January 2014

# Automated Usability Testing Of Websites Using Link Structure

Christoffer Odegaard Korvald

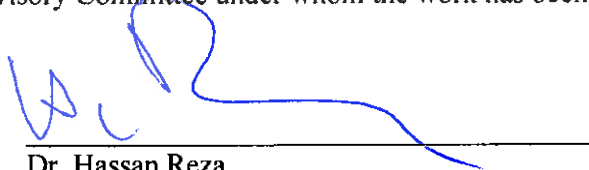# Automated Usability Testing of Websites using Link Structure

By

Christoffer Ødegaard Korvald
Bachelor of Arts, University of North Dakota, 2012

A Thesis
Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
December
2014
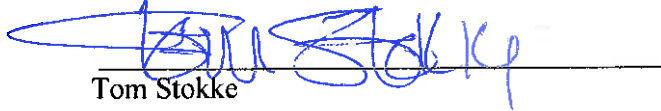
This thesis, submitted by Christoffer Ødegaard Korvald in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

Dr. Hassan Reza

Dr. Wen-Chen Hu

Tom Stokke

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies of the University of North Dakota and is hereby approved.

Wayne Swisher,
Dean of the School of Graduate Studies

September 17, 2014
Date

# PERMISSION

Title                Automated Usability Testing of Websites using Link Structure

Department      Computer Science

Degree            Master of Science
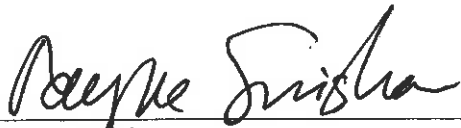

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.



Christoffer Ødegaard Korvald

August 27, 2014

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. Hassan Reza, Dr. Wen-Chen Hu and Tom Stokke for being members of my committee. I am so thankful for their guidance and support during my time in the master's program at the University of North Dakota. I would also like to thank all the faculty and staff at the Computer Science Department for the help they provided during my undergraduate and graduate career.

# ABSTRACT

Websites keeps getting more important in business and other aspects of society. Making the websites as usable as possible is crucial as difficult to use systems tend to frustrate users, which might lead to users leaving or lost revenue for a business. Usability testing is needed to identify and fix those issues. Manual tests in usability labs can be very time consuming and costly. An automated system could reduce time and cost of testing, but are often too focused on one aspect to give a clear view of what needs to be fixed. A system to improve this is needed. 4 separate modules focusing on different aspects of testing the information structure and navigation of a website are implemented and tested. The modules are combined in a system that gather the results from each module and provide a better overview of the usability issues of a website.

# CHAPTER I
# INTRODUCTION

It has become very important for every company and organization to have an online presence today. It is not only a valuable source of information for users, but can be the livelihood for a company. With the growing number of websites, the complexity and amount of information on the Internet keeps growing. Keeping all that information organized and accessible to the end user is a challenging task for a website owner. With the growing number of users and websites the competition is getting harsher and the website owner will as a result have to work harder to retain or attract new users. Wu and Offutt  (Wu & Offutt, 2002)  mention how there appears to be no "site loyalty" when it comes to websites, which can make it very hard to retain users when they are very likely to move on to another website of better quality. This is one of the reasons why making sure a website meet user expectations is very important and failure to do so can result in lost revenue.

User satisfaction with websites comes down to many factors and usability is just one of them. If a site is perceived by a user to not be very usable it will most likely discourage them from completing their transaction or coming back at a later time. Most users navigate to a website with a specific goal in mind. The goal might be to purchase goods, gather information, communicate with other users, and so on. To aid the user in accomplishing the goal the website should be organized in a logical manner. Performing

tests to uncover potential usability issues is therefore crucial to any website owner. Usability testing is usually done in two ways (classical and automated) (Rukshan & Baravalle, 2012) and will be discussed in further detail later.

The problem with existing work is that they either have a manual process that requires a lot of time and resources to perform or automated testing that might not uncover the issues. The few that focus on automated usability testing have a very narrow view of the issues and therefore cannot uncover some of the other issues with a website or they still require some sort of user interaction or expert knowledge.

There is a need for a system to combine some of these efforts into a larger system that works with information already available such as the website graph / information architecture. The proposed system will introduce several modules that can be used separately to perform automated usability testing, but combined in a larger system will give a better view of usability issues with a website. Harty stated that automated usability testing could uncover many types of issues if combinations of several techniques are used (Harty, 2011). The proposed method is not supposed to be used as the only measure of usability of a website, but should be able to uncover usability issues related to the organization of the pages (information architecture), and can be used in combination with other types of usability testing techniques.

The system works by crawling a website to generate a website graph where the pages are nodes and links are edges. This graph can then be augmented with information about the actual usage collected from web server logs. Using the website graph features such as number of nodes, number of edges and how connected each node a machine learning model is first trained and later used to predict the usability score of a website.

The machine-learning model will also be able to identify which features are more important in relation to usability so the website owner can focus on those in potential redesign efforts. The last part of the system provides the website owner with a tool to inspect specific paths from one page to another and identify issues with the path. The links on those pages can then be rearranged to optimize the path a user takes through the website. Two extremes in how websites are organized are shallow (Figure 1) and deep (Figure 2). A shallow graph would give the user a shorter path to the destination, but more choices at each page. A deep graph provides fewer choices at each page, but results in a longer path. The system will identify the different graph types and usability issues that might be present in the website.
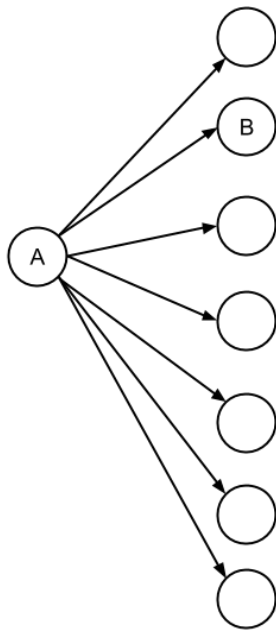


Figure 1. Example of shallow graph
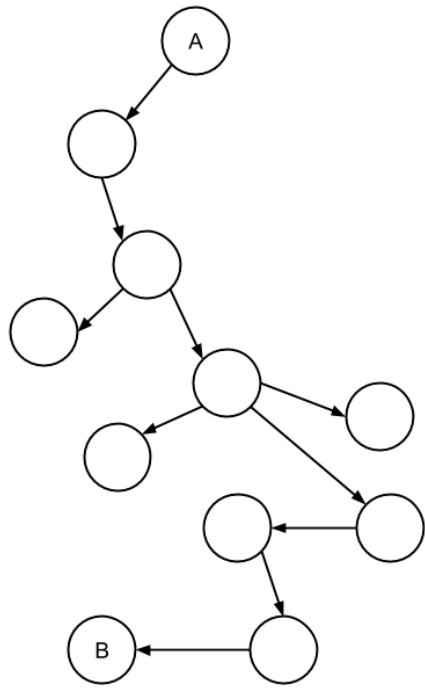
Figure 2. Example of deep graph

# CHAPTER II
# RELATED WORKS

## Usability testing

There are multiple definitions for usability, but Seffah, Donyaee, Kline and Padda attempted to consolidate this into a single model called Quality in Use Integrated Measurement (QUIM) which includes 10 factors: efficiency, effectiveness, productivity, satisfaction, learnability, safety, trustfulness, accessibility, universality and usefulness (Seffah, Donyaee, Kline, & Padda, 2006) . The factors are further broken down into 26 criteria that can be measured. The method proposed in this paper will only focus on a subset of those 10 factors: efficiency, productivity, satisfaction and learnability.

