JobScan

Automated CV-Vacancy Matching
and
Improved Search in a Vacancy Database

Anuj Gupta
April 2012
Abstract

In the last few years there appears to be a paradigm shift in the job recruitment field, from physical contact between recruiters and job seekers to e-recruiting, with online job boards providing the meeting place for both parties. As job boards have gained in popularity so have the number of vacancies and curricula vitae being posted on these sites. Whilst this wealth of online vacancies and CVs provides great opportunities to recruiters and job seekers alike, locating relevant information on the job boards has become an increasingly tedious task.

This thesis describes an attempt to alleviate this situation by creating a web application that aids both recruiters and job seekers. Recruiters are offered the possibility to match a CV to all vacancies in the database, providing them with possible leads. Job seekers will also appreciate this functionality as it provides them with job suggestions and saves them time searching through many vacancies. Job seekers are also provided with improved means to search an online vacancy database, through offering them advanced (boolean) search capabilities and improved result ranking. In both cases the effort required to find suitable matches is reduced.

A local vacancy database was created by retrieving vacancies from a number of popular Dutch job boards. Data extractors were implemented to automatically extract features such as the function title, job requirements text and required education, skills and competences from the retrieved vacancies, after which the vacancies were clustered on the requirements text using the cosine similarity measure and k-means. The cosine similarity measure was also used as part of the CV vacancy matching algorithm and to improve result ranking when querying the vacancy database.

Evaluation of the working prototype demonstrated that whilst some improvement to the data extractors is desirable, the prototype was able to show that improved result ranking and relevant CV-vacancy matches are achievable when CVs and vacancies are of high quality.

Anuj Gupta (1069527)
Man-Machine Interaction Group
Faculty of EEMMCS
Delft University of Technology

Members of the supervising committee
prof. dr. drs. L.J.M. Rothkrantz
ir. H. Geers
dr. ir. P. Wiggers
Acknowledgements

First and foremost I would like to thank my supervisor, Leon Rothkrantz, for his support, guidance, critical eye and his broad range of knowledge, which gave great assistance in completing this thesis. I will miss our lively discussions, whether they were related to the thesis or not.

I am indebted to my parents, sister and brother in law for supporting me through this endeavour. They listened, they cooked, they pushed me to keep going and never gave up. Thank you from the bottom of my heart and expect a much awaited home cooked meal from me!

For showing me how e-recruiting sites look from the recruiter's point of view, for providing me with insights into what a recruiter looks at when matching a curriculum vitae to a vacancy and sharing with me some of the good and bad points of existing e-recruiting tools, I would like to extend a special thank you to André da Silva and Robbert van Spronsen.

Last but not least, I would like to thank my friends for keeping close, spurring me on and providing a necessary distraction from my work. I haven't seen many of you as much as I would have liked and I'm sure I've bored and confused you on countless occasions talking about some paper I had just read or function I had just implemented. Thanks for being there.
# Table of Contents

1 Introduction..................................................................................................................................................1  
1.1 Societal Relevance................................................................................................................................3  
1.2 Research Challenges..............................................................................................................................6  
1.3 Methodology...........................................................................................................................................7  
1.4 Motivation...............................................................................................................................................8  
1.5 Thesis Outline.........................................................................................................................................9  
2 Theoretical Background................................................................................................................................11  
2.1 Related Work.........................................................................................................................................17  
2.2 Interviews with Recruiters....................................................................................................................20  
3 Architecture and Algorithm..........................................................................................................................23  
3.1 System Architecture...............................................................................................................................24  
3.2 Comparing Vacancy Visual Features: Digital versus Printed Vacancies...........................................26  
3.3 Vacancy Retrieval Component.............................................................................................................29  
3.4 Vacancy Parser Component..................................................................................................................30  
3.5 Vacancy Clustering Component............................................................................................................33  
3.6 Curriculum Vitae Component...............................................................................................................37  
3.7 Matching Component............................................................................................................................38  
3.8 Ranking Component..............................................................................................................................40  
3.9 Vacancy Database..................................................................................................................................42  
3.10 User Interface.......................................................................................................................................43  
4 Implementation and Testing..........................................................................................................................49  
4.1 Hardware, Software, Tools and Libraries..............................................................................................49  
4.2 Vacancy Retrieval Component.............................................................................................................52  
4.3 Vacancy Parser Component..................................................................................................................53  
4.4 Vacancy Clustering Component............................................................................................................65  
4.5 Curriculum Vitae Component...............................................................................................................71  
4.6 Matching Component............................................................................................................................72  
4.7 Ranking Component..............................................................................................................................76  
4.8 Vacancy Database..................................................................................................................................77  
4.9 User Interface.......................................................................................................................................78  
5 Experiments and Results..............................................................................................................................83  
Experiment 1: Vacancy Parser Accuracy....................................................................................................83  
Experiment 2: Vacancy Clustering Speed...................................................................................................87  
Experiment 3: Vacancy Cluster Accuracy...................................................................................................90  
Experiment 4: Matching CV to Vacancy.....................................................................................................95  
Experiment 5: Result Ranking Keyword Search Results...........................................................................98  
6 Conclusion..................................................................................................................................................101  
6.1 Goals and objectives.............................................................................................................................101  
6.2 Future Work / Improvements................................................................................................................105  
Bibliography..................................................................................................................................................107  
Appendix........................................................................................................................................................111  
A: Databases................................................................................................................................................111  
B: Links..........................................................................................................................................................116  
C: Stop Word List.......................................................................................................................................118
List of Figures

Figure 1: Monster Job Search Email for "Functional Administrator" ........................................... 4
Figure 2: Monster Job Search Email for "Architect" .................................................................... 4
Figure 3: Search Engine Result Presentation ............................................................................... 14
Figure 4: LinkedIn Recommended Jobs Email ............................................................................ 19
Figure 5: LinkedIn Job Search Portal .......................................................................................... 20
Figure 6: Diagram Legend ............................................................................................................ 23
Figure 7: System Architecture Component Overview ................................................................. 25
Figure 8: Vacancy Retrieval Component ..................................................................................... 30
Figure 9: Vacancy Parser Component .......................................................................................... 31
Figure 10: Vacancy Clustering Component .................................................................................. 34
Figure 11: Curriculum Vitae Component .................................................................................... 38
Figure 12: Matching Component ................................................................................................. 38
Figure 13: CV, Vacancy Matching Vectors .................................................................................. 40
Figure 14: Match Score ............................................................................................................... 40
Figure 15: Ranking Component .................................................................................................... 41
Figure 16: Interface Overview ..................................................................................................... 43
Figure 17: Keyword Search GUI .................................................................................................. 44
Figure 18: Keyword Search Results ............................................................................................. 44
Figure 19: CV Management GUI .................................................................................................. 45
Figure 20: Mock-up: RSS Feed Management GUI ....................................................................... 46
Figure 21: Mock-up: System Management GUI ........................................................................... 46
Figure 22: CV-Vacancy Match Management GUI ........................................................................ 47
Figure 23: Vacancy Parser Component Data Extractor Results Overview ..................................... 47
Figure 24: System Wide Methods ................................................................................................ 51
Figure 25: Basic Database Interaction Methods ........................................................................... 51
Figure 26: Vacancy Retrieval Component Information Flow ......................................................... 52
Figure 27: Read/Update RSS feed ................................................................................................. 53
Figure 28: Vacancy Retrieval ....................................................................................................... 53
Figure 29: Vacancy Parser Component Information Flow ............................................................ 54
Figure 30: Data Extractor Methods ............................................................................................. 56
Figure 31: parsedADS Class ......................................................................................................... 57
Figure 32: Markup Removal Information Flow .............................................................................. 57
Figure 33: topWords Method ....................................................................................................... 59
Figure 34: Data Extractor Information Flow .................................................................................. 60
Figure 35: Vacancy Clustering Component Information Flow ....................................................... 65
Figure 36: Dutch Stemmer Information Flow ................................................................................ 66
Figure 37: Dutch Stemmer Method .............................................................................................. 66
Figure 38: Dictionary Methods ..................................................................................................... 66
Figure 39: Dictionary Creation Information Flow .......................................................................... 67
Figure 40: Vacancy T/TF Vector Generation Information Flow ..................................................... 68
Figure 41: Main Clustering Methods ............................................................................................. 69
Figure 42: Cluster Vacancy Multi-Thread ...................................................................................... 70
Figure 43: Calculate New Cluster Mean Multi-Thread .................................................................... 70
Figure 44: Curriculum Vitae Component Information Flow ......................................................... 71
Figure 45: CV T/TF Generation Methods ..................................................................................... 71
Figure 46: Matching Component Information Flow ........................................................................ 72
Figure 47: CV-Vacancy Matcher Methods ...................................................................................... 73
Figure 48: Ranking Component Information Flow .......................................................... 76
Figure 49: Vacancy Database Columns ........................................................................ 78
Figure 50: User Interface Component ........................................................................ 78
Figure 51: Search Interface Methods .......................................................................... 79
Figure 52: Binary Search Operators ............................................................................ 79
Figure 53: Results CV-Vacancy Match ....................................................................... 81
Figure 54: Cluster Size Occurrence Rate (K=150, #Vacancies=3035) ....................... 88
Figure 55: RSS Feed .................................................................................................. 111
Figure 56: Unparsed Vacancy .................................................................................... 112
Figure 57: Retrieved Vacancy Status Codes ............................................................... 112
Figure 58: Parsed Vacancy Status Codes .................................................................. 112
Figure 59: Parsed Vacancy ....................................................................................... 113
Figure 60: Curriculum Vitae ..................................................................................... 113
Figure 61: Dictionary Term ....................................................................................... 114
Figure 62: Vacancy Text Monograms ....................................................................... 114
Figure 63: Vacancy Text Bigrams .............................................................................. 114
Figure 64: Vacancy Text Trigrams .............................................................................. 114
Figure 65: CV-Vacancy Match .................................................................................. 115
List of Tables

Table 1: Source: http://www.whatjobsite.com/An introduction to job boards and job sites.htm ........2
Table 2: Percentage of Ads in the Press and the Internet Mentioning the Following Criteria ..........26
Table 3: Percentage of Ads Mentioning Information Markers in France and the UK ..........27
Table 4: Percentage of Ads Mentioning Selection Criteria in France and the UK .................27
Table 5: Overview of Visual Eye Catchers on Job Boards .................................................28
Table 6: Matching Lists .............................................................................................32
Table 7: K-means: Number of Operations ....................................................................37
Table 8: Characteristic Features ..................................................................................39
Table 9: JobTrack Sample Vacancy Identifiers ............................................................54
Table 10: VKbanen Sample Vacancy Identifiers ............................................................54
Table 11: Recognised Languages ..................................................................................59
Table 12: Dictionary Entry ............................................................................................67
Table 13: CV Database Contents ..................................................................................71
Table 14: Vacancy Parser Component Accuracy ............................................................84
Table 15: Vacancy Clustering Speed (time in s) .............................................................88
Table 16: Ranking Component Queries .........................................................................99
Table 17: Stop Word List ..............................................................................................118
1 Introduction

Everyday, thousands of people world wide visit job agencies in an effort to find a (new) job. Job agencies are facilitators in the marketplace matching job seeker profiles to vacancies, leaving companies and job seekers time to spend on other activities. The task of matching job seekers to vacancies has so far been a task largely driven by human judgement. Recruiters review thousands of curricula vitae, scan their memory and portfolios for candidates matching a profile and make a personal judgement on which candidate(s) to recommend for an open position.

Print media has always been an important recruitment channel, however the Internet has brought about a change in the recruitment process by replacing print media as the most important recruitment channel [Malinowski et al]. Job seekers often no longer visit job agencies, but send a digital copy of their curriculum vitae (CV) to parties whom they believe can help them. Job boards, such as Monster have become enormous marketplaces for job seekers and job advertisers to find each other.

Job boards should assist the employment process. Constructed properly, job seekers should be able to easily locate relevant vacancies and job advertisers should be able to locate potential candidates to fill their vacancies. However, the popularity of job boards has brought about new problems. The steady influx of new users, together with their CVs, and vacancies, has made it difficult to keep up with the daily job board updates. Antiquated search options and extensive manual effort to review search results are making it difficult to efficiently use job boards. Additionally, job board user interfaces treat all users equally, making it difficult if not impossible for users to search for a vacancy/CV in a manner reflecting their personal knowledge, experience and education level. As a result, specialist job boards such as AcademicTransfer¹ are popping up to provide the same service, but only to a specifically targeted group or industry. These specialist job boards take all forms, from sector based jobs, such as financial and ICT, to academic and community based (e.g. an expatriate job board). The result is that job seekers are having to spend more and more time finding relevant job boards, searching for vacancies/creating a profile on these job boards. Meanwhile job advertisers are having to spend more time and money advertising on all the relevant job boards to make sure that they are getting the necessary exposure to fill their vacancies.

¹ http://www.academictransfer.com
Table 1: Source: [http://www.whatjobsite.com/An introduction to job boards and job sites.htm](http://www.whatjobsite.com/An introduction to job boards and job sites.htm)

<table>
<thead>
<tr>
<th>Types of Job Boards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job board</td>
</tr>
<tr>
<td>Generalist job site, e.g. Monsterboard</td>
</tr>
<tr>
<td>Specialist or Niche Board, e.g. AcademicTransfer</td>
</tr>
<tr>
<td>Network Board</td>
</tr>
<tr>
<td>Social Network Board, e.g. LinkedIn</td>
</tr>
</tbody>
</table>

Job search via the Internet has shifted the first impression and calling card of a job seeker to his/her CV. Initially, the only information known about the job seeker is what is presented within the CV. This shift of first contact has also made it commonplace for recruiters to only meet potential candidates before sending them to their first interview. In the past a job seeker and recruiter would first meet to discuss the desires of and possibilities for the job seeker before looking for appropriate jobs. This shift puts additional pressure on the job seeker to make his/her CV stand out above the rest, as it is now down to the quality of the CV to leave a lasting first impression, and not a snazzy outfit or a smart rhetoric, to the detriment of the quality of the CV in some cases.

Public user profiles on social networking sites offer recruiters insight into some of the more personal aspects of potential candidates that normally only come to light during a face to face interview, but this does require additional work from the recruiter's part when an interesting candidate is found. Perhaps to help ease this process, or help job seekers create a more complete profile on the job board, Monster has added the possibility for users to connect their LinkedIn profile to their Monster account.

Advances in the technologies offered by job boards seem few and far apart, whilst the scale they operate on has increased at a staggering rate. Without improvements for both job seekers and advertisers, it can only be a matter of time before it becomes too cumbersome for people to use job boards. New tools are required to let users take advantage of the vast amount of vacancies and CVs stored on the job boards. Improved search queries, result ranking and presentation, and even result recommendations are but a few of the areas in which job boards should strive to improve efficiency, effectiveness and the user experience.

This thesis describes the efforts taken to improve the search experience by:

i. Creating a prototype system to automatically match a database of vacancies and CVs;
ii. Improving manual search by reducing the amount of irrelevant results;
iii. Improving ordering of (relevant) results through a better scoring algorithm.
1.1 Societal Relevance

E-recruitment tools should assist all parties in the recruitment process and not hinder them. In the study *Internet job search and unemployment durations*, Kuhn and Skuterud show that “ceteris paribus, Internet job search does not correlate with shorter unemployment durations” [Marchal et al], suggesting that better tools are required.

The rapid growth in the use of job boards has made it difficult and time consuming for both job seekers and advertisers to manually sift through all the daily updates. The torrent of (poorly qualified) candidates and the costs of having to sort through them compromises the expected economic efficiency of recruitment websites as quoted by [Maurer et al 2011] quote [Dineen et al 2007].

“Badly designed recruitment sites are costing US companies $30 million every day” [Maurer et al 2007] quote [Pastore 2000]. Research by [Jansen et al] showed that a large portion of user's problems with e-Recuiting sites stemmed from issues with the search engine layout, i.e. job seekers being unable to easily find the required search options, and irrelevant results being returned when searching using multiple keywords. They also found that formatting attractiveness could be more important than usability, though search functionality and matching were ranked as most important.

By making the search tools more powerful, making more effort to hide irrelevant and low quality results and improving the result presentation and ranking, users will be able to spend their time on job boards more efficiently, doing what they came to do, namely find a relevant vacancy or candidate. This leads us to the following problem definition:

**Problem Definition:**

“When searching a vacancy/curriculum vitae database, can the quality of search results be improved by using automatically extracted metadata, automatically generated clusters of similar vacancies and creating a ranking algorithm that uses the metadata to better score results and can vacancies be matched to CVs by creating a matching algorithm that uses the generated data?”

1.1.1 From the point of view of the job seeker

Much time is wasted browsing vacancies for appropriateness. To name but some problems, suitable vacancies may be missed due to a bad choice of search terms (keywords), ambiguity in the search terms, vacancies being incorrectly assigned a low rank causing them to appear far down the results list or the vacancies may be described badly making them difficult to locate.
1 Introduction

To be able to fully use all the features of job boards requires going through a registration process. This process includes the creation of a user profile and uploading one's CV. Though not difficult, the process of creating a good profile can be time consuming, discouraging many users from putting in the effort to create a good profile. Luckily, some job boards give users feedback on the completeness of their profile. Monster shows a profile status indicating completeness, and LinkedIn not only give a profile completeness score, but also provides tips on how to achieve a complete profile based on what has already been entered. When one considers that there is no single standard set of information desired by all job boards, it becomes clear that the process of registering, creating and maintaining a high quality profile on multiple job boards is a tedious process.

There are a huge amount of job boards containing thousands of vacancies. Even if a job seeker was to focus on a handful of job boards in the search for a new job, the job seeker may have to browse through the description of tens if not hundreds of vacancies, looking for ones that match the job seeker's interests and personal profile. To aid in this search process, some job boards offer job seekers the option of receiving a daily/weekly summary email of vacancies with function titles matching self-chosen keywords. The quality of the received summary is highly dependant on the chosen keywords. Few results are received if the job seeker is too specific in his/her choice of keywords, and too many (irrelevant) results received when the keyword choice is too general.

Figures 1 & 2 show portions of a daily email summary for a functional administrator and an architect. Even though the query for a functional administrator is quite specific, results are returned for application administrators. When looking at the architect summary, strange results, such as a senior infrastructure systems administrator and an embedded software engineer, appear as it is not clear what type of architect is desired, e.g. software, infrastructure, etc.
Ideally, a job seeker should only need to submit his/her curriculum vitae to a job board, and the job board should return all vacancies for which the job seeker could be interested and for which he/she meets the requirements/profile.

1.1.2 From the point of view of the job advertiser

Job advertisers waste a huge amount of time browsing through either their own and/or commercial databases containing the curricula vitae of job seekers, trying to find matches for their vacancies. Discussions with two recruiters brought forward their need for more advanced search capabilities, such as the use of wildcards, to better describe their search strings.

Besides the time spent searching a (local) CV database, a lot of time is spent sifting through all the responses to a vacancy to extract the relevant ones. [Maurer et al 2011] quote Internet misuse may contribute to long joblessness (2003) as saying that 92% of recruiters and 71% of HR managers feel that most resumes do not match the job description.

There are a large number of job boards, and advertising on all is not only economically non-viable for most parties due to the costs of advertising, but also due to the time and effort required to maintain the vacancies on all the sites.

Additionally, searching for suitable candidates in online CV databases (job boards) through the sole use of keywords may not provide the desired candidates. Ambiguity, poor profiles/descriptions, or different terms describing the same set of skills cause issues for retrieval. Misconceptions often occur. When a job advertiser searches for CVs containing the keyword “accountancy”, does the keyword refer to CVs of candidates with experience in accountancy or with previous employment within an accountancy firm (potentially not accountancy related, but for example as a secretary)?

Job boards focus on keyword based search on what the user is looking for, but often neglect to offer the facilities to omit results by entering keywords that should be ignored. Without providing operators such as the boolean NOT, it would not be possible to get a list of search results for candidates with experience as a “search consultant” whilst excluding candidates with experience as a “desktop search consultant”.

Ideally, a job advertiser should only need to create vacancies containing a description of the skill set that potential applicants should possess on a job board, and the job board should return a list of suitable candidates, ranked and sorted by relevancy.
1.2 Research Challenges

The aim of this project is to create a prototype tool that supports job seekers and advertisers in their decision making process by removing irrelevant search results and presenting the remaining results ranked by some relevancy measure. A number of sub problems must be tackled to achieve this goal:

1. Automatically retrieve vacancies from the Internet and store them in a local database.
   A local database is required to store vacancies that are retrieved from Internet job boards to create a dataset for the vacancy parser component. To complete this step:
   1.1. Determine what needs to be stored in the database besides the vacancy and set up the database.
   1.2. Implement a component to automatically retrieve vacancies from selected job boards and store the retrieved vacancies in the local database.
   1.3. Run the vacancy retrieval component to fill the unparsed vacancy database.

2. Automatically decompose vacancies and store the results in a local database.
   To search for vacancies and match them to CVs useful information must first be extracted from the vacancies and stored for use by the matching component. For this step:
   2.1. Design a model of the features/data that are to be extracted from the stored vacancies.
   2.2. Set up a database in which to store the extracted features/data, determined in step 2.1.
   2.3. Implement a component to extract the data from step 2.1 from the vacancies retrieved during step 1.3 and store this information in the local database.
   2.4. Run the vacancy parser component to fill the parsed vacancy database.

3. Implement a local database in which to store curricula vitae and their (automatically) extracted features.
   A database with some decomposed CVs is required to test the matching component when matching CVs to vacancies. To achieve this:
   3.1. Design a model of the features/data to be extracted from the CVs.
   3.2. Set up a database in which to store the CV text and the extracted features/data, determined in step 3.1.
   3.3. (Manually) insert some sample CVs into the CV database.
   3.4. Implement a component to extract the data determined during step 3.1 from the CVs stored during step 3.3 and store the results in the local database. Off-the-shelf software to decompose CVs into their constituent parts may be used.

4. Automatically cluster decomposed vacancies into groups of related vacancies.
   To aid in reducing the amount of irrelevant search results, reduce the search space and to improve the overall quality of search results, the decomposed vacancies should be clustered according to similarity. To achieve this:
   4.1. Implement a component to cluster the decomposed vacancies from step 2.4.
   4.2. Cluster the vacancies from step 2.4.

5. Design and implement an algorithm to match vacancies to CVs and vice versa.
Given a vacancy, the prototype system should suggest relevant CVs and vice versa. Matches should be scored by some relevancy measure.
5.1. Determine what features should be used by the matching algorithm.
5.2. Design and implement the matching algorithm.
6. Design and implement an algorithm to rank search results.
   When querying the vacancy database, results should be scored/displayed based on some relevancy measure.
6.1. Determine which features to use for ranking search results.
6.2. Implement a ranking component that uses the features from step 6.1.
7. Test the implemented components.
   Perform experiments on the components implemented in steps 1 through 6 to determine their performance. The components to be tested are the vacancy retrieval, vacancy parser, vacancy clusterer, CV parser, matcher and result ranking components.

Outside the Scope of this thesis
Decomposed CV texts are required by the matching algorithm. The CVs will be decomposed manually or by using off-the-shelf software. There is a wide variety of high quality commercial offerings available to achieve CV decomposition. It is therefore outside the scope of this thesis to create such a component.

1.3 Methodology
To successfully complete the research challenges set out earlier, a combination of literature research in the field of information retrieval, vacancy make-up (important aspects) and a database of vacancies/CVs are required.

A literature study was undertaken to find the state of the art in the field of information retrieval, specifically concerning metadata generation, document clustering and document retrieval techniques. This not only provides a theoretical basis, but aids in building a picture of what (meta)data will need to be extracted from the vacancies/CVs to cluster, match and retrieve them from the database.

A database of vacancies is required to provide a test set for experimentation. A tagged test set would be ideal, but a freely available solution was not found. An untagged vacancy database will be created by scraping vacancies from a number of popular Dutch job boards. A collection of at least 5,000 – 10,000 vacancies is desirable as this should enable enough variety in the vacancies to test the components of the prototype system.

