement using Porter Stemming.

The resulting dataset containing the job and associated skills is `jobs_id_skills.csv`. We have on average 4.3 skills per job (this is low, but can be explained by our small skills dataset, and restrictive matching method). To observe how our matching performs, we tested it with a few jobs to see how the related jobs title are similar (see Figure. 7).

2.3.3 Skills similarity. After extracting keywords from the job offers and matching them with skills, we had to find a metric that describes well the relationship between keywords in the job offers. This metric will be used to associate a weight to the edge between two skills in the graph.

Co-occurrences. The first metric we implemented was to count the total number co-occurrences of skills in all job offers. Therefore, in our graph, the edge between two skills corresponds to the number of times these two skills occur together in a job offer. The approach succeeds to highlight skills that are connected, but give advantage to skills that appear in most job offers even if they are not strongly connected with a specific set of skills.

Normalized co-occurrence. This second approach aims to prevent having the most occurring skills like "German" have a high rating because of their high frequency in the dataset. Now, for a pair of skills, the score corresponds to the number of co-occurrences divided by the average number of occurrences of each two skills.

$$w(s1, s2) = \frac{2 \times (\#s1 \& s2)}{\#s1 + \#s2} \tag{1}$$

With $w(s1, s2)$ the weight between skills $s1$ and $s2$, $\#s1\&s2$ the number of jobs where $s1$ and $s2$ are present, and $\#s_i$ the number of jobs where s_i is present.

2.3.4 EPFL courses analysis. The last task consisted of computing EPFL courses metrics using the graph of skills thanks to collections in ArangoDB that map the EPFL courses to Wikipedia concepts.

After loading the courses, nodes from courses to concepts, and the concepts, we tried to match those concepts with the job market using the skills. The goal was to compute centrality metrics for the different course concepts to see how different courses tackle very specific areas or are bridging between multiple domains. We were also interested to see which software of programming language is taught by each course and how much those are then needed for the job market (Figure. 9 shows popular software and programming languages in the job market). Nevertheless, none of the skills in the skills graph were present in the courses related concepts. These results happened late in the project, we did not have time to solve this issue. We discuss this in the section 4.

3 RESULTS

In this section, we present the different results obtained in the project starting from the dataset generated and then discussing the generated graphs and visualizations.

3.1 Datasets

At the end of each processing step, we computed and saved a dataset to facilitate future work and reproducibility. We summarise here the different datasets generated:

- Job categories (id, name, Wikipedia concepts): `csv.nosync/categories_id_name_wiki.csv`
- All job ids: `csv.nosync/jobs_id.csv`
- Mined HTML pages: `html.nosync/`
- Job offers full data: `csv.nosync/jobs_all.csv`
- Cleaned job id and description: `csv.nosync/jobs_id_description.csv`
- Translated job id and description: `csv.nosync/jobs_id_description_translated.csv`
- Job id, description, and keyBERT keywords: `csv.nosync/jobs_id_description_bertify.csv`
- Job id and associated skills: `csv.nosync/jobs_id_skills.csv`
- Skills with Wikipedia concepts and keywords: `csv.nosync/final_skills.csv`

Other datasets:

- Job offers with NLTK extraction terms: `csv.nosync/jobs_id_description_nltkfy.csv`
- Skills datasets found online: `csv.nosync/skills/`
- Intermediary translation datasets (french, italian, german): `csv.nosync/translation/`
- Job description sentences hashing: `csv.nosync/hashing/`

3.2 Graph of skills

Two different graphs of skills were built during this project, corresponding to the two skills similarity metrics used (co-occurrences without and with normalization). The graph with normalization provides better results and corresponds to our final result (where the other is used to compare). The resulting graphs are heavily connected as they only require one co-occurrence between two skills to build an edge. To improve the readability in the graph visualizations and highlight the strong links between certain nodes, we built a filtered version of the graphs that only contains the most important edges between the skills. These filtered versions are used for the visualizations.

Here is are the graphs and visuasualizations created:
Graph:

- `default.gefx`: Graph with co-occurrences weights

- `default_filtered.gexf`: Graph with co-occurrences weights filtered (edges with at least weight = 100)
- `norm.gexf`: Graph with normalized co-occurrences weights
- `norm_10.gexf`: Graph with normalized co-occurrences weights with only the edges of 10% highest weights
- `norm_1.gexf`: Graph with normalized co-occurrences weights with only the edges of 1% highest weights

Visualizations:

- `visualization_default_filtered.gephi`: Visualization of graph for co-occurrences filtered (edges with at least a weight of 100).
- `visualization_norm.gephi`: Visualization of graphs for normalized co-occurrences with the Top 10% and Top 1% of edges.

In the different visualizations, the node size corresponds to their degree, and the color corresponds to the skill category.

From the visualizations, we can observe how normalizing the weights improved our graph. Indeed, in the co-occurrences weights filtered graph (see Figure. 10), we observe the most frequent skills to have the highest degree and have high weights with many nodes (e.g. "German", "Management", or "Teamwork"). In the normalized graph (see Figure. 1 and Figure. 16), this is different. More nodes have higher degree, and for example keywords related to computer science that are strongly connected to their domain have a high degree (e.g. "Programming", "Python", or "Cloud"). This technique achieves the expected result having a high edge score between rare but strongly connected skills.

We can observe how "Python" and "Computer Hardware" in `norm_10` graph or "Java" in `norm_1` graph are well connected to skill in their domains (see Figures. 12,14, and 17). Also, we see how a more general term like "Computer Science" (in `norm_10`) connects nodes from it's domain area and but also outside "Leadership" or "Entrepreneur" (see Figure. 13). On the opposite, in the graph `default_filtered`, the compute science node has only few connections to nodes with high degree.

In general, this graph of skills succeeds to show relationships between similar skills in the job market. Nevertheless, the graph is still heavily connected, and filtering nodes based on weight required to see clusters appear. Also, the small size of the skills dataset makes it hard to observe clearly defined and distinct domain areas. Indeed, our approach fails to get all the information from the job offers, as from the 60 keywords extracted, we only get on average 4.3 skills per job.

4 DISCUSSION

Through the realization of this project, numerous design and implementation choices were made, some because of technical difficulties, time, or resource constraints. In this section, we discuss the difficulties faced during the project, the chosen methods, and future work on the project.

Wikify API. Due to extensive computation time and issues with the API server at the end of the project, the keyword extraction using the Wikify API was not investigated further. The mapping between job description and Wikipedia concept could be used to directly match the job offers with the set of skills and enable to cover a much larger dataset of Wikipedia concepts.

Skills dataset. The manual building of skills dataset was a mistake. Even though it enabled to reduce the noise of the data, and have a straighforward solution for the skills, the solution does not scale and required an extensive amount of time to compute. Also it failed to overlap with the concepts of the EPFL courses. Finally, the definition of skills being complex, the selected terms were based on our judgment that is not the most knowledgeable. A better definition on the skills dataset, using the O*NET database could strongly improve the graph.

Skill to Wikipedia matching. In this project, we also built manually (using a semi-automated program) the matching between skills dataset and list of Wikipedia articles. Using metrics like "Levenstein" distance, or the EPFL Graph and Wikify keyword to concept mapping could enable to scale this associated, and match multiple Wikipedia concepts to skills.

Job description and skill keywords matching. The natural language can often be ambiguous, and the same meaning can be found in numerous words in different texts. Therefore, one of the challenges of this project was to be able to match all different meanings of the same skill with it. To accomplish this, two simple solutions were used (add a list of related words to skills keywords, and use stemming). Nevertheless, word embedding techniques are known to provide good results and are a more scalable approaches. Using BERT or GloVe models to compute embedding, and adapted metrics should improve the keyword matching and increase the number of skills found per job offer.

Job description keywords score. The job offer keywords extracted using keyBERT have a score representing the importance of the keyword in the text. This score was not used in the project, but could impact the weight of the skills in a job offers, and therefore the nodes/edges in the graph of skills. This emphasizes the idea that in a certain job offer, certain skills are more important than others.

Position/Job-specific skills. We can expect certain jobs to require a certain type of skills. To improve the job description to skill matching, we could make use of the job title (or job description) to build a set of job-specific skills. Then by matching the different job offers to the associated job/position, we can add implied skills for certain position that are not seen in the job offer description. This would enrich the skills list when the job description does not provide sufficient information about the required skills, and increase the average number of skills per job offer. It would also enable to categorize the different job offers and have a better understanding of what type of job requires what skills.

EPFL courses analysis. Through this project, we failed to focus on the concepts related to EPFL and rather worked on more general skills. Due to our small dataset size, this leads our graph to be completely disconnected from the EPFL courses concepts. An increase in the skills dataset, or adding skills found in the course's concepts could fix this issue. Also, we observed that none of the "Software" or "Programming" skills added and taught at EPFL were not present in the list of course's related concepts. For example,

the "Java" concept is not connected to the "Introduction to programming" or "Object-oriented programming" classes even if those are teaching programming in java. The current courses' concept representation is mostly based on the course description. Student feedback, learning supports, or introduction course material could be used to enrich the concepts or skills associated with the course, and potentially help the graph and concepts to be connected.

5 CONCLUSION

Through this project, we were able to build ready-to-use datasets of job offers from the Swiss job market and skills. By testing different NLP techniques to extract keywords from the job offers, we could observe the different skills that are the most common requirements in the job market. Finally, using the our graphs visualizations, we illustrated the relationship between skills in the job market.

ACKNOWLEDGMENTS

Many thanks to Francisco Pinto for following the project, providing feedback and helpful suggestions throughout the semester.

