Toward Online Matching of Europe's Job Market

1 Motivation

Our overall objective with this study is to investigate how the European job market could operate if people would really move freely throughout Europe. This idea has been inspired by the 2012 Nobel Prize in Economics being awarded to Gale and Shapley, for their research on the stable marriage problem and its implications for matching markets. Finding the stable marriage between online vacancies and social media profiles seemed the perfect challenge to approach while competing for the Norvig Web Data Science Award.

1.1 Participants

Our team brings together the Information Access Group of CWI and high-tech SME TextKernel. We combine CWI's expertise on semi-structured document retrieval and MapReduce with TextKernel's knowledge on multi-lingual semantic recruitment technology. Textkernel's product portfolio includes tools for advanced text processing in the human resources domain, and data streams consisting of profiles and CVs on the supply side and job vacancies from job boards on the demand side.

2 Task and Objectives

The final purpose of our study is to collect the data to match job-seekers and vacancies across Europe (e.g., matching vacancies in Northern European countries with job-seekers from Southern European countries). The objective for the competition is to assess the feasibility of replacing and/or extending Textkernel's domain- and country-specific focused crawling methods by more generic techniques using the Commoncrawl data, facilitating the company to include more data sources and expand their technology to include a wider range of languages.

The approach covered so far creates a pipeline to extract vacancies and profiles from the data set, leaving vacancy ranking and solving the stable marriage problem for future work. We (1) explore general statistics of the dataset, (2) filter the crawl by a high recall (but low precision) first pass filter to identify candidate vacancies in top domains of interest, (3) select the most likely vacancies using a subsequent high-precision classification phase, and (4) extract social media profiles from Europe's three major professional networking platforms.

3 Domain statistics

The first step is to generate overview statistics to estimate the number of vacancies and profiles we can expect to extract. Based on the metadata files, we find that the crawl includes 264 different top-level domains, representing 1.2B (11187032078) web pages from 29M (29987331) hosts. The top 10 domains, in decreasing order of size, are `.com`, `.de`, `.org`, `.net`, `.uk`, `.nl`, `.it`, `.info`, `.ru`, and `.br`. These top 10 domains account for more than 75% of all internet hosts and 77% of web pages, with the `.com` domain alone accounting for more than 51%.

4 Extracting vacancies

Textkernel has existing in-house solutions for Dutch, German and French online vacancies, and has started development of a new English pipeline in the context of this project. To extract vacancies, we

deploy a two stage process. A first pass filters potential vacancies at high-recall and low-precision, followed by a (more expensive but) high precision subfiltering step. The first stage deploys (proprietary) lists of vacancy keywords provided by Textkernel. We identify the web pages that match two or more keywords from these lists. Our experiments use an (existing) Dutch and a (newly created) English keyword list.

4.1 The first pass filter

We now present the implementation steps to carry out the first pass vacancy filtering in Hadoop, as applied to the Commoncrawl data. For Dutch vacancies, we consider the `.nl`, `.com`, and `.org`. For English, filtering was restricted to a subset of eight Northern European top domains, `.de`, `.nl`, `.uk`, `.se`, `.fi`, `.at`, and `.no`.

a) Obtain URLs of candidate vacancies

A MapReduce job processes the `Text` files from the corpus. The vacancy keywords are stored in a plain text file, made available to the mappers using a `Distributed Cache` job configuration. During the mapper's class setup, we read the filter from the `Distributed Cache` and store the vacancy keywords in a `List` data structure. The `Map` function visits the `URL/Text` value pairs, and produces the vacancy keyword frequencies. Matched vacancy keywords and their corresponding frequencies are maintained in a `HashMap <keyword , frequency>`. Output is in the format `<(domainID, URL), (matched_KW, frequency)>`, where the first tuple (bracketed) forms the key and the second the value. Since at least two keywords must match for a page to be in the output, the key will be repeated, which after grouping by the key gives output of the format:

```
<(domainID, URL),
        {(matched_keyword1, frequency1),
         (matched_keyword2, frequecy2), ...,
         (matched_keyword_n, frequency_n)}
>
```

b) Obtain ARC files that contain candidate URLs

First, from the metadata files, we create a look-up index to store information about each URL's location and offset in the ARC files.

Using Pig-Latin, the output from step (a) is joined with the lookup index to find the subset of ARC files containing candidate vacancy URLs. We then extract the `domainIDs` and (unique) `URL`s using Pig-Latin, followed by applying Pig-Latin's "replicated" join.

The Table below presents the statistics obtained for the English vacancies:

Domain	#Pages	#Vacancies	%
de	56006004	4549383	8.12
nl	17334927	1790398	10.33
uk	42289119	15889965	37.57
se	6261311	479751	7.66
fi	2238493	148343	6.63
at	4606741	403742	8.76
no	2903424	183894	6.33

c) Extract HTML for candidate URLs

From step (b) we have a list of URLs and the ARC files containing them, that together let us extract the HTML files for the candidate vacancy pages. We store the file with URLs in the `Distributed Cache` to make it available to the `Map` functions. Input key/value pairs for the map functions are the URL/ARC record. The mappers extract the HTML content from URLs in the candidate list, outputting `<URL, HTML>` as its key-value pairs.

4.2 Stratified Sampling

The next step in our project produces a sample of 100,000 candidate URLs as training data to develop a new high-precision classifier for English language vacancies. From the list of URLs identified in task a) above, we create a proportional stratified sample as follows.