In addition to the literature study, interviews will be held with recruiters to ascertain if/how there current needs are met by internal/external vacancy/CV databases such as Monster. The recruiters' input will be used as the basis of the elements to match the vacancies and CVs on, i.e. to discover the key vacancy terms.
Lexicons can greatly assist data extractors to help locate and extract relevant information. Some lexicons will be created by extracting the fixed value lists (such as industry on a job board search form) of key vacancy terms, whilst others will be obtained from relevant organisations. Co-occurrence statistics from the vacancy texts will be used to find relevant terms for the lexicons and data extractors.

Data extractors will be created to extract the key vacancy terms from the vacancies stored in the vacancy database. The data extractors will use a combination of NLP and regular expression parsing techniques to discover and extract the desired information.

Besides the database of unparsed vacancy texts, a database of parsed vacancies is required by the clustering and matching algorithms. The parsed vacancy database will be created using the unparsed vacancy database as input, and transforming it using the created data extractors.

A database of CVs is required as a test set for experimentation. A small test set will be created by asking volunteers to donate their CV as well as searching the Internet for publicly available CVs. As the test set of CVs will be small, they will be decomposed manually. As many commercial products already exist to decompose a CV into its constituent parts, a component to do this will not be developed.

1.4 Motivation

The purpose of job boards should be to provide job seekers and job advertisers with a set of tools that enables both parties to reach their employment goals as quickly and efficiently as possible. Currently, this is not the case, as bad quality search results (irrelevant, missing and results not being sorted by relevancy) hamper both parties in achieving their goals. One reason for this can be attributed to the large influx of (new) vacancies on job boards. To give an idea of the problems faced by job seekers, statistics gathered by monitoring vacancy feeds of five Dutch job boards over a two week period in April 2011 showed a total of 9043 vacancies being added. This total includes potential duplicate vacancies across job boards and the reposting of vacancies by job advertisers to get them extra exposure in the RSS vacancy feeds or higher up the results list.

Ex-colleagues from the recruitment firm MHI provided additional food for thought by telling me about some of their problems when using job boards for recruitment purposes. Also relevant are problems they encounter using (internal) recruitment databases which often lack advanced search options to support them in obtaining relevant curricula vitae with the minimum amount of effort.

Additional motivation for this topic was found from personal experiences and frustrations in navigating through hundreds of vacancies on job boards trying to find a suitable new job.
1.5 Thesis Outline

This thesis is structured in six parts: “Theoretical Background”, “Architecture and Algorithm”, “Implementation and Testing”, “Experiments and Results”, “Conclusion” and “Appendices”. In turn, these parts are structured in chapters.

The theoretical background part first discusses various issues troubling job boards and their users, suggested improvements and related work. The first chapter discusses similarities and differences between digital and printed vacancies. The second chapter summarises discussions with recruiters. Here the focus is on obtaining insights into the needs of recruiters when using e-recruiting systems. The good, the bad and the ugly in used systems are highlighted along with desired features currently found to be missing/lacking in quality.

The second part: “Architecture and Algorithm” describes the model of the proposed research and the algorithms that are applied to implement this model.

In the part: “Implementation and Testing” the implementation of the vacancy retrieval and matching prototype will be described. This includes all tools necessary to create the proposed prototype.

The fourth part: “Experiments and Results” describes the experiments that are performed on the working prototype and presents/comments on their results.

The fifth part: “Conclusion” provides a conclusion of all work done for this research project. A look is taken to see if the research challenges outlined in chapter 1.2 have been met, possible improvements are suggested and recommendations are made for future research. This part concludes the main body of the thesis.

In the final part the appendices can be found. The database tables and their structures are presented, a list of links to various resources used and/or discussed within this project and the stop words list used internally by the parser.
Information retrieval systems exist in many different shapes and sizes, catering to different needs and each attempting to solve a different piece of the retrieval puzzle. With the vast quantity of information available and being generated on a daily basis, the future of information retrieval is looking at applying semantics to help manage the information. Semantics have the possibility of greatly aiding the search experience, by improving retrieval (recall), result ranking (precision) and data/result visualisation. While using semantic information may seem as straightforward as making domain specific knowledge available to the IR system, this is also where the main problem of semantic search begins.

A key technique to improving search is through the use of metadata, as it enables us to describe the contents of a resource and use this information for purposes such as placing the resource into categories.

To generate metadata, resources can be annotated manually, but this is a time consuming process that is error prone and the keywords chosen to annotate the resource are likely to be subjective in nature. Also, a user searching resources annotated by others may not achieve the desired results, as he will search using his own knowledge/vocabulary of the searched domain. Limiting the choice of keywords that resources can be annotated with, by forcing the user to select the keywords from a domain specific dictionary, may alleviate the situation, but this has the possible drawback of forcing users to choose imperfect annotations when better descriptive keywords are missing from the dictionary.

Automatic metadata generation may provide a solution, but the quality of the metadata is highly dependant on the the quality of the metadata generators.

Currently, a lot of automatic metadata generators focus on generating keywords that summarise the main concepts in documents. Newer approaches attempt to extract contextual information from the documents depending on the resource type.

C-PANKOW [Cimiano et al] is a context-driven automatic semantic annotation tool that has the major benefit of working unsupervised, i.e. without having to train the system with a hand labelled data set, and works without making assumptions on document structure. This is a huge benefit over supervised/machine learning approaches. “First, machine learning approaches inducing extraction rules for each concept from training data typically do not scale to large numbers of concepts as Semantic Web ontologies consist of. Second, in order to annotate with respect to a few hundred concepts, a training set in the magnitude of thousands of examples needs to be provided, an effort that probably not many people are willing undertake. Third, machine-learning based approaches rely on the assumption that documents have a similar structure as well as content, an assumption which seems quite unrealistic
considering the heterogeneity of the current web” [Cimiano et al].

A disadvantage of this and most other automatic metadata generation techniques is the requirement for a domain specific ontology. Ontology creation is a vast and time intensive undertaking, but enables the possibility to find associations between concepts and the ability to calculate similarity/distance between concepts.

Of special note is how metadata changes over time. Many articles focus on the use and benefits of using metadata but fail to recognise how changes in language usage will effect the quality and accuracy of metadata as a language evolves. If metadata is seen as a static element added to describe resources, then changes in word usage/meaning will have a direct impact on retrieval. Automatic metadata generation may solve part of this problem, as it makes it possibly to apply new metadata to all resources in a library using updated metadata generators. It will however likely be time consuming to regenerate metadata for all resources in a large library, though the benefits of this undertaking may outweigh the extra effort.

As mentioned earlier, manually annotating resources takes a lot of manual effort which can be alleviated using automatically generated metadata. A possible alternative to using metadata is to use query expansion during search. Query expansion enables a user to input query terms as known to the user, and lets the IR system expand the query terms to search using related keywords and concepts. To enable use of related concepts, domain specific ontologies are required. These ontologies could also be used for metadata generation.

The major benefit of query expansion is a shift in paradigm from pure keyword matching to matching on a semantic/conceptual level. Experiments done by [Malaisé et al], investigating the interaction between automatic annotation and query expansion on a large cultural heritage archive, showed improved search results.

[Schumacher et al] created a fact retrieval engine that “focuses on the knowledge base in order to find the perfect answer” and performed query rewriting by augmentation, using context, thesauri and user assigned names. Given a suitable knowledge base, a CV/vacancy matching system could apply a fact retrieval engine to extract desired facts from the CVs/vacancies, however, as is mostly the case with modern approaches, a knowledge base is required. Unfortunately for the CVs/vacancies this is not readily available.

[Scheir et al] attempt to employ techniques from associative information retrieval to find relevant material, even if no semantic information is available for these resources. In the network, concept nodes are associated by means of semantic similarity, and document nodes are associated by means of textual similarity. Concepts and documents are associated if the concept is used to annotate the document. They compute textual document similarity by taking the twenty-five highest term weights. While this may work for medium to large sized documents, it is unlikely that key terms will occur multiple times in a short document. Directly applying this to the requirements section of vacancies, it is noticed that the requirements are mentioned only once, rendering this approach useless.

[Chirita et al] created a system called P-TAG, creating personalised annotation tags. They
found that using latent semantic indexing would likely provide better results for calculating document similarity than cosine similarity, but that latent semantic indexing is computationally too expensive. Through experimentation they found that cosine similarity performed well for large documents and term frequency provided the best metric.

Once an IR system has executed a search query, displaying results ranked in order of relevance is possibly the most important part of an IR system. Users hope to see the desired search result on the first few pages of the results list. Displaying the results that the user is seeking on anything past the 2nd or 3rd page will likely cause many users to modify their query in the hope of not having to first review more pages of less-/irrelevant results. Results appearing further down the list have been ranked lower, so users may also decide not to view results past the first few pages as they may be deem these results to be irrelevant.

There is little to no linkage between resources on job boards making context aware ranking difficult. [Aleman-Meza et al] believe that just as ranking of documents is a critical component of today’s search engines, ranking of relationships will be essential in tomorrow’s semantic search engines that would support discovery and mining of the Semantic Web. [Aleman-Meza et al] take the approach of making the relevancy of results depend on a context defined by the user. Results should therefore be ranked closer to the actual domains of interest of the user. They perceive the requirements to rank semantic relations to be:

1. semantic associations need to be filtered according to their perceived relevance (user's domain of interest),
2. a customisable criterion.

This again raises the issue of having to create an ontology, this time to describe the user's preferences. However, given such an ontology and a domain specific ontology for the data being searched, it enables the user to search an information store using terms which are personal to the user, whilst the IR system is able to modify the query terms using the ontology to search an information store using related terms to perform a more universal search.

[Basile et al] note that relevance computation is primarily driven by a basic string-matching operation. They suggest an improvement using N-levels Document Representation, combining keyword with semantic information. Currently the big problem with this approach is that this requires a long pre-processing time and results in substantially slower query response times (+40%). The biggest disappointment of their approach was the results for relevance and mean average precision. The experiments approached the results of stemming alone, with only one experiment having a marginally higher score for relevance.

After evaluating a variety of algorithms and post-processing techniques for finding new articles on the web relevant to news broadcasts, [Henzinger et al] found that “filtering articles by similarity to the caption text and similarity with each other gives a large improvement in precision”. Ranking results by similarity between results therefore seems to provide a possible basis for improving result ranking and removing irrelevant results.

It is obvious that more work needs to be done on relevancy ranking. However, ranking the
results in order of relevancy may not be the only means of displaying search results in an optimum fashion. Data visualisation techniques and techniques to browse through data/results may offer great benefits.

Most current search engines display their search results as a list ranked by some relevancy measure. Figure 3 shows a classic example from the Google search engine. Each result has a title, displays the link to the result and a snippet of text, which is a sentence/paragraph from the resource that closest resembles the query or a text chosen by the website administrator. The question is whether this is the optimum way to display results.

![Google Search Results](image)

**Figure 3: Search Engine Result Presentation**

A number of different approaches, such as concept maps and mind maps, have appeared that attempt to facilitate visualisation of information through knowledge representation.

DeepaMehta [Richter et al] for example applies the only ISO standardised approach, topic maps. DeepaMehta applies lessons learned on how the human mind stores and associates knowledge to create a more natural search experience. Of special relevance, when solving complex problems, is having access to problem-related knowledge, complexity reduction (through clustering) and avoiding cognitive overhead. This is crucial, as they found that “the number of items we can consciously deal with at a time is limited to seven plus or minus two”.

Why can't this approach of data visualisation be applied to result visualisation? [Smith et al] thought just that and created FacetMap, an interactive query-driven visualisation. Smith et al researched the differences between the dominant paradigm for searching and browsing large data stores (a scrollable list of search results in response to textual search term input) and their visual metaphor. Through a small set of experiments they found that the most desirable
situation for search is likely to be a combination of graphical and textual search.

Reviewing some commercial state of the art systems gives some insight into what is currently commercially viable outside the realms of research labs and also gives some insight into the amount of effort required to create a large scale semantic information retrieval system.

*Cognition* [Dahlgren][Dahlgren et al] claims to have spent 100 person years of lexicographical work into creating Cognition's semantic databases. This is no small feat and gives an idea of the amount of work that is required to create an expansive knowledge base.

Cognition performs ranking on a list of retrievals containing documents in which the query concepts are found together in a sentence. These results are then relevance ranked as follows:

1. Documents where query terms exactly match documents terms are ranked first.
2. Documents where some query terms match exactly, and others match conceptually ranked second,
3. Ranked last are documents with only conceptual matches to the query terms.

*ConceptMap* [Edmonds] on the other hand generates a unique signature (fingerprint) for each document which is then used to compare concepts between document and provide an efficient means for clustering and indexing.

The key features of ConceptMap include that:

- Documents are represented by a small fixed length numeric vector which is used as the signature and is generated from the concepts in a document;
- Signatures form a space in which similar documents can be detected with great computational efficiency. Documents that are similar in meaning and topic are close in concept space;
- ConceptMap includes support classes to create, index and locate signatures, and thus similar documents in log(N) time;
- The measure of similarity can be adapted to have a range of characteristics including finding near duplicates and variants of documents, or finding documents with similar topics;
- The algorithm used is parallelisable and scalable.

Speed looks to be one of the greatest advantages of ConceptMap's implementation, as processing time is claimed to grow with the log of the number of documents, compared to linearly for competitors. The question arises whether the signatures can adequately describe the documents. ConceptMap's main use looks to be in clustering documents, as the clusters ConceptMap creates can be used to reduce the size of the dataset the query is executed on. Alternatively, an entire cluster could be returned, letting the user browse the cluster.

Over the years various metrics, such as the Damerau-Levenshtein distance, Dice's Coefficient and the Jaccard similarity coefficient, have been devised to calculate the distance between two
Theoretical Background

strings. Each metric has its own strengths and weaknesses. The Damerau-Levenshtein distance calculates distance by counting the minimum number of transformations necessary to transform one string into another. It is efficient for short strings, but due to its $O(M \cdot N \cdot \max(M, N))$ complexity, where M and N are strings lengths, it is not suitable for large texts. Dice's Coefficient measures similarity based on the number of common terms between two documents as a proportion of the total number of words in both texts. The Jaccard coefficient measures similarity as the proportion of words both have in common versus the words they do not have in common.

In practice, one of the most used combinations of algorithms to create document clusters involves using the cosine similarity measure to measure document similarity and the K-means algorithm to create clusters of similar documents. This can in part be attributed to the perform of the cosine similarity measure. [Lewis et al] found that cosine similarity outperformed both the Dice and Jaccard metrics for word vector strategies.

The cosine similarity measure calculates the similarity of two vectors by measuring the cosine of the angle between them. Thus, a document must first be represented as a vector with term frequencies for each term in the document. TF-IDF is commonly used as the frequency measure as it overcomes the shortcomings TF and IDF.

When K-means [MacQueen] is used as a partitional clustering algorithm it involves building clusters of similar objects based on a cluster centroid, where the cluster centroid is the average of all objects within the cluster [Steinbach et al][Premalatha et al]. All objects within a cluster are deemed to be closer to this cluster centroid than all other cluster centroids. All objects within a cluster should be more similar to each other than with objects within other clusters, however this is not necessarily the case. Two similar objects may fall into separate clusters, even if you would expect them to fall within the same cluster. The reason for this is partially dependent on the number of clusters ($k$) and the initial cluster centroids chosen at the start of the clustering process. However, during the clustering process, cluster centroids are likely to change as new centroids are found that better describe the $k$ clusters. Fine-tuning may be required to chose an “optimal” value for $k$ and as well as choosing a good set of initial cluster centroids.

Relevant and accurate metadata looks to be a key component in increasing overall recall and precision of an IR system. While manual metadata generation may let users better describe the content, the process of manually annotating the documents is time-consuming, error-prone, prone to ambiguity and may hamper retrieval when personal annotation choices do not match the (domain) knowledge of other users. Automatic metadata generation is therefore the preferred choice. Metadata will be based on the gathered domain information and improvements in the metadata generators can be used to regenerate metadata for the entire dataset without requiring human intervention to manually re-annotate the entire dataset. This will create a uniform view for all users and helps satisfy research challenge 2.3.

Performance (both speed and accuracy) of the retrieval process can be greatly improved through document clustering. Techniques such as Mind/Concept Maps provide an innovative approach to search for- and browse through information, however their emphasis lies more on
locating data through an iterative process of browsing through results till the desired resource is found. As the desired result is closer to a recommendation system where relevant results are displayed directly, these techniques do not offer the ideal results. K-means will be used as the clustering algorithm as its track record shows that it offers good quality results at a relatively low computational cost. Another tried and tested technique, the cosine similarity measure, will be used to calculate document similarity. This should also provide good results for relevancy ranking as dissimilar documents will be ranked low as opposed to a high ranking for similar documents. Research challenges 4, 5 and 6 should benefit from this approach.

2.1 Related Work

Quite unexpectedly the majority of the research found regarding e-recruiting comes from the fields of marketing and psychology. Whilst seeming problematic at first, reading this research provided insights into the problems with existing e-recruiting sites and techniques.

[Nelson et al] note that the reason that few studies in the domain of advertised job skills in IT/IS may be due to the use of an expert panel to independently analyse and classify texts, where later meetings must be held to resolve any differences. This issue should be resolved by advances in Internet retrieval software and statistical software packages.

In “A Comparative Study of IT/IS Job Skills and Job Definitions” [Nelson et al] outline their approach in their ongoing study where they are evaluating “the skills required for IT/IS jobs by analyzing a broad set of online jobs”. They use job advertisement search engines to retrieve only IT/IS vacancies, and then parse the contents. “The jobs and their associated skills are extracted based upon skill classifications described in previous research”. Latent Semantic Categorization (LSC) is used to extract text artefacts (e.g. advertised job skill text). Whereas traditional categorical analysis analyses selected keywords, LSC analyses entire paragraphs to group job skills that co-occur. The resulting clusters group jobs by their similarities in advertised job skill requirements. The ability to cluster over entire paragraphs is a major benefit of using LSC over other clustering techniques.

[Vega] views the problem of matching job offers and job search requests as a problem of semantic matching. A relatively straightforward approach using NLP techniques is taken where a knowledge base, domain specific dictionary and general dictionary are used to determine semantic closeness. Unfortunately no test results or the quality of the system is discussed.

[Malinowski et al] attempt to answer the question “How can the selection of individuals be supported or improved with IS support?”. Their approach is using a recommender system to find a person-job (P-J) fit. Besides the use of the basic information in the CV and job description, they believe that the preferences of the recruiter and candidate are also part of the equation. Expanding on their earlier work [Keim] discusses a framework to match human resources using a recommender system for decision support and to reduce information
overload. System training was performed using 250 ratings and testing using 50 ratings from 30 CVs and 10 job profiles. Validation performed on 100 job profiles showed 88% of the classifications as accurate within a 95% confidence interval. Whilst these results are very impressive, it would be interesting to see how the system performs on a significantly larger/real world dataset.

[Rafter et al] also sees merit in using recommendations. Their system however bases its recommendations on what similar users have previously liked and uses a cluster-based automated collaborative filtering approach “as it can exploit transitive relationships between profiles that otherwise do not overlap”. For this, high quality profiles are required. The authors note this was their first step and that a more extensive evaluation is required, as they had no automated method to confirm their results.

Integrating job search with social networking looks to be the next logical step in job search, as professional networking sites such as LinkedIn and Xing contain extensive professional user profiles that could easily be used for data mining and matching purposes. [Keim] sees the emergence of job boards combining with social networks requiring new mechanisms to search these mixed structures.

Whilst working on this thesis the large players on the recruitment/social networking market did not stand still. Monster released a Facebook application called BeKnown. The application is an extension to Facebook providing a professional social networking environment within Facebook, whilst still being separate from the private Facebook network. In recruitment circles BeKnown is being seen as an attempt by Monster to create a rival to LinkedIn. The job feed in BeKnown can be populated by a user query, but BeKnown also recommends vacancies based on the profile the user creates within BeKnown. Details of the matching algorithm are undisclosed.

From the recruiter's point of view BeKnown has a major problem when searching for candidates. Unless a candidate is already connected to the recruiter, a search for candidates meeting certain skills will not return said candidate. Passive recruitment is therefore difficult or unlikely, meaning that the value of this service to recruiters is questionable.

LinkedIn's long awaited job recommendation system was also launched. This is a two part system consisting of a job recommendation email and a job search portal. Figure 4 shows an example of their recommended jobs email. What is immediately apparent is the lack of coherency between the recommended jobs. It seems that job recommendations are being based in part on the user's social network and not only on the skills/previous employment of the user. This is a rather puzzling choice as the user's skill set should surely outweigh the industries/skills of the people in the user's social network.

The second part of the job recommendation system is a simple job search portal. Here the user can enter a user defined query, but more interestingly, below is a list of recommended jobs (figure 5). These recommendations look to be based on the skill set and previous positions held by the user.
Figure 4: LinkedIn Recommended Jobs Email
2 Theoretical Background

Figure 5: LinkedIn Job Search Portal

2.2 Interviews with Recruiters

Interviews were held with two recruiters from two different companies targeting two different sectors, to get some insight into how they use job boards, what features they find work well, which require improvements and which features are missing.

CVs are a vital ingredient for recruiters to do their work and a key source for new CVs are job boards. Job seekers upload their CVs to the job board's database, making them available to the recruiters. Recruiters can search this database, but there is little to no intelligent search functionality implemented to assist the recruiters in locating interesting CVs. One reason for this is that uploaded CVs are not automatically decomposed into their constituent parts. Recruiters are therefore limited to basic keyword search when looking for CVs. Job seekers don't always leave their CV behind on job boards, but create a (complete) personal profile instead. The problem is that creating such a profile is a time-consuming process for the job seeker, especially if this needs to be done for numerous job boards, and so is used little and may not be complete. If implemented properly, recruiters could be presented with finer and/or more advanced search capabilities, but till now this does not seem to be the case.

Applicants on the specialist job board www.oilcareers.com must recreate their (entire) CV on the site. I was informed that this is one of the few job boards with more advanced search capabilities, but this is only possible because the applicants have had to create a personal
profile, manually specifying details from their CV, using the site's forms. Effectively the site is using the applicants to parse the data instead of using software to automatically decompose the CVs. As a result, the site's administrators can focus on the retrieval aspect of the site.

With the enormous number of new CVs being uploaded to job boards on a daily basis, recruiters need a way to be notified of new CVs which are relevant to them. A common mechanism which is employed to achieve this is the ability to store queries on the job board which are then run on a schedule, notifying the recruiter of matching new user profiles. A problem with this approach is that the underlying search engine performs a basic keyword search which is unable to achieve a high precision and recall rate.

Results from large sites such as Monster have been deteriorating due to the large influx of CVs and the lack of intelligence in the search/ranking algorithm. When searching with multiple keywords unrelated matches are often found. This may be attributed to a lack of checking of coherence between query terms. A search query of “software architect” (with or without boolean operators and/or quotation marks) may return CVs containing results such as, “solutions architect”, “naval architect” and “workflow architect”. The recruiters note that it appears that this may be due to some job boards no longer responding correctly when performing boolean search, but they could not verify this.

Atlas4Jobs, a specialist job board created by Atlas, a Dutch recruitment company, performs some form of advanced matching, as notifications of new automatic matches are of high quality. How this is achieved could not be disclosed.

A nice feature which Atlas4Jobs contains is query term suggestions. Suggestions are presented as an auto-complete list in a similar fashion when searching using the Google search engine.

Search term suggestion is also available through a taxonomy of function titles. OilCareers' taxonomy is extensive, but is not used often by the recruiter I spoke to. Some other sites offer some basic taxonomies.

When asked what key feature the recruiters are missing from job boards they both agreed that a quality aspect to the search results is required. Currently it is unclear if there is any logic to the presentation of results. A better scoring/ranking system for result presentation is therefore highly desirable.

When asked if the recruiters would like a system which when given a new vacancy outputs the best matching candidate(s) in the database, the answer was not necessarily. As long as the system could provide an accurate list of e.g. top 20 matches, which are ranked and scored, then the recruiter only needs to spend some time reviewing which of those candidates the recruiter thinks is the most suitable for the vacancy. Their desire seems to focus more on a decision support system than a tool that aims to take most of the work out of their hands. One reason for this is that the recruiters often have a lot of intangible information at their disposal (e.g. social skills) which would have to be entered, quantified and considered in a recommendation system that aims to replace most of the recruiter's searching and matching skills.
3 Architecture and Algorithm

Creating a prototype system to retrieve, automatically decompose and match vacancies against CVs requires combining (advanced) parsing, natural language processing, clustering and retrieval techniques. Datasets (CVs and vacancies) are required to develop and test the system, whilst efficient and high-quality algorithms are required to process the information held within the datasets.

