Job searching engine based on Gale-Shapley stable matching algorithm

Piotr Staniów

Department of Computer Science
Faculty of Fundamental Problems of Technology

Politechnika Wrocławska

A thesis submitted for the degree of

inżynier

Supervisor: D.Phil. Mirosław Korzeniowski

Abstract

To be devised. This section might be removed from the final version.

DRAFT COPY ONLY

Contents

Chapter 1

Introduction

Information Technology labour market is quite distinct from others in terms of how people tend to seek for a job. Position requirements for a programmer are often very concise and precise, therefore it is feasible to quantify skills and requirements an applicant has.

In recent years, a handful of job searching websites dedicated solely to programmers emerged on the internet[1] and presented a rather unique approach to filtering jobs. It is now preferable to put an emphasis on programming languages, technologies, libraries and frameworks which are involved in the post, rather than let users search by keywords and skim through a block of text within the offer.

1.1 Overview

In this particular case, especially at the beginning of recruitment process, a crucial part of matching an applicant and an employer is to ensure that some basic requirements are fulfilled. This gives way to providing a job search automation.

The both sides of the market can establish some matching based on applicant's and employer's requirements and offerings, however it cannot be guaranteed that participants do not change their mind. Informally, Gale-Shapley *stable matching algorithm* is an algorithm which outputs the most preferable pairs from two distinct sets, so that no other pair would rather be formed. The algorithm and its

[1]Examples of this approach are: www.filltr.pl and www.nofluffjobs.com

application is thoroughly discussed in the first chapter.

The whole process of finding a suitable job ought to be straightforward and therefore preferably a user to machine interaction should be kept at the bare minimum. General architecture and possible use cases of the web application created over the course of the thesis are widely described in the second chapter.

Not a single website might have been built without some technologies. An overview of programming languages, frameworks and libraries used in the project will be presented in the third chapter. Its general structure is also explained in this part, as well as the motivation behind it.

In the last chapter, guidelines to the installation process are presented in line with a few possible further improvements which can be introduced. Moreover, the project offers a REST API. Its structure, access permissions and communication description is shown in Appendix A.

1.2 Motivation

The need for automation of job searching process is motivated by a variety of problems it may solve. The first advantage of using the stable matching algorithm is that it forms pairs of employers and applicants such that there is neither employer nor applicant which would rather work with someone else and the one also prefers that matching.

One can easily imagine the situation in which an applicant is offered a job by *company A*, which at first is accepted, however in the near future *company B* may also offer a job to the applicant. As a result the candidate may decide to turn down the first proposal and incline towards the second one, which may increase the company A costs of recruitment process. Also the opposite situation is possible when it is the company which finds more suitable candidate for the post.

Furthermore, finding an IT job may turn out to be a daunting task, due to the fact that one has to skim through a vast number of announcements, yet rarely is the list represented and filtered only by a quantifiable set of skills and requirements. The approach presented in the following project will generate propositions for both companies and applicants without an intense involvement. In turn, this will reduce the process of finding a suitable job to a mere procedure of filling in the profile and waiting for the outcome.

1.3 Related work

1.3.1 Algorithm usage in similar problems

1.3.2 Review of job searching engines

Only a subset o the most popular solutions available in Poland is discussed.

1.3.2.1 LinkedIn

1.3.2.2 Pracuj.pl

1.3.2.3 GoldenLine

1.3.2.4 Filltr

1.3.2.5 Nofluffjobs

Chapter 2

Gale-Shapley stable matching algorithm

The Stable Matching problem is occurs in a number of disciplines, including mathematics, economics and computer science. The earliest known matching scheme originated in 1952, when National Resident Matching Program was launched in United States, in order to provide reliable and fair method of matching graduating medical students to their first hospital assignments. [2] Independently, the such an algorithm was devised by David Gale and Lloyd Shapley in 1962 [1]. The most basic variant of the problem it solves is formulated in the following paragraph.