REFERENCES

[1] *EPFL Graph*. [Online]. Available: https://graphsearch.epfl.ch/
[2] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
[3] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: https://ipython.org
[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USA: USENIX Association, 2004, p. 21.
[5] J. Russo. Avoiding webscraping throttling using python and tor as a proxy. [Online]. Available: https://boredhacking.com/tor-webscraping-proxy/
[6] N. Palani, *Selenium Webdriver: Software Automation Testing Secrets Revealed Part 2*. Schaumburg, IL: eBooks2go, 2016.
[7] R. V. Chandra and B. S. Varanasi, *Python requests essentials*. Packt Publishing Ltd, 2015.
[8] L. Richardson, "Beautiful soup documentation," *April*, 2007.
[9] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134
[10] M. Danilk, "Python langdetect package: port of google's language-detection library from java to python." [Online]. Available: https://github.com/Mimino666/langdetect
[11] Google, "Google Translate API Pricing." [Online]. Available: https://cloud.google.com/translate/pricing
[12] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html
[13] M. Majlis, "Wikipedia api python." [Online]. Available: https://github.com/martin-majlis/Wikipedia-API
[14] E. Center for Digital Education.
[15] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
[16] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.
[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.
[18] M. Grootendorst, "Keybert: Minimal keyword extraction with bert." 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4461265
[19] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162
[20] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

APPENDIX

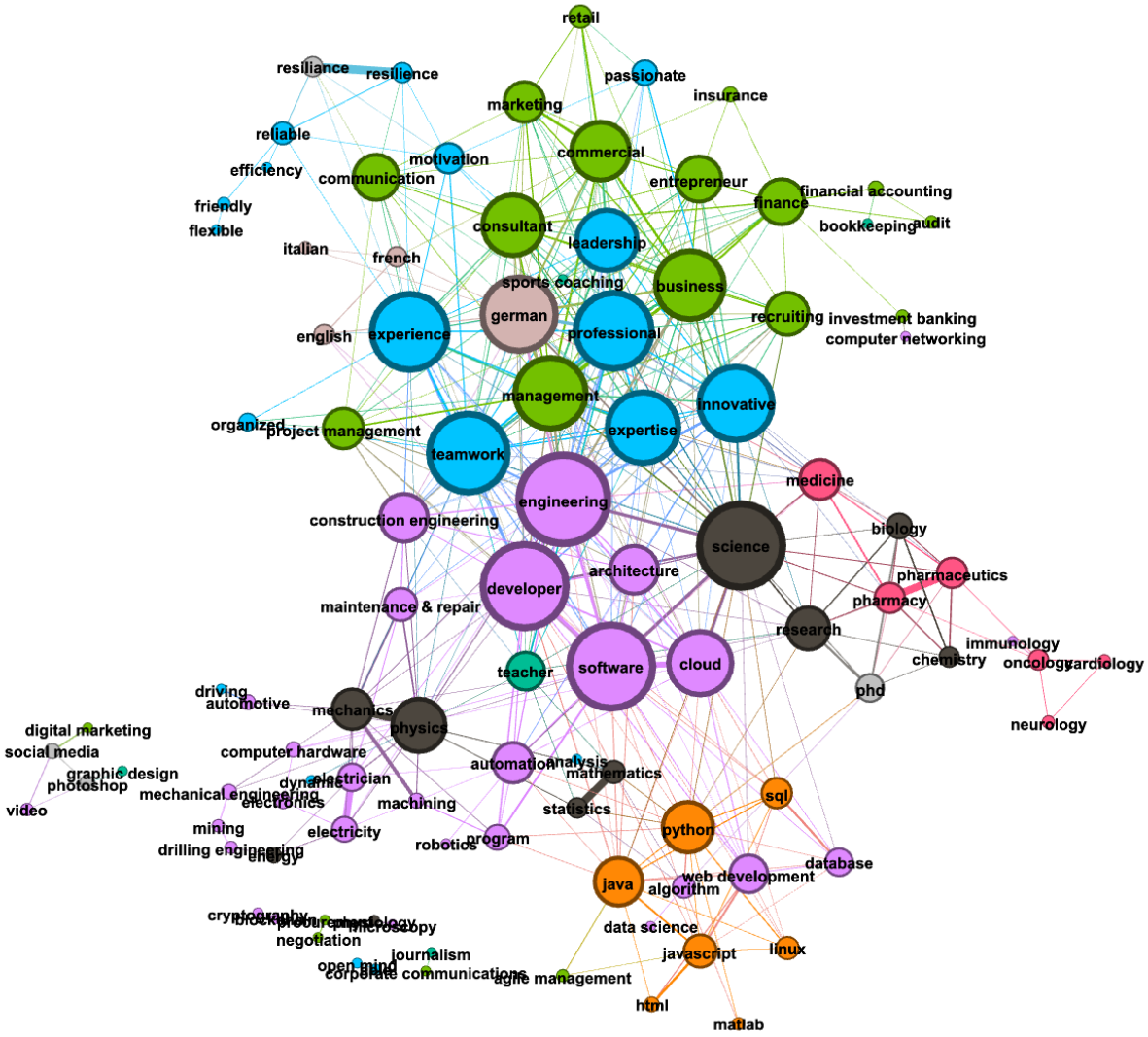

Figure 2: Skills Graph using intermediate skills dataset (Normalized and Filtered Top 10%)