This chapter describes the different components which together form the prototype system. First an overview of the overall architecture is given. Secondly, the visual features used to present different content within digital and printed vacancies are discussed, along with their similarities and differences. Lastly, the various components of the prototype are each discussed individually including discussion of sub-components such as lexicons and a Dutch stemmer.

Diagram Legend

Fig 6: Diagram Legend
3.1 System Architecture

The prototype is made up of six key components (a vacancy retriever, vacancy parser, vacancy clusterer, curriculum vitae parser, matcher and user interface) that each perform a different role in the chain of tasks leading to a match between a vacancy and a CV or presenting the results of a query on the vacancy database. Lexicons are present to assist the text parsers to locate and extract relevant data. Various databases are used along the way to store vacancy and CV texts, information extracted from them and the matches.

1. The **vacancy retrieval component** is responsible for retrieving vacancies from the Internet, using RSS vacancy feeds to locate the (new) vacancies, and storing the retrieved vacancies in the local database for use by the vacancy parser component.

2. The **vacancy parser component** parses the vacancies, extracting desired text using the provided lexicons, and stores the parsed vacancies and extracted data in a database for use by the vacancy clustering and matching components.

3. The **vacancy clustering component** takes the parsed vacancies in the parsed vacancy database and partitions them into $k$ groups of similar vacancies using the cosine similarity measure and K-means clustering. The resulting cluster information is appended/updated in the database entries.

4. The **curriculum vitae component** parses the CVs provided to the system, decomposing the CV into its constituent parts and storing this data in the CV database for use by the matching component.

5. Given a curriculum vitae, the **matching component** returns related vacancies sorted by some relevancy measure. When provided with a vacancy, the matching component returns related CVs sorted by some relevancy measure.

6. The **ranking component** takes the results from keyword search and attempts to remove irrelevant and low quality results.

7. A **user interface** is required to perform keyword search on the vacancy database generated by the vacancy parser component, but also to view vacancy-CV matches or to alter system preferences such as the dictionary.
Figure 7: System Architecture Component Overview
3.2 Comparing Vacancy Visual Features: Digital versus Printed Vacancies

Vacancies contain a variety of features which can be extracted. However, which features are useful and should be extracted and how can they be found? Research into the make up of digital and printed vacancies should give clues into the visual cues used to describe various features, but should also help create an overview of which features are (commonly) found.

It is clear when comparing printed job advertisements to their digital counterparts that there is a disparity between the two. [Marchal et al] note that “internet sites have an effect on the content and format of the information presented in the job ads”. The presence of predetermined input fields may also encourage their usage.

Table 2, taken from [Marchal et al], shows the results from a chi-square test highlighting some differences between printed and digital job advertisements.

Table 2: Percentage of Ads in the Press and the Internet Mentioning the Following Criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Press (N = 800)</th>
<th>Internet (N = 400)</th>
<th>Significance of the differencea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>62</td>
<td>81</td>
<td>***</td>
</tr>
<tr>
<td>Experience duration</td>
<td>22</td>
<td>41</td>
<td>***</td>
</tr>
<tr>
<td>Education level &gt; Bac.</td>
<td>48</td>
<td>61</td>
<td>***</td>
</tr>
<tr>
<td>Foreign language</td>
<td>25</td>
<td>32</td>
<td>**</td>
</tr>
<tr>
<td>Salary mentioned</td>
<td>22</td>
<td>49</td>
<td>****</td>
</tr>
<tr>
<td>Salary amount</td>
<td>5</td>
<td>30</td>
<td>****</td>
</tr>
</tbody>
</table>

* In this table and in the following ones, statistical significance was assessed with chi-square tests. * p < .05; ** p < .01; *** p < .001.

These (and other) standardised markers are easily picked up and used by search engines to perform matching calculations. Kuhn and Skuterud notice that in using standardised markers, “non-observable” features, such as career paths or particular skills are adversely effected [Marchal et al].

There are also differences in job advertisements based on country. “British ads characteristically contain very few markers allowing recruiters to select candidates by profile, but offer precise information on salary (considered to be an essential benchmark), benefits and job-status. French ads are, on the contrary, marked by the diversity and profusion of criteria
aimed at screening job-seekers prior to accessing the offers, whereas the job-seekers themselves have relatively little information on the employment conditions proposed” as displayed below in tables 3 and 4 [Marchal et al].

### Table 3: Percentage of Ads Mentioning Information Markers in France and the UK

<table>
<thead>
<tr>
<th>Marker</th>
<th>France (N = 400)</th>
<th>UK (N = 400)</th>
<th>Significance of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary amount</td>
<td>30</td>
<td>61</td>
<td>***</td>
</tr>
<tr>
<td>Additional benefits</td>
<td>7</td>
<td>36</td>
<td>***</td>
</tr>
<tr>
<td>Full/part time</td>
<td>6</td>
<td>27</td>
<td>***</td>
</tr>
</tbody>
</table>

* p < .05; ** p < .01; *** p < .001.

### Table 4: Percentage of Ads Mentioning Selection Criteria in France and the UK

<table>
<thead>
<tr>
<th>Criteria</th>
<th>French ads (N = 400)</th>
<th>British ads (N = 400)</th>
<th>Significance of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>20</td>
<td>1</td>
<td>***</td>
</tr>
<tr>
<td>Education</td>
<td>73</td>
<td>27</td>
<td>***</td>
</tr>
<tr>
<td>Experience</td>
<td>81</td>
<td>75</td>
<td>*</td>
</tr>
<tr>
<td>Foreign language</td>
<td>32</td>
<td>4</td>
<td>***</td>
</tr>
</tbody>
</table>

* p < .05; ** p < .01; *** p < .001.

The difference in content and visual layout between printed/digital job advertisements could suggest that there is a difference in purpose between the two. This may have to do with the cost of running (national) advertising campaigns in printed media or with the limited space and styling capabilities that are available to the advertisers. Whilst there is an overlap between the information presented in printed/digital job advertisements, printed advertisements seem to focus more on informing the reader about the presence of a vacancy within a company, whilst digital advertisements focus more on presenting specific details about the vacancy such as the desired skill set of applying candidates.

### Digital Vacancies

Digital job advertisements come in all shapes and sizes and their make up seems to be determined in part by the target audience of the job board.

The more general the vacancies of a job board are (e.g. Monster), the more freedom the
advertisers appear to have in the presentation of their vacancies. Sites such as Monster let advertisers style and layout their vacancies as they wish. This is likely to let advertisers express their corporate identity through the vacancy. A company logo, colour scheme, what information to include, a job summary box, etc. are all free to be used by the advertiser. A visual inspection of randomly chosen vacancies shows that (large) job agencies and companies are especially fond of presenting their vacancies in a format that displays some form of corporate identity.

At the other extreme are job boards such as AcademicTransfer and VKBanen. Inspection of vacancies on these sites shows that all vacancies are presented in the same way. Advertisers seem to have no say in the visual presentation of their vacancies. The only distinction they can make is in the content. Some may see this as a disadvantage, as this will make it more difficult to make a vacancy stand out from the rest, but presenting all the vacancies using the same visual layout and style does mean that advertisers are likely to put more effort into the content of the vacancy to make them stand out. Academic job boards are visually sober, but make up with a simple, well defined and clear structure, making them easy to navigate.

**How do job boards catch the attention of job seekers?**

Table 5 shows an overview of some of the techniques employed by advertisers on different job boards to draw attention to various bits of information.

Table 5: Overview of Visual Eye Catchers on Job Boards

<table>
<thead>
<tr>
<th>Site</th>
<th>Logo use</th>
<th>Colour use</th>
<th>Layout &amp; Styling</th>
<th>Summary Box</th>
<th>Corporate Identity</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic Transfer</td>
<td>Yes</td>
<td>Function Title, Headings</td>
<td>Fixed</td>
<td>Key features</td>
<td>Only Logo</td>
<td>URLs to advertiser's own website, vacancy and email address</td>
</tr>
<tr>
<td>VKbanen</td>
<td>Yes</td>
<td>Function Title, Headings</td>
<td>Fixed</td>
<td>Key features</td>
<td>Only Logo</td>
<td>Google Map of vacancy location</td>
</tr>
<tr>
<td>Monsterboard</td>
<td>Yes</td>
<td>Rarely used</td>
<td>Fully customisable</td>
<td>No</td>
<td>Only Logo</td>
<td>URLs possible → show up as a link</td>
</tr>
<tr>
<td>JobTrack</td>
<td>Yes</td>
<td>Function Title</td>
<td>Fixed</td>
<td>No</td>
<td>Only Logo</td>
<td></td>
</tr>
<tr>
<td>Nationale Vacature Bank</td>
<td>Yes</td>
<td>Function Title</td>
<td>Fully customisable</td>
<td>Optional</td>
<td>Fully customisable</td>
<td>Headings/Job title highlighted with larger bold text in the default layout</td>
</tr>
</tbody>
</table>
Printed Vacancies

Just as with digital job advertisements, printed job advertisements also come in various shapes and sizes, but very generally, there are either small or large (full page) printed advertisements. Besides the obvious cost of advertising, the choice of advertisement size is also determined by the page size of the publication in which the advertisement is to be printed (e.g. A4 journal, tabloid format newspaper).

Smaller job advertisements tend to contain very little information on the actual vacancy. Generally these advertisements only convey the job title, number of FTEs, the company slogan and/or a catchy phrase. The attention within the advertisement is focussed on the function title. Highlighting the title (in bold and/or a colour), placing it in a (coloured) bounding box, bullet pointing it, are but some of the techniques encountered in printed media that aim to draw the attention of the viewer to the function title.

Larger job advertisements often contain much more information than the smaller advertisements and can portray more of the characteristics of digital job advertisements. Like their digital counterparts, these advertisements often contain specific information regarding skill requirements, information on the company and contact details for further information/applying for the vacancy. Besides the eye catchers used by small advertisements, the larger display space available to large advertisements also gives more opportunity to employ corporate styling.

3.3 Vacancy Retrieval Component

The vacancy retrieval component (VRC) is tasked with obtaining (all) vacancies from job boards using available RSS feeds and storing the retrieved vacancies in a local database.

The original idea to retrieve vacancies from job boards was to create an agent that would interact with job board websites, locating and populating the vacancy search form, and using it to obtain a list of vacancies that can subsequently be retrieved and stored in a local database. Initial research into the source code of various job boards' homepages revealed that creating a generic agent to crawl/scrape these sites would take a lot of work compared to the returns offered.

The main goal of this component is to retrieve vacancies from selected job boards. As most job boards advertise their new vacancies using RSS feeds that contain links to the new vacancies, the act of locating and retrieving the vacancies can be simplified greatly by using the available RSS feeds. As a result the choice was made to use the RSS vacancy feeds that most job boards contain to obtain the vacancies.
3. Architecture and Algorithm

Figure 8: Vacancy Retrieval Component

The first part of this component periodically checks supplied RSS feeds for updates. When the feed contains new entries the details including the URL of the vacancy are stored in the database.

The second part of this component periodically checks the database populated by the RSS feeds to identify new vacancies which have not yet been retrieved. The new vacancies are retrieved in small batches to ensure that the job boards are not overloaded by too many requests and are subsequently stored in the unparsed vacancy database.

3.4 Vacancy Parser Component

The Vacancy Parser Component is responsible for extracting useful information, such as function title and industry, from the retrieved vacancy texts and writing these results back into the database.
Before the vacancy parser component can extract useful content from a vacancy, preprocessing takes place to “clean” each vacancy. The cleaning process is done to help remove document markup (HTML tags and scripts) from the vacancy. The markup must be removed as it will not only slow down processing of the vacancy, but at a later stage will also interfere with the document clustering component.

Outright removal of all HTML tags (and their contents) is not possible as either too much or too little content will be removed along with the HTML tags. Choices will therefore have to be made on how to handle different HTML tags depending on whether they have the potential to enclose useful data.