There exist a wide variety of techniques to uncover usability issues. Each technique focusing on different areas, requiring different information and can be performed at different stages of software development. Nielsen presents four basic ways of evaluating user interfaces: automatically, empirically, formally and informally (Nielsen, 1994). He further goes on to discuss a number of usability inspection methods:

- Heuristic evaluation

- Cognitive walkthroughs

- Formal usability inspections

- Pluralistic walkthroughs

- Feature inspection

- Consistency inspection

- Standards inspection

A single evaluator at a time is required while performing some of the methods; while others like pluralistic walkthroughs require multiple evaluators.

Evaluating a user interface can be done using methods such as heuristic evaluation or usability testing in a laboratory with actual users. In studies by Jeffries and Desurvire they find that it is not an either or scenario (Jeffries & Desurvire, 1992). Heuristic evaluation is performed by experts and could identify a lot of issues early on and would not require actual users to be involved. On the other hand the usability tests conducted with users in a laboratory was able to uncover a lot of the same issues as the heuristic evaluation in addition to others. This technique would also be able to uncover the issues real users would have with the system, reducing the time spent fixing issues that would not have a real impact on the overall usability of the system. Because the different techniques have different strengths they suggest that a mix of several techniques should be used when evaluating the usability of a system to uncover a wide variety of issues.

Scholtz discussed the advantages and disadvantages of the different approaches to usability evaluation and the phases of usability engineering (requirements analysis, design/testing/development, and installation) (Scholtz, 2004). The main benefit of a model based evaluation approach is that once the model has been defined it can be used repeatedly without much extra cost. One disadvantage of the model base approach is that it can be difficult to define the model initially. The process of defining the model is very time consuming as well. User-centered evaluations on the other hand are beneficial

6

because actual users are involved and the results can often uncover specific usability issues of a system that model based evaluation might overlook. One of the disadvantages of user-centered evaluations is that is very time consuming and expensive to administer the test. Another approach is expert-based evaluation and is usually less time-consuming and less expensive than the user-based approach. This method is possibly not as accurate as there are typically a few individuals reviewing the site and this might be very time consuming for larger systems. Research has found that $10 \pm 2$ is the optimal number of test users needed to discover 80% of usability issues (Hwang & Salvendy, 2010).

The differences between automated and classic usability evaluation techniques are many. Automated usability evaluation could reach a larger number of subjects with a larger geographic demographic compared to classic usability evaluation. The automated approach focuses on breath compared to depth (Rukshan & Baravalle, 2012).

Another issue issue with user-based usability evaluation is that the presence of an observer in a laboratory affects the subject and their emotion, performance and physiological measures (Sonderegger & Sauer, 2009). Sonderegger and Sauer also show that the set-up of the laboratory could affect responses from subjects.

**Evaluating link structure**

Evaluating the link structure of a website could be done using information about user behavior (Zhou & Chen, 2002). Zhou and Chen's approach starts with first defining a link structure model. This model is represented as a weighted directed graph where the nodes represent the web pages and the hyperlinks represent the edges. The data from web logs is used to determine user behavior and calculate the edge weights. The model could then be used to calculate the website complexity using Association Degree and

Convenience Degree of page pairs. Kung, Liu and Hsia have a similar approach as well where they create a model from the link structure. Instead of a directed graph they create a finite state machine (FSM), which they call a Page Navigation Diagram (Kung, Liu, & Hsia, 2000) .

The breath and depth of menu design influence task complexity (JACKO & SALVENDY, 1996) . They did a study on the following menu structures: $2^2$, $2^3$, $2^6$, $8^2$, $8^3$ and $8^6$ ($X^Y$ where X is the number of choices at each level and Y is the number of levels). Increased menu depth resulted in perceived complexity of a task. Campbell discussed 4 characteristics that describe complexity: multiple paths, multiple outcomes, conflicting interdependence among paths and uncertain/probabilistic linkages (Campbell, 1988).

## Machine learning in usability testing

The work of Oztekin, Delen, Turkyilmaz and Zaim where they compare four different models (multiple linear regression, decision trees, neural networks, and support vector machines) and how they performed on the data collected is a great example of how machine learning techniques have been utilized in usability testing (Oztekin, Delen, Turkyilmaz, & Zaim, 2013) . A checklist called UseLearn was used in collecting data focusing on factors such as error prevention, visibility, flexibility, accessibility, etc. The data was collected from the users of an online cell biology course. They trained and analyzed the results from the different models using 10-fold cross-validation. They showed that a multi-layer perceptron neural network performed better than the other methods with their data. The ability to identify the important features that had the biggest impact on the overall usability score was also discussed.

**Web crawling**

Web pages connected with hyperlinks make up a website. A wide variety of technologies/languages such as HTML (Hypertext Markup Language) for structuring the document, CSS (Cascading Style Sheets) for the look and feel and some times JavaScript for frontend logic are used in making the website. On the backend a programming/scripting language is commonly used for generating the dynamic content that is saved on file or in a database. A user can navigate from one web page to another using hyperlinks that connect the pages. A crawler would be able to navigate the website in a similar manner by identifying the links on each page and following them. There are many issues with crawlers and depending on the implementation and parameters used the crawler might behave very differently (Cothey, 2004). The ability to crawl an entire website depends on a number of factors like: forms (where the input given might lead to different pages), client-side validation and server-side manipulation as discussed by Marchetto,Tiella, Tonella, Alshahwan and Harman  (Marchetto, Tiella, Tonella, Alshahwan, & Harman, 2011) . Liu, Janssen and Milios worked on creating a smarter crawler based on user data and Hidden Markov Models (HMM). This method would give the ability to crawl the most relevant pages first (Liu, Janssen, & Milios, 2006) .

**Web log mining**

Web servers often record every request made to the server in a log file. The log file can then be parsed for usage patterns to identify how a user moved through a website.