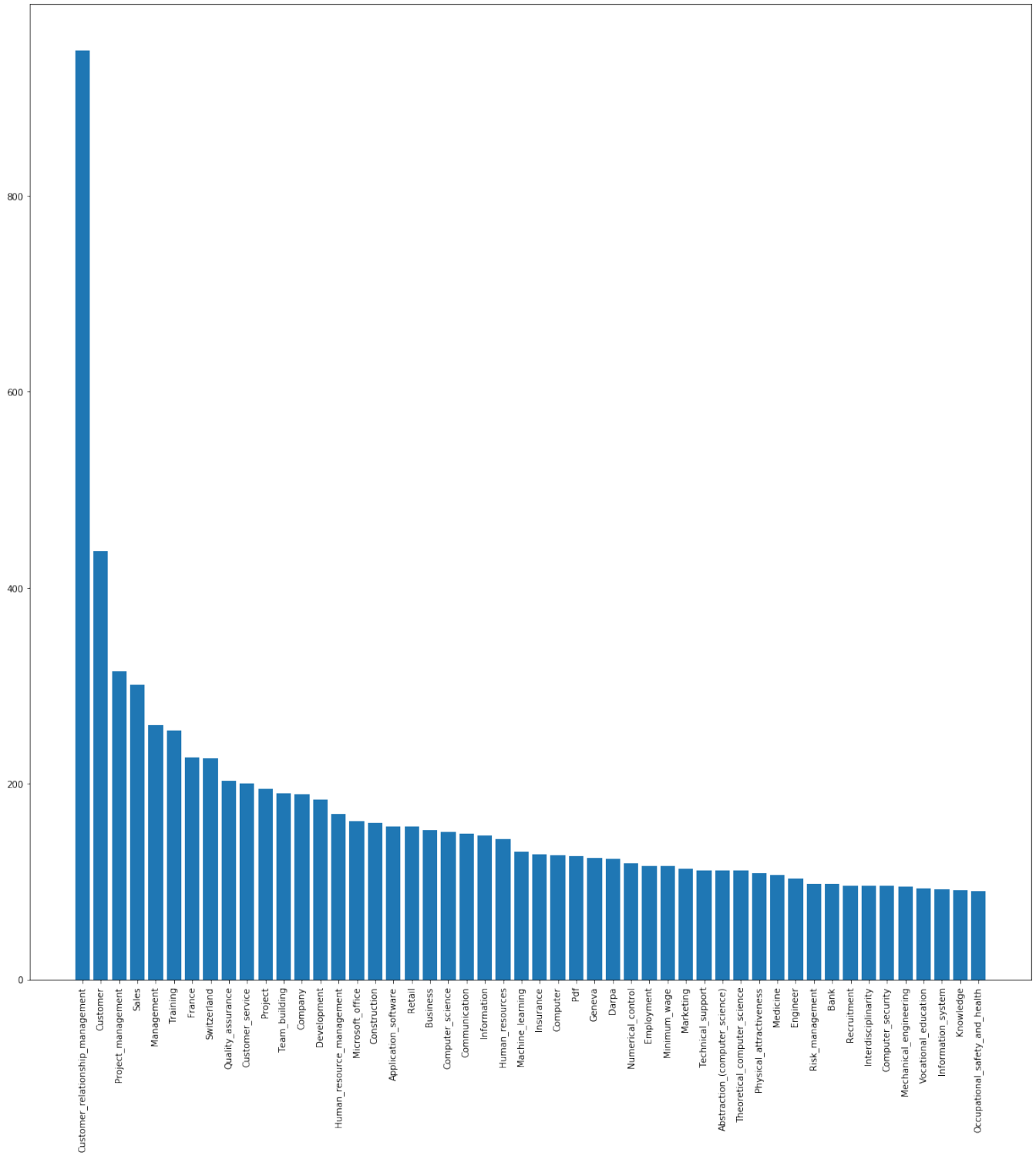

Figure 3: Job description most frequent keywords (Wikify API) - first 1000 jobs

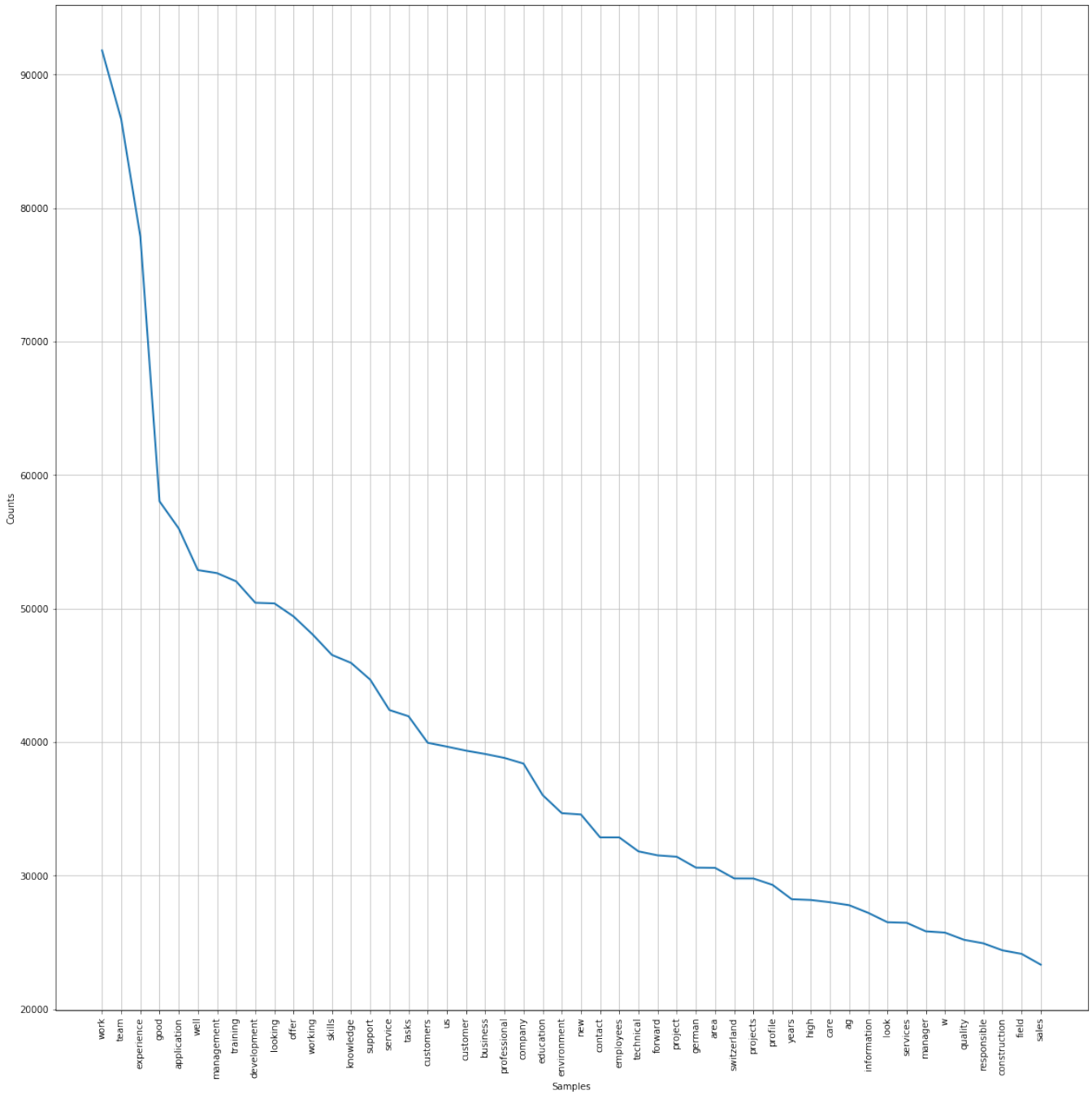

Figure 4: Job description most frequent keywords (nltk) - 64k jobs

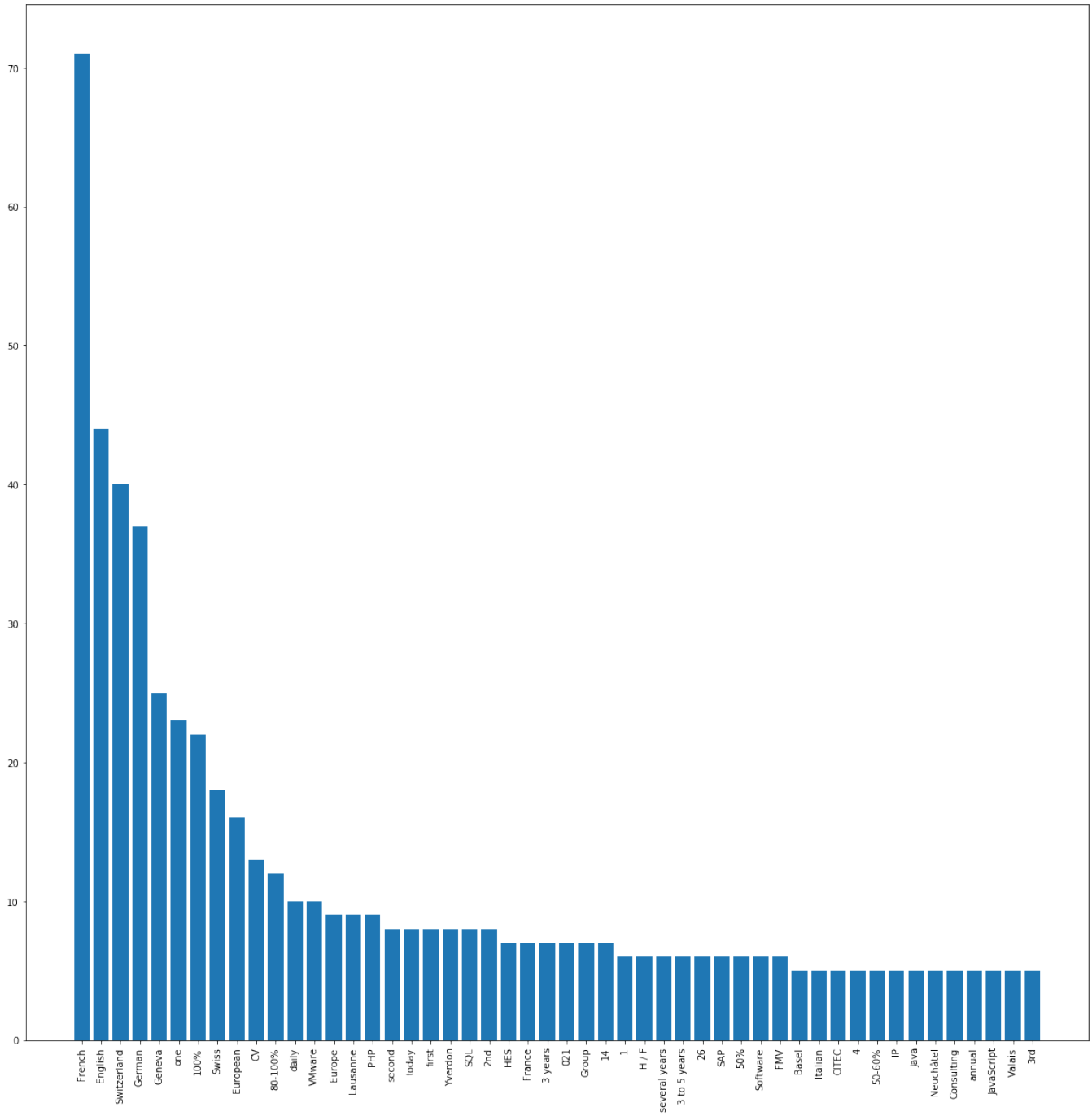

Figure 5: Job description most frequent keywords (SpaCy) - first 100 jobs

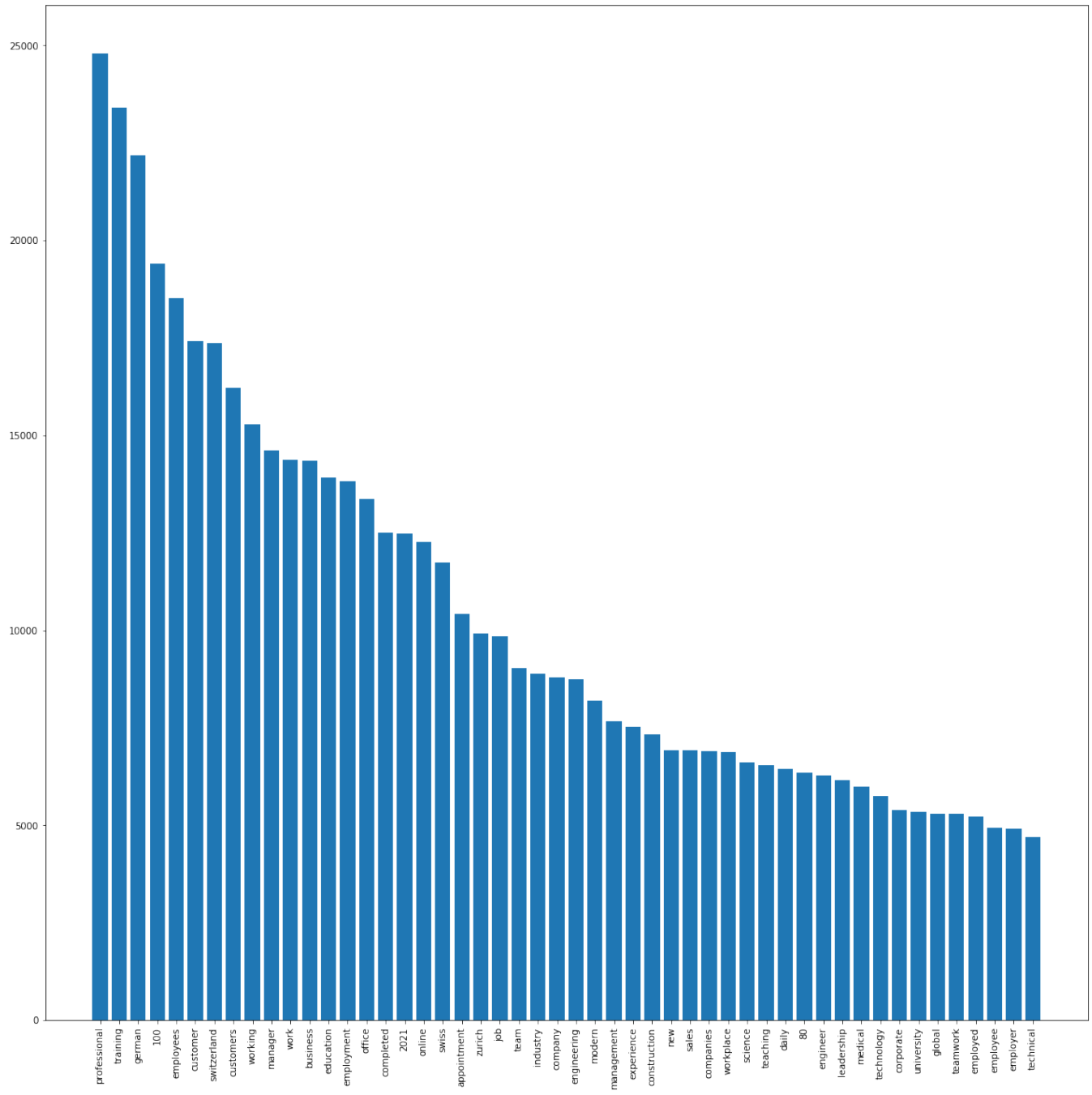

Figure 6: Job description most frequent keywords (keyBERT) - 64k jobs

```
In [238]: for i in [software_engineer_id, electrical_engineer_id, hr_manager_id]:
              print(f"Job title: {jobs_df.loc[i, 'title']}")
              print(f"Similar jobs: ")
              display(show_similar_offers_title(i, 0.7)[:10])
```

```
Job title: Ingénieur logiciel JAVA (H/F)
Similar jobs:

['Architecte Technique',
 "Intégrateur d'applications (DevOps Java / Angular) - Confirmé",
 'Architecte logiciel Java (H/F)',
 'Ingénieur logiciel JAVA (H/F)',
 'Responsable amélioration continue',
 'Consultant ETL TALEND',
 'Consultant Technicien Migration',
 'Ingénieur informatique JAVA Cammunda/BPM (H/F)',
 'Gestionnaire de projets en Romandie',
 'Consultant secteur Logistique - Transport 100% (H/F)']