Once the vacancy text is in an unformatted plain text form there is still a document character encoding issue to contend with. Western web pages are generally encoded using either ISO-8859-1 or UTF-8 character encoding. To efficiently parse and compare the vacancies, it is necessary to choose a single character encoding for all stored vacancies. A simple database query checking which character encoding is used most showed that ISO-8859-1 encoding is the most prevalent. However, initial tests to transcode a random set of vacancies from UTF-8 to ISO-8859-1 encoding gave display errors. By analysing the HTML source of the vacancies and researching this issue further on the Internet it became apparent that the character encoding set by the tag:

```
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

This tag is ignored by many browsers when the character encoding is (also) sent in the HTTP headers. The result of this action is that the character encoding sent in the HTTP header is generally used (when present). As a result a search on used character encoding in the stored vacancies does not give any conclusive results. Articles on transcoding text between UTF-8 and ISO-8859-1 suggest using UTF-8 due to its backward compatibility with ASCII. All vacancies will therefore be transcoded to UTF-8 character encoding. This is to ensure that vacancies erroneously tagged with the incorrect character encoding are also converted to UTF-8.
HTML encoded characters (e.g. &amp;#39;) will be replaced by their UTF-8 counterpart and all accented characters will be replaced by their non-accented counterpart. This last step ensures that identical words spelt with or without the accents will be reduced to the same written form.

This concludes the pre-processing stage. The vacancy's contents are now passed to each extractor. Each extractor is responsible for locating and retrieving a specific type of information from the vacancy. Extractors will be created for the types of information listed in table 6.

<table>
<thead>
<tr>
<th>Pre-defined</th>
<th>Manually Created</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry</td>
<td>Industry</td>
<td>Industry</td>
</tr>
<tr>
<td>Driving License</td>
<td>Driving License</td>
<td>Driving License</td>
</tr>
<tr>
<td>Work Experience</td>
<td>Work Experience</td>
<td>Work Experience</td>
</tr>
<tr>
<td>Hours per week</td>
<td>Hours per week</td>
<td>Hours per week</td>
</tr>
<tr>
<td>Education Level</td>
<td>Education Level</td>
<td>Education Level</td>
</tr>
</tbody>
</table>

Regular expressions are used to locate information. Regular expression matching is assisted by utilising the manually created lexicons. Once all extractors are complete, the results are written into the database ready for the matcher component.

**Lexicons**

Lexicons are used to locate data within the vacancy text. Lexicons provide an easy method to locate desired data by letting the parser search for members of the lexicon. As no existing tagged dataset(s) of vacancies or lists of identifiers for locating desired data in vacancy texts are readily available, a choice had to be made on what information is required, how it will be used, and how it will be obtained.

Regular expressions (regex) offer a quick, powerful and relatively straightforward technique to either locate data in texts from pre-defined lists, or to search texts for alpha-numeric patterns.

When searching for data to extract there are two types of information, static information, such as Dutch driving license types or education level, and dynamic information, such as salary.

Where information is static, lists were compiled to match the text against. Much of the contents of the static lists were obtained by viewing the source code of the job boards' search forms, locating and extracting the lists of pre-defined search selection lists. The lists obtained from various job boards were then merged into a single list that is fed to the parser. Other lists,
such as the list of Dutch driving license types were freely available on the internet.

Where information is dynamic, it is important to have a method of locating the desired content even when the exact values are not known. For these types of information it is necessary to be able to search for identifiers which indicate the contents of the surrounding information. When searching for the salary in Dutch vacancies, the word “salaris” or “maandsalaris” (salary or monthly salary) generally preceded the value of the salary. Lists of such identifiers were compiled by hand by manually viewing a large number of vacancies.

A list of Dutch stop words is desired to reduce the search space and improve recall/precision, for the clustering and matching algorithms, by removing non-descriptive high frequency words.

### 3.5 Vacancy Clustering Component

The vacancy clustering component is responsible for grouping similar vacancies. Clustering is done in the aim of improving the efficiency of the matching algorithm. Instead of comparing a CV to all vacancies in the database, the matching algorithm will be able to select relevant clusters and only match against all vacancies contained within these clusters. This should greatly reduce the time required for a CV to be matched, as it will only be matched against a subset of all vacancies in the database. More specifically, a CV will only be matched to the subset of vacancies which should be closest to the CV, instead of matching against all vacancies in the vacancy database.

There are different methods and identifiers to cluster vacancies on. One method is to only select those vacancies from the database for comparison which match some pre-selection criteria, such as a particular industry, which in turn matches the same category/categories as the CVs that are being matched against. It should be possible to implement this technique with relatively little effort, as most job boards offer the possibility to customise the RSS vacancy feed and receive the results filtered by some pre-defined selection criteria such as the industry the vacancy is advertised in.

Though it could be advantageous to receive results in this manner, there are some drawbacks. If a vacancy text contains multiple industries which the vacancy pertains to, the vacancy will turn up in each custom feed. The vacancy will then be retrieved multiple times (once per feed that it appears in), causing duplicates within the database. There is also no guarantee that the listed industries are relevant to the vacancy or even correct.

As mentioned earlier, job boards do not share a standardised list of selection values for fields such as industry. Using the industry field would therefore require creating a mapping between equivalent values in vacancies from different job boards. There could however be problems when trying to map values which overlap each other as a choice would then have to be made which mapping “best” approaches the intention of all advertisers.
To avoid this problem it has been chosen to retrieve all vacancies, without performing any filtering from the side of the job board, and to locally cluster all new vacancies once they have been parsed and stored in the database. Though this makes it more difficult to query the database to return all financial jobs (unless listed as an industry within the vacancy), this approach should make it possible to create a more accurate grouping of the vacancies.

Besides its use in reducing the size of the dataset used by the matching algorithm, clusters of vacancies could also be used to reduce processing time for free text search queries on the vacancy database.

Clustering vacancies also has the advantage of helping solve ambiguity issues. Many companies use different job titles to describe the same job. Similarly, the same job title often means different things to different people. By clustering vacancies using the job requirements text, and not the job title, vacancies will be clustered by similarity of the required skill set, irrespective of the function title. Searching for a specific job title could then return all relevant vacancies even if the function titles differ. This opens up new avenues to see how job titles...
and requirements relate.

Clustering is performed on the vacancy's requirements text, but before we can cluster the vacancies a measure of similarity between vacancies is required. Cosine similarity is used as the similarity measure.

\[
\text{cosine similarity} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}},
\]

where \( n \) is the length of the vacancy's term frequency vector, and \( A \) and \( B \) are two vacancies.

Stemming performs an important role when generating a vacancy's term vector as it has the potential to greatly reduce the size of the term vector. Stemming “is a process for removing the commoner morphological and inflexional endings from words (in English). Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems”\(^2\). A beneficial side effect of the stemming process on the term vector is its reduced memory footprint and the reduced processing time when comparing term vectors.

The best known stemming algorithm is the Porter Stemming Algorithm, by M.F. Porter, which was created for the English language. For this project a Dutch language stemmer is required. The applied stemming algorithm is an implementation of the Kraaij-Pohlmann [Kraaij et al] stemming algorithm as implemented for use in the Drupal content management system. The implementation used by Drupal contains some improvements over the original algorithm, namely:

- Including more consonants in undoubling operations;
- Correctly remove apostrophes (e.g. pagina’s, pages in Dutch);
- Correctly strip accented suffixes;
- Convert s/f to z/v when removing double vowels in the last syllable.

The term and term frequency vectors are created in three stages. During the first stage a vector of all the unique words from the requirements text is generated. The terms within the vector are stemmed and all contained stop words are removed. During the second stage invalidated terms from the global term dictionary are removed from the vector of terms. The term vector is generated using the vocabulary of the global dictionary which describes the allowed terms. Lastly, the term frequency vector and the term frequencies are normalised for the number of words in the text.

The dictionary is nothing more than a list of all stemmed terms, their occurrence frequency in the processed vacancies and whether they should be used in the term/term frequency vectors. To initialise the dictionary terms will be collected from a random sample of vacancies, e.g. the

\(^2\) http://tartarus.org/martin/PorterStemmer/
first 1000 vacancies in the parsed vacancies database.

Terms with high occurrence frequencies are unlikely to be key terms. For this reason terms occurring in more than roughly 10-15% of the vacancies (or with high occurrence counts) should be invalidated and should not be used in the term/term frequency vectors.

The vacancies are clustered using a slightly modified version of the tried and tested K-means algorithm. The standard implementation of the K-means algorithm recalculates the cluster centroid (mean) after each new observation, in this case each vacancy, is added to a cluster. Considering the current size of the parsed vacancy database (~17,000 vacancies), this could potentially lead to a huge number of calculations when adding new vacancies.

The best case scenario, leading to the least number of calculations, would be a situation where every cluster has an equal amount of members, i.e. all vacancies are evenly distributed between all clusters:

$$ nk + \sum_{n=2}^{nk} \frac{(n-1)nk}{2} + 1 $$

The worst case scenario would be if all vacancies are added to one cluster, leaving all other clusters untouched. This would result in:

$$ nk + \sum_{n=2}^{n} \frac{(n-1)n}{2} + 1 $$

calculations, where \( n \) is the number of vacancies to cluster and \( k \) the number of clusters.

Given the above formulas for calculating the best and worst case scenarios we discover that the number of calculations performed during the worst case scenario stays fairly constant when varying the number of clusters, whilst the best case scenario has the most to gain by achieving an even distribution of cluster sizes (see table 7).

A problem with clustering is how to achieve an even distribution of cluster sizes. For one, this requires that there is an even distribution of each type of vacancy as well as a similar quality in the requirements text of the vacancies. This is unlikely to be the case. There is no easy way to figure out the optimum distribution, as we do not have a priori knowledge of the vacancies which are yet to be clustered.

It could be argued that we could try to choose the optimum initial cluster centroids by basing their selection on a combination of facts stored within the vacancies, such as the industry and job title and their combined occurrence within the parsed vacancy database. However, this approach would only help us identify a possible better starting point for the clustering algorithm, it does not guarantee that the clusters will be more uniformly distributed.
Table 7: K-means: Number of Operations

<table>
<thead>
<tr>
<th>k</th>
<th># vacancies</th>
<th>best case</th>
<th>worst case</th>
<th>best case 250</th>
<th>worst case 250</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1051</td>
<td>2,000K</td>
<td>194M</td>
<td>181K</td>
<td>1,7M</td>
</tr>
<tr>
<td>20</td>
<td>1051</td>
<td>517K</td>
<td>194M</td>
<td>361K</td>
<td>1,7M</td>
</tr>
<tr>
<td>40</td>
<td>1051</td>
<td>173K</td>
<td>194M</td>
<td>722K</td>
<td>1,8M</td>
</tr>
<tr>
<td>50</td>
<td>1051</td>
<td>141K</td>
<td>194M</td>
<td>903K</td>
<td>1,8M</td>
</tr>
<tr>
<td>75</td>
<td>1051</td>
<td>121K</td>
<td>194M</td>
<td>1354K</td>
<td>1,8M</td>
</tr>
<tr>
<td>10</td>
<td>16980</td>
<td>8,200M</td>
<td>816B</td>
<td>335M</td>
<td>3,3B</td>
</tr>
<tr>
<td>20</td>
<td>16980</td>
<td>2,000M</td>
<td>816B</td>
<td>669M</td>
<td>3,3B</td>
</tr>
<tr>
<td>40</td>
<td>16980</td>
<td>512M</td>
<td>816B</td>
<td>1,339M</td>
<td>3,3B</td>
</tr>
<tr>
<td>50</td>
<td>16980</td>
<td>328M</td>
<td>816B</td>
<td>1,673M</td>
<td>3,3B</td>
</tr>
<tr>
<td>500</td>
<td>16980</td>
<td>12M</td>
<td>816B</td>
<td>3,356M</td>
<td>16,7B</td>
</tr>
<tr>
<td>10</td>
<td>50000</td>
<td>208,000M</td>
<td>20,8T</td>
<td>8,4B</td>
<td>84B</td>
</tr>
<tr>
<td>20</td>
<td>50000</td>
<td>52,000M</td>
<td>20,8T</td>
<td>16,8B</td>
<td>84B</td>
</tr>
<tr>
<td>40</td>
<td>50000</td>
<td>13,000M</td>
<td>20,8T</td>
<td>33,6B</td>
<td>84B</td>
</tr>
<tr>
<td>50</td>
<td>50000</td>
<td>8,000M</td>
<td>20,8T</td>
<td>42B</td>
<td>84B</td>
</tr>
<tr>
<td>1000</td>
<td>50000</td>
<td>71M</td>
<td>20,8T</td>
<td>84B</td>
<td>839,4B</td>
</tr>
</tbody>
</table>

It is proposed to modify the K-means algorithm to restrict recalculating the cluster centroids till a fixed number of vacancies from the parsed vacancy database have been clustered. The last two columns of table 11 give the number of calculations when the new cluster centroids are calculated after every 250 unclustered vacancies from the parsed vacancy database are assigned a cluster. Though these figures generally show worse values for the best case scenario, the worst case scenario is many magnitudes lower. On average this should result in better execution times.

### 3.6 Curriculum Vitae Component

The aim of this component is to parse a CV into its constituent parts and store this information in the CV database.

Software to decompose curricula vitae, stored in a variety of file formats, into their constituent parts has been available commercially for some time now. It is not the intention of this project to reinvent the wheel. It is assumed that in a large scale test, or a production environment, off-the-shelf software will be linked to the CV component to parse the CV data. For this prototype a CV database will be created storing a few manually decomposed CVs. This approach should prove sufficient for testing purposes.
3.7 Matching Component

The matching component is responsible for scoring each parsed (vacancy, CV) pair in the database for closeness, i.e. how well a CV matches the requirements of a vacancy and vice versa. The list of highest scoring value pairs indicating the best matches. As with the vacancy clustering component, the cosine similarity measure will be used to calculate the closeness of the (vacancy, CV) pair.

Most vacancies contain a set of skills that a candidate must possess. As a complete decomposition of the vacancies to the level of recognising all skills, proficiencies, software applications, etc., will not be available due to the time required to create the required
knowledge base, matching will take place using the characteristics listed in table 8.

<table>
<thead>
<tr>
<th>Field</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function/Title</td>
<td>Medium</td>
</tr>
<tr>
<td>Education Level</td>
<td>High</td>
</tr>
<tr>
<td>Years of Experience</td>
<td>High</td>
</tr>
<tr>
<td>Skills / Proficiencies</td>
<td>High</td>
</tr>
<tr>
<td>Languages</td>
<td>High</td>
</tr>
<tr>
<td>Driving License</td>
<td>Low</td>
</tr>
</tbody>
</table>

The choice of characteristic features (table 8) is quite obvious, but also reflects the mental matching employed by the recruiters that discussions were held with. One key difference however is the inclusion of the driving license in the matching algorithm. They did not deem this as necessary as they both stated that they believed in most cases that a lack of driving license would not be an issue. I however argued that there are jobs, such as a travelling salesman or a driver, where this is a key requirement. I have therefore chosen to include the driving license in the matching algorithm, be it with a lower weight.

A perfect match for the education level is when the required level in the vacancy matches the (highest) achieved education level in the CV. However, what should be done when a candidate has a (higher) education level than the required level? This could be viewed either as an advantage or a disadvantage. Having achieved a higher level of education than required by the vacancy could indicate that the candidate is over-qualified, whereas a lower level of education may indicate the opposite. A candidate who is one step away from the educational requirements, e.g. a BSc instead of a MSc, could have their score modified to indicate this. Different values, both positively and negatively modified should be tested to determine the overall effects on the matchings.

All required and optional languages requirements are extracted from the vacancy text. No distinction is currently made between the two. This means that the languages listed in the database for a parsed vacancy including the original vacancy text:

“The candidate must be fluent in English, Dutch and Spanish. Knowledge of French and Italian would be seen as a bonus”,

would list all five languages in the text. As it is highly uncommon to come across vacancies requiring more than three languages, the weight for languages should be 1.0 for a perfect match (up to 3 languages). When the candidate has knowledge of more (or other) languages than those listed, the language score should be boosted slightly to indicate this.

Once the features in table 8 have been gathered for both the CV and the vacancy their closeness needs to be scored. All features except the requirements (vacancy)/skills (CV) texts can be scored and compared by checking if the terms match. For example, if the vacancy's language requirements is the same set (or a subset) of the languages presented in the CV.
Matching the vacancy's requirements texts against the skills presented in the CV is a different matter. The skill set within the CV is the CV text excluding personalia, hobbies and interests, etc. To score the similarity between the two the cosine similarity is determined.

The obtained similarity scores are written as a vector and can be adjusted using a weight vector defining the importance of each feature from table 8. The importance of each item gives an indication of how the weight vector should modify the values within the vacancy/CV vectors. Once the weight vector has been applied the vector's elements are summed to provide an overall score.

\[
CV \text{ Vector} = \begin{bmatrix}
\text{Title} \\
\text{Education Level} \\
\text{Years of Experience} \\
\text{Skills/Proficiencies} \\
\text{Languages} \\
\text{Driving License}
\end{bmatrix}, \quad \text{Vacancy vector} = \begin{bmatrix}
\text{Title} \\
\text{Education} \\
\text{Experience} \\
\text{Profile} \\
\text{Languages} \\
\text{Driving License}
\end{bmatrix}
\]

**Figure 13: CV, Vacancy Matching Vectors**

\[
\text{Score} = \begin{bmatrix}
\cos \text{similarity}(\text{Title}_{CV}, \text{Title}_{vac}) \\
\text{similarity}(\text{Education Level}_{CV}, \text{Education}_{vac}) \\
\text{similarity}(\text{Years of Experience}_{CV}, \text{Experience}_{vac}) \\
\cos \text{similarity}(\text{Skills/Proficiencies}_{CV}, \text{Profile}_{vac}) \\
\text{similarity}(\text{Languages}_{CV}, \text{Languages}_{vac}) \\
\cos \text{similarity}(\text{Driving License}_{CV}, \text{Driving License}_{vac})
\end{bmatrix} \cdot \begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4 \\
w_5 \\
w_6
\end{bmatrix}
\]

**Figure 14: Match Score**

The importance of values in the CV and Vacancy vectors can be fine tuned by adjusting the values in the weight vectors (W₁ to W₆).

When scoring each (vacancy, CV) pair a minimum value of the cosine similarity is desirable to indicate a match. This sets a lower threshold for the number of matching terms required to constitute a match between a vacancy and a CV. A lower threshold is also necessary to not fill the database with matches that have no real use or relevance. The value for the lower threshold will need to be fine-tuned through experimentation as there is no way to predict what the value of the cut off point should be based on the used datasets.

### 3.8 Ranking Component

Designing and implementing a new and improved algorithm to rank search results is no easy task. This is made more difficult due to the lack of domain specific dictionaries and/or
knowledge bases.

Ranking search results whilst filtering out incorrect or low quality matches is a key component to any search system to ensure that only relevant results are displayed.

![Figure 15: Ranking Component](image)

The approach taken by web search ranking algorithms, such as Google's PageRank, will not work since the stored vacancies do not link to each other. This is because the vacancy database can been seen as a document library where no links between library members is defined. Discovering features that can be used to link vacancies could potentially be exploited to cluster related search results.

When searching a job board there are two key elements that job seekers are likely to search with, firstly, a function title and secondly, a skill(set).

The function title is the key feature that vacancies are advertised on and which job seekers receive their results on. If the query is for a function title, vacancies should be grouped by title as long as the requirements of the vacancies are related. The problem is that function titles for the same position often differ between companies, so simply grouping results by function title will not be a sufficiently good method to assist ranking search results. Clusters of vacancies based on function title could provide solace.

If ranking results from a search for a particular skill(set) we look for a match between the query terms and the vacancy requirements text. The location of the skill(set) terms within the vacancy's requirements text might indicate its importance to the function. Key skill are likely to be lists before less important skills, hence the closer the match is to the beginning of the requirements text, the higher the score should potentially be.

Ranking a match for desired education level can be viewed differently by different parties. A vacancy advertised as accepting candidates with different education levels, e.g. HBO or WO\(^3\), will receive different values from different job seekers. A job seeker with a HBO degree may see this as a vacancy where the advertiser is looking for someone with at least their education

---

\(^3\) A higher education polytechnic degree (HBO) vs University degree (WO)
level (i.e. a challenging position), whereas a job seeker with a WO degree may see this as a vacancy for someone with a lower or equal education level (i.e. perhaps not challenging enough). Ranking could take advantage of this if the education level is part of the query.

Considering the search domain, there is something to be said about company names. If they can be identified within the vacancy, then vacancies can be clustered using them. An automated lookup on the company may also make additional information regarding the industry of the vacancy available. The company logo (URL to the logo) used within vacancies creates coherence between vacancies. What type of coherence would have to be deduced separately, as the advertiser could for example be a (general) job recruitment agency or a company seeking to expand its workforce in different parts of the company, but it does provide an initial entry point to identifying the company.

On the basis of the above, the results of the keyword search will be ranked by calculating the cosine similarity score between the highest scoring result and the remaining results. The results will then be sorted by the cosine similarity score, in descending order, and results falling below a certain threshold may be removed from the results list in their entirety. The user will be able to perform keyword search with or without the ranking component enabled.

### 3.9 Vacancy Database

A database of vacancies is required to test and develop various components of the system. Whilst there are many different accredited datasets, such as the TREC datasets\(^4\), available for information retrieval research and testing, unfortunately no such dataset seems to be freely and readily available for the job search and matching domain. This means creating an own vacancy database.

Running the vacancy retrieval component generates a vacancy database. The retrieved vacancies are untagged, i.e. they are not annotated with metadata describing their contents. The obtained HTML vacancies will be stored in the vacancy database excluding any external files, such as images, that they link to. The external files do not add value, as they add elements for the visual display and interaction with the vacancy. They do not contain any vacancy text.

Each entry within the vacancy database should contain the vacancy's details as advertised within the RSS feed as well as details on whether/when the vacancy has been retrieved, the RSS feed that the vacancy is associated with, and once retrieved, the vacancy itself.

\(^4\) [http://trec.nist.gov](http://trec.nist.gov)
3.10 User Interface

User interfaces are required to let users interact with different parts of the system. The figure below gives an overview of the functionality which can be operated by means of the graphical user interface. As most of the workings of the system are automated, the user interfaces are limited to searching the vacancy database, adding a CV to the CV database and reviewing the results of matching a CV to the stored vacancies. Other aspects of the system can be managed by directly manipulating the relevant database or source file(s).

![Interface Overview](image)

Figure 16: Interface Overview

3.10.1 Keyword Search

A simple interface to perform keyword search on the vacancy database is required. The search interface (figure 17) will allow the job seeker to input query terms and view the results (figure 18).

When performing keyword search, MySQL's built-in relevancy measure will be used to score and order the search results. The user will also have the option to execute the same query but have the results ranked by the ranking component described in chapter 3.8.

The layout of the search and result interfaces take inspiration from Google and a study performed by [Joho et al]. Each result will show the vacancy title, industry (if available) and a snippet of text from the vacancy. [Joho et al] found that adding the top ranking sentence (TRS) and a thumbnail of each result to the result list has a positive influence on query reformulation and relevance assessment. It is questionable whether adding a thumbnail of
each vacancy will improve relevance assessment and so will be left out. Adding a snippet in the form of the TRS is useful, but considering the results are of vacancies, the first few lines of the vacancy's description will be shown instead. This is likely to give the user an equally good indication of the contents of each result.

![Keyword Search GUI](image17.png)

**Figure 17: Keyword Search GUI**

![Keyword Search Results](image18.png)

**Figure 18: Keyword Search Results**

### 3.10.2 Curriculum Vitae Management

A simple interface is required to manage the database containing CVs. Functionality should include adding new CVs to the database and updating/removing CVs from the database.

Using a data entry form the user should be able to insert/update the CV decomposition data, i.e. education level, languages, skills (*CV Text* in diagram below), function titles, driving license, years of experience and an identifier for the vacancy.
### Manage CV

![CV Management GUI](image)

**Figure 19: CV Management GUI**

#### 3.10.3 Manage RSS Feeds

To manage the RSS feeds an interface is desired which lets the system administrator view the existing RSS feeds, make alterations to them and add new ones.

The basic details that need to be managed are the name of the website that the RSS feed is from, the feed name, the feed URL and the date format used within the feed.
3.10.4 Manage System Back end

Using the general management interface the system administrator can manage different aspects regarding the running of the system, see interface mock-up (figure 21). Items include scheduling vacancy cluster (re-)generation, managing the dictionary and updating the stop words list.

3.10.5 Matcher: Explore CV-Vacancy Matches

Using this interface the user can view the vacancies that best match the chosen CV or (re-)start the matching process. The results shown are the results of running the matching component, matching CVs to vacancies. Using the interface should be as simple as selecting a
CV and then choosing to view existing match results or to re-match the selected CV and view the new results.

The reverse, matching vacancies to CVs has not been implemented as there are only a few CVs present to test the system with. The procedure is identical to that defined in chapter 4.6 for matching CVs to vacancies with the difference that each vacancy should be compared to all CVs. In the case of the CV-vacancy match each CV is matched to each vacancy cluster and then all vacancies within the best matching clusters.

![Manage CV-Vacancy Match](image)

Figure 22: CV-Vacancy Match Management GUI

### 3.10.6 Vacancy Parser Component Data Extractor Results

A simple interface to display a vacancy and the data that has been extracted from it (see figure 23) is desirable to view data extractor results as the data extractors are developed and tested. This interface should provide an overview of the results of each individual data extractor and show the original vacancy.
4 Implementation and Testing

This chapter describes the implementation phase of the project, motivating design choices and decisions where relevant. First, the hardware, software, tools and libraries used within the project are discussed. Next, the implementation of each component, the creation of the lexicons, and the matching and ranking algorithms are discussed in detail.

4.1 Hardware, Software, Tools and Libraries

The prototype has been created as a web service as opposed to an executable application. PHP, MySQL and an Apache web server form the core components required to run the prototype.

The combination of Apache, MySQL and PHP is a popular choice for hosting web services. Together they provide a mature, stable, tried and tested platform, able to run on a variety of operating systems. They are also free, so no licenses or expensive development tools are necessary to develop and operate an application hosted by such a system.

Platform portability and independence is a great advantage of using MySQL and PHP. Any operating system and web server capable of running a web server supporting PHP and MySQL can be used to host the service. Additionally, this approach makes the offered services accessible using any browser on any platform without having to create, compile and install a version of the prototype for each supported platform, creating a portable and platform independent system. Lastly, system speed is not a performance criterion for this prototype, so the lower speed of running the PHP code over native C or Java code is not relevant.

On the development side, creating the prototype as a web service has the major advantage of enabling a faster implementation. Data can be output (dumped) directly to a browser window, removing the need to implement a GUI to display the output when testing implemented code.

PHP was chosen as the scripting language to implement the prototype in. Benefits of implementing the code in PHP include native support for the MySQL database, regular expressions, a library to connect and communicate using the HTTP protocol (libcurl) and the many free libraries available which offer a variety of new and/or improved functionality. An added advantage of using a scripting language is that the code is compiled at runtime. Running (updated) code is merely a matter of loading the source file into- or refreshing the
browser window, saving time compiling a complete application or module. Scripting errors are displayed within the browser, informing the programmer of the location and nature of the error, so no separate or special development tools are required to locate errors. Testing of the PHP code was performed as it was developed. Each function was tested to check that it exhibited expected behaviour. This was done by calling each function using valid, as well as invalid data and checking the output. The results were output to the browser window.

WampServer\(^5\) version 2.1, a freeware Windows installation package containing Apache, MySQL and PHP was used to quickly get the back end up and running. PsExec\(^6\), part of Microsoft's SysInternals, is used to launch separate PHP threads under Windows. This is the only software that needs to be installed for the prototype to run.

Being a web application, the system hardware and software requirements are quite loose. Any Operating System with a web server capable of executing PHP code, and a MySQL database, can be used. As hardware requirements for most web servers and MySQL are low, even a 15 year old, 300Mhz computer with 512MB of RAM should work, though the minimum amount of required RAM will depend on the number of vacancies in the largest vacancy cluster. Data storage requirements are dependant on the amount of vacancies that will be stored. The initial test database came to ~650MB for 17,764 unparsed vacancies. The faster the computer and the more RAM available, the faster the system will be able to parse new vacancies, perform matchings and execute search queries.

Development and Operation Environment

The prototype has been developed on a desktop containing a Dual Core 2.6GHz AMD X2 5200+ processor, 2GB RAM and running a 32bit copy of Windows Vista Home Premium and WampServer 2.1. Additional testing occurred on a system containing a 1.3GHz Intel U4100 processor, 4GB RAM and running a 64bit copy of Windows 7 Home Premium and WampServer 2.1.

The PHP code was written using the text editor Notepad++, an advanced text editor that supports PHP syntax highlighting. Any other text editor could have been used. An IDE with PHP support, such as Eclipse, could have been used, but was dismissed due to personal preference. As code was developed and tested in small increments the loss of time locating the source of errors greatly limited the advantages of developing using an IDE.

The PHP code was stored in a local SVN server (VisualSVN server\(^7\)) and managed using TortoiseSVN\(^8\) a SVN client that integrates into the Windows file manager.

Firefox and Internet Explorer were used during development and testing to output results and test system behaviour. This choice of browsers was arbitrary.

SQLyog, a graphical interface capable of interacting with MySQL databases was used during

---

5 http://www.wampserver.com
6 http://technet.microsoft.com/en-us/sysinternals/bb897553
7 http://www.visualsvn.com
8 http://tortoisesvn.tigris.org
development and testing. It was used to quickly set up the databases, insert/modify/delete test data and view the contents of the various databases.

Some PHP methods are created to support the working of multiple system components. These have been stored in their own source files (see below). These include simple methods to format and output variables in the browser window, whilst others provide the basic connection to the database.

```
<table>
<thead>
<tr>
<th>GeneralFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>int refreshInterval</td>
</tr>
<tr>
<td>int getPageInterval</td>
</tr>
<tr>
<td>int refreshIntervalE</td>
</tr>
<tr>
<td>int getPageIntervalE</td>
</tr>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>string newline()</td>
</tr>
<tr>
<td>string printVar(int/string var)</td>
</tr>
<tr>
<td>date parseDate(string date, string dateformat)</td>
</tr>
<tr>
<td>string strip_selected_tags(string s, string tags, bool stripcontent)</td>
</tr>
<tr>
<td>string htmlDelBR(string s)</td>
</tr>
<tr>
<td>array strArrayToLower(array a, string encoding)</td>
</tr>
<tr>
<td>int getLine(string haystack, string needle)</td>
</tr>
<tr>
<td>string printArr(array a)</td>
</tr>
<tr>
<td>int indexOf(needle, haystack)</td>
</tr>
<tr>
<td>string htmlCleanup(string s)</td>
</tr>
<tr>
<td>string replaceAccents(string s)</td>
</tr>
<tr>
<td>string array2csv(array a)</td>
</tr>
<tr>
<td>array csv2array(string csv)</td>
</tr>
</tbody>
</table>
```

**Figure 24: System Wide Methods**