Srivastava et al. provides a general survey of Web Usage mining and the techniques, methods, challenges and benefits of different web usage mining approaches.

9

They claim web mining can be divided into three different categories: structure mining, usage mining and content mining (Srivastava, Cooley, Deshpande, & Tan, 2000) . They also discuss different approaches to extract the data from server side web access logs and some of the challenges like a multiple user can access the web server from the same IP address, etc. In addition they provided a list of projects in the web usage-mining field, which showed that a majority of them used an access log captured on the server side compared to proxy or client side.

Joshi and Krishnapuram demonstrated a method of clustering user sessions based on pair-wise dissimilarities, which was done using a fuzzy clustering algorithms that outperforms association rule based approaches (Joshi & Krishnapuram, 2000) . They also discuss how to extract individual user sessions from the access logs by using the IP address and time of request in addition to filtering out unwanted entries such as errors, other request methods than "GET" and resources such as images. Work has also been done to dynamically improve the hypertext structure based on the data provided by usage mining (Masseglia, Poncelet, & Teisseire, 1999) .

Catledge and Pitkow found in their three-week study that the mean between each user interface event was 9.3 minutes and that events occurring over 25.5 minutes apart would be considered separate sessions (Catledge & Pitkow, 1995) .

# CHAPTER III
# METHODOLOGY

## System overview

Previous work shows that to get a clear picture of the usability of a website or system in general multiple techniques should be utilized  (Jeffries & Desurvire, 1992; Nielsen, 1994) . This is why a system with multiple modules where each part deals with different data and produce different reports can be more powerful when put together.

The system presented is composed of multiple modules that on their own could be used to produce results, but as a whole it can generate more detailed reports. Figure 3 shows how the modules are connected and what type of data is required for each module. The "Get structure" module is a web crawler that takes a website and generates a directed graph with nodes representing web pages and edges representing the links connecting the separate web pages. Usage processing takes web logs collected by a web server and identifies how users are navigating though the website graph. The Path analyzer can be used to analyze specific paths in a website graph using the output from both of the aforementioned modules. The artificial intelligence (AI) module takes the structure to extracts features, which can be used with training data to train a machine-learning model. The model can then be used to predict the overall usability score of the website in addition to identify which features are more important and should be focused on in a

11

possible restructuring of a website. All the modules will be described in more detail in the following sections.
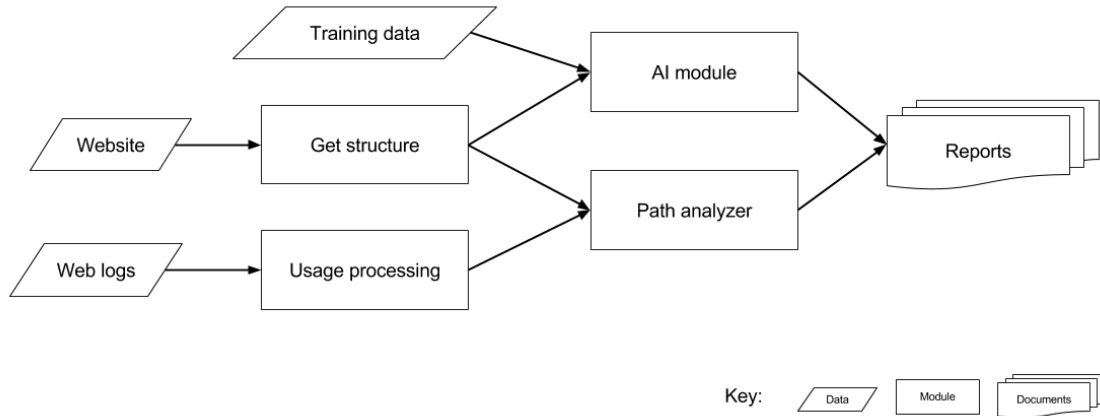


Figure 3. System overview

## Get structure of website

How the pages are linked together and organized can be referred to as the website graph or information architecture and is what will be used to determine the level of usability of a website. Manually creating this graph is time consuming and some times close to impossible with a very large website. Therefore an automated approach would be preferred to cut down the time it would take to create a website graph.

The simple crawler that was created takes a domain name like http://python.org to keep the crawling contained within the domain and a start URL like http://python.org/about/ to indicate where to start the crawling. It is preferred to crawl the entire website, but in very large sites this can be very time consuming. In a website graph with N number of nodes where every page is linking to every other page and to itself there will be $N^2$ number of edges.

A simple Breadth-First Search approach was taken for simplicity. A Breadth-First approach works by exploring all the nodes connected to the current node first before moving on to the next level. A Depth-First approach is different where it explores as deep as possible before moving back up and repeating the approach. Some work suggest that using a Breadth-First approach will discover the more important pages quicker because they are not likely to be hidden deep inside the structure (Najork & Wiener, 2001) . Time to crawl the entire website will depend on bandwidth and number of pages and hyperlinks. The results are stored in a database to allow the next steps to work with the data without having to wait for the crawler to run on the website every time.

As mentioned earlier there are many factors that influence a crawlers ability to successfully parse the entire website (Marchetto et al., 2011). If there are pages in the overall website graph that are not linked together with the rest of the graph, this implementation of a crawler would no be able to reach those pages. A user navigating though a website using the hyperlinks would not be able to reach those pages either, so this would indicate a usability issue the website owner would have to correct if they wanted users to access those pages.

Algorithm 1: Crawling algorithm
Input: website domain, start url
Output: website graph $G$
Steps:

1. Add start url to queue $Q$

2. Pop url from $Q$

3. Add a node *N* to the website graph *G*

4. Get all links from webpage pointed to by current url and add them to *Q* and the website graph *G*

5. Repeat steps 2-4 until *Q* is empty

The website graph is created from the data gathered during crawling with first adding all the pages as nodes N. Then adding edges E to represent the link from node *i* to node *j*. Algorithm showing the simple crawling method and graph generation is shown in Algorithm 1. An example representation of a graph created using this technique in Figure 4. Using a similar technique as described by Zhou and Chen (Zhou & Chen, 2002) weights could be added to the edges for further analysis, but that is the responsibility of the usage-processing module.
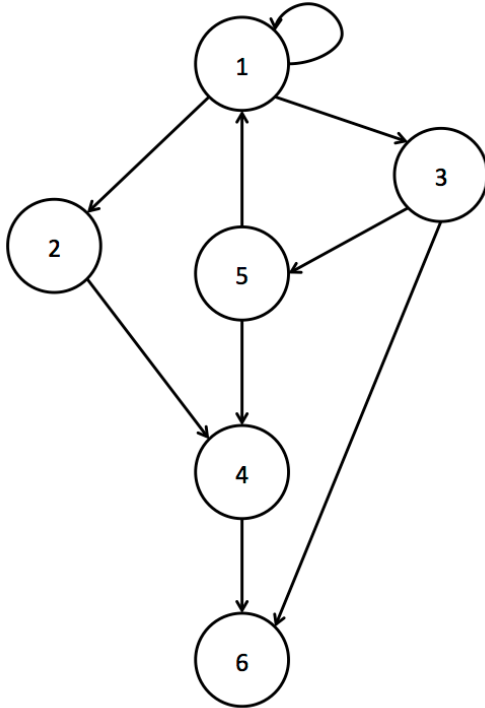