Job title: Chef(fe) Projets en énergie de traction
Similar jobs:

['Disponent administratif électroménager (h/f) 100%',
 'Chef(fe) Projets en énergie de traction',
 'Responsable de Projet Sécurité (h/f) 80-100%',
 'Spécialiste en gestion de données énergétiques',
 'Chef de projet en installations électriques',
 'Responsable de projet ELE (H/F)',
 'Responsable de projet ELE (H/F)',
 'Conseiller en sécurité électrique (H/F) ou Chef de projet (H/F)',
 'Chef de projet en technique ferroviaire 80-100% (f/h)',
 'Un(e) chef(fe) de projets senior en électricité [Suisse - Canton de Vaud ou Genève]']

Job title: Consultant (w/m/d) Controlling & Finance
Similar jobs:

['Chef/Cheffe de projet senior 80-100%',
 'Assistant Manager or Manager - Finance Transformation 80-100%',
 'Manager - SAP FI/CO Consulting 80-100%',
 'Senior IT Quality Consultant',
 'Internship - Consulting - Finance Strategy & Transformation 100%',
 'Operational Excellence Specialist in Banking (Geneva)',
 'Assistant Manager or Manager – Finance Transformation',
 'Service Owner Digital Health (w/m)',
 'Strategisches IT-Management – Berater und Unternehmer (m/w/d)',
 'Technischer Berater ITSM / Software Engineer']
```