```
<table>
<thead>
<tr>
<th>dbFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>dbconnect()</td>
</tr>
<tr>
<td>dbdisconnect()</td>
</tr>
<tr>
<td>executeQuery(string query)</td>
</tr>
<tr>
<td>inDB(string table, string column, string value)</td>
</tr>
<tr>
<td>inDB2(string table, string col1, string val1, string col2, string val2, string operator)</td>
</tr>
</tbody>
</table>
```

**Figure 25: Basic Database Interaction Methods**
4.2 Vacancy Retrieval Component

The first step in retrieving vacancies using RSS feeds is to locate relevant RSS feeds and storing the links to the feeds, together with the job board name, a name identifying the feed and the date format used by the feed. These feeds could be stored and managed through a RSS feed database, but due to the limited number of RSS feeds that will be used during this project, the desired feed data will be hard coded.

The next step is to retrieve and parse the contents of each feed. cURL (Client URL Library), a PHP library for connecting and communicating with different servers using various protocols, is used to download the RSS feeds (figure 26a).

Once a RSS feed has been retrieved, the contents need to be parsed. As RSS feeds have a standardised structure based on XML, the DOM structure of the feed is used to locate the required data from each feed item. Retrieving each entry is simply a matter of iterating through each item tag and extracting the fields title, description, link (URL to the vacancy), guid and pubDate (publish date).

![Diagram of Vacancy Retrieval Component Information Flow](image)

*Figure 26: Vacancy Retrieval Component Information Flow*
Each retrieved entry is stored in the vacancy database if it is not already present there.

It was noticed that the RSS feeds are updated more frequently between 8am and 7pm. Therefore, the VRC monitors the feeds more frequently for updates between these hours.

The last stage of the VRC is to retrieve and store the actual vacancies (figure 26b).

For each unprocessed URL entry in the vacancy database, the vacancy was downloaded using cURL and the database entry updated with the vacancy (web page), retrieval date and the status updated to 1, indicating a successful retrieval, or -1 in the case of a failure.

Up to twenty-five URLs of unprocessed vacancies are retrieved from the database and processed one at a time. The limit of twenty-five vacancies per run was imposed to make sure that the job boards are not overloaded with requests in case a large number of new vacancies are advertised in the updated RSS feed. The twenty-five vacancy limit can be altered if desired.

Chapter 4.8 provides details on the implementation and contents of the vacancy database achieved by running the vacancy retrieval component.

### 4.3 Vacancy Parser Component

The vacancy parser component is one of the most important components of the prototype. In the next few paragraphs the process of extracting data from the vacancies is described. In brief, an unparsed vacancy is given as input to the vacancy parser component. Here, the HTML code is removed and the data extractors run. The resulting parsed vacancy is then stored in the parsed vacancy database.
The first step in building the parser was to analyse the source code of a random selection of vacancies from each of the job boards whose vacancies have been retrieved. The aim of the analysis was to create lists for each content field, containing the tags and their identifiers that surround/precede relevant information. The belief was that the “id” portion of the html tags could be used to identify the subsequent/contained information.

For example, vacancies from Jobtrack\(^9\), heavily use the “id” portion of <DIV> and <SPAN> tags to identify the enclosed contents (see table 9).

<table>
<thead>
<tr>
<th>Identified Data</th>
<th>SPAN tag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Title</td>
<td>ctl12_JobOpeningNameLabel</td>
</tr>
<tr>
<td>Hours per week</td>
<td>ctl16_JobOpeningDisplayWorkingHoursLabel</td>
</tr>
<tr>
<td>Industry</td>
<td>ctl16_JobOpeningDisplayFunctionCategoryLabel</td>
</tr>
<tr>
<td>Education Level</td>
<td>ctl16_JobOpeningDisplayEducationLevelLabel</td>
</tr>
</tbody>
</table>

On the other hand, vacancies on VKbanen\(^10\) use a combination of section headers, identified between <h3> </h3> heading tags, and <dl>, <dt> and <dd> definition list tags to identify specific contents (see table 10).

<table>
<thead>
<tr>
<th>Identified Data</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Description</td>
<td>&lt;h3&gt;Functieomschrijving&lt;/h3&gt;</td>
</tr>
<tr>
<td>Requirements</td>
<td>&lt;h3&gt;Profiel&lt;/h3&gt;</td>
</tr>
<tr>
<td>Offer</td>
<td>&lt;h3&gt;Aanbod&lt;/h3&gt;</td>
</tr>
<tr>
<td>Sector</td>
<td>&lt;dt&gt;Sector&lt;/dt&gt; &lt;dd&gt;Industrie / Techniek, Zakelijke dienstverlening, IT / Automatisering&lt;/dd&gt;</td>
</tr>
</tbody>
</table>

\(^9\) [http://www.jobtrack.nl](http://www.jobtrack.nl)

\(^10\) [http://www.vkbanen.nl](http://www.vkbanen.nl)
Careerbuilder\textsuperscript{11} vacancies are built up in various ways. There is no one template used for all vacancies. Some vacancies identify upcoming content using comment tags, e.g.

\begin{verbatim}
<!-- Job Description Area -->
<!-- End Description Area -->

and

<!-- Requirements Start -->
<!-- Requirements Area -->
\end{verbatim}

, whereas other vacancies may use <DIV> or <SPAN> tags, or a combination of the two.

As Monsterboard allows advertisers to customise their advertisement template, vacancies found on Monsterboard also used a variety of techniques to identify the tags.

Having compiled a list of identifiers work was started to create a separate data extractors for each job board. As programming of these data extractors progressed, it became apparent that the plan to create a separate set of data extractors per job board, using the DOM structure and HTML tag identifiers to locate and extract the desired information would make things easy when the identifiers are present. However, when the identifiers are lacking, some intelligence and a lexicon would be necessary.

Running the Monsterboard data extractors on a random sample of vacancies retrieved from Monsterboard, it became ever more apparent that the data extractors would work on some vacancies and not on others, the reason often down to different advertisement templates. This meant that some vacancies had specific differences that would have to be accounted for. This would create extra work and the question of how to catch all exceptions arose.

The solution was to create “generic” data extractors which don't rely on the layout or HTML tag information, but work by searching for keywords and/or then searching neighbouring data for the desired content. The advantage of this approach is that it should work for any unformatted vacancy text from any job board, meaning that individual extractors do not need to be constructed and maintained for every job board that will be supported. The disadvantage is that more work will be required to make each data extractor aware of the content it is searching for.

For this approach the plain text of the vacancy, and none of the HTML markup are required. The HTML markup would therefore have to be removed before the data extractors can do their job. This led to a change in the parsing strategy:

\footnotesize{\textsuperscript{11} http://www.careerbuilder.nl}
Given a HTML formatted vacancy:

1. Remove all content except the contents between `<body>` and `</body>`
2. Remove HTML tags that contain no useful content, e.g. `<script>` and `<hr>`
3. Remove all remaining HTML tags excluding their content
4. Remove all contents after the end (Contact Us) of the vacancy
5. Convert all remaining content to ISO-8859-1 encoding
6. Pass this cleaned text to each individual data extractor
7. Store results in database.

<table>
<thead>
<tr>
<th>extract</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td>int display</td>
</tr>
</tbody>
</table>

| **Called External Methods** |
| string getContent(string s, int display) |
| string getDescription(string s, int display) |
| string getDrivingLicense(string s, int display) |
| string getEducation(string s, int display) |
| string getExperience(string s, int display) |
| string getFunctionTitle(string s, int display) |
| string getHours(string s, int display) |
| string getIndustry(string s, int display) |
| string getLanguages(string s, int display) |
| string getProfile(string s, int display) |
| string getSalary(string s, int display) |

*Figure 30: Data Extractor Methods*

To help keep a clear overview of the code, each method for each data extractor is stored in its own source file. These files are included in the extract method to make their method available.

Once all the data has been extracted from a vacancy a parsed ADS object (see below) is created to store the extracted data in. In turn, this object is used to insert the data into the parsed vacancy database.
4.3.1 Markup Removal

Though all markup was removed from each vacancy, how the tags were removed was dependant on what type of content is contained between the opening and closing HTML tags.

The input of the markup removal is the unparsed vacancy web page as it was downloaded from the job board. After processing, the vacancy text, excluding the HTML markup is returned.

A tag such as `<SCRIPT>` should be removed in its entirety, including the contents between the opening and closing tags. This tag does not contain vacancy text and so is not required for further processing.
Most other tags, such as `<TABLE>`, `<DIV>` and `<UL>`, which have an opening and closing tag may contain vacancy contents. In this case the tags should be removed, but not the contents between the opening and closing tags.

Lastly, tags such `<BR>` and `<HR>`, which don't have a separate closing tag can be removed outright as they do not contain any information in themselves.

### 4.3.2. Lexicons

The data extractors rely in part on lexicons to locate and extract certain information. This section describes the implementation of the lexicons and various considerations and decisions that had to be made during their implementation.

The lists in table 6 are stored as arrays in a file that is loaded by the parser. It is a deliberate choice not to store this data in a database, as it is less expensive to load this data from a file as constants which are included in the parser functions than to query the database for the lists each time the parser is executed.

When creating the lexicon, some decisions needed to be made. For example, should all possible languages be detected or only a subset? Should all possible degrees (by name) be detected, or only the degree levels (a finite list)?

The easiest section to implement was the list of Dutch driving license types, as this is a freely available list on the Internet.

Next the list of educational degree levels was compiled. This is quite a short list. This is in sharp contrast with the list of possible degrees which is large and constantly changing, as new courses are created and others are scrapped. Two lists of Dutch degrees were found, HODEX and CROHO. HODEX is a list compiled of degrees offered by participating educational institutions and CROHO is a list of higher education degrees (known as WO and HBO degrees in The Netherlands) as listed in the Ministry of Education's database of accredited degree courses. The problem with both lists is that even when combined, the resultant list does not provide a complete list of all degree courses, not in the least due to non higher education degrees being excluded. It can be argued that a large portion of the advertised vacancies are for persons with a higher education degree, and so the use of either CROHO and/or HODEX will result in a large portion of the required degree courses being covered. Given more time, the compiled list could be used to discover the context degree courses appear in in the text, and so be used to create a generic a parsing component. Currently they have not been used.

When creating the list of languages to detect, the choice was made to only detect a small number of languages (listed in table 11 below). There are a number of reasons for this. Firstly, the vacancies that are retrieved and processed are Dutch vacancies and are highly likely to only contain language requirements for European languages. Manually querying the database with a large number of different languages proved this point, and makes a short list of languages viable for this prototype. To be complete, all languages should be included, but for
the purposes of this prototype, it does not add any extra value. A second reason for only
including a subset of all world languages is that there are two word forms for each language.
One in the proper noun form, e.g. engels (English), and the other is the adjective form, e.g.
engelse (English). Both forms of the languages need to be identified, and unfortunately there
is no fixed rule for converting the noun form into the adjective form. Generally, it is just a
matter of adding an “e” behind the noun form, but this is not a generic solution. Perhaps the
best method of creating a complete list of languages is to obtain a list of all proper noun forms
of languages and to list the exceptions to the rule when converting to the adjective form. This
complete list can then be used for language detection. Lastly, in English, languages are written
starting with a capital letter, as they are proper nouns. This aids their detection in free text, as
one can look for all capitalised words and filter from there. In Dutch, proper nouns are not
capitalised, so no such pre-filtering is possible.

<table>
<thead>
<tr>
<th>Table 11: Recognised Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
</tr>
<tr>
<td>Dutch</td>
</tr>
<tr>
<td>English</td>
</tr>
<tr>
<td>French</td>
</tr>
<tr>
<td>German</td>
</tr>
<tr>
<td>Polish</td>
</tr>
<tr>
<td>Portuguese</td>
</tr>
<tr>
<td>Spanish</td>
</tr>
<tr>
<td>Swedish</td>
</tr>
</tbody>
</table>

There are algorithms that aid in extracting the subject of a text. As vacancies have a fixed set
of topics that are covered, that generally begin with the title of the topic, the method
employed to locate the topics is by matching against a list of pre-defined identifiers for each.
An example is the list for commonly used names for the heading for education level,
“opleidingsniveau”, “opleiding” and “educatie”. Once the identifier is located, the topic of
the next block of text(s) is known and can be handled by the appropriate data extractor. These
lists were created for the headings listed in table 6.

A research method used to speed up locating interesting words, but also interesting words
surrounding desired content was by creating a function called topWords. Given a text,
topWords stores all mono-, bi- and trigrams in a database, effectively gathering co-occurrence
statistics, incrementing the occurrence counter for any mono-, bi- or trigram already found in
the database. topWords was applied to all downloaded vacancies in the database. topWords
assisted in gaining a better understanding of the context that key terms occur in.

<table>
<thead>
<tr>
<th>topWords</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>void topWords(string s)</td>
</tr>
</tbody>
</table>

Figure 33: topWords Method
The list of Dutch stop words was created by merging Oracle's\(^\text{12}\) and the Hogeschool Utrecht's\(^\text{13}\) lists of Dutch stop words.

To help further reduce the size of the vocabulary of all vacancy texts a separate list with some common and non-descriptive words (with regards to a vacancy's requirements), such as *informatie*, *functie* and *contactgegevens* was compiled using topWords. When merged with the Dutch stop words list, this should help reduce vacancy texts to their core, increasing processing throughput during clustering and reducing memory requirements.

### 4.3.3 Data Extractors

The data extractors locate and extract relevant data from the vacancy text. They lean heavily on the use of regular expressions to locate and extract the desired information. The data extractors work on plain, unformatted text, as all markup has been removed during the pre-processing phase. The output for each extractor is the specific data that was being sought, or nothing if no match was found. The only exception is the function title extractor which extracts the data between the `<title></title>` HTML tags before the HTML tags are removed from the vacancy.

![Diagram of Data Extractor Information Flow](image)

*Figure 34: Data Extractor Information Flow*

In some ways the approach taken to extracting the desired data can be seen as performing named entity recognition. This is only possible due to the earlier construction of domain specific information about the information that is sought for extraction.

**Function Title Extractor**

It has been observed that the function title of the vacancy is generally also advertised between the `<title></title>` HTML tags of the vacancies. For this reason the decision was made not to try to locate the function title based on either an ontology of function titles or looking at other HTML markup such as heading tags, e.g. `<h1>`. The list of possible job titles is also constantly evolving, so obtaining a complete and accurate list will be difficult, and is likely to

---

\(^{12}\) http://download.oracle.com/docs/cd/B28359_01/text.111/b28304/astopsup.htm#i634823

\(^{13}\) http://www.catalogus.hvu.nl/webopac/helpteksten/stopwoorden.htm
be outdated quickly.

One general problem with the function titles is that other descriptions, such as job location, salary or company name are often also included in this text. To resolve this issue, after extracting the text between the title tags, additional filtering is performed on the text to try to only keep the actual function title and remove all other text.

Vacancy Description Extractor

The vacancy description is generally the first block of text found in a vacancy, and normally has a heading identifying it. The two key words identifying the vacancy description text were found to be “functieomschrijving” and “omschrijving”. Once these keywords are found in the text the subsequent portion of text will be the description.

To extract only the description the next heading must be found. Since the HTML tags have been removed from the vacancy text it is not possible to search for the next heading by looking at the HTML formatting. The solution is to search for other known heading names that come after the description, and then extract the text between the first and second heading names. This resulted in the following query:

```
/(?:functieomschrijving|taakomschrijving|bomschrijving|b|functie-inhoud|functie inhoud|(?:de|je) functie|functie::jouw baan)(?:;)?(.*)(?:functie-eisen|functie eisen|\buw profiel\b|wij vragen\de eisen|eisen:\(?:jou|je|jouw) profiel|jouw achtergrond\wij zoeken)/si
```

The above query does not catch all vacancy descriptions as the second list of headings to search for may not be present in the text. Also, considering this prototype is working on a large number of vacancies from a limited number of sites, it was thought useful to take advantage of a comment tag found on one of the job boards. If the above query resulted in no match, the following match was attempted:

```
/((?:functieomschrijving|\bomschrijving|\b|functie-inhoud|functie:)\[\[:space:\]\]*(\n|\r\n))(.*<!-- Einde Functieomschrijving -->)/si
```

In case this also yields no results, then an attempt is made to extract all text between the vacancy description heading and the job offer heading:

```
/(?:functieomschrijving|bomschrijving|b|functie-eisen\b|\buw profiel\b|wij vragen\de eisen|eisen:\(?:jou|je|jouw) profiel|jouw achtergrond\wij zoeken)/si
```

It could be that none of the identifiers used to find the beginning of the vacancy description are present in the vacancy. In such a case one last attempt is made to extract the description by taking all the text from the beginning of the vacancy till the requirements section:
In the event that none of the above patterns matched, all the vacancy contents are stored as being the vacancy description.

**Job requirements Extractor**

Extracting the job requirements from the vacancy text is implemented using the same approach as that used for the vacancy description.

First an attempt is made to extract all text between the requirements section and either the job offer, contact details, employment terms section or a stemmed version of the Dutch word for apply for vacancy (sollicit). This is done using the following two regex queries:

```
(?:functie(-|\s)?eisen)(?:heb jij)?eisen:(?::|;)?(.*?)(?:wat )?(wij bieden)bieden wij)biedt:aanbod\|contact\|contactinformatie\|contactgegevens\|sollicit\|arbeidsvoorwaarden))/si
```

and

```
/(?:bprofiel|ben jij(?:::)?|uitdagingen:jouw achtergrond\je hebt een)(.*)\n(?:wat )?(wij bieden)bieden wij)biedt:\|aanbod\|contact\|contactinformatie\|contactgegevens\|sollicit\|arbeidsvoorwaarden))/si
```

When this yields no results this is likely to be because we cannot find any identifiers for the end of the text (job offer, contact details section, etc). In this event it has been opted to retrieve all text from the requirements heading till the end of the vacancy:

```
/(?:functie(-\s)?eisen[[:space:]]*\n\r\n)?.*/si
```

One last attempt is made to retrieve the correct section using the following pattern:

```
/(?:wij zoeken)(?:wij vragen)(?:heb jij)(?::)?(?:wat )?(wij bieden)bieden wij)biedt:aanbod\|contact\|contactinformatie\|sollicit\|arbeidsvoorwaarden))/si
```

If none of the above queries result in a match then the entire vacancy text is stored as being the vacancy's requirements. This has been done to ensure that there is text to cluster and match on at a later stage. This would not be possible if this section was left blank.
**Industry and Education Level Extractors**

Creating the data extractors for industry and education level were all implemented in a similar fashion.

Firstly, the domain specific information gathered earlier was stored in arrays, and then a straightforward regex query looking for any of the array items was executed on the vacancy text. For these cases, each regex query is simply the array contents of the domain specific information, each entry separated by a “|”.

\/(keywords)/si

The initial implementations first looked for a keyword such as industry, before searching the following block of text for the domain specific information. This did not work (well) on all sites and vacancies, as different keywords and formatting were used. Through experimentation it was discovered that leaving out the search for the identifier and simply performing the regex query with the domain specific information worked better, faster and was simpler to implement.

**Work Experience Extractor**

Attempting to extract the required number of years of prior work experience based solely on the domain specific information gathered through analysing job board search forms worked well but not in all cases. The work experience data extractor therefore first attempted to locate the desired data using the query below, where *keywords* denotes the domain specific information.

\/(keywords)/si

If no results are found by this query it does not mean that there are no work experience requirements. Where the work experience was written as a requirement within the vacancy text, two different written forms for this requirement were noticed, the number of years experience was either written in numerical (e.g. 3) or written (e.g. three) form. As a result the vacancy is re-checked, first for the requirement in numerical form:

\/[0-9]+(?: jaar ([a-z ]*)?(?:werk)?ervaring)/si

and if no results are found in written form:

\/(een|twee|drie|vier|vijf|zes|zeven|acht|negen|tien)\s*\(?[0-9]+\)?(?: jaar ([a-z ]*)?(?:werk)?ervaring)/si
Language Extractor
Required languages in the vacancy text could not be located by simply matching against a list of all worldly languages. The main reason for this is that language requirements mostly occur within a block of vacancy text. Vacancy text also often also contains company details, such as “a Dutch company”. Simply matching against the language list resulted in a huge number of false positives.

The solution was to search for keywords that accompany language requirements, and then search the surrounding text for languages. Words such as “mondelinge” (verbal), “schrijfelijk” (written) and “communicatie” (communication) are often found in the text a few words before or after the language.

For a discovered language to be accepted as a language requirement, at least one identifying keyword needs to be in the vicinity of the language text unless more than one language is found in the vicinity of another. This is because the context of a language cannot be verified without considering the surrounding text. When two or more languages are in each other's vicinity, then it can be assumed that the list of required languages is being summed up.

Hours Extractor
The text indicating the number of hours per week that an applicant must be willing to work for were either found to precede identifying text, e.g. 32-40 hours per week, where hours per week is the identifier, or the requirements are listed in words such as “full time” or “parttime”.

As a result the vacancy text is first searched for the pattern:

```
/((\[0-9 \-a\]{1,})(identifiers))/si
```

where the identifiers are once again separated by a “|”. The numbers in the results are then extracted to obtain the working hour requirements.

Once this step is complete, the vacancy text is scanned for the presence of the fixed keywords (full time, parttime, etc.).

Driving License Extractor
In The Netherlands, except for a few special cases, the different driving license (rijbewijs in Dutch) types, e.g. for a car, motorcycle or car with a large trailer, are denoted by a code, e.g. B, D and E. Using the list obtained from the internet and the knowledge gained from having manually viewed a large random sample of vacancies, it was clear that Dutch vacancies publish the driving license requirements as rijbewijs X, where X is the type. There were however a few exceptions, namely for truck drivers, forklift operators and chauffeurs. Combining this information, the following regex pattern was constructed to locate the driving license type required:
4.4 Vacancy Clustering Component

In a nutshell, the vacancy clustering component takes the vacancy requirements text from the parsed vacancy database and uses this text to create clusters of related vacancies. The parsed vacancy database is then updated with the cluster information so that the matching algorithm can take advantage of the clusters, selecting only relevant clusters in an attempt to reduce the search space. Cosine similarity is used as the vacancy similarity measure, whilst K-means is used as the clustering algorithm.

![Vacancy Clustering Component Information Flow](image.png)

**Figure 35: Vacancy Clustering Component Information Flow**

**Stemming**

The implementation of the Dutch stemmer is an improved version of the Kraaij-Pohlmann stemmer as implemented for use in the Drupal content management system\(^\text{14}\). The Drupal Dutch stemming module was implemented in PHP but included calls to Drupal functions. To be able to use the Drupal Dutch stemming module, the Drupal specific code was removed and replaced by equivalent PHP code after which the stemmer was tested on some sample texts.

The input of the Dutch stemmer is the vacancy requirements text which has previously been

\(^{14}\) [http://drupal.org/project/dutchstemmer](http://drupal.org/project/dutchstemmer)
extracted from the unparsed vacancy. After stemming, the stemmed text is output.

**Figure 36: Dutch Stemmer Information Flow**

![Diagram showing the process of extracting and processing data through a Dutch Stemmer.

**Dictionary**

Before any term/term frequency vectors can be created a vector of allowable words is required. Loading this code in a web browser immediately initiates the dictionary update procedure.

**Figure 38: Dictionary Methods**

For each parsed vacancy in the db:
1. Create a term vector of all words in the requirements section of the vacancy.
2. Remove all stop words.
3. Apply Dutch stemmer (suffix -es/-as not stemmed).
4. Get counts for each word in the term vector.
5. Update dictionary.

Initially all terms are considered to be valid, i.e. *valid* is set to 1.
The above steps were taken to create the dictionary using the first 1051 vacancies in the parsed vacancy database. This resulted in a total of 10,108 terms. Terms with high occurrence frequencies were manually invalidated. This resulted in 10,054 valid terms. The dictionary should be recreated periodically to ensure that changes in term usage are captured.

It could be wise to automatically invalidate any terms that occur in at least 10-15% of the vacancies as one could argue that these terms do not add any extra value. To achieve this an extra column (nDocs) would need to be added to the dictionary indicating how many vacancies the term occurs in. By multiplying the number of vacancies by some threshold (e.g. 15%) we obtain a threshold indicating when terms should be invalidated. Updating the dictionary to reflect this would be as trivial as executing a simple SQL query, e.g.