Figure 4. Example graph created by the module

The directed graph is defined as follows:

$$G=(N,E)$$

$$N = \{N_i : i \in \{1,n\}\}$$

$$E = \{E_{i,j} : i,j \in \{1,n\}\}$$

This module could also be expanded or replaced by a new module where the user could define the graph manually instead of crawling an existing website. Creating the website graph manually would allow testing earlier in the development process of a website, even at the design stage. By identifying issues before the website is implemented can reduce the time it would have taken to restructure the system. A manual approach like this could be time consuming in systems with a large amount of pages or a system that is highly dynamic and the pages/links change constantly.

**Artificial Intelligence Module**

Developing heuristics to be used in usability testing can be a difficult task and often end up being based on the implementer's intuition and experience. The heuristics might be very useful in some cases, but some times they might not be accurate at all. What a normal user perceives as usable might be completely different than what an expert user who is defining the heuristics.

Expert systems, which work with rules usually defined by domain experts, can generate great results when used in many areas. Some research shows that expert systems and machine learning models generate pretty similar results when applied to a credit

score prediction (Ben-David & Frank, 2009) . The expert system is only as good as the rules defined by domain experts. Domain experts might not always be available and the expertise will vary from person to person. On the other hand, machine-learning models require a lot of existing data to be trained, which might not always be available. A machine-learning model will usually be able to generate more accurate predictions, as more data is made available.

Because of the issues with expert systems a machine learning method was implemented to evaluate the website graph features and give a usability score based on previous data. Normal users would rate a website as usable or not and with a large enough data set a model could be trained and successfully identify if a website is usable or not just based on the website graph features.

Some features were extracted from the website graph to be used by the machine learning models. The five features used in this system are as follows, but could be expanded in future studies:

1. Number of nodes (pages)
2. Number of edges (unique links between pages)
3. Average out degree
4. Average in degree
5. Graph radius (the minimum eccentricity of the graph)

$$numNodes = |N|$$

$$numEdges = |E|$$

$$avgOutDegree = \frac{1}{|N|} \sum_{n \in N} \text{outDegree}(n)$$

16

$$avgInDegree = \frac{1}{|N|} \sum_{n \in N} \text{inDegree}(n)$$

Those website graph features were chosen because of their relation to the

information architecture and navigation of a website. The number of nodes, number of

edges, average in/out degree say something about how connected and complex a website

is, and the graph radius is related to the maximum distance from the start page to any

other page in the website.

Training the machine-learning model is a challenging task and often requires a

very large data set to be able to predict the usability of a new website not already

categorized as good or bad. As a large dataset does not exist for this purpose a dataset

was generated to test the method. A program was successfully created to randomly

generate website graphs and extract features and a usability score. The website graphs

created fall into 2 different size categories (large and small) which refers to the number of

nodes/pages in the graph. Small graphs contain between 25-50 nodes, while large graphs

contain between 100-200 nodes. The two size categories are then divided up into two

groups based on their connectivity (high and low). A high connectivity means that the

nodes in the graph are highly connected and therefore providing more choices for paths

through the graph. Low connectivity represents websites with fewer links between the

nodes, which in turn constricts the user to certain paths through he graph. The scoring is

done using a simple 5-star system (0-5) where 5 stars means the website is very usable

and 0 stars mean poor usability. The score for the different categories of test graphs is in

Table 1. These numbers and scores where chosen after looking at a random selection of

websites, but could easily be changed if there was a need for this in future studies. The

beauty of the system is that it will adapt as more data is provided and produce more accurate results. The scores are based on previous experience and best practices to give a baseline for testing. Scores could be collected by asking users to rank the perceived usability of websites. This data would in turn reflect the actual preferences of the users and give more accurate results. As most of the machine learning models require a large amount of data to give reasonable predictions a random graph generator will be used while testing this module. Certain features were made more important to see if the model would be able to distinguish them from the rest of the random data. An example of this training data set is shown in Table 2.

Table 1. Website graph types and score

|  | High | Low |
|---|---|---|
| **Large** | 2 - 4 | 0 - 2 |
| **Small** | 4 - 5 | 3 - 4 |

Table 2. Example training data set

| Score(0-5) | Feature #1 | Feature #2 | Feature #3 | Feature #4 | Feature #5 |
|---|---|---|---|---|---|
| 4 | 545 | 293,803 | 20 | 65 | 2 |
| 2 | 167 | 9,863 | 92 | 50 | 4 |

A variety of different models from a python machine-learning library called scikit-learn (Pedregosa et al., 2011) were trained, tested and compared to identify the model that worked best with this data set. Using 10-fold cross validation the different

models were evaluated with a logloss scoring function with a variety of data-set sizes to identify the best model (Korvald, Kim, & Reza, ) . A linear regression model was the most consistent and performed well even with smaller data set sizes.

When the machine-learning model has been trained it is ready to be used to categorize new websites. The method consist of three steps to generate the reports for a website owner:

1. Get features of a website graph
2. Run the characteristics through the trained model and get score
3. Rank the features to determine which features of the website graph has the biggest impact on overall usability