Figure 7: Examples of jobs with similar set of keywords

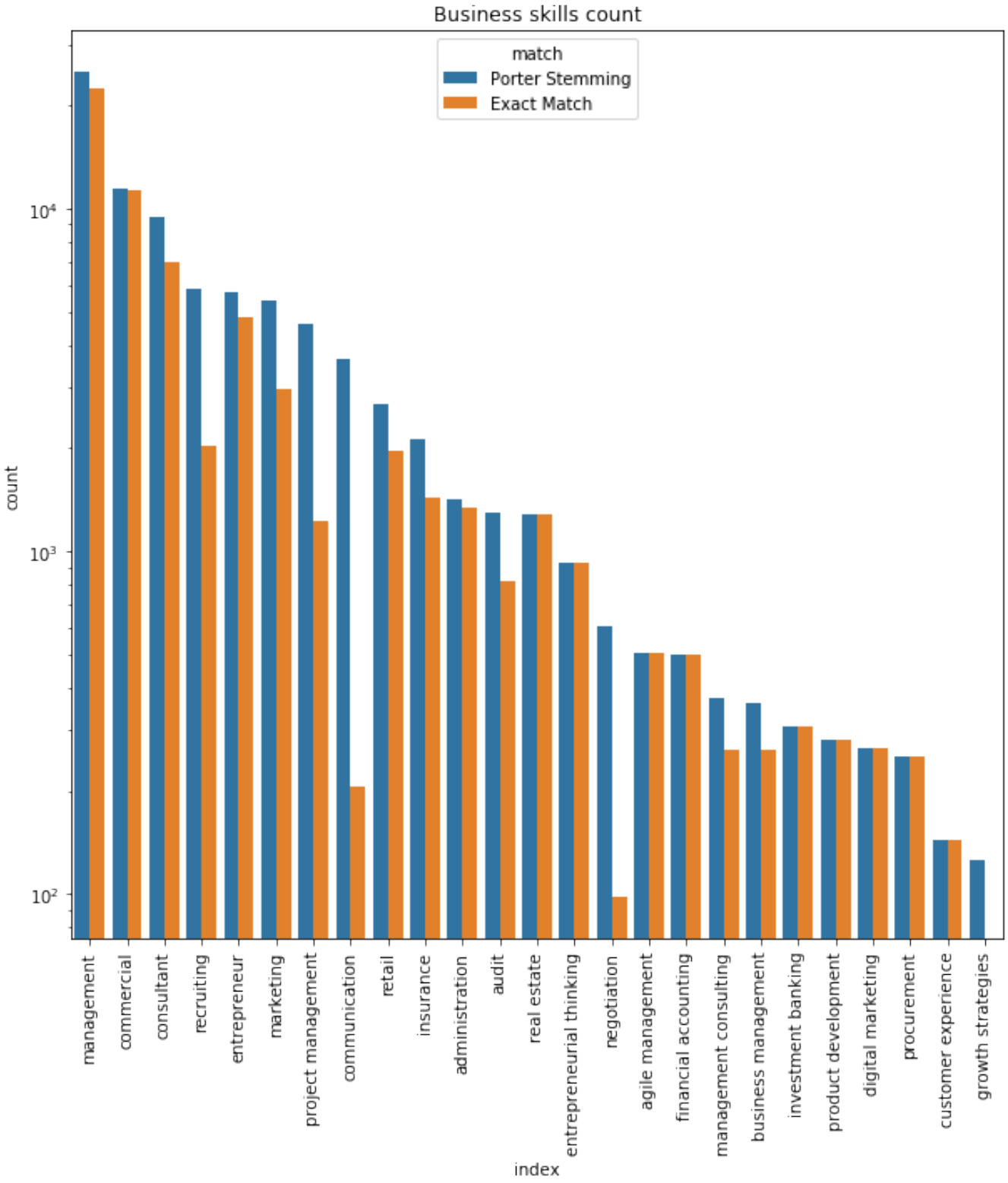

Figure 8: Business skills histogram using Exact matching vs. Porter Stemming

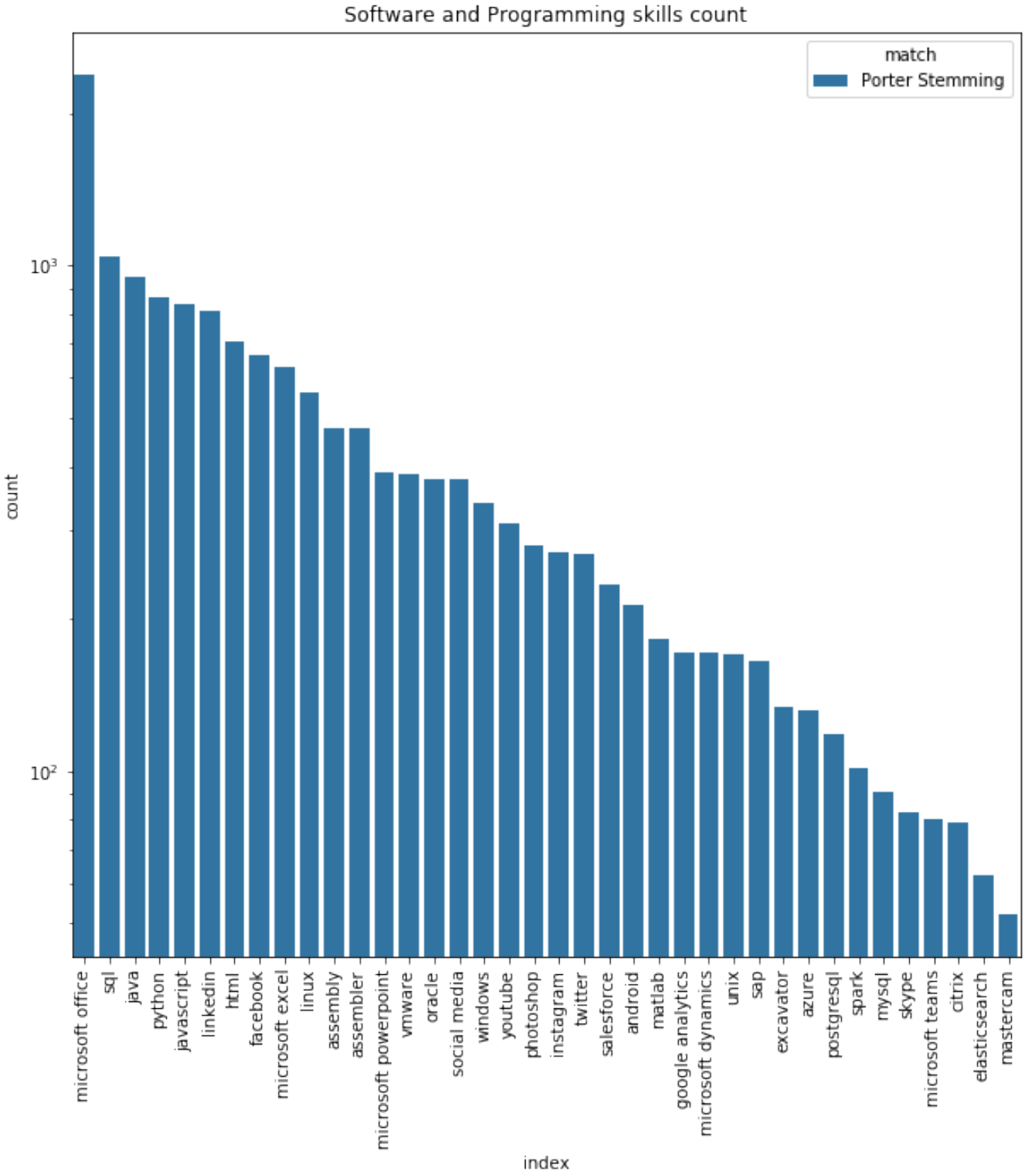

Figure 9: Programming and Software skills histogram

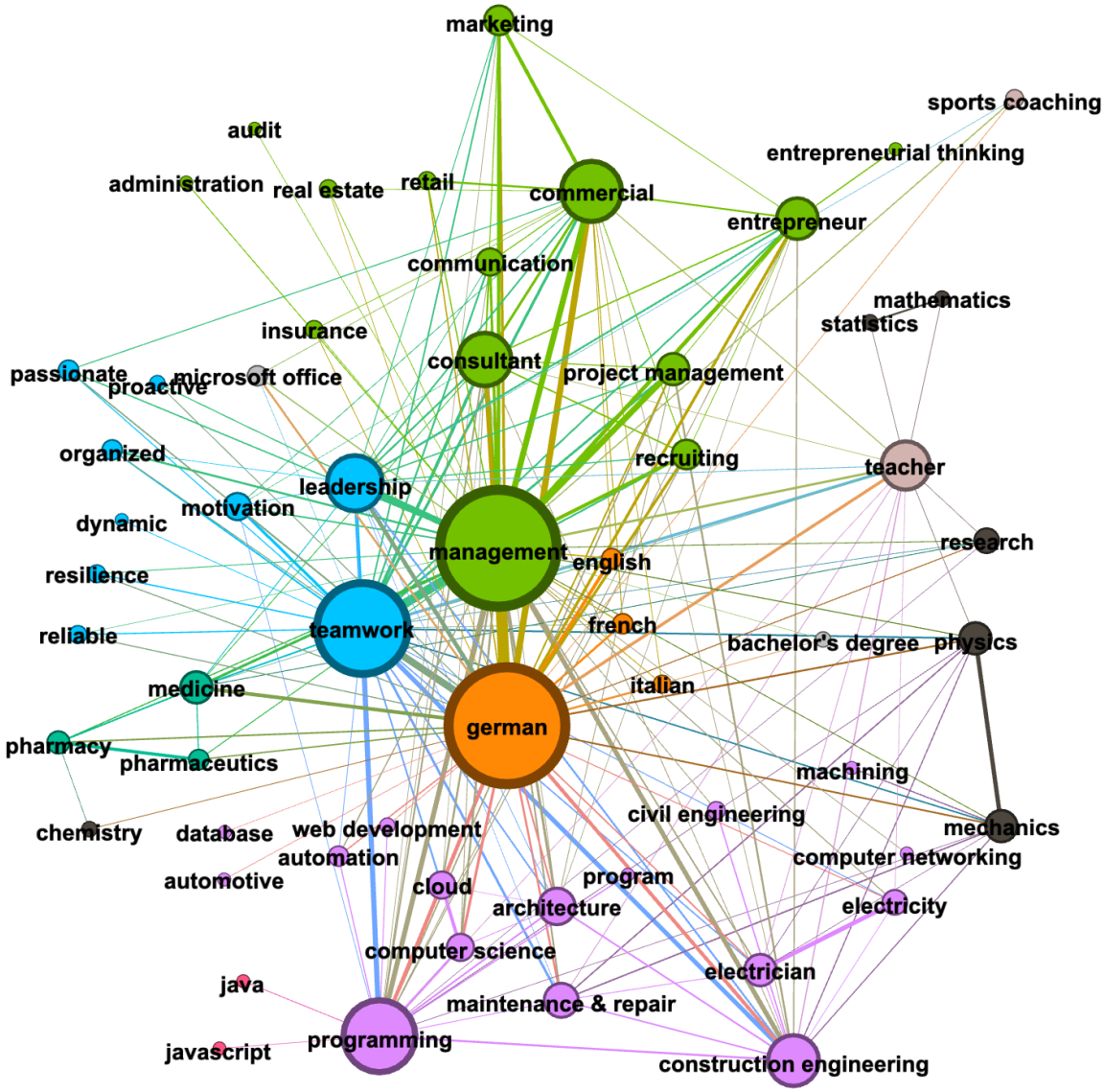

Figure 10: Graph of skills with co-occurrences weights filtered (edges with a least weight of 100)