```
UPDATE dictionary SET valid=0 WHERE nDocs>threshold
```

The drawback of recreating the dictionary is that the term frequency vector of all vacancies in the parsed vacancy database will have to be recreated as it is likely that new terms will appear till some critical mass is reached. Another issue is how to deal with terms that have been invalidated by the user and not by the system, especially if we would like to automatically alter the validity of terms based on their occurrence in all the stored vacancies.

**Table 12: Dictionary Entry**

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Term</th>
<th>Frequency</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pre-processing (term/term frequency vector creation)
Pre-processing of the vacancy is required before it can be clustered. Pre-processing involves the creation of the term and term frequency vectors of the vacancy.

1. Retrieve the vacancy requirements text from the parsed vacancy database.
2. Pre-process the text to remove punctuation, blanks, stop words and invalidated words from the dictionary.
3. Create the term vector.
4. Create the term frequency vector.
5. Update the vacancy in the database with the term and term frequency data.

Clustering
As mentioned earlier, a slightly modified version of K-means is used as the clustering algorithm. Given $K$ initial clusters and a vacancy ID, the steps which are performed to cluster a parsed vacancy are (also see figure 10):

1. Calculate the cosine similarity of the vacancy to the centroid of all $K$ clusters.
2. Assign the vacancy to the cluster with the highest cosine similarity score.
3. Update the vacancy in the database with the cluster information.
4. Update cluster centroids after every 250 vacancies have been clustered.

The system administrator has the choice of supplying the K-means algorithm with cluster centroids when starting the clustering process. When no centroids are supplied $K$ centroids are randomly assigned. When the centroids are randomly assigned, no attempt is made to optimise the initial cluster choice.

If the standard procedure of recalculating the cluster centroids had been adopted as specified in the K-means algorithm, then the cluster centroid would have to be recalculated after every
single vacancy is added to a cluster. This is a computationally expensive process. To reduce the execution time of the clustering process, it was chosen to recalculate the cluster centroids at set intervals. Whilst this may result in slightly less accurate results, it should help reduce the number of calculations in the worst case scenario to more acceptable levels.

It was chosen to recalculate the cluster centroids after every \(250\) vacancies were added into the clusters (total vacancies, not per cluster). The last two columns of table 7 show the best and worst case scenarios using this approach. We can clearly see that each time the number of clusters is doubled, so does the number of calculations performed by the best case scenario, eventually overtaking the best case scenario of the standard K-means approach. However if we compare the worst case scenario's, we see that this approach should on average only require a fraction of the calculations compared to standard K-means. It is also clear that above a certain value of \(k\) the worst case scenario of this approach will become more costly than the standard K-means approach. This however only occurs for large values of \(k\).

The initial implementation of the clustering algorithm was single threaded. This meant that the clustering algorithm did not make use of any extra processor cores that were available. As \(k\) increased the time to re-cluster the vacancies (see Experiment 2, table 15) began to take unacceptable proportions. At this point it was chosen to allow the clustering algorithm to take advantage of multi processor cores. As the chosen implementation re-calculates the cluster centroids after every 250 vacancies are inserted into clusters, two steps can be multi-threaded.
Firstly, inserting the batch of 250 vacancies into the most appropriate cluster, and secondly, calculating the new cluster centroids after every 250 vacancies have been inserted into the clusters.

<table>
<thead>
<tr>
<th>clusterVacancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>int threadID</td>
</tr>
<tr>
<td>[int] centroids</td>
</tr>
<tr>
<td>[int] vacIDs</td>
</tr>
<tr>
<td>Methods</td>
</tr>
</tbody>
</table>

*Figure 42: Cluster Vacancy Multi-Thread*

<table>
<thead>
<tr>
<th>calcNewMean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>int oldCentroidID</td>
</tr>
<tr>
<td>int newCentroidID</td>
</tr>
<tr>
<td>[int] newCentroidTF</td>
</tr>
<tr>
<td>[int] vacIDs</td>
</tr>
<tr>
<td>[int] matrix</td>
</tr>
<tr>
<td>Methods</td>
</tr>
</tbody>
</table>

*Figure 43: Calculate New Cluster Mean Multi-Thread*

**Insert vacancies into clusters**
To be able to process multiple vacancies at once, the number of vacancies per batch (250 in this case) is equally divided amongst the number of threads one wishes to run. These batches of vacancies are then run in parallel on different processor cores.

**Calculate new cluster centroids**
Calculating a new cluster centroid has no impact on any vacancies/clusters other than the cluster being processed. This means that the process of calculating a new cluster centroid can be run in parallel for as many clusters/threads as one wishes.

The problem arose that PHP does not contain support for running multiple threads, however, there was a solution. It is possible to run PHP scripts from the commandline using *php-cgi* and it is possible to execute programs from within PHP using the *exec()* function. The next issue was passing variables and obtaining thread status (active, complete, etc). By combining use of *php-cgi* and *exec()* and setting up a database table to maintain running thread details (actions and status) it became possible to run parallel threads.
4.5 Curriculum Vitae Component

The curriculum vitae component (CVC) takes a CV as input and stores it, decomposed into its constituent parts in the CV database. As mentioned earlier, there are already many commercial companies offering software that decomposes CVs stored in different file formats. In a large scale test such commercial software would be used, but for the purposes of this prototype, a few CVs will be manually decomposed and stored in the CV database. This should prove sufficient for testing the prototype.

![Curriculum Vitae Component Information Flow](image)

*Figure 44: Curriculum Vitae Component Information Flow*

<table>
<thead>
<tr>
<th>cvFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>[string] stopWordsV</td>
</tr>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>(string,string) generateCV_TF_vacID(int cvID)</td>
</tr>
<tr>
<td>(string,string) generateCV_TF(string s)</td>
</tr>
<tr>
<td>int TF(int value, string key, int nTerms)</td>
</tr>
</tbody>
</table>

*Figure 45: CV T/TF Generation Methods*

Part of the process of inserting the CV into the CV database involves the generation of the term/term frequency vectors of the CV text. This is the same procedure as generating these vectors for the vacancy text (figure 39), except that it is performed on the CV text.

The database that stores the decomposed CVs contains a plain text copy of the CV's contents along with the extracted information. The CV database was created to store the information displayed in table 13 for each CV.

<table>
<thead>
<tr>
<th>Table 13: CV Database Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personalia</td>
</tr>
<tr>
<td>(Highest) achieved education level</td>
</tr>
<tr>
<td>Languages</td>
</tr>
<tr>
<td>Skills / Proficiencies</td>
</tr>
<tr>
<td>Job/Function titles</td>
</tr>
<tr>
<td>Driving License</td>
</tr>
<tr>
<td>Job History</td>
</tr>
<tr>
<td>Plain text CV</td>
</tr>
</tbody>
</table>
15 CVs with completely different profiles/industries have been manually decomposed and inserted into the CV database using the CV management GUI (figure 19). This should be sufficient to test the matching component.

4.6 Matching Component

The matching component is responsible for locating the vacancies in the database that closest match the contents of the CV. The score of a match is calculated by individually scoring the closeness of the characteristic features (table 8) of the CV/vacancy, applying a weight vector and then finally summing the individual scores (see figure 14). The highest scoring match signifies the best match.

Figure 46: Matching Component Information Flow
**Procedure**

Match CV to Vacancy. Given a CV characteristic vector:

1. Calculate cosine similarity of CV's skill set to the requirements text of the vacancies' cluster centroids. Note the \( n \)-highest scoring cluster centroids.
2. For each vacancy in the \( n \)-highest scoring cluster centroids (from step 1):
   2.1 Calculate cosine similarity between the CV's skill set to the requirements text of the vacancy.
   2.2 Calculate the similarity between the remaining characteristic features (see table 8).
   2.3 Apply the weight vector.
   2.4 Sum all the individual scores.
3. Store the \( m \)-highest scoring matches in the matches database.

**Figure 47: CV-Vacancy Matcher Methods**

Attempting to match a CV against all the vacancies stored within the vacancy database would be both time consuming and counter productive. As the vacancies have already been split into clusters of related vacancies, these clusters can be used to reduce the search space to the clusters with the highest cosine similarity score between the skill set defined within the CV and requirements text of the vacancy cluster centroids. The clusters with the highest cosine similarity scores should contain the vacancies with the best matches, so these clusters are selected.
The weight vector enables fine-tuning of the scoring function to add strength to certain features. For example, matching achieved education level requirements is considered more important than matching the job title.

The final score of the similarity between a vacancy and a CV is based on the sum of similarity scores between the individual matching features, modified by a weight vector. Below the procedure explaining how each individual feature is scored is outlined.

**Cluster Selection**
Cluster selection is done by selecting those clusters with the highest cosine similarity score between the CV's skills/proficiencies and the vacancy's requirements text. The various characteristics of the CV are then matched to all the vacancies stored within the selected clusters.

These selected clusters should contain the best matches, though outliers in clusters that weren't selected could potentially yield a better result. It is however too computationally expensive and inefficient to try all combinations of CVs and vacancies. This makes apparent the importance of good quality vacancy clusters to reduce the size of the dataset to match each CV against.

**CV Skills/Proficiencies vs Vacancy Requirements**
The cosine similarity score between the CV's skills/proficiencies and the vacancy's requirements text is calculated. The terms CV skills/proficiencies are used loosely to describe all text from the CV that describes a person's skills that should be used to match a vacancy against. Currently this includes all information about:

- the education followed (degree, course, certificate);
- job descriptions (job history often contains key skills gained and used);
- languages;
- IT skills;
- driving license;
- any other relevant skills.

While one might argue that including information such as the languages here will cause them to be counted double, the difference in absolute score will be negligible due to individual terms in the term frequency vector only having a tiny effect on the final score.

**Languages**
The similarity score of the language vectors is achieved by looking at how many languages are in common between the two. If the CV contains more languages than requested, the score is boost slightly for each extra language. On the other hand, if the vacancy contains more languages than contained within the CV, the score is negatively boosted for each missing
language.

\[
\text{score}_\text{language} = |\text{intersection}(\text{languages}_\text{vacancy}, \text{languages}_\text{CV})| + |\text{languages}_\text{CV} \setminus \text{languages}_\text{vacancy}| \times \text{score boost} - |\text{languages}_\text{vacancy} \setminus \text{languages}_\text{CV}| \times \text{score boost}
\]

**Function/Title**

Whereas a vacancy should only contain one function title, a CV will contain a function title for each position held. To calculate the similarity score the cosine similarity between each function title in the CV and the vacancy's function title is calculated. The highest cosine similarity score is regarded as the final score.

\[
\text{score}_\text{title} = \max \left( \text{for each title} \left( \text{cosine similarity}(\text{title}_\text{CV}, \text{title}_\text{vacancy}) \right) \right)
\]

**Education Level**

The score for the education level is calculated by looking at the highest achieved education level in the CV and comparing it to the highest requested education level in the vacancy. If the levels match, then a perfect score is achieved. If the CV contains a slightly higher education level, then a lower score is achieved. If the CV contains a slightly lower achieved education level than that requested within the vacancy, then a low match score is achieved. In all other cases a matching failure is assumed, and a score of 0 is applied.

The concept of a slightly higher and lower score is achieved by sorting the recognised education levels into groups of equivalent degree levels, and then seeing if the two compared education levels fall into neighbouring groups.

**Years of Experience**

The score for the years of experience is straightforward. The experience field within the CV record in the CV database stores the total number of years experience of all the jobs listed in the CV. This is compared to the minimum and maximum requested number of years experience in the vacancy. If the candidate falls within the minimum and maximum, a perfect score is achieved. If the candidate is one year over or under qualified, a slightly lower score is assigned. If there are more than two years difference, a very low match score is assigned, as the vacancy may still be of some interest.

**Driving License**

The similarity score of the driving license vectors is achieved by looking at how many of the desired driving license types (qualifications) match. If the CV contains more driving license qualifications than requested, the score is boosted slightly for each extra qualification. On the other hand, if the vacancy contains more driving license qualifications, the score is negatively boosted for each missing one.
\[
\text{score}_{\text{driving license}} = |\text{intersection}( \text{driving license}_{\text{vacancy}}, \text{driving license}_{\text{CV}}) | \\
+ |\text{driving license}_{\text{CV}} \setminus \text{driving license}_{\text{vacancy}}| \times \text{score boost} \\
- |\text{driving license}_{\text{vacancy}} \setminus \text{driving license}_{\text{CV}}| \times \text{score boost}
\]

**Weights**

The values of the weights have initially been assigned according to their importance, knowledge which was gathered earlier (see table 8). It is useful to conduct experiments to investigate the effects of different values for the weights, however this is dependant on the quality of the initial results.

**Match Threshold**

In a production system it would be useful to filter out low scoring matches so that the results only contain high quality matches. Low scoring results would not be stored if they fall below a predetermined threshold. The ability to use a threshold has been implemented, however the threshold value has been set to 0 so that all results can be viewed to determine if the matching/scoring function is working as expected. Viewing all results will also help determine a potential value for the threshold.

### 4.7 Ranking Component

The ranking component is responsible for ranking the search results from the keyword search performed through the search GUI.

![Figure 48: Ranking Component Information Flow](image)
**4 Implementation and Testing**

### 4.8 Vacancy Database

Implementing the vacancy database (*crawledfeedurls*) was a case of inventorising what data the vacancy retrieval component would return and also determining what additional data might be useful to store besides the retrieved vacancy.

When a new vacancy is stored in the database it is assigned a unique ID. The vacancy RSS feed details (title, description, URL, GUID and PubDate) and the ID of the RSS vacancy feed that the vacancy belongs to are added. Once the vacancy has been retrieved and stored in the database, the crawl status, crawl date, and page contents (HTML page, excluding external contents such as images) are updated to reflect that the vacancy has been retrieved.

When viewing a random sample of vacancies that were downloaded to test the vacancy retrieval component it became clear that there were errors in displaying the downloaded vacancies due to the document encoding used (character set). It was decided to add an extra column to each vacancy in the database containing the character encoding of the vacancy. This could then be used at a later stage by the vacancy parser component to determine how to replace accented characters with their non-accented counterparts. With slight modification each vacancy in the vacancy database is stored with the information displayed in figure 49.

The current vacancy database contains 17,764 vacancies retrieved from five job boards\(^{15}\) between 29-03-2011 and 23-04-2011. This should provide sufficient data for development and testing purposes.

The vacancy retrieval component was run for a few hours per day between the selected dates on the general job board RSS feeds. It is therefore possible that duplicate vacancies exist. These duplicates are possible within the same job board (reposting a vacancy), but duplicates

\(^{15}\) Monsterboard, JobTrack, Nationale Vacaturebank, VKbanen and GemeenteBanen
may also occur if a vacancy is posted to multiple job boards. Effects of the duplicates on system performance should be negligible as long as they are limited in quantity.

The vast majority of the vacancies, 17,058 out of 17,764 vacancies, were retrieved from Monsterboard, JobTrack and Nationale Vacaturebank.

<table>
<thead>
<tr>
<th>crawledfeedurls</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>URL</td>
</tr>
<tr>
<td>GUID</td>
</tr>
<tr>
<td>PubDate</td>
</tr>
<tr>
<td>FeedID</td>
</tr>
<tr>
<td>CrawlStatus</td>
</tr>
<tr>
<td>CrawlDate</td>
</tr>
<tr>
<td>Encoding</td>
</tr>
<tr>
<td>PageContents</td>
</tr>
</tbody>
</table>

*Figure 49: Vacancy Database Columns*

### 4.9 User Interface

Various user interfaces have been implemented to interact with the prototype. As the main focus of this project has been on giving users the ability to search through the created vacancy database and automatically match CVs with vacancies, only the user interfaces for these components have been implemented, together with the interface to maintain the CV database. Interfaces for other aspects of the system, such as managing the RSS feeds (figure 20) and managing the system back end (figure 21), have only been implemented as dummy interfaces to give an idea of how a production system would look. Due to this choice, changes to the system such as modifying the weight vector of the matching component must be performed by editing the source file(s).

*Figure 50: User Interface Component*
4.9.1 Keyword Search

The interface for performing keyword search is implemented in full. The keyword search interface provides the user with a single search field (figure 17) where query terms are typed.

When searching the vacancy database the user has the choice of displaying the results sorted using only MySQL's built in relevancy measure or also using the ranking component described in chapter 4.7.

In either case the search results are displayed in a fashion similar to Google's search results (figure 18). Three pieces of information are displayed as part of every result. First the title is displayed. Clicking the title displays the vacancy. Next, when available, the industry/industries extracted from the vacancy are displayed. Lastly, a snippet of text, the first 280 characters of each vacancy are displayed to give the user an idea of the vacancy's contents. As mentioned earlier, using the top ranking sentence as the snippet has been found to have a positive influence on query reformulation and relevance assessment [Joho et al], but for the purposes of this project, displaying the first portion of vacancy text should provide equally good results.

If a user wishes to create an advanced search query, the user has the possibility to use MySQL's boolean search operators\(^\d\). An overview of the operators is given below. When a user enters multiple query terms without using any search operators the OR function is assumed between query terms.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>AND</td>
<td>chipsoft +sap</td>
</tr>
<tr>
<td>-</td>
<td>NOT</td>
<td>chipsoft -sap</td>
</tr>
<tr>
<td>&gt;</td>
<td>Increase relevance score for term</td>
<td>chipsoft &gt;sap</td>
</tr>
<tr>
<td>&lt;</td>
<td>Decrease relevance score for term</td>
<td>chipsoft &lt;sap</td>
</tr>
<tr>
<td>(</td>
<td>Group</td>
<td>(chipsoft sap)</td>
</tr>
<tr>
<td>~</td>
<td>Negate term</td>
<td>~chipsoft sap</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>Match entire phrase</td>
<td>&quot;sap-srm&quot;</td>
</tr>
</tbody>
</table>

---

4.9.2 Curricula Vitae Management

The tasks that a user can perform from the CV management interface (figure 19) are adding, updating and deleting a CV from the CV database.

When adding/updating a CV, the term and term frequency vectors are immediately generated and inserted into the database along with the CV. The same procedure and dictionary is used to generate the T/TF vectors for the CVs and vacancies (see chapter 4.4, figure 39). The basic steps are:

1. Validate the form data.
2. Generate term/term frequency vectors.
3. Update the CV database.

4.9.3 Matcher: Explore CV-Vacancy Matches

The matching interface is used to view and discover the vacancies that best match a CV. Using this interface the user can select a CV to view the list of best matching vacancies of- or manually (re-)start the matching process for the selected CV.

The resulting implementation is a basic form where a CV can be selected after which the best matching vacancies are displayed. Buttons are present to (re-)start the matching process. The resulting matches are displayed in a similar fashion as when performing keywords search on the vacancy database. Results are ordered using the matching algorithms scoring function (figure 14) in order of best to worst match.
Manage CV-Vacancy Match

![CV-Vacancy Match Interface]

Figure 53: Results CV-Vacancy Match
5 Experiments and Results

This chapter describes the experiments that were undertaken and their results. The experiments were formulated based on the problem definition defined in chapter 1.1 and the research challenges outlined in chapter 1.2. They are performed to investigate the quality of the data extractors, the vacancy clusters and the matching/ranking algorithms. Experiments were also conducted on the vacancy clustering component to investigate what effect varying the number of clusters has on the execution time of the clustering algorithm.

Each experiment begins by outlining the aim of the experiment and the test procedure. Where relevant, information on the system hardware and platform the experiments are run on is described. Next the results are presented and discussed.

Experiment 1: Vacancy Parser Accuracy

Aim:
The aim of this experiment is to subjectively measure the accuracy of the vacancy parser component by selecting random vacancies from the unparsed vacancy database, feeding them through the vacancy parser component and manually comparing the unparsed vacancies to the parsers' results.

Procedure:

1. Select 20 vacancies at random from each of the following three job boards (Monsterboard.nl, JobTrack and Nationale Vacature Bank).
2. For each vacancy verify that each data extractor extracted all, most, some or none of the relevant information. Assign a score, between 0 and 3 or n/a, to the results of each data extractor. A score of 0 denotes a complete failure, 3 a (near) perfect match and n/a if there was no relevant data to extract for the given data extractor. The score is lowered if irrelevant entries appear in the data extractor results.

Once the scores have been assigned, a total score for each tested vacancy is calculated. This procedure tests the accuracy of the ten developed data extractors. The total score for each reviewed vacancy is calculated as follows:

\[
\frac{\sum_{i=1}^{10} \text{score data extractor}_i}{\text{no. of data extractors with valid data}}.
\]
Where a value of n/a is assigned, the value for this data extractor is ignored, i.e. no value for the score of the given data extractor is summed and the value of the no. of extractors with valid data is reduced by 1. This is done to ensure that if a score of 0 is assigned, this indicates a complete mismatch which negatively effects the total score for the given vacancy.

The overview screen created to assist in the development of the data extractors (figure 23) is used to review the randomly selected vacancies for accuracy as it provides an overview of the extracted data (left side) and the original unparsed vacancy (right side).

**Results:**
Table 14, summarises the results of executing the vacancy parser component on 20 random vacancies from each of the three mentioned job boards (60 total vacancies).

### Table 14: Vacancy Parser Component Accuracy

<table>
<thead>
<tr>
<th>Site \ Data Extractor</th>
<th>Title</th>
<th>Driving License</th>
<th>Languages</th>
<th>Education</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monster</td>
<td>2.70</td>
<td>2.00</td>
<td>3.00</td>
<td>2.78</td>
<td>1.78</td>
</tr>
<tr>
<td>JobTrack</td>
<td>2.90</td>
<td>1.50</td>
<td>3.00</td>
<td>3.00</td>
<td>2.34</td>
</tr>
<tr>
<td>NVB</td>
<td>2.43</td>
<td>3.00</td>
<td>3.00</td>
<td>2.75</td>
<td>2.70</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>2.68</strong></td>
<td><strong>2.17</strong></td>
<td><strong>3.00</strong></td>
<td><strong>2.84</strong></td>
<td><strong>2.27</strong></td>
</tr>
<tr>
<td><strong>Occurrences</strong></td>
<td><strong>60</strong></td>
<td><strong>6</strong></td>
<td><strong>22</strong></td>
<td><strong>51</strong></td>
<td><strong>59</strong></td>
</tr>
<tr>
<td><strong>Average Usage</strong></td>
<td>100.00%</td>
<td>10.00%</td>
<td>36.67%</td>
<td>85.00%</td>
<td>98.33%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Site \ Data Extractor</th>
<th>Salary</th>
<th>Industry</th>
<th>Experience</th>
<th>Description</th>
<th>Hours</th>
<th>Total Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monster</td>
<td>3.00</td>
<td>2.66</td>
<td>2.77</td>
<td>1.78</td>
<td>2.58</td>
<td>81.59%</td>
</tr>
<tr>
<td>JobTrack</td>
<td>3.00</td>
<td>2.58</td>
<td>3.00</td>
<td>2.43</td>
<td>3.00</td>
<td>90.13%</td>
</tr>
<tr>
<td>NVB</td>
<td>3.00</td>
<td>2.56</td>
<td>2.43</td>
<td>2.75</td>
<td>3.00</td>
<td>88.57%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>3.00</strong></td>
<td><strong>2.60</strong></td>
<td><strong>2.73</strong></td>
<td><strong>2.32</strong></td>
<td><strong>2.86</strong></td>
<td><strong>86.76%</strong></td>
</tr>
<tr>
<td><strong>Occurrences</strong></td>
<td><strong>7</strong></td>
<td><strong>36</strong></td>
<td><strong>22</strong></td>
<td><strong>60</strong></td>
<td><strong>40</strong></td>
<td><strong>363</strong></td>
</tr>
<tr>
<td><strong>Average Usage</strong></td>
<td>11.67%</td>
<td>60.00%</td>
<td>36.67%</td>
<td>100.00%</td>
<td>66.67%</td>
<td>60.50%</td>
</tr>
</tbody>
</table>

Looking at the total accuracy scores for each job board we see that Monster scores the lowest accuracy at just under 82% and JobTrack and NVB\(^ {17} \) are the winners at respectively 90% and 89%.

One reason that Monster achieved a lower score can be attributed to the profile data extractor incorrectly extracting the requirements section of the vacancy text. Monster allows job advertisers to personalise the look and feel of their vacancies, resulting in a large quantity of vacancies with a variety of layouts. This issue is not specific to Monster, as all the tested job boards suffer from this problem. It is just more apparent with Monster. The cause for the incorrect extraction of this data differs between the vacancies, but reasons include:

1. the word *functie-eisen* (job requirements/profile in Dutch) being mentioned in other

---

\(^ {17} \) Nationale Vacaturebank
parts of the vacancy. This caused an incorrect body of text being recognised and extracted as the job requirements text.
2. errors in the HTML code of the vacancy. This caused the pre-processing phase to incorrectly remove content as well as HTML tags from the vacancy.
3. no clear identifier being used to indicate the job requirements section. Without a clear identifier only the content makes clear what the current section entails.
4. the general job description was often not announced, resulting in the data extractor often not being able to locate the beginning of the job description and the requirements sections (or any other section).

Looking at all the job boards, a number of the mismatches were caused by the vacancy templates used by the job advertiser. Some advertisers (intermediaries, job agencies, etc.) had a brief description of their company and mentioned that they offer jobs for people with a higher education degree (e.g. HBO, WO) while the vacancy itself seeks an individual with an WO degree. In this case the data extractor would locate all the mentioned degrees even the listed ones (HBO in this case) which are not relevant to the vacancy.

The data extractors for the vacancy description and requirements texts need improvement. The lack of fixed templates means that more effort needs to be put into locating and extracting these sections. More vacancies need to be reviewed to locate additional markers.

One would expect the title data extractor to have a perfect score (3) as the <title> HTML tag is used to locate and extract this data. Whilst the title tag does help locate the title text, the title data extractor attempts to remove irrelevant data, such as salary, location of the job and vacancy reference code, from the title before storing it in the database. This score therefore signifies how well the actual function title was extracted from the title text.

Though the sample size for this test is relatively small, 60 total vacancies reviewed, a few things become immediately apparent about the frequency certain data occurs in:

- The driving license is only specified in 10% of the vacancies;
- An indication of salary in 11.67% of the vacancies;
- Language and Experience requirements are only specified in just over a third of the vacancies;
- Industry of the vacancy is only specified in 60% of the cases.

The low occurrence rate for the driving license requirement was to be expected, as interviews with recruiters had made clear that this requirement is rarely specified unless essential for the job. The low score for this data extractor was not anticipated, however this turned out to be a combination of the low number of vacancies with this requirement and a vacancy from JobTrack being parsed incorrectly due to broken HTML code. The “broken” vacancy resulted in the driving license requirement being removed with much of the vacancy text during pre-processing. If the HTML code had not be broken, this data extractor would have scored 2.67 out of 3 instead of 2.17 out of 3.

The low occurrence rate for salary indication came as a surprise. The manual vacancy review
showed that vacancies often indicated the salary would be competitive, but failed to give an indication. This suggests that a competitive salary is not seen as a salary specification and so is not extracted. Reasons for not adding a salary indication will likely differ but may include reasons such as not wanting to put off good applicants with a higher salary expectation from applying, leaving room to negotiate the salary according to the applicant's experience and qualifications and making it more difficult for competitors to know what it would cost to “steal” their employees.

It is clear that work needs to be done to improve the overall quality of the data extractors, especially when extracting the vacancy requirements text and the vacancy description. New approaches to locating the beginning and ending of these blocks of text need to be found. Perhaps the make up of the first few words of text directly after the block identifier will provide new insights into obtaining their location.

The vacancies on Monster accounted for almost 40% of the occurrence counts of each data extractor. Looking at the occurrence counts for only Monster, the data extractors ran a total 70.5% of the time as opposed to 50.5% for NVB and 60.5% for JobTrack. This may suggest that the vacancy descriptions on Monster are of a higher quality and may also suggest that the vacancies are for jobs where a higher level of experience/education is required.

The results also show that the key elements/sections that form a vacancy description are the title, skill requirements (specifically education level) and a description of the vacancy. The type of job (e.g. full time, part time, permanent, contractor) and industry the vacancy is in are often also included. As vacancies from Dutch job boards (in Dutch) are expected in the system, a low value for the number of vacancies with language requirements can be expected. A Dutch vacancy is unlikely to list Dutch as a required language, and will likely only list foreign languages for applicants working in a multilingual environment.

As more and more vacancies were reviewed for content and parser accuracy, it became clear that the quality of vacancy descriptions varies between job boards. The “Nationale Vacature Bank” for example, often contained vacancies containing little more than a job title and a few words describing the required skill set. Whilst this means that the data extractors have less work to do, the clustering process will likely experience difficulties as these vacancies contain little to no text that is characteristic of the vacancy.
Experiment 2: Vacancy Clustering Speed

Aim:
The aim of this experiment is to see how the execution time of the clustering algorithm is affected by varying the number of clusters and vacancies.

Procedure:

1. Define the number of clusters $k$ for this run.
2. Run the vacancy clustering component on the parsed vacancy database.
3. Repeat steps 1 & 2 for varying numbers of vacancies.

As varying values for the number of clusters are chosen the initial cluster centroids are not manually assigned but chosen at random from the parsed vacancy database. This approach could potentially lead to a few clusters containing most of the vacancies, as many of the cluster centroids may express fairly unique vacancies in the database that have little in common with any other vacancy in the database. This would then adversely affect execution time as clusters with a high number of vacancies require a higher than average number of calculations when recalculating the cluster centroid.

As mentioned earlier, the implementation of the K-means algorithm has been altered slightly. Instead of recalculating the cluster centroid after every addition to the cluster, the centroids of all clusters are recalculated after a total of $250$ vacancies have been clustered (in total, not per cluster).

This experiment was conducted on a computer running Windows Vista Home Premium 32bit with a 2.6 GHz AMD X2 5200+ processor and 2GB of RAM.

After the first set of test runs it was decided to add multi-thread support to the K-means algorithm. In the ideal situation adding multi-thread support to the clustering algorithm can decrease processing time by up to half each time the number of available processor cores (parallel serviceable execution threads) is doubled.

Results:
What immediately pops out of the results is the difference in execution times between two runs of the experiment using the same initial parameters. The difference in execution time between runs ranges from 4-10%. An obvious reason for differences in runtime is due to the choice of initial cluster centroids. Whilst the number of clusters, vacancies and the dictionary size are kept constant, the initial clusters are not pre-defined, but chosen at random during each run, resulting in a different sets of clusters of varying sizes.

Extending the K-means algorithm from single to dual execution threads results in a speed up between 75-110%. One would not expect that running two parallel threads automatically translates to a halving of execution times. Cluster sizes vary, resulting in some threads taking
longer to complete than others. On average, doubling the number of execution threads should reduce the total processing time by almost half, given that the computer has enough resources available to efficiently service each execution thread.

<table>
<thead>
<tr>
<th>K</th>
<th># Vacancies</th>
<th>Single threaded</th>
<th>Dual threaded</th>
<th>Multi-thread Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Run 1</td>
<td>Run 2</td>