Terms: perfect matching, instability, etc. Restrictions of G-S algorithm and extensions. Time complexity. Reference to proof of correctness.

Let M, W denote two sets of distinct agents, each of them having some ordered preference list of elements from the opposite set. An outcome of the algorithm is a set of ordered pairs $S = \{(m, w) \in M \times W\}$ having a following property: For pair $(m, w) \in S$ there are no elements $m' \in M, w' \in W$ such that *m would prefer w' to w* and *w would prefer m' to m*.

2.1 Properties of stable matching

We will be concerned about some variation on the algorithm which can handle following assumptions:

1. Number of members within two groups can be different

2. An element from one group can be matched to multiple elements from the latter one, i.e. for agent e there is some arbitrary n such that $(e, x_1), ...(e, x_n) \in S$
3. It is possible that some agents remain without a pair: in such case an applicant would rather remain unemployed than choose any offer, as well as an employer may decide not to employ anyone.
4. Ties on a preference list are allowed
5. There are some forbidden pairs

2.2 Algorithm design

Algorithm design is depicted in two subsections:

2.2.1 Pseudocode

2.2.2 Implementation details

Chapter 3

Project architecture

3.1 Deployment diagram

Possibility of separating frontend and API serving layers should be discussed here, as well as bare minimum configuration with vagrant virtual machine. This diagram will present hardware for the system, installed software and middleware used to connect components.

3.2 Component diagrams

Component diagrams showing general architecture and components

3.3 Sequence diagrams

Alternatively, also activity diagrams will be in this subsection.

Usage scenarios and the logic behind.

3.4 Use cases

Possible use cases for employer, applicant and administrator.

3.4.1 Employer registration process

3.4.2 Applicant registration process

3.4.3 Log in functionality

3.4.4 Employer registration process

3.4.5 Posting an offer

3.4.6 Preview of generated stable matching

DRAFT COPY ONLY

Chapter 4

Technologies used and architectural solutions

4.1 Environment

It could be stated here that there are many other solutions and the choice was driven by personal experiences and knowledge.

4.1.1 Virtual machine served by Vagrant

What is vagrant? What is it used for? What are boxes? Which box is used?

4.1.2 Puppet

What is puppet? What are alternatives? Why is it suitable for a quick deployment of a brand new environment? What puppet modules are used? PostgreSQL, et

4.1.3 Python 3

Small excerpt about Python and its version. What programming paradigm is represented by Python? Why it is recommended to use Python 3.x instead of Python 2.x?

4.1.4 Python virtualenv

What are virtualenvs? Why is it worthwhile to use it? How to install all dependencies with one-liner? (pip)

4.2 Frameworks and libraries

4.2.1 Django

What is Django? What are alternatives in Python? What are general alternatives?

Flask, Pyramid Java EE, Play! Framework, Ruby on Rails, NodeJS (?)

4.2.2 Django REST Framework

What is REST? What does this framework provide? What are components that should be created to have REST API? What were the problems with deserialization?

4.2.3 Django Allauth

What is the package used for? How does OAuth2 work? /marginnoteOAuth2 description might be removed from final version How Google Plus authentication may be obtained? What were obstacles that lead to the decision about not using skills from LinkedIn?

4.2.4 Django REST Auth

Who has made it? What is it used for? Why is it complimentary to the package above?

4.2.5 Unittest

What are unit tests? Why is it important to write unit tests? What is tested within this project?

4.2.6 Other dependencies

What can be found in requirements.txt? Optional

Chapter 5

Conclusions and further development

5.1 Further development

Matching for couples that would like to work in the same geographic locations: the problem is NP-complete. However NRMP for the most cases achieved the solution quickly.

Appdx A

.1 REST API

.1.1 General structure

.1.2 Permissions

.1.3 Communication via JSON

References

[1] J. M. Kleinberg and É. Tardos. *Algorithm design.* Addison-Wesley, 2006. 4

[2] National Resident Matching Program. Nrmp description. `http://www.nrmp.org/about/`. Accessed: 2015-11-22. 4