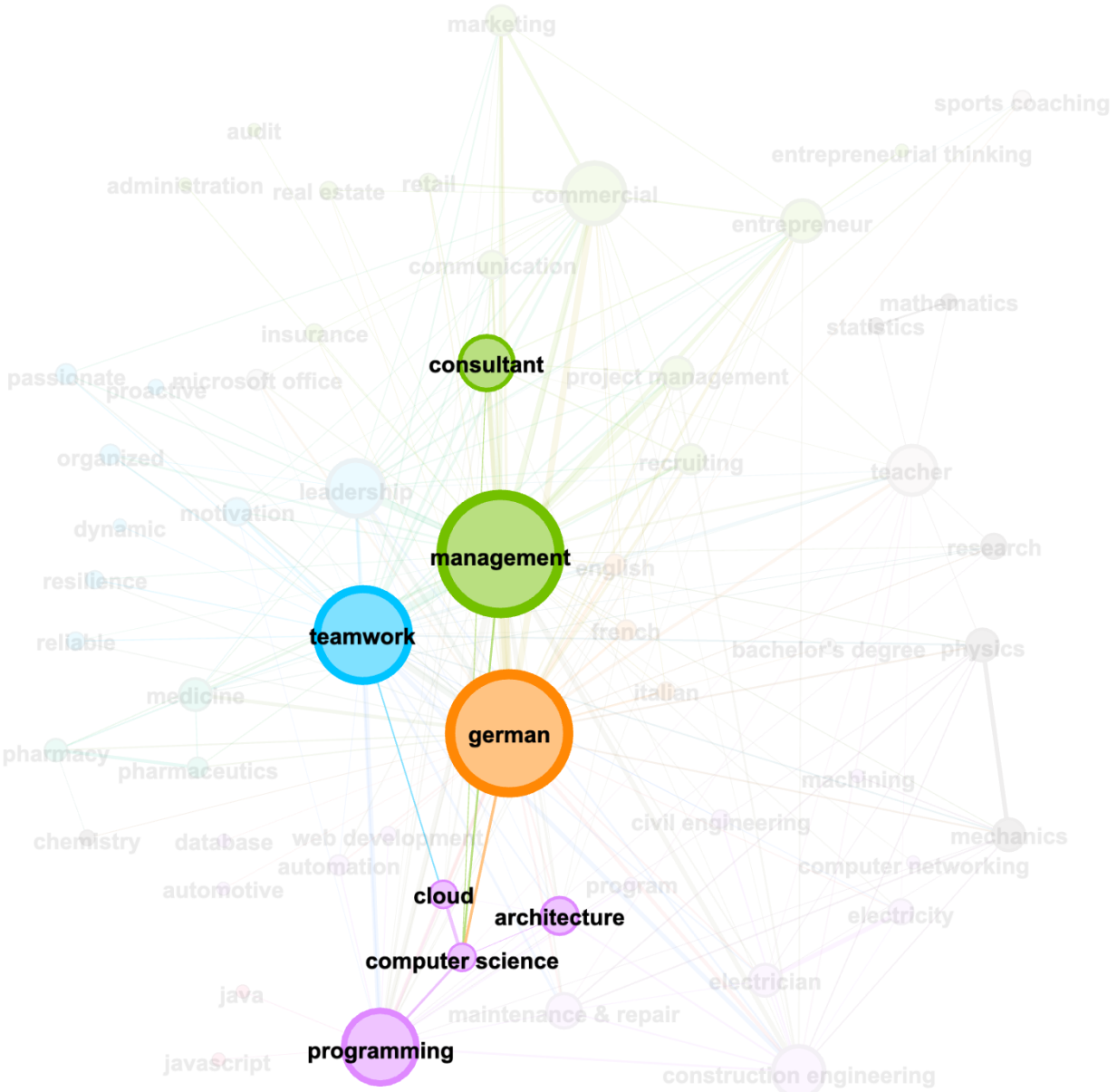

Figure 11: "Computer Science" skill connections in Co-occurrences graph filtered

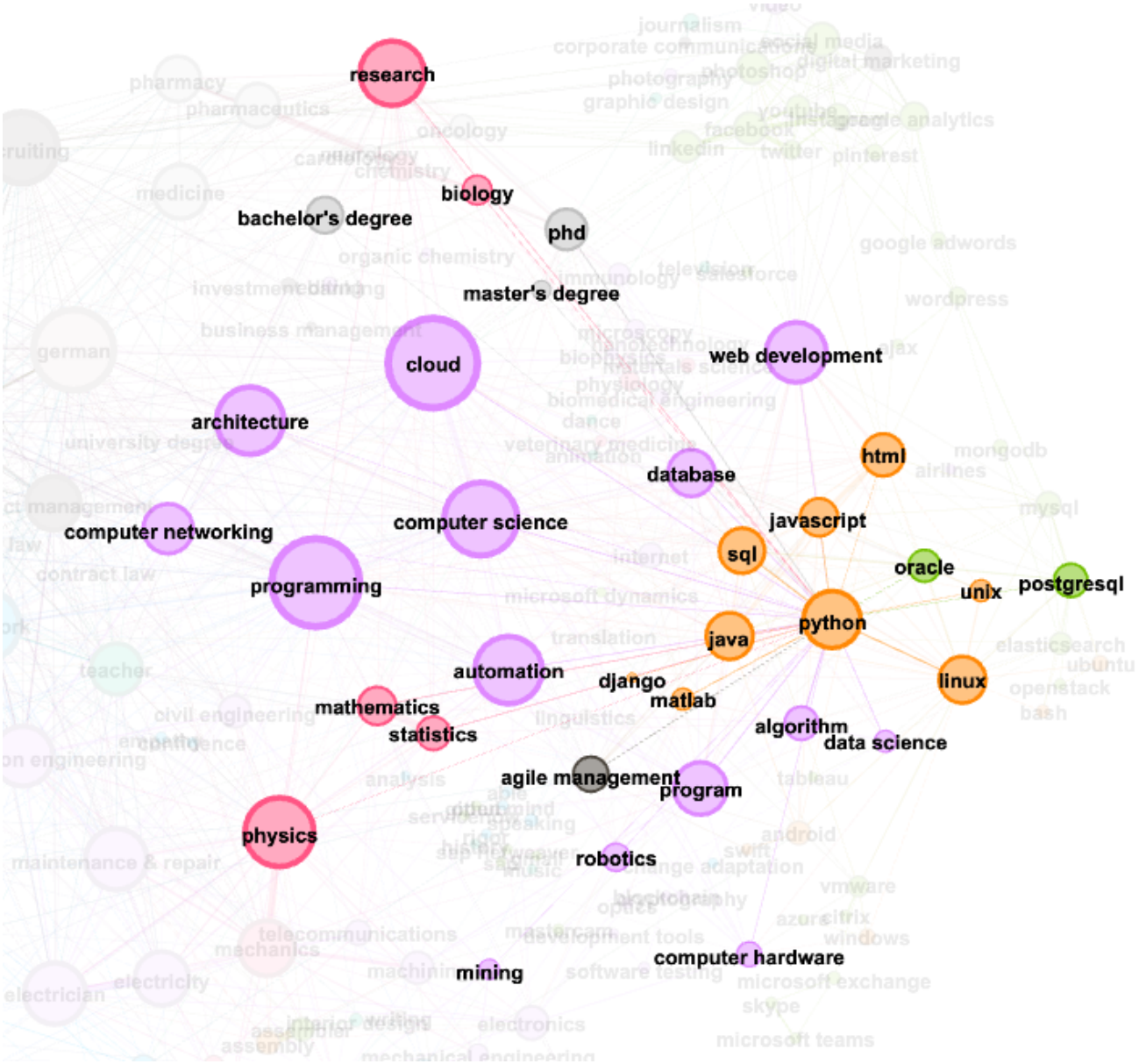

Figure 12: "Python" skill connections in Normalized Co-occurrences graph Top 10%

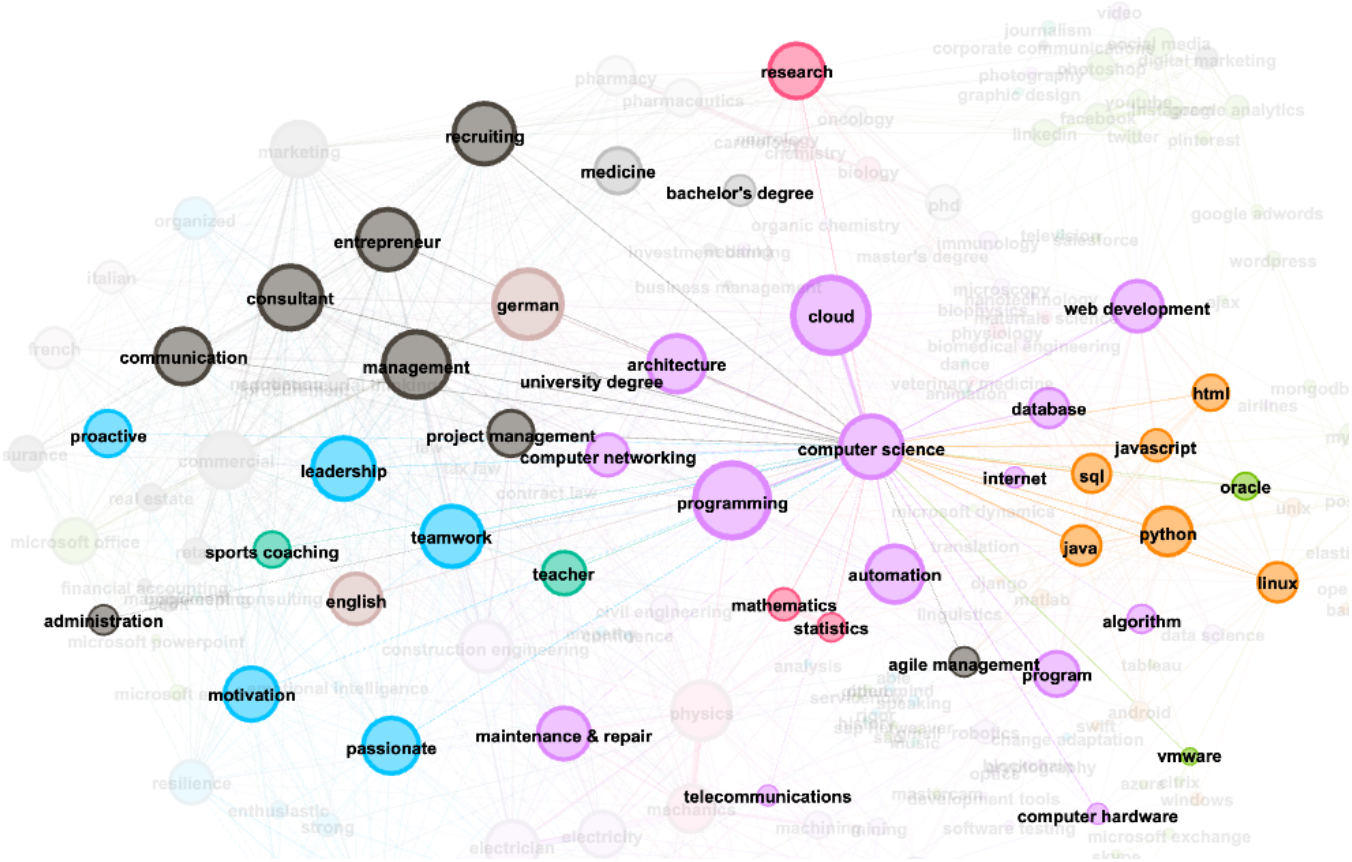

Figure 13: "Computer Science" skill connections in Normalized Co-occurrences graph Top 10%