<td>Run 1</td>
</tr>
<tr>
<td>10</td>
<td>1051</td>
<td>2609</td>
<td>2808</td>
<td>1286</td>
</tr>
<tr>
<td>20</td>
<td>1051</td>
<td>4759</td>
<td>4423</td>
<td>2472</td>
</tr>
<tr>
<td>40</td>
<td>1051</td>
<td>8135</td>
<td>8430</td>
<td>4363</td>
</tr>
<tr>
<td>50</td>
<td>1051</td>
<td>10048</td>
<td>10170</td>
<td>5490</td>
</tr>
<tr>
<td>75</td>
<td>1051</td>
<td>13910</td>
<td>14111</td>
<td>7980</td>
</tr>
<tr>
<td>100</td>
<td>1051</td>
<td>18168</td>
<td>18079</td>
<td>10787</td>
</tr>
</tbody>
</table>

The results also show that the execution time of the K-means algorithm is directly proportional to K and the number of vacancies being clustered. Doubling the number of clusters results in a doubling of the total execution time. Tripling the number of vacancies triples the total execution time.

![Cluster size occurrence rate](image)
each cluster would contain 20 vacancies. We find that half the vacancies are contained in clusters with 2-28 vacancies, whilst the remaining vacancies are found in clusters with 29-106 vacancies.

Overall speed of the clustering algorithm is not ideal as PHP was not designed for such intensive number crunching. However, the possibilities and capabilities of the K-means algorithm have been shown. In a real world scenario, where speed and resource efficiency is important, a Java/C(++) component could be created to perform the clustering.

It is interesting to see how the clustering algorithm scales in relation to processor speed. The experiment for $k=50$, with 1051 vacancies and a dictionary size of 10054 words was executed on a laptop running Windows 7 Home Premium 64bit with a 1,3 GHz Intel U4100 processor and 4GB of RAM. Halving the processor speed increased execution time by roughly 67%. Initially one would expect a doubling of execution time when halving the processor speed, however, the difference could be attributed to various factors such as using a different Windows version (Windows Vista 32bit vs Windows 7 64bit), the difference in total system RAM, a difference in bandwidth between the processor and the memory, other architectural differences between AMD and Intel processors and/or differences between 32/64bit systems.
Experiment 3: Vacancy Cluster Accuracy

**Aim:**
The aim of this experiment is to verify that vacancies are clustered together correctly and that newly added vacancies are assigned to the correct cluster.

**Procedure:**
1. Cluster vacancies into $K$ clusters.
2. Subjectively check how closely vacancies in the cluster relate.
3. Repeat for different values of $K$.

The challenge with this experiment is to comment on the accuracy of the vacancy clusters. The number of clusters, $K$, the variety in job functions, the number of vacancies, the quality of the vacancy text, the size of the term dictionary and the set of initial cluster centroids are but some of the factors which influence the quality of the found clusters.

When the number of clusters is small, many unrelated vacancies will be mapped to the same cluster as it is highly likely that a cluster centroid for the given vacancy does not exist. As more and more unrelated vacancies are added to a cluster, the cluster centroid shifts resulting in groups of clusters which may have similar contents.

When the number of clusters is high, the likelihood of related vacancies being assigned to the same cluster increases as long as the choice of initial cluster centroids reflects the variety in vacancies stored in the vacancy database.

Dictionary size and quality is paramount to good quality clusters. Terms which are not in the dictionary are not used by the clustering algorithm. New vacancy types may therefore not be correctly identified and as a result will likely not be grouped with related vacancies. A dictionary of all possible skills, tool sets, applications, knowledge areas, etc. is therefore desirable. Unfortunately, as the size of the dictionary is increased, so is the size of the vacancy term vector, slowing down processing time and increasing memory requirements when calculating new cluster centroids. A dictionary of 10,000 stemmed terms extracted from the vacancies is used.

**Results:**
Analysis of cluster cohesion is based on the vacancy titles. Vacancies with similar titles should end up grouped together, assuming that the vacancy requirement texts are similar.

**K = 75, 1051 vacancies**
With 75 clusters there are likely to be too few clusters to create unique groups of related vacancies. The combination of the initial choice of random cluster centroids and recalculating the cluster centroids after every 250 vacancies have been added is also likely to cause dramatic changes to cluster composition as vacancies are added.
Looking at vacancy titles of cluster members we generally find groups of related vacancies. Vacancies often look to be grouped either by industry or by function. Below is a shortened summary of industries/functions and function titles found within some of the clusters (the Dutch names have been translated):

- **Sales:**
  - Sales advisor, car salesman, sales manager, sales person;
  - Shop manager, sales person, sales support, sales advisor.

- **Marketing:**
  - Marketeer, senior brand marketing, assistant brand & sales marketing, marketeer employment market.

- **Construction:**
  - Project engineer, planner concrete, civil builder, foreman, installer, interior builder, draftsman.

- **Financial:**
  - (Team manager) credit/debit.

- **Hair salon:**
  - Salon head, hairdresser.

- **Carpenter:**
  - All round, student, foreman.

- **Production (factory):**
  - Various.

- **Car mechanic:**
  - Car technician, MOT mechanic.

- **Healthcare:**
  - Maternity nurse, home care, dentist, nurse, healthcare consultant, home care nurse, orthopaedic nurse, corporate wellness.

- **Transport / Logistics:**
  - Transport / logistics planner;
  - Distribution driver, forklift truck driver, logistics employee;
  - Drivers (car / lorry).

- **Management:**
  - Manager, General manager, region manager, store manager.

- **Cleaner:**
  - Cleaner, household help.

- **Gardener.**

- **Account manager / Relation management.**

- **Engineering**
  - Electrical engineer, engineer, service engineer, welder, hardware engineer.

These are but some of the groups of professions found within the clusters. Looking at cluster composition we may find clusters of “similar” professions, for example, one cluster contained cooks, chamber maids/cleaners, and starter administrative assistants, whereas another cluster
contained different types of engineers, for example, welders, electrical engineers, geotechnicians, production engineers, etc.

However, not all clusters contained vacancies which appear to be related, for example, one cluster contained an information analyst, HR advisor, sales person and financial advisor mortgages. Though sometimes found clustered together, IT vacancies are often found spread across a large number of clusters. The question arises why this is the case.

Considering the large variety of vacancies, it takes time for the cluster contents/centroids to settle. Once “enough” similar vacancies are grouped together, any new similar additions should be added to that cluster. The question is, how many vacancies is enough? This is likely to be related to the number of clusters and the degree of variety in vacancies.

Cluster accuracy could be greatly improved by recalculating the cluster centroid after every update to the cluster. The drawback of this approach is that it is computationally expensive. However, considering that the clusters still show groups of professions, it can be argued that the approach taken delivers an acceptable trade-off. Perhaps recalculating cluster centroids should happen often when initialising the clusters and cluster size is small, but as cluster size increases, the frequency of recalculating cluster centroids can be reduced, as the cluster centroids should have settled.

\[ K = 100, 1051 \text{ vacancies} \]

The increase in the number of clusters clusters pays dividends as reviewing the cluster contents shows that there is less disparity between the cluster members. The clusters members show a greater cohesion between function title and/or industry. The cluster containing healthcare vacancies now only contains one vacancy outside this industry.

We notice that some groups of professions/industries which were spread thinly before are now concentrated in a much smaller number of clusters. A good example of this are the IT, cook and financial vacancies. An interesting change is that vacancies for people with little to no education (e.g. summer jobs, a paper round and bar staff) are found within the same cluster.

As mentioned earlier, the improvement in cluster membership is in part due to the choice of initial cluster centroids. Re-starting the clustering process from scratch will lead to a completely different set of clusters, though it should still result in groups of similar vacancies within the clusters.

\[ K = 150, 3035 \text{ vacancies} \]

It was chosen to increase the number of vacancies to cluster from 1051 to 3035 for this test. We immediately notice two things when looking at the cluster members, firstly, we once again find groups of similar function titles with a cluster, and secondly, we find seemingly unrelated groups of vacancies within a cluster. This last observation can be explained in part by the increase in number of clustered vacancies from 1051 to 3035. This is almost a tripling of the number of vacancies whilst the number of clusters has only increased by 50%. This means that a higher number and variety of vacancies must be mapped onto a relatively smaller number of clusters.
The increase in vacancies has made new groups of professions, such as teachers, paper boy, secretaries and receptionists, appear within the clusters. The new clusters once again show that there are groups of professions that cluster well, e.g.:

- Teachers;
- Medical/Healthcare;
- Draftsman;
- Drivers (lorry, forklift, etc.);
- Mechanics.

The IT vacancies however are spread much more thinly across the clusters again. It could be that the dictionary does not contain enough terms relevant to the IT industry.

The entire application assumes that only Dutch language vacancies are processed. However, there is a small number of English language vacancies advertised within the vacancy feeds. No pre-processing has been done to identify and remove these vacancies from the results. Interestingly and quite unexpectedly, the English language vacancies are clustered together. In hindsight this is logical, as the vocabulary of the English language vacancies is barely shared with the Dutch language vacancies.

**K = 150, 4000 vacancies, 12369 dictionary terms**

For the final cluster accuracy test the clustering parameters were changed slightly. Whilst the number of clusters remained at 150, the dictionary was recreated using 10,000 parsed vacancies to extract term and term frequencies from. In turn, this was used to create the term and term frequency vectors for 4,000 vacancies which were then clustered.

Recreating the dictionary using 10,000 vacancies as a corpus resulted in 37,343 terms in the dictionary. All terms occurring only once in the 10,000 vacancies can be discounted from further usage (invalidated), as this means that there are currently no other vacancies using these terms. Invalidating these terms reduced the valid terms in the dictionary by 15,667. Similarly, terms occurring twice/thrice were invalidated, resulting in a further reduction of 9,029 valid dictionary terms. 278 out of 348 terms occurring over 400 times in the vacancies were also invalidated as their high occurrence rate reduces the intrinsic value of these terms. It was chosen not to invalidate all terms occurring over 400 times as some were deemed useful to describe vacancy contents, e.g. “accountant” and “transport”.

Manually reviewing cluster contents shows that cohesion between cluster members has improved. Clusters of professions/job functions identified in previous tests appear again, as well as some new or improved clusters. Legal jobs are clearly grouped closely together as well as tax advisers and security guards. The most obvious improvement is in IT vacancies. In previous tests these vacancies were spread thinly across clusters, however now clear groups are seen:

- .NET C# developers/testers;
- PHP developers/testers;
5 Experiments and Results

• SQL / Oracle developers/database administrators;
• Java developers/scrum masters;
• Virtualisation specialists;
• IT consultants.

Most interesting is not only that the IT vacancies are now clustered, but they are clustered together by requirements. The software development vacancies do not all appear in the same cluster, however, most .NET C# vacancies, for example, do appear in the same cluster. This pattern repeats itself for the functions mentioned above.

The question arises where the improvement has come from. Clearly the choice of initial cluster centroids and the order in which vacancies are added to these clusters has a strong effect on how cluster membership develops. An important change with this test run was recreating the dictionary using a corpus of 10,000 vacancies. It is more likely that the improved list of dictionary terms, and the resulting term (frequency) vectors, picked up many missing IT related terms enabling a better comparison of texts.

**Overall Conclusion**
Large clusters contain vacancies with a large variety of job functions/industries. This suggests that these vacancies either have very few characteristic features in common and/or the requirements text has not be extracted properly leading to similarities in the extracted vacancy text from the HTML vacancy template. This disparity may also be due to the high variety in vacancies being mapped to a small number of clusters.

As expected, increasing the number of clusters increases the quality of the cohesion between cluster members as a finer granularity of cluster members is achieved. However, this does not tell us what the optimal number of clusters is. The variety in stemmed vacancy titles or the industry of the vacancy may give us a clue.

The initial cluster centroid choice greatly impacts cluster formation. It could be useful to investigate improvements in the choice of the initial cluster centroids. Instead of randomly choosing \( k \) centroids, it could be beneficial to ensure that the initial centroids are not related.

The final test shows that increasing the dictionary size has a clear impact on improving cluster quality. Adding more terms to the dictionary should further improve cluster accuracy, however this will come at the cost of the speed that vacancies are clustered at and the memory requirements of the system. A better approach may be to create separate clusters for (all) industries, identify the industry that a vacancy pertains to, and then cluster the vacancy in this sub group using a domain specific dictionary. This should not only create better clusters, but would also allow vacancies to be described using only those terms that relate to the industry, possibly reducing the size of the term vector.
Experiment 4: Matching CV to Vacancy

Aim:
The aim of this experiment is to see whether we can match a CV to relevant vacancies. Of importance is whether the found matches (e.g. top 10 list) are relevant and ordered as expected.

Procedure:
1. Select a decomposed CV.
2. Run the matching component on the selected CV.
3. Review the result list and comment on the quality.
4. If required, modify the weight vector and repeat steps 1-3.

Subjectively see if the matchings are relevant. Initially the test match threshold is set to 0 so that all results are returned. This is to see if results are being assigned a low match score when a high score is expected. This approach also makes it possible to investigate a possible value for the threshold without having to obtain it through experimentation.

Results:
Initial testing resulted in very strange matches. The first step in investigating this issue was to see how the scores for individual features of each match were scored. This showed that the problem came from the cosine similarity score between the vacancy requirements text and the CV's skill set. The cosine similarity scores were on average at least a factor 3-5 smaller than the scores for other features. As a result the weight for the skills match was increase by a factor 10 to add weight to this feature. Making this changes offered a slight improvement to the results, but in general the results were still poor.

The effects of cluster quality can also not be ignored. Each CV is first matched against all cluster centroids after which the CV is matched to the members of the 10 closest matching clusters. The centroids of the clusters reflect the average of the cluster members. How vacancies are distributed between the clusters therefore greatly impacts which clusters will be selected. Querying the vacancy database for the term “debit” revealed 20 clusters. This means that at least 10 clusters containing relevant results will be ignored. Also, the clusters containing this term may not even be selected as the remaining terms in the term vector co-determine which clusters are selected during the cosine similarity calculation.

Further investigation showed that CV quality plays enormous parts in the generally low quality of the search results. The CVs displaying decent matches were CVs which listed many exact skills and certificates, such as SAP, Windows Server, etc. The bad quality matches came from CVs which were simple, i.e. containing little to no certificates, software packages or descriptions of the performed tasks during previous employment.

The debit-credit administrator matched poorly. Many results were for sales positions. This
could be in part because of the word “verkoop” (sales in Dutch) occurring often in the CV making this a heavily weighted term.

The CV of a starting HR consultant was poor. Lack of experience and skills meant that matching to a relevant CV would be difficult. This was reflected in the results where no HR vacancies or other seemingly relevant results were found.

The secretary's CV gave some decent results in the top, but also some very strange results, e.g. sales, transport planner and even a cleaner.

At first glance the results of matching the CV of a manager in the food & beverages industry seemed wrong, as there were many administrative vacancies listed. After reviewing the CV it became clear that these results could be considered relevant. The skill set contained within the CV are comparable to someone performing these tasks and so the results could be considered accurate.

Which CVs resulted in decent matches? The CVs of a DTP-er, Junior Controller, Sales Manager, a production worker and building maintenance engineer delivered decent results. Vacancies relating to the CV profile were found within the top 20 matches.

The truck driver CV and my personal CV exhibited the best matches. The truck driver CV is quite unique in the fact that it is the only CV with the word “chauffeur” (driver in Dutch). As only vacancies for drivers contain this term, the matches are of high quality. As for my CV, compared to the sample CVs, it is extensive in the lists of skills, proficiencies and performed tasks. Almost all the results are what I would expect to see. Software developers (for the inter-/intranet), system-/network administrators and some consultancy jobs are to be found, with the system administration jobs leading the pack.

The point at which there is a clear crossover from relevant vacancies to completely irrelevant vacancies varied per CV. One reason is that vacancy cluster sizes vary and if a cluster with many irrelevant vacancies is chosen, only a few results will be relevant. It was clearly noticeable that for the better matching CVs the percentage of overall relevant vacancies was substantially higher. Irrelevant matches for these CVs started at a score of between 40-60% of the highest scoring match.

When the CVs were inserted into the database it was chosen to insert the majority of the CV text (education, skill set, job history, achieved certificates, languages, etc.) as the CV profile text to match the vacancies against. The reasoning behind this was that this is also contained within the vacancy requirements text and so a closer cosine similarity score should be found. Whilst this is correct, it also removes weights from the remaining terms and further boosts the scores for terms (e.g. the languages) which are being scored separately as well. It was chosen to retry the experiment keeping only the followed education, achieved certificates, job history and skill set. The effect of this was an overall small though noticeable improvement.

**Overall Conclusion**
Clusters play a key role in this approach to matching CVs to vacancies. It is therefore
paramount to maximise cluster quality. Considering the variety and quality of stored vacancies the first step to improvement should be to create finer clusters of vacancies. Just increasing the number of clusters or improving initial seeding of cluster members will help, but only to a certain point. A big disadvantage of this approach is that increasing the number of clusters also increases processing time when comparing new vacancies to the cluster centroids. Instead, cluster groups should be created for individual domains, e.g. industry or sector, and vacancies should be clustered within the appropriate domain. Naturally this means that the domain of the vacancy would have to be discovered.

Another improvement is utilising the k-means algorithm in the manner it was intended, i.e. updating the cluster centroid after each insertion to the cluster. Currently the cluster centroids are re-calculated after 250 vacancies are inserted (split over the 150 clusters). This means that cluster centroids may shift greatly after the centroids are re-calculated.

Domain specific dictionaries are essential to potentially reduce the size of the term (frequency) vectors whilst improving their quality.

The number of stored vacancies and their variety also greatly effects the quality of the matches. Results are likely to change, for the better or worse as the number of vacancies in the database increases.
Experiment 5: Result Ranking Keyword Search Results

Aim:
The aim of this experiment is to see if the quality and ordering of keyword search results, performed on the parsed vacancy database using MySQL's built-in relevancy measure, can be improved by applying the ranking component to the search results.

Procedure:
1. Create a search query.
2. Execute the query with and without the ranking component enabled.
3. Subjectively compare the quality of both results lists.
4. Repeat steps 1-3 for a number of different queries.

Results:
This experiment was conducted on 10 queries (see table 16). Whilst this may seem a small sample, a pattern to the results emerged quite quickly.

The first and most noticeable observation throughout the experiment was the difference in how and where results were found within the result list once a query was re-executed with the ranking component enabled. Whilst the search results using only MySQL's relevancy measure generally contained good results, the placement of these results within the results list was often questionable. Junior and senior functions for example were often found throughout the results list, jumbled through each other. When the same queries were performed with the ranking component enabled, it was immediately noticeable that in general the junior functions were grouped together and the senior functions were grouped together. A query on the terms chipsoft and sap resulted in medical sectors vacancies appearing at the top of the results list with the ranking component enabled, whilst these entries were scattered across the results list when the ranking component was switched off.

There was no noticeable degradation in the quality of the ordering of the results when the ranking component was enabled. Even though the result ordering was not always perfect, it was definitely not worse.

The ranking is based on the highest scored result from the MySQL results list, so a mismatch here will greatly effect the results of the ranking component. Results will still be grouped together, however the top ranking results may not seem the obvious ones. Going back to the example of a senior/junior function, depending which of these is returned as the top result, the remaining results will be ordered accordingly, junior or senior functions first. As vacancies are added/removed from the database, the same query may cause the results to change unpredictably. This partly shows the importance of using boolean search operators in a query. Unfortunately, as discovered during the interviews with recruiters, this is not always possible.
Interestingly it was also noticed with some queries that English language results were being pushed towards the bottom of the results list when the ranking component was enabled. This is likely due to the low overlap in vocabulary between the Dutch and English language vacancies.

### Table 16: Ranking Component Queries

<table>
<thead>
<tr>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>chipsoft sap</td>
</tr>
<tr>
<td>&quot;windows server 2008&quot; +mcitp exchange</td>
</tr>
<tr>
<td>(&gt;senior belastingadviseur)</td>
</tr>
<tr>
<td>&quot;brand manager&quot;</td>
</tr>
<tr>
<td>zorg consultant</td>
</tr>
<tr>
<td>&quot;net developer&quot;</td>
</tr>
<tr>
<td>(&gt;junior) +&quot;business controller&quot;</td>
</tr>
<tr>
<td>&quot;applicatie beheerder&quot;</td>
</tr>
<tr>
<td>php mysql ajax</td>
</tr>
<tr>
<td>tekenaar autocad solidworks inventor</td>
</tr>
</tbody>
</table>
6 Conclusion

This chapter presents an overall conclusion of the work done for this thesis as well as future recommendations that arose while working on the thesis research.

Using the research challenges defined in chapter 1.2 it will first be discussed if and how these challenges have been met. Next, future work and possible improvements are discussed.

6.1 Goals and objectives

In the introduction the following high level goals were defined to demarcate the research done for this thesis:

1. Automatically retrieve vacancies from the Internet and store them in a local database.
2. Automatically decompose vacancies and store the results in a local database.
3. Implement a local database in which to store curricula vitae and their (automatically) extracted features.
4. Automatically cluster decomposed vacancies into groups of related vacancies.
5. Design and implement an algorithm to match vacancies to CVs and vice versa.
6. Design and implement an algorithm to rank search results.
7. Test the implemented components, i.e. the vacancy retrieval, vacancy parser, vacancy clusterer, CV parser, matcher and result ranking components.

A detailed overview will now be given of the results of each goal.

6.1.1 Automatically retrieve vacancies from the Internet and store them in a local database

The vacancy database has been created to store vacancies that are retrieved from online job boards. It currently stores 17,764 vacancies retrieved from five job boards (see chapter 4.7 for details).

The initial idea to create an agent to crawl job boards and retrieve vacancies by interacting with the job boards' search forms was dropped due to its relatively high complexity (in relation to the returns offered). It was replaced by a component (chapters 3.3/4.2) that retrieves vacancies using RSS vacancy feeds offered by job boards. This method provides a simpler and more generic means to retrieve vacancies. An added benefit of this approach is that most job boards offer the possibility to add selection criteria to the vacancy feeds,
meaning that pre-filtered sets of vacancies can be retrieved if desired.

The choice of characteristics (see figure 56) to store in the vacancy database, besides the vacancy (HTML page), is determined in part by the information found for each RSS feed element.

6.1.2 Automatically decompose vacancies and store the results in a local database

The retrieved vacancies are useless unless the information stored within them can be extracted and stored for use further down the processing pipeline. The choice for what data to extract was based on information gathered during interviews with recruiters and common sense. Once these features had been determined the database could be implemented and filled by creating the vacancy parsing component that locates and extracts the features using a separate data extractor for each feature.

The overall performance of the data extractors was positive (see table 14), considering that they do not require a fixed template or tagged training set to work, and enabled the creation of a usable decomposed vacancy database.

The extractors for the general description and requirements text require improvement. The variation in page layout/template, especially on Monsterboard, causes issues determining the boundaries of these sections within some of the vacancies. In some cases the blame falls on the HTML code of the retrieved vacancies. Not all vacancies had valid HTML code, causing the pre-processing stage to inadvertently remove HTML tags along with vacancy content. More research needs to be performed to find better identifiers for these sections.

The function title is extracted from the HTML <TITLE> tag after which an attempt is made to remove irrelevant terms, such as location/salary of the vacancy from the function title as these details should not be stored as part of the function title. Whilst the results of the vacancy title data extractor are good, more work could be done to remove the irrelevant information from the title. Improvements could be achieved by looking for an overlap between the function title and vacancy text, as the function title is often repeated at the beginning of the vacancy text without the offending extra details.

6.1.3 Implement a local database to store curricula vitae and their (automatically) extracted features in

As discussed earlier, commercial software exists that decomposes curricula vitae into their constituent parts. As it is not the purpose of this thesis to reinvent the wheel, it was chosen to manually decompose a handful of CVs for testing purposes and directly insert the results into the CV database. A user interface (figure 19) is present to add/update CVs to- and delete CVs from the CV database.

The CV database was implemented using the criteria used by the matching algorithm (table 8)
append by some fields required by the matching component (figure 60).

### 6.1.4 Automatically cluster decomposed vacancies into groups of related vacancies

After researching document clustering techniques and similarity measures the choice was made to cluster the vacancies using K-means clustering and cosine similarity. This was both due to their proven track record in the document clustering field and their time complexity.

The single thread clustering speed was quite slow, but to some degree this had to be expected as clustering is performed using PHP code and not a more efficient programming language such as C/Java. This however was not a performance criteria for this research. Considering new cluster centroids are not calculated after every insertion, but only after every 250 insertions a further improvement in speed was expected. Multi-threading the clustering process to two parallel threads almost doubled throughput making clustering times more acceptable (table 15). It is very possible that the large term frequency vectors (>10K terms) of float values was also partially responsible for the slow performance.

Experiments conducted on the effects of the ratio between the number of clusters and the number of vacancies to cluster hinted that the lower the ratio between the two, the better the cluster quality.

Dictionary size affected cluster quality as the valid dictionary terms (i.e. not stop words, excluded words, etc.) are used to create each vacancy's term/term frequency vectors which are used in determining vacancy similarity. This was made apparent by the improvement in cohesion between cluster members when comparing the results of the final two tests of the third experiment where the number of valid dictionary terms was increased from ~10K to ~12K after rebuilding the dictionary using a corpus of 10,000 vacancies.

The question arises how best to reduce the size of the term vectors whilst improving cluster quality. Reducing the size of the float values of the term frequencies could help, as well as a better partitioning of the data. Term counting against domain specific dictionaries could be used to identify the domain of the vacancy after which the term/term frequency vectors would be created using the dictionary of the associated domain. This should also enable more accurate clustering, as vacancies would then be better partitioned within the assigned domain using only those terms specific to the domain at hand.

### 6.1.5 Design and implement an algorithm to match vacancies to CVs and vice versa

The core of the matching is performed using the cosine similarity measure between the CV's skill/proficiency text and the vacancy's requirements text. The remaining features used by the matching algorithm (table 8) reflect the mental matching performed by recruiters, with the addition of the driving license for reasons mentioned earlier, translating to the scoring function presented in figure 14.
Initially the CV skill/proficiency text included internships and features of the matching algorithm (table 8), however, this turned out to be a less than optimal idea, and after some experimentation it was chosen to only keep the job history, education, obtained certificates and skill set. This improved results slightly.

Whilst saving processing time, the choice to calculate the new cluster centroid after every 250 vacancies were assigned to the 150 clusters meant that cluster quality, while good, was suboptimal.

The presence of domain specific dictionaries would make it possible to maintain separate clusters of vacancies (per domain) and so potentially improve the clustering and matching processes.

To some degree the results of this experiment were disappointing as higher quality matches were expected beforehand. In hindsight, the results are not as bad as they seem. It has been made ever more apparent how important it is to have a high quality CV that describes all the skills a person possesses, as the matches that did deliver good results were those for CVs containing ample information about the skill set of the individual.

### 6.1.6 Design and implement an algorithm to rank search results

Ranking search results would be trivial if the vacancy clusters were of such high quality that we knew that all results coming from a single cluster are relevant. Unfortunately this is not the case.

Ranking is performed by taking the results of the keyword search, taking the best result and ranking all remaining results according to the cosine similarity score between them and the best result. This was a very simple technique to implement and the positive effects were immediately noticeable. Related results were grouped close(r) together with the results most similar to the top ranking result nearest the top.

Removal of irrelevant results using a lower threshold for the cosine similarity score was not used as the quality of the results found during experimentation were generally high. Obtaining a value for the lower threshold requires further investigation as results from a larger vacancy database are likely to have more irrelevant results show up.

The importance of using boolean search operators was also found to be very important to obtain the desired results. Whilst this is an obvious statement, it is not obvious why this is not available on all job boards and has even been disabled on some.

**Contribution to the Field**

Whilst the data extractors constructed to extract information from the vacancies are not perfect, the retrieved and parsed vacancies stored within their respective databases offer anyone wishing to do research into vacancy composition, clustering and data-mining a corpus

---

Page 104
of almost 18,000 parsed and unparsed vacancies to work with. This was unavailable at the onset of the project.

For those that wish to create their own vacancy dataset, the vacancy retrieval component provides a means for them to retrieve and store the vacancies using vacancy RSS feeds.

### 6.2 Future Work / Improvements

The performed experiments show that there is much room for improvement. How the improvements can be achieved is open to debate and clever solutions. It is clear that quality and performance can be improved during data extraction, vacancy clustering and matching.

Adding reviewing capabilities, i.e. manually check an unparsed vacancy and its parsed counterpart and fix any faults, is a method to improve the quality of the parsed results. Data-mining the manual alterations could provide clues to improve the data extractors. Alternatively, some form of machine learning could be applied to improve the quality of the extraction process based on the manual alterations. Manually reviewing every parsed vacancy is highly time consuming, so this could prove a good means to improve accuracy with minimal effort.

The current implementation lacks domain specific knowledge. This knowledge has the potential to both simplify the vacancy term frequency vector as well provide a better mapping between specific skills and what these skills entail. Imagine the mapping below:

```
Word Processing → MS Word, Word Perfect, Open Office Writer, AbiWord.
```

Instead of the term frequency vector having an entry for each word processor, it could simply have an entry “word processing”. This reduces the size of the term frequency vector, reducing the execution time of document similarity calculations. Obtaining/Creating the domain specific knowledge is the critical component to utilising this possibility.

Currently the list of stop words is general and applied to vacancies from all domains. If the domain of a vacancy could be determined before the term frequency vector is created, then a two stage removal of stop words could be applied to reduce the size of the term frequency vector. First a general stop word list could be applied to all vacancies, which removes words such as “the”, “it” and “of”. Next, a domain specific stop word list could be applied to further reduce the size of the term frequency vector. The catch here is how to determine the domain of a vacancy.

The implemented term frequency vectors are made up of float values. Though this results in finer results when calculating document similarity, this also increases processing times. It could be interesting to investigate how the processing time and document similarity calculations relate to the accuracy of the float value.
Considering how large a term frequency vector can get with a large dictionary it becomes obvious why some commercial software, such as Scientio's ConceptMap, creates a fingerprint/signature of data which us used in clustering and making the information searchable.

Improvements to the clustering process are desirable to speed up the process and improve accuracy. Currently there is no limit on the number of members a cluster can have. Surely there is a point at which an individual cluster should be divided into one or more new clusters. Should the metric be cluster size, or should a cluster be split when the difference between outliers within a cluster reaches a certain threshold? Alternatively, why not introduce multi-level clustering? Instead of choosing a high initial value for $k$, choose a lower value, and then divide the cluster members into sub-groups of finer clusters. This could be repeated forming multiple levels creating a tree-like structure. This has the potential to greatly reduce the number of calculations when calculating the new cluster centroids and when performing vacancy-CV matching.

Currently the ranking algorithm only uses the highest scoring result to rank the remaining results. It could be interesting to investigate how results can be ranked using the $n$-best results. This would assist discovery of related results and could open the potential of result grouping or suggesting additional vacancies based on the selected one. The challenge would be in finding a method to use the cosine similarity score lists of the $n$-best results and combine these into a meaningful ranked results list.

Reviewing the ranked results of keyword search on the vacancy database revealed a desire to re-order results based on an interesting vacancy from the results list. Whilst the surrounding vacancies are likely to be similar due to the use of cosine similarity in ranking, it would be interesting to investigate how the results list would evolve if it were re-ranked using the selected vacancy as the top ranked result. This may result in (completely) different results appearing close to the selected vacancy.

Currently job boards lack personal contact. A job seeker visits a job board, searches for appropriate vacancies, views the vacancies and perhaps applies to some or contacts the advertiser for further information on the vacancy. It is strange that there is no feedback on the profile of users viewing a vacancy. This is highly useful information for a job advertiser as it provides information on the profile of the people viewing the vacancy. This could be used to see if the target audience is being reached and if not, it can be used to alter the vacancy description or, if applicable, the CV-vacancy matching algorithm. Alternatively it could be used to discover target audiences which were overlooked.
Bibliography


Bibliography


Pacchioli, D.: Smart Search, Penn State Online. http://www.rps.psu.edu/0305/search.html


A: Databases

This section briefly describes the implementation of the database tables created for this project. These tables hold all the information retrieved from the Internet (e.g. the downloaded vacancies), the generated data (e.g. dictionary) and the matches between the CVs and vacancies.

RSS feeds
This table holds details on the vacancy RSS feeds which are used by the vacancy retrieval component to retrieve vacancies from the job boards. These RSS feed details are used to manage the feeds. Each RSS feed is identified by a unique ID (FeedID), the name of the website (SiteName), the URL of the feed (FeedURL), the name of the feed (FeedName), the character encoding used by the feed (Charset) and the date format (DateFormat).

<table>
<thead>
<tr>
<th>rssfeeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeedID : int(3)</td>
</tr>
<tr>
<td>SiteName : char(50)</td>
</tr>
<tr>
<td>FeedURL : varchar(500)</td>
</tr>
<tr>
<td>FeedName : char(50)</td>
</tr>
<tr>
<td>Charset : char(20)</td>
</tr>
<tr>
<td>DateFormat : char(30)</td>
</tr>
</tbody>
</table>

Figure 55: RSS Feed

Unparsed vacancies
As the RSS feed data is retrieved, the details of each item within the RSS feed is stored in the crawledfeedurls table. These are the unparsed vacancies. Each vacancy is assigned a unique ID (ID) and the contents of the RSS feed entry are stored in the fields Title, Description, URL, GUID and PubDate. The ID the associated RSS Feed is stored in FeedID.

Once a vacancy has been retrieved, the vacancy's record is updated with an HTML copy of the vacancy (PageContents), retrieval date (CrawlDate), vacancy (PageContents) and an updated status (CrawlStatus) indicating that the vacancy has been retrieved.
Parsed vacancies

Once the data extractors have done their job on an unparsed vacancy, the decomposed vacancy is stored in the parsedvacancies table. Besides containing the ID of the original vacancy (crawledfeedurlID), each parsed vacancy is assigned a new, unique ID (vacID). Fields exist to hold the data acquired by each data extractor. At a later stage, when pre-processing for the clustering process has generated term and term frequency vectors for the parsed vacancy, the record is updated with the term and term frequency vectors.

The status field is used to identify the stage of the clustering process that the vacancy resides in. An overview of the codes is displayed below.
CV
The CV database table is the counterpart of the database table storing the parsed vacancies (figure 59). It stores the entire CV text as well as the decomposed features of the CV's text that are required by the matching algorithm.

The status field is currently not used. It is present in case automated CV-vacancy matches are to be performed as batches in the background and the system administrator wants to keep track of which CVs have been processed.
Appendix

Dictionary
The dictionary contains all the terms which are recognised and utilised when generating the term and term frequency vectors for all parsed vacancies. Each entry contains a word (term), its total occurrence count (freq) over all the vacancies which were used to build the dictionary with and a validity flag (valid). When valid is set to 1, the term is considered to be valid, and is used in the term/term frequency vector. When valid is set to 0, the term is considered invalid, i.e. irrelevant, overused or a stop word, and is ignored when generating the term/term frequency vectors.

<table>
<thead>
<tr>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>term : varchar(50)</td>
</tr>
<tr>
<td>freq : int(10)</td>
</tr>
<tr>
<td>valid : tinyint(1)</td>
</tr>
</tbody>
</table>

*Figure 61: Dictionary Term*

topWords
The three topWords database tables store co-occurrence statistics (mono-, bi- and trigrams) on the vacancies stored within the vacancy database. These are statistics that were generated after downloading the initial set of 17,764 vacancies. These statistics helped formulate the stop words list and gain insight into word usage when creating the data extractors.

<table>
<thead>
<tr>
<th>topWords1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID : int(10)</td>
</tr>
<tr>
<td>w1 : varchar(255)</td>
</tr>
<tr>
<td>count : int(10)</td>
</tr>
</tbody>
</table>

*Figure 62: Vacancy Text Monograms*

<table>
<thead>
<tr>
<th>topWords2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID : int(10)</td>
</tr>
<tr>
<td>w1 : varchar(255)</td>
</tr>
<tr>
<td>w2 : varchar(255)</td>
</tr>
<tr>
<td>count : int(10)</td>
</tr>
</tbody>
</table>

*Figure 63: Vacancy Text Bigrams*

<table>
<thead>
<tr>
<th>topWords3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID : int(10)</td>
</tr>
<tr>
<td>w1 : varchar(255)</td>
</tr>
<tr>
<td>w2 : varchar(255)</td>
</tr>
<tr>
<td>w3 : varchar(255)</td>
</tr>
<tr>
<td>count : int(10)</td>
</tr>
</tbody>
</table>

*Figure 64: Vacancy Text Trigrams*

Matches
The matches database table stores records for the performed matches. Each record contains the ID of the CV that has been matched, the vacancy IDs of the n best matching vacancies from the highest scoring match to the lowest, and the match scores similarly sorted. vacIDs and scores are comma separated value strings.
## Matches

<table>
<thead>
<tr>
<th>cvID</th>
<th>: int(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vacIDs</td>
<td>: text</td>
</tr>
<tr>
<td>scores</td>
<td>: text</td>
</tr>
</tbody>
</table>

*Figure 65: CV-Vacancy Match*
B: Links

Academic Transfer
http://www.academictransfer.com

CareerBuilder
http://www.careerbuilder.nl

CROHO (Central Register Higher Education Degree Courses)

Dutch Porter Stemmer (Snowball)
http://tartarus.org/martin/PorterStemmer/

Dutch Porter Stemmer (Drupal)
http://drupal.org/project/dutchstemmer

Dutch Stop Words (Hogeschool Utrecht)
http://www.catalogus.hvu.nl/webopac/helpteksten/stopwoorden.htm

Dutch Stop Words (Oracle)
http://download.oracle.com/docs/cd/B28359_01/text.111/b28304/astopsup.htm#i634823

HODEX (Higher Education Data Exchange)
http://www.hodex.nl

Job Board Types
http://www.whatjobsite.com/An introduction to job boards and job sites.htm

JobTrack
http://www.jobtrack.nl

Monsterboard
http://www.monsterboard.nl

Nationale Vacature Bank
http://www.nationalevacaturebank.nl

Notepad++
http://www.notepad-plus-plus.org

PsExec
Appendix

TortoiseSVN
http://tortoisesvn.tigris.org

TREC datasets
http://trec.nist.gov

VisualSVN Server
http://www.visualsvn.com

VKbanen
http://www.vkbanen.nl

WAMP Server
http://www.wampserver.com
## C: Stop Word List

**Table 17: Stop Word List**

<table>
<thead>
<tr>
<th>aan</th>
<th>daar</th>
<th>eveneens</th>
<th>ja</th>
<th>niet</th>
<th>ten</th>
<th>waar</th>
</tr>
</thead>
<tbody>
<tr>
<td>aangaande</td>
<td>daargaan</td>
<td>evenwel</td>
<td>je</td>
<td>niets</td>
<td>tenzij</td>
<td>waarom</td>
</tr>
<tr>
<td>aangezien</td>
<td>daarin</td>
<td>gauw</td>
<td>jezelf</td>
<td>noch</td>
<td>ter</td>
<td>wanneer</td>
</tr>
<tr>
<td>achter</td>
<td>daarna</td>
<td>ge</td>
<td>jij</td>
<td>nog</td>
<td>terwijl</td>
<td>want</td>
</tr>
<tr>
<td>achterna</td>
<td>daarnet</td>
<td>gedurende</td>
<td>jizelf</td>
<td>nogal</td>
<td>thans</td>
<td>waren</td>
</tr>
<tr>
<td>af</td>
<td>daarom</td>
<td>geen</td>
<td>jou</td>
<td>nu</td>
<td>tijdens</td>
<td>was</td>
</tr>
<tr>
<td>afgelopen</td>
<td>daarop</td>
<td>gehad</td>
<td>jouw</td>
<td>oa</td>
<td>toch</td>
<td>wat</td>
</tr>
<tr>
<td>al</td>
<td>daanaanlangs</td>
<td>gekund</td>
<td>jouwe</td>
<td>of</td>
<td>toen</td>
<td>we</td>
</tr>
<tr>
<td>aldaar</td>
<td>dan</td>
<td>geleden</td>
<td>juist</td>
<td>ofschoon</td>
<td>toenmaals</td>
<td>weer</td>
</tr>
<tr>
<td>aldus</td>
<td>dat</td>
<td>gelijk</td>
<td>jullie</td>
<td>om</td>
<td>toenmalig</td>
<td>weg</td>
</tr>
<tr>
<td>alhoewel</td>
<td>de</td>
<td>gemoeten</td>
<td>kan</td>
<td>omdat</td>
<td>tot</td>
<td>tegens</td>
</tr>
<tr>
<td>alias</td>
<td>den</td>
<td>gemogen</td>
<td>klaar</td>
<td>omhoog</td>
<td>totdat</td>
<td>wel</td>
</tr>
<tr>
<td>alle</td>
<td>der</td>
<td>geweest</td>
<td>kon</td>
<td>omlaag</td>
<td>tussen</td>
<td>weldra</td>
</tr>
<tr>
<td>allebei</td>
<td>des</td>
<td>gewoon</td>
<td>konden</td>
<td>omstreeks</td>
<td>u</td>
<td>welk</td>
</tr>
<tr>
<td>alleen</td>
<td>deze</td>
<td>gewoonweg</td>
<td>krachtens</td>
<td>omtrent</td>
<td>uit</td>
<td>welke</td>
</tr>
<tr>
<td>alles</td>
<td>die</td>
<td>haar</td>
<td>kunnen</td>
<td>omver</td>
<td>uitgezonderd</td>
<td>werd</td>
</tr>
<tr>
<td>als</td>
<td>dikwijls</td>
<td>had</td>
<td>kunt</td>
<td>onder</td>
<td>uw</td>
<td>wezen</td>
</tr>
<tr>
<td>alsnog</td>
<td>dit</td>
<td>hadden</td>
<td>later</td>
<td>ondertussen</td>
<td>vaak</td>
<td>wie</td>
</tr>
<tr>
<td>altijd</td>
<td>doch</td>
<td>hare</td>
<td>liever</td>
<td>ongeveer</td>
<td>vacature</td>
<td>wiens</td>
</tr>
<tr>
<td>altoos</td>
<td>doen</td>
<td>heb</td>
<td>maar</td>
<td>ons</td>
<td>vacatures</td>
<td>wier</td>
</tr>
<tr>
<td>ander</td>
<td>door</td>
<td>hebben</td>
<td>mag</td>
<td>onszelf</td>
<td>van</td>
<td>wij</td>
</tr>
<tr>
<td>andere</td>
<td>doorgaand</td>
<td>heeft</td>
<td>me</td>
<td>onze</td>
<td>vanaf</td>
<td>wijzelf</td>
</tr>
<tr>
<td>anders</td>
<td>dus</td>
<td>heeft</td>
<td>meer</td>
<td>ook</td>
<td>vandaan</td>
<td>wil</td>
</tr>
<tr>
<td>anderszins</td>
<td>echter</td>
<td>hem</td>
<td>men</td>
<td>op</td>
<td>vanuit</td>
<td>worden</td>
</tr>
<tr>
<td>baan</td>
<td>een</td>
<td>hen</td>
<td>met</td>
<td>opnieuw</td>
<td>vanwege</td>
<td>wordt</td>
</tr>
<tr>
<td>behalve</td>
<td>eens</td>
<td>het</td>
<td>mezelf</td>
<td>opzij</td>
<td>veel</td>
<td>zal</td>
</tr>
<tr>
<td>behoudens</td>
<td>eer</td>
<td>hier</td>
<td>mijn</td>
<td>over</td>
<td>veelerz</td>
<td>ze</td>
</tr>
<tr>
<td>beide</td>
<td>eerder</td>
<td>hierbeneden</td>
<td>mijn</td>
<td>overeind</td>
<td>verder</td>
<td>zei</td>
</tr>
<tr>
<td>beiden</td>
<td>eerder</td>
<td>hierboven</td>
<td>mijnent</td>
<td>overigens</td>
<td>vervolgens</td>
<td>zelf</td>
</tr>
<tr>
<td>ben</td>
<td>eerlang</td>
<td>hierin</td>
<td>mijner</td>
<td>pas</td>
<td>vol</td>
<td>zelfs</td>
</tr>
<tr>
<td>beneden</td>
<td>eerst</td>
<td>hij</td>
<td>mijzelf</td>
<td>precies</td>
<td>volgens</td>
<td>zich</td>
</tr>
<tr>
<td>bent</td>
<td>elk</td>
<td>hoe</td>
<td>misschien</td>
<td>reeds</td>
<td>voor</td>
<td>zichzelf</td>
</tr>
<tr>
<td>bepaald</td>
<td>elke</td>
<td>hoewel</td>
<td>mocht</td>
<td>rond</td>
<td>vooraf</td>
<td>zij</td>
</tr>
<tr>
<td>betreffende</td>
<td>en</td>
<td>home</td>
<td>mochten</td>
<td>rondom</td>
<td>vooral</td>
<td>zijn</td>
</tr>
<tr>
<td>bij</td>
<td>enig</td>
<td>hun</td>
<td>moest</td>
<td>sedert</td>
<td>vooralsnog</td>
<td>zjine</td>
</tr>
<tr>
<td>binnen</td>
<td>enige</td>
<td>hunne</td>
<td>moesten</td>
<td>sinds</td>
<td>voorbij</td>
<td>zo</td>
</tr>
<tr>
<td>binnenin</td>
<td>enigszins</td>
<td>iemand</td>
<td>moet</td>
<td>sindsdien</td>
<td>voordat</td>
<td>zodka</td>
</tr>
<tr>
<td>boven</td>
<td>enkel</td>
<td>iets</td>
<td>moeten</td>
<td>slechts</td>
<td>voordezen</td>
<td>zonder</td>
</tr>
<tr>
<td>bovenafl</td>
<td>enkele</td>
<td>ik</td>
<td>mogen</td>
<td>sommige</td>
<td>voordien</td>
<td>zou</td>
</tr>
<tr>
<td>bovendien</td>
<td>enz</td>
<td>ikszelf</td>
<td>na</td>
<td>spoedig</td>
<td>voorheen</td>
<td>zouden</td>
</tr>
<tr>
<td>bovengenoemd</td>
<td>er</td>
<td>in</td>
<td>naar</td>
<td>steeds</td>
<td>voorop</td>
<td>zowat</td>
</tr>
<tr>
<td>bovenstaand</td>
<td>erdoor</td>
<td>innmiddels</td>
<td>nabij</td>
<td>tamelijk</td>
<td>vooruit</td>
<td>zulke</td>
</tr>
<tr>
<td>bovenvermeld</td>
<td>etc</td>
<td>inzake</td>
<td>nadat</td>
<td>te</td>
<td>vrij</td>
<td>zullen</td>
</tr>
<tr>
<td>buiten</td>
<td>even</td>
<td>is</td>
<td>net</td>
<td>tegen</td>
<td>vroeg</td>
<td>zult</td>
</tr>
</tbody>
</table>