We first compute the total number of URLs for each domain using Pig-Latin, and distribute the resulting summary using the `Distributed Cache`. From each domain, we sample `R * DomainURLCount` URLS, where `R = SampleSize / Sum DomainURLCount` (summing over the domains of interest). At the map stage, we first filter for the domains of interest, sampling in the reduce stage by throwing a (virtual) dice for each URL, keeping the URL if the dice throw results in 1. We keep throwing dice until the desired sample size is reached.

The results of the sampling process are summarized in the following table:

Domain ID	*#candidateURLs*	*#sampleURLs*
de	4549383	19404
nl	1790398	7636
uk	15889965	67774
se	479751	2046
fi	148343	633
at	403742	1722
no	183894	784

5 Extracting Profiles

Profile extraction consists of the following stages:

- Extract pages in `linkedin.com`, `viadeo.com`, and `xing.com` domains from the ARC files;
- Extract the profiles contained in these results;
- Restrict the profiles to people located in Greece, Italy, Spain, Portugal, Ireland, Poland, Turkey, Cyprus, Romania, Bulgaria, and Hungary.

We proceed as follows:

a) Determine if a page is a profile page

We first created accounts in all three professional platforms to find out how profiles can be identified in the data.

Profile pages from `viade.com` or `xing.com` contain `/profile/` as pattern in their URL paths. To accurately identify the profiles from `linkedin.com`, we had to rely on the presence of their *required fields* in the HTML source of the HTML pages, such as `full-name`, `given-name`, `family-name`. We used a `Jsoup` object for parsing the HTML, giving the following results:

	#Pages	*#Profiles*	*#Profiles in*

			countries of interest
`LinkedIn.com`	49406	4864	112
`Viadeo.com`	48324	24776	102
`xing.com`	44492	17988	1101

b) Obtaining geo-location information:

The Jsoup-parsed HTML is also used to derive a geo-location value from the HTML source. Each professional networking site uses their own geo-location field: `linkedin` uses locality, while `viadeo` and `xing` use `country-name`.

The following Table summarizes the number of profiles identified in each professional networking site for the countries of interest:

	`LinkedIn.com`	`Viadeo.com`	`Xing.com`
Greece	3	4	9
Italy	26	36	316
Spain	27	26	538
Portugal	9	10	11
Ireland	16	8	10
Poland	6	5	19
Turkey	7	5	155
Cyprus	0	0	0
Romania	14	6	14
Bulgaria	3	2	10
Hungary	1	0	19
	112	**102**	**1101**

The results presented should be interpreted as preliminary, as various sources of noise in the processing have not yet been adequately addressed. For example, we only used English location names, and have not addressed naming variations like (Germany (en), Deutschland(de)) and (Iran, Islamic republic of Iran).

6 Coding Patterns and Techniques

We now focus on specific implementation techniques to carry out our study on SARA's BigGrid Hadoop cluster.

6.1 MapReduce Patterns

We used a few specific patterns and data processing ideas from Jimmy Lin and Chris Dyer's book *Data-intensive Text Processing with MapReduce*.

a) In-map combiner

We used to reduce the amount of intermediate results from map tasks, by combining values of the same key.

b) Key pair or value pairs

We used this technique when we found that we need to group some data together in the keys or the

values. For example in task for finding candidate URLs we wanted the final output to be in the form: <(domainID, URL), {(matched_keyword1, frequency1), (matched_keyword2, frequecy2), ..., (matched_keyword_n, frequency_n)}>

c) user-specified initialization code

To read cached files, to set and retrieve configuration values.

6.2 Executing external software in Hadoop

The high-precision classifier for the second pass of the vacancy extraction process is proprietary production code, that we would like to execute at the cluster (to avoid pumping around large subsets of the Commoncrawl data).

The classification software (referred to as *Textractor*) takes an HTML file as input and returns a semi-structured representation of the vacancy in XML as output. Running the software involves unzipping an archive that includes its binary distribution, setting paths and other system variables, and starting a server that listens on local ports, and does the actual classification.

As we do not have administrative privileges to install software outside the Hadoop environment, we have to run the code as a sub-process of our MapReduce jobs. To accomplish this, we experimented with three possibilities: running the code directly from java, using Pig streaming, and, using the Hadoop streaming interface. We settled for the third option, of using the Hadoop streaming interface. Hereto, the software is repackaged as `.jar` file, and uploaded to the Hadoop file system, using the `cacheArchive` option to distribute the code and make it available to the mappers. A `bash` script sets the required paths and environmental variables, and starts and stops the server. We embedded a small `python` script within the `bash` script to find the current file being processed and to pass it to the classifier.

While the resulting framework to run third-party software seems to work adequately *in principle*, we have not yet succeeded in getting Textractor to execute correctly on the Hadoop cluster. After running into problems executing the 32-bit codebase on the 64-bit machines, SARA staff has been so kind to install 32bit libc support on the cluster; however, a lack of time did not let us resolve the next problem we stumbled upon, with opening local networking ports on the cluster nodes. We expect that it is definitely feasible to get their high precision classifier software to execute correctly and this way finalize the data acquisition process, but unfortunately will not have completed this step before the competition deadline closes.

7 Conclusion

So far, we have accomplished three main tasks. We estimated domain statistics, identified vacancy pages in the crawl by deploying a high-recall low-precision first stage filter (restricted to top-domain countries of our interest), and extracted professional networking profiles originating in a list of countries of interest. We are just about ready to run the second phase process, which will let us finalize the vacancy data extraction step.

While the project is not yet finished, participation in the challenge has been exciting. We have learned important techniques and acquired the skills to work with the MapReduce framework, Pig-Latin, Hadoop-streaming, using external software on the cluster, and using Hadoop in concert with other scripting languages.