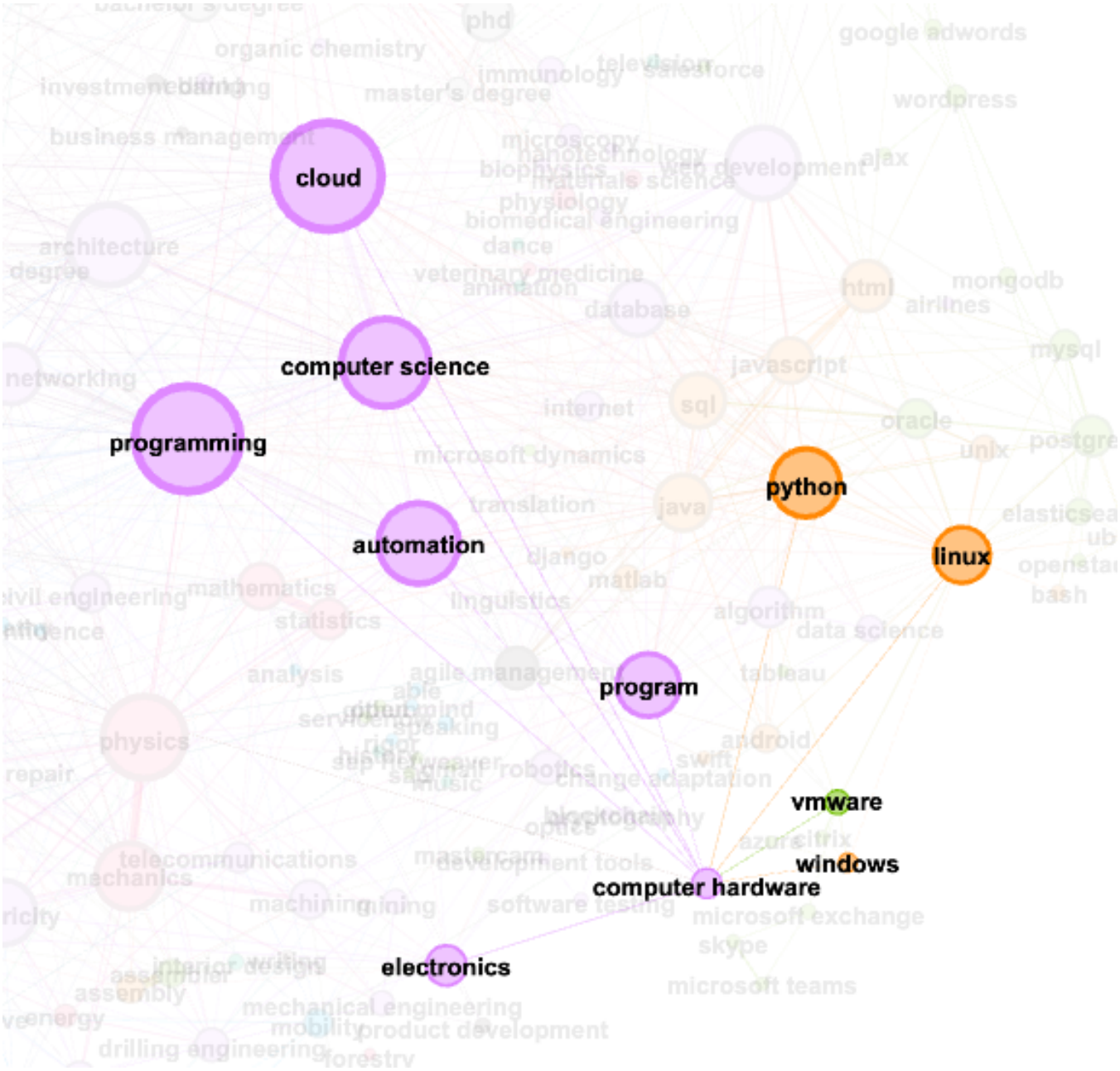

Figure 14: "Computer Hardware" skill connections in Normalized Co-occurrences graph Top 10%

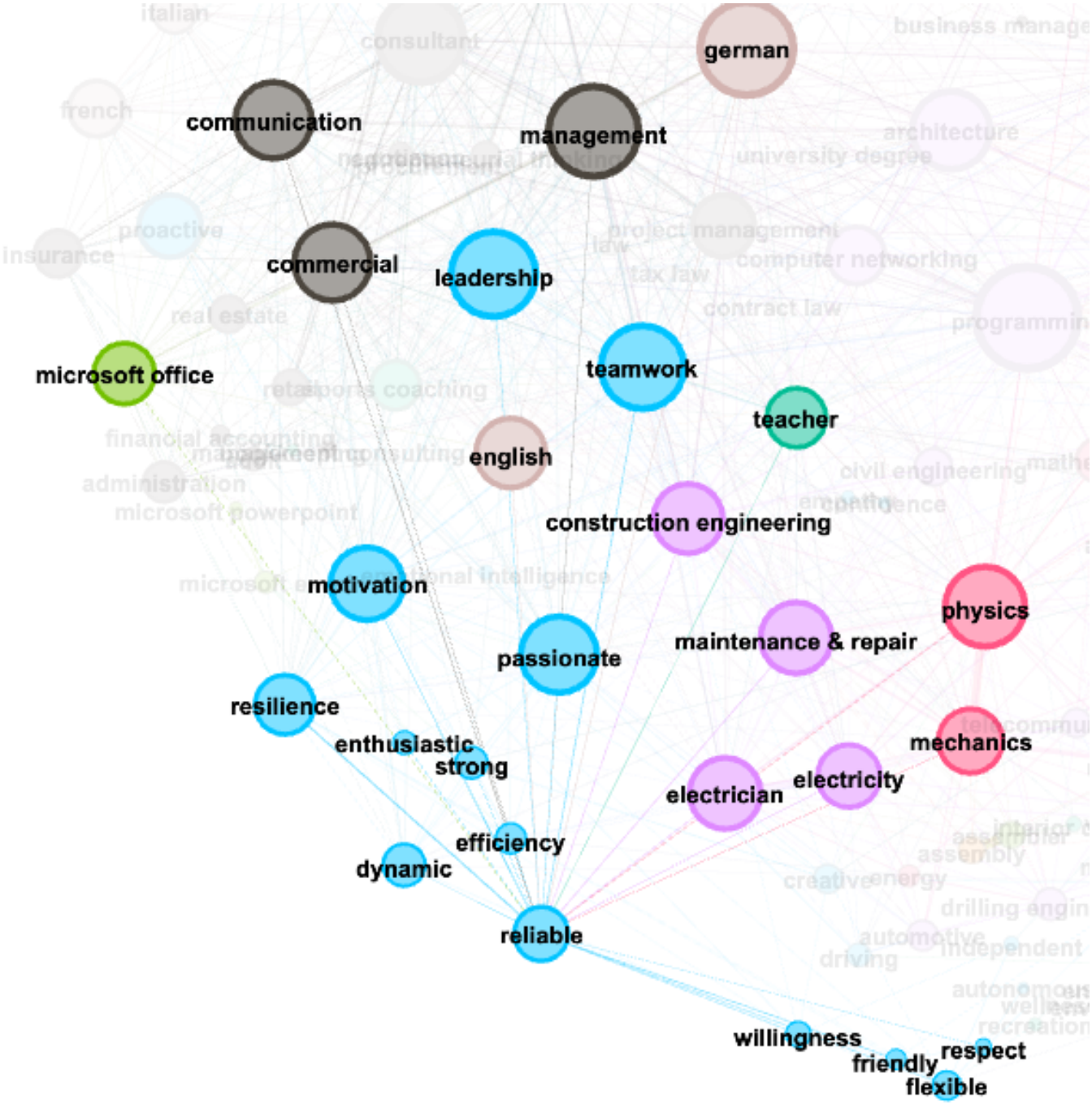

Figure 15: "Reliable" skill connections in Normalized Co-occurrences graph Top 10%