The trained model takes the features and predicts the usability score (0-5). This will give the website owner a general idea if there is a need for major changes. If the result from the prediction indicated that the usability was bad, the trained model could be used to determine which features should be focused on in the redesign of the website. Spending a lot of time fixing issues that will not have a great impact on the overall usability of a website might not be viable in a situation with limited resources or time constraints. Therefore the redesign efforts should be focused on the most important factors that influence the usability. An example of ranking feature importance is shown in Figure 5.
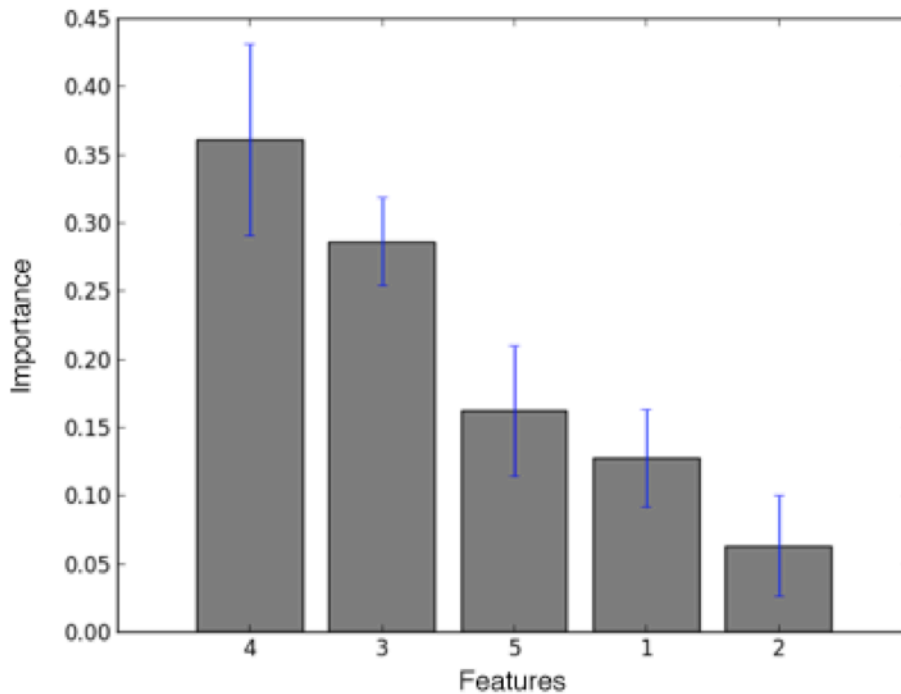
Figure 5. Example of features ranked by importance

## Usage processing

Insight into the usability of a website can be derived just by looking at the structure of the website, but what is even more interesting is how the website is actually being used by the users. Traditionally recording usage patterns can be done in a usability lab where the user is physically present with a human monitoring them and/or some sort of software recording every click made or eye movement. There are multiple problems with this and one of them is the observer effect, which means that the users are not acting the way they would in their normal environment (Sonderegger & Sauer, 2009) . When the users are not acting like they normally would, the data collected does not represent real-world use. A non-intrusive recording of the usage patterns would be preferred and this can be done by the web server on the backend.

20

A simple, yet powerful method would be to utilize the information already gathered by the web servers. Most web servers keep some sort of access log, which has the information on every request made. The raw access logs are stored differently based on web server configuration. Each line in the access log corresponds to a single resource requested by the user. When the user requests a web page it will often also have to request other resources such as images, style sheets, JavaScript, etc. that all make up the page. The log is therefore full of entries that do not mean anything in regards to figuring out usage patterns. Because of all the noise in the data set the unwanted entries will have to be filtered out. Another problem with the access logs is that some browsers will keep a local cache of some pages/resources to reduce load time. If a user is clicking the back button in a browser it could load that from the local cache and the server would not be able to record this.

A common log format (often referred to as NCSA Common log format) usually contain the following pieces of information:

- Remote IP address

- User identification if server requires user authentication.

- Time and date of request

- Request method (GET, POST, PUT, DELETE)

- Request URL

- HTTP version

- HTTP status code

- Content-length (size) of the document transferred

21

Example entry in a log:

127.0.0.1 user-id [5/Oct/2000:13:55:36-0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

Algorithm 2. Extract user sessions from web access:

1) Read file

2) Filter entries

   a) Extract the parts needed using regular expressions

   b) Only POST and GET requests are kept

   c) Everything with HTTP code over 300 is discarded

   d) Only web pages are kept

   e) Date is parsed to a UNIX timestamp

3) Records are organized by IP address

4) For each IP address

   a) Sort by time

   b) If time between two requests are over X seconds, it is recorded as a new
      transaction.

   c) For each user session

      i. Update the global navigation list with navigation from page X to
         page Y

5) Output global navigation list to screen or file

The program defined in Algorithm 2 will read in the content of a file and put each
line in a list. The list now contains all the entries from the original access log. Because

22

we are only interested in a subset of the entries in the log it will be filtered on multiple fields. Each line will be parsed using a regular expression to extract each field according to the common log format described earlier.

The first filter will remove all the requests that are not of the method POST or GET. This is to ensure we only get the requests made by the user while navigating the web page. PUT and DELETE are normally used with RESTful services, which has nothing to do with the user navigating through a website.

The second filter will remove all requests with an HTTP status code over 300 as they are related to requests where something went wrong. The HTTP status codes are 1xx (informational), 2xx (success), 3xx (redirection), 4xx (client error), 5xx (server error).

The last filter is more complex as it tries to filter out any resources that are not pure HTML pages based on the URL in the request. This is done by checking for known file extensions for web pages (.html, .php, .aspx, etc.) discarding images, videos, style sheets, JavaScript or other files needed to render a single web page.

After unwanted requests are filtered out the time stamp is converted to a Unix timestamp for easier sorting later and added to the list of filtered requests for further processing.

The filtered requests are then stored in a hash table where the IP address is the key and the value is a list of all requests associated with that IP address. For the sake of simplicity we are assuming that only one user will be accessing the website from same IP address at one time. This is not always the case and could possibly result in two or more user sessions being combined into one. This is not that common and the issue will therefore be ignored in the current implementation.

For each IP address the requests are sorted by time from oldest to newest request. This will yield a list of requests in the order the user accessed them and is crucial to identifying the path he/she took through the website.

The next step is to break those requests up into user sessions. A user session is defined as a list of requests made by that user in one sitting. If the time between two requests is over a certain threshold (30 min in this case) it is considered as a new user session. The sessions are then used to determine how many times a user navigated from page X to page Y. If the user session only includes one entry it will be discarded because the user never navigated between two pages.

If we have a user session with pages A, B, C, D this would result in 3 entries in the global navigation list where A → B, B→C and C→D will be counted. If we have another user session with A, B, C, E we get A→B, B→C and C→E.

The resulting global navigation list is then printed to the screen or saved to a file. The output from this algorithm can then be used to calculate the probability of a user at page X navigate to a page Y. An example of a global navigation list is provided in Table 3.

Table 3. Example of a global navigation list

| Path | Number |
|------|--------|
| A → B | 2 |
| B → C | 2 |
| C → D | 1 |
| C → E | 1 |

Parallelizing it using Message Passing Interface (MPI) so it can run on multiple

nodes increased the performance of this method  (Gropp, Lusk, & Skjellum, 1999) . The

external library mpi4py was used to utilize MPI with Python. It tries to mimic the

functions in the C++ MPI implementation  (Dalcín, Paz, & Storti, 2005; Dalcín, Paz,

Storti, & D'Elía, 2008) . There is some coordination that needs to be done between nodes

because user sessions cannot be broken up over several nodes in the current

implementation. The master node will read in the file and divide the data in equal chunks

that are sent to the other nodes for processing. The filtering and timestamp conversion is

done on separate nodes until the results are sent back to the master node. The master node

assembles the requests and organizes them by IP address. The data is then divided and

sent to the other nodes again with keeping requests from one IP together to prevent user

sessions breaking up. User sessions are then calculated in parallel before sending it all

back to the master node for reassembly and output to screen/file.