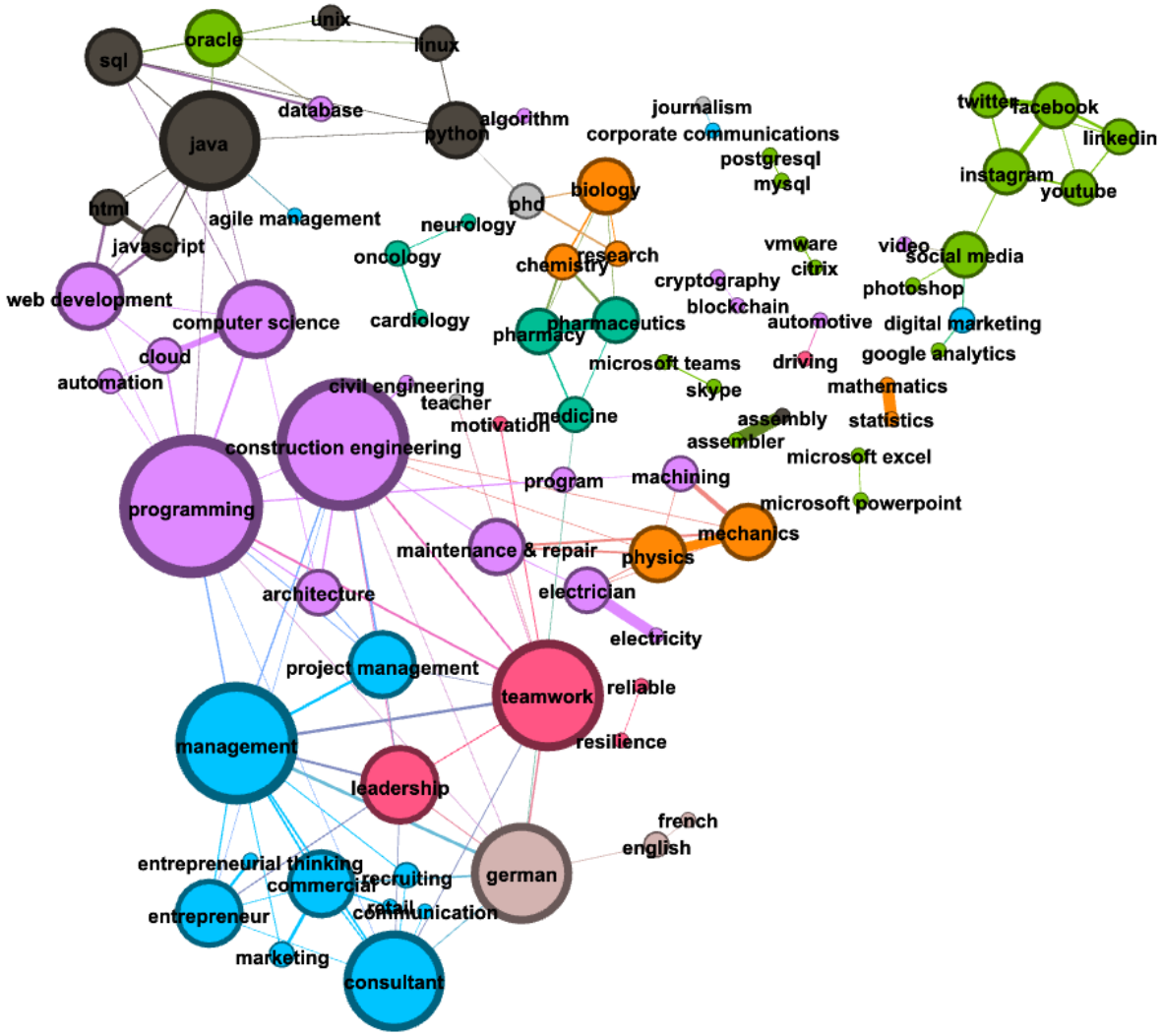

Figure 16: Graph with Normalized Co-occurrences weights Top 1% of edges

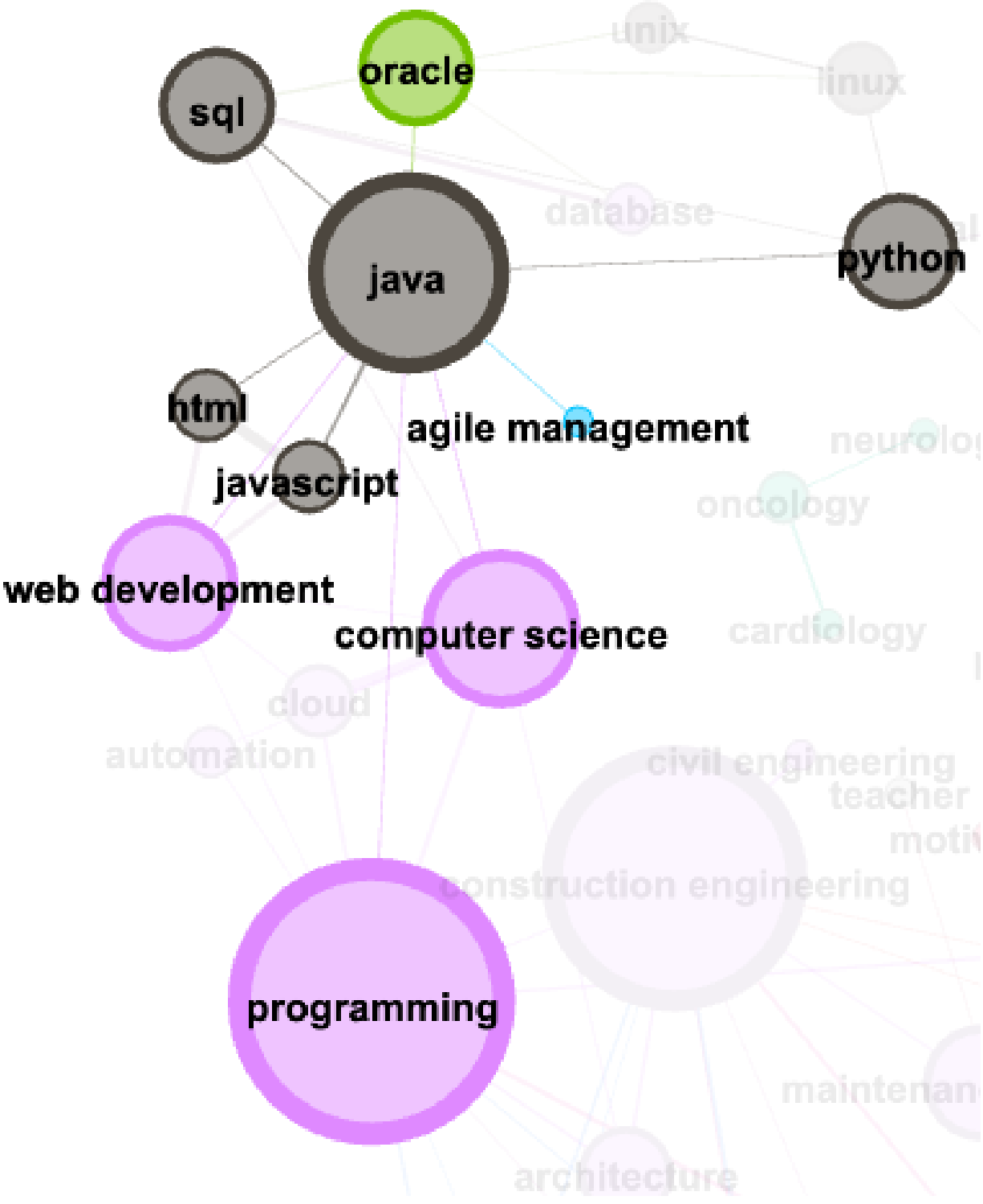

Figure 17: "Java" skill connections in Normalized Co-occurrences graph Top 1%

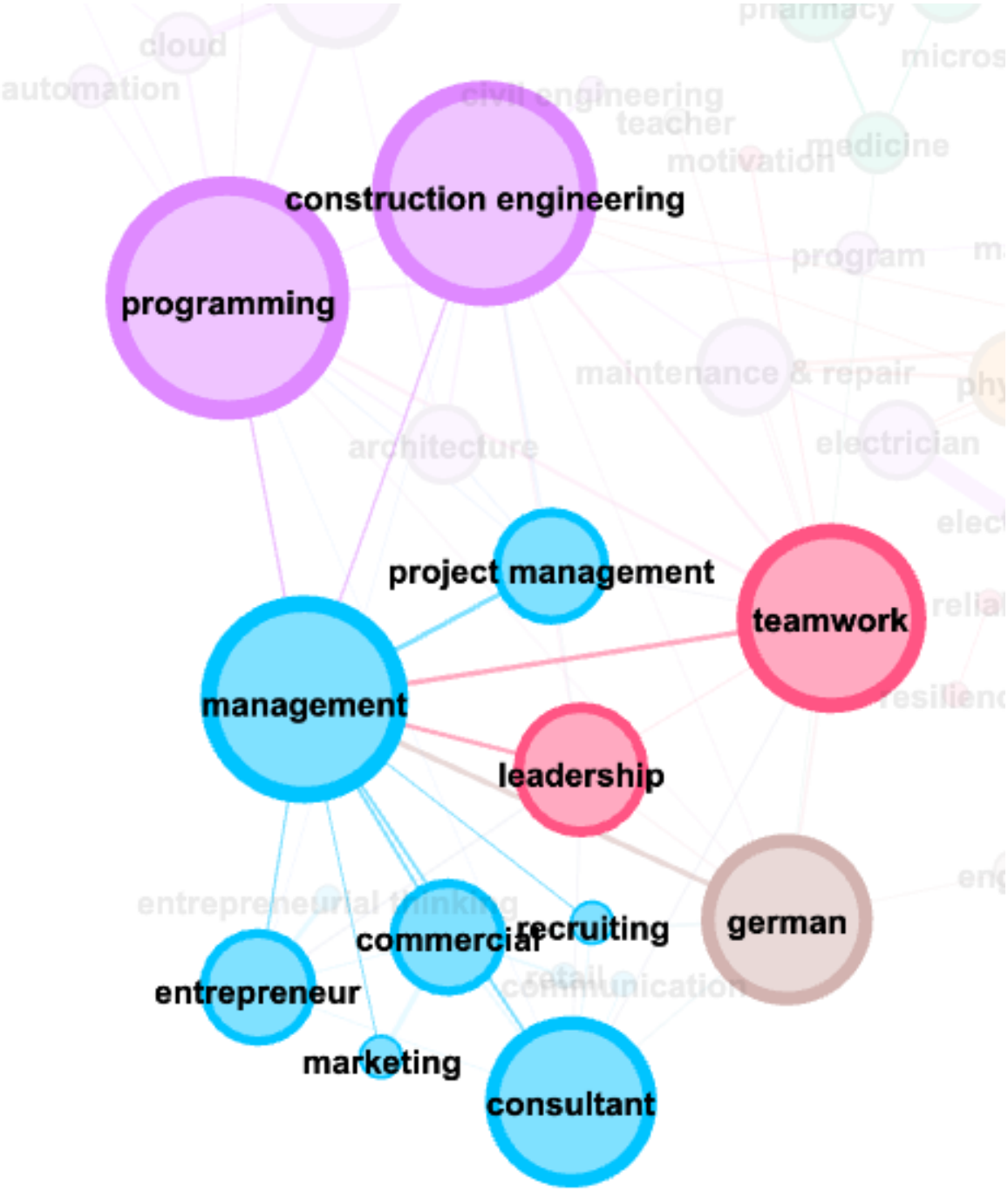

Figure 18: "Management" skill connections in Normalized Co-occurrences graph Top 1%