### Path analyzer

Website owners will often times need to know how well a campaign is

performing or how the users are navigating through the website to get to certain pages.

To do this a tool to analyze specific paths through a website is needed. The tool would

identify the most likely path a user is to take from page A to page B. This is computed by

performing a simple shortest path algorithm like Dijkstra's algorithm  (Cormen,

Leiserson, Rivest, & Stein, 2001) . If the specified path is not what the website owner

intended, this is a sign that something needs to be changed in order to modify the users

behavior. The website owner could then add a direct link between page A and B or

change other pages along the path to increase the likelihood of the user to choose the intended path.

When users are navigating through a website they go from page to page using the hyperlinks provided on each page. To identify usability issues related to the navigation you could look at specific paths from node A to node B.  If there is a website graph looking like the one presented in Figure 4 you might expect the user to navigate from node 1 to node 6 through node 3 like shown in Figure 6. While this looks like the shortest path from 1 to 6, this might not be the most likely path of a user. The user might be navigating in a roundabout way like illustrated in Figure 7. There are many different reasons why the user is navigating in this manner instead of the expected path. Links might not be obvious enough, the layout/design of the page might hide the link to node 3, the text of the links might not be descriptive enough and so on.

The path analyzer module will be able to identify those navigation path issues and show the website owner the most likely path of a user from one page to another.

The module is getting the most likely path of a user by calculating the shortest path through a website graph. This is improved upon using the probabilities gathered by the usage pattern module to give each edge in the graph a specific weight. Using methods such as Dijkstra's algorithm(Cormen et al., 2001) the module will be able to predict with higher accuracy how the users of a website are moving though the website graph. Using this information a website owner might want to improve upon the structure/design of the website to make certain paths more probable.
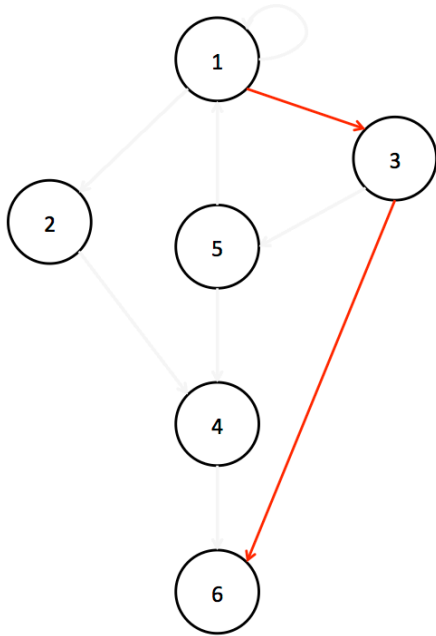
Figure 6. Expected path through a website graph
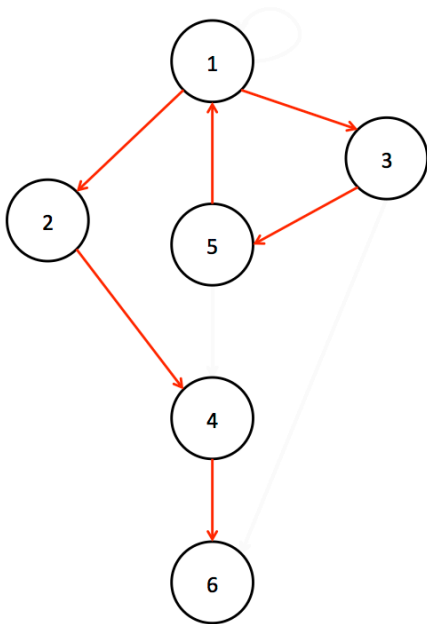


Figure 7. Actual path through website graph

## Combining the modules

The modules were combined in a system presenting the results to a tester on one screen. This was implemented as a simple website pulling the information from the different modules. A database was used to save the results from each module as much as possible to reduce the amount of information needing to be recalculated every time. The system has two main screens. In Figure 8 a screenshot of the main screen of the system with a list of all the websites tested with hyperlinks to more information and usability test results. The screenshot in Figure 9 show the most important part of the system, where it presents statistics from the website graph with a usability score generated by the machine-learning model. This screen also has the path analyzer where paths between specific pages can be shown. More screenshots from specific tests are shown in 0.
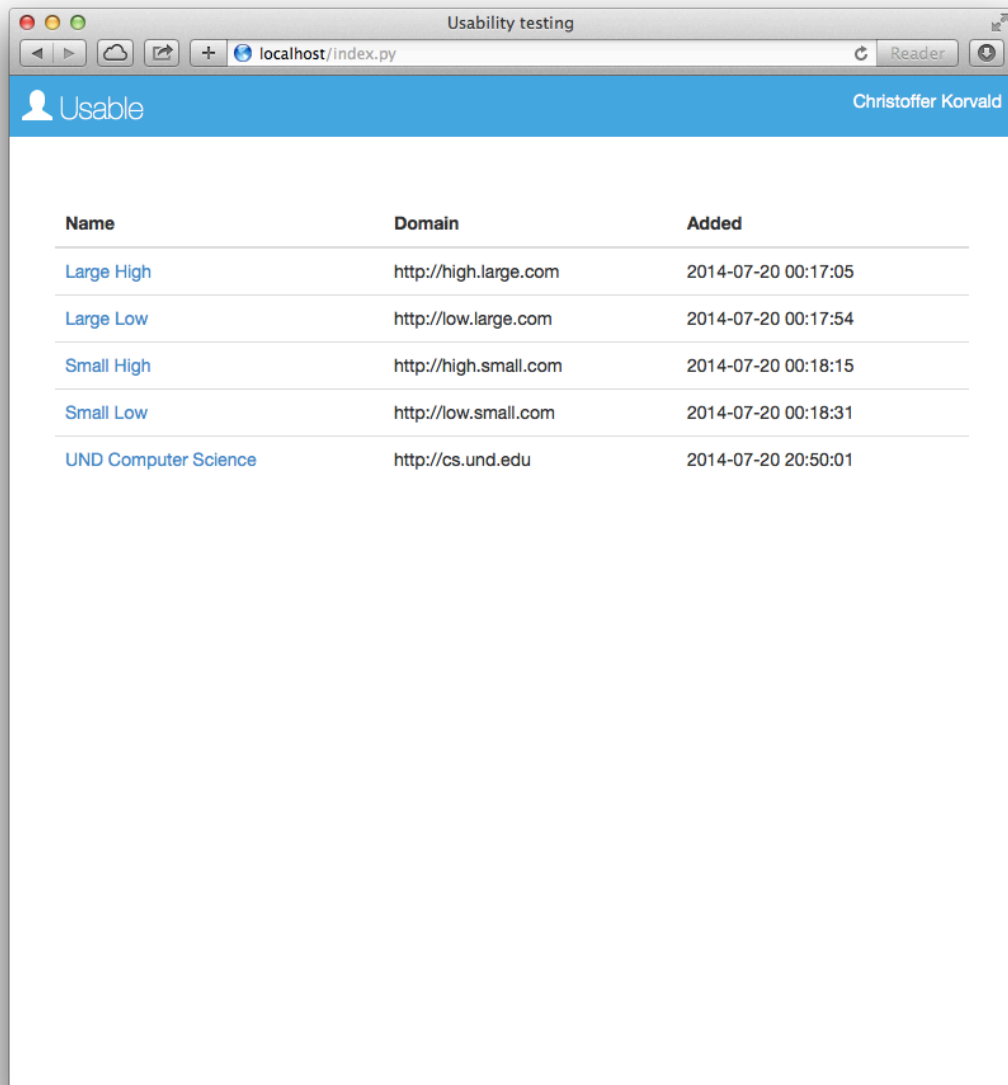
Figure 8. System screenshot (list of websites)
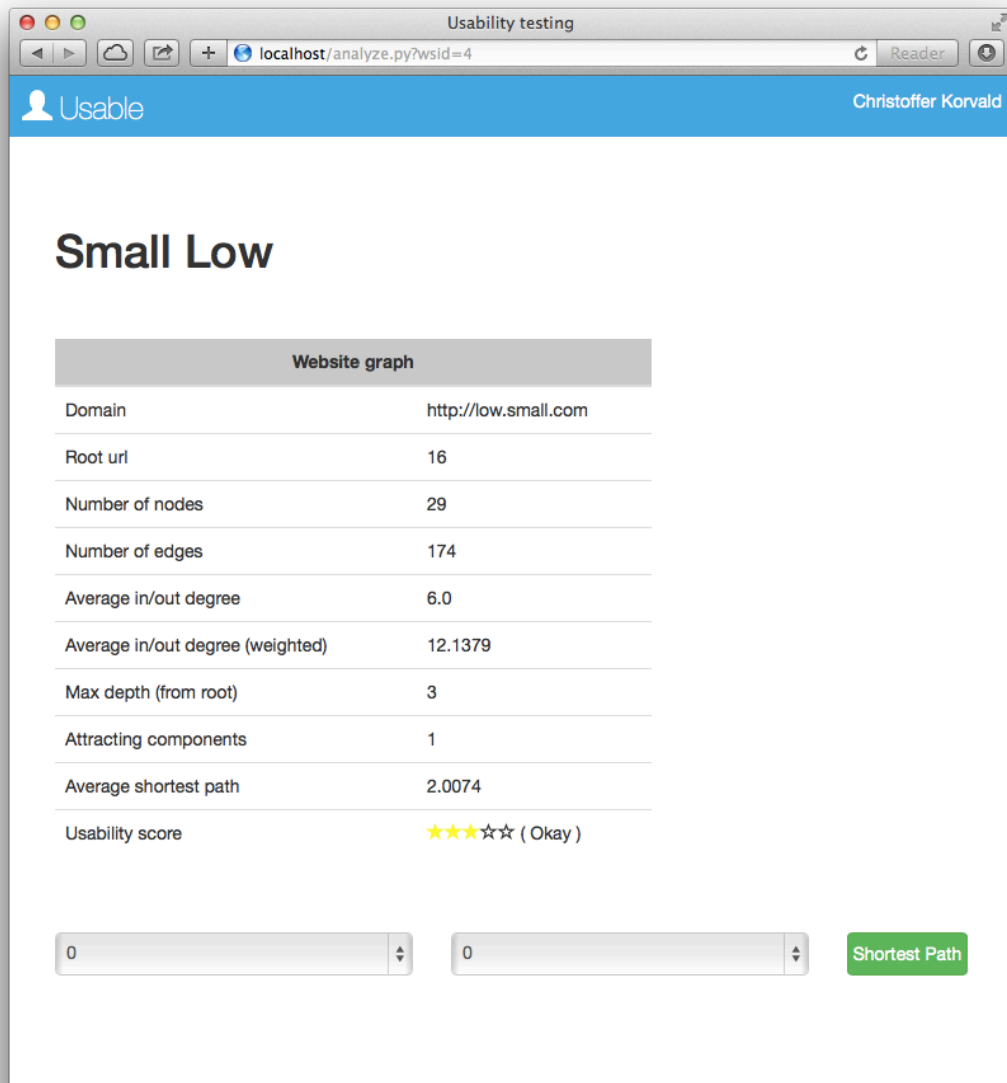
Figure 9. System screenshot (details of a website)

# CHAPTER IV
# RESULTS

**Get structure of website**

To test the crawler module a smaller website was chosen for simplicity, but the steps shown could be performed on a larger website as well. The UND Computer Science website was a good candidate as it has a fairly low number of pages and a simple structure. It is the website for the Computer Science Department at the University of North Dakota and contain information for current and prospective students about different majors and program details. It also has some news, contact information for faculty and staff as well as other information related to the department or program of study. A screenshot of the front page is provided in Figure 10.
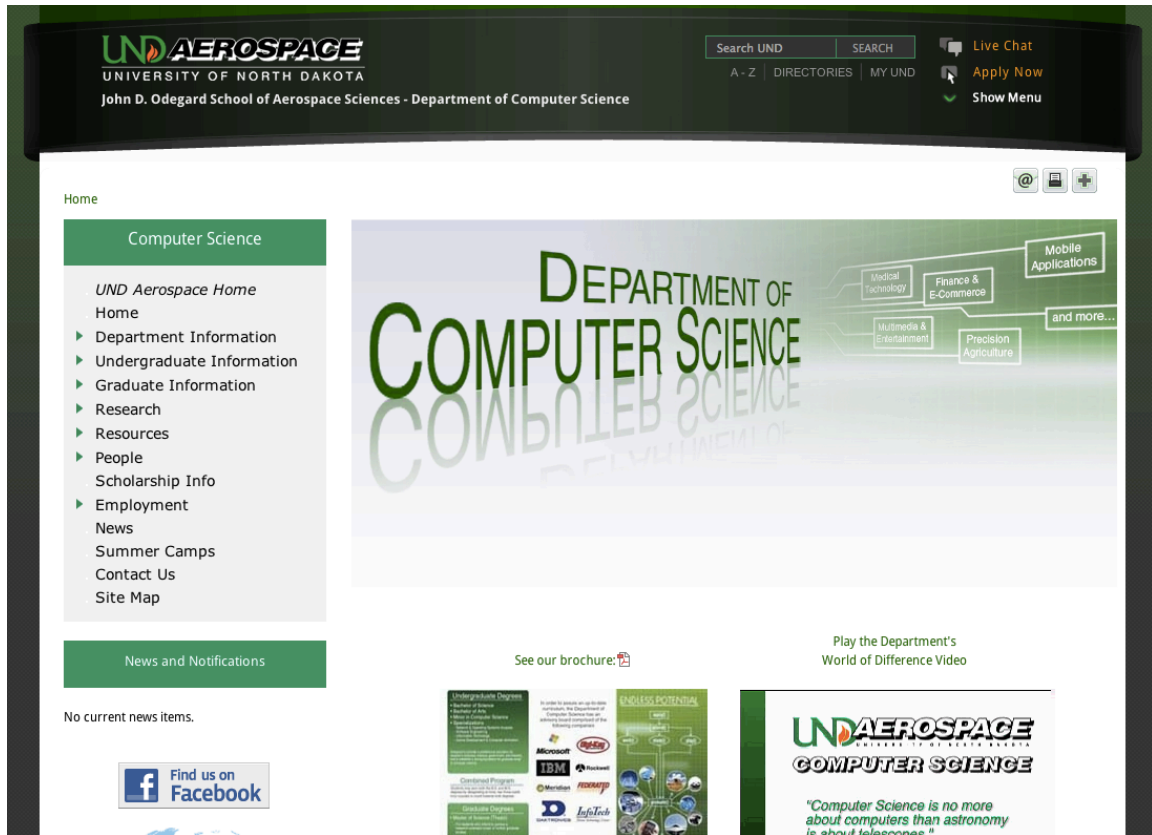
Figure 10. UND Computer Science Website

Crawling a website can be challenging because very few websites adheres correctly to all the standards. There were some issues with crawling the website and one of them was that links pointing to the same page were written differently on certain pages. Example of this is a link to Default.aspx that was also written as default.aspx, which would record these as separate pages while in reality they are the same. This is hard to fix as web servers treat URLs differently. Some use case-sensitive URLs, while others don't. The host name (domain) can be specified in both upper and lower case (Berners-Lee, Masinter, & McCahill, 1994) , but most of the time it's converted to lowercase, as it doesn't matter. Because some pages are saved more than once it might

not give completely accurate representation of the actual website, but since this is occurs rarely it will be ignored in this test.

Running the crawler identified 86 web pages with 2767 connections. Each connection has a value (or weight) associated with it that represents the number of links from one page to another. The maximum depth from the front page was 3, meaning it would take maximum of 3 clicks to navigate from the front page to any other page on the website provided the user took the shortest path. The website graph was successfully saved to the database and a visualization created and shown in Figure 11. Because of the number of nodes and edges it becomes very hard to give a clear visualization of the graph, but in smaller graphs it can be very helpful.
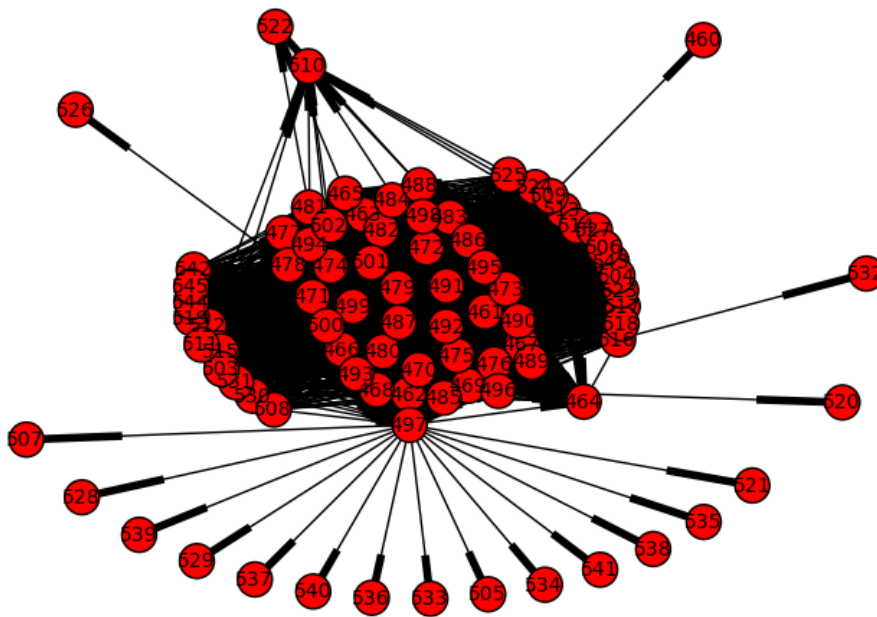


Figure 11. UND Computer Science Website Graph

## Artificial Intelligence

As described in **Error! Reference source not found.**, the machine-learning model has been trained with a large dataset that was randomly generated. The range of scores given to each type of website graph is shown in Table 1.

When running 4 types of randomly generated graphs (Large-High, Large-Low, Small-High, and Small-Low) through the model it provided the usability scores shown in Table 4. When comparing the 4 test graphs with the ranges used in the training data it is clear that they all fall within the correct ranges. Therefore showing that the model could successfully predict the score of a website graph based on previous training data. As expected the small graph with high connectivity got a high score of 5, while a large graph with low connectivity got a low score. Running the UND Computer Science website through the machine-learning model gives a score of 2 because it falls into the large size category and is not highly connected.

Table 4. Machine learning prediction results

| Size | Connectivity | Usability score |
|------|-------------|-----------------|
| Large | High | 4 |
| Large | Low | 1 |
| Small | High | 5 |
| Small | Low | 3 |

The machine-learning model can also be utilized to indicate which of the website graph features are most important when it comes to the usability score. Running the randomly generated training data from before it gave the following ranking:

1. Feature 1: Number of nodes (pages)

2. Feature 2: Number of edges (unique links between pages)

3. Feature 4: Average in degree

4. Feature 3: Average out degree

5. Feature 5: Graph radius (the minimum eccentricity of the graph)

This can also be seen in Figure 12 with the standard deviation shown with blue lines on each bar. The value corresponds to the percentage of importance it has in determining the usability score. It shows that the two most important factors with this test data is the number of nodes and number of edges. This corresponds to the results shown in Table 4 where a low number of nodes and high number of edges (high connectivity) gave a better score than high number of nodes and low number of edges (low connectivity). To improve the usability score of a website, the main focus should be to increase the connectivity as this will have the biggest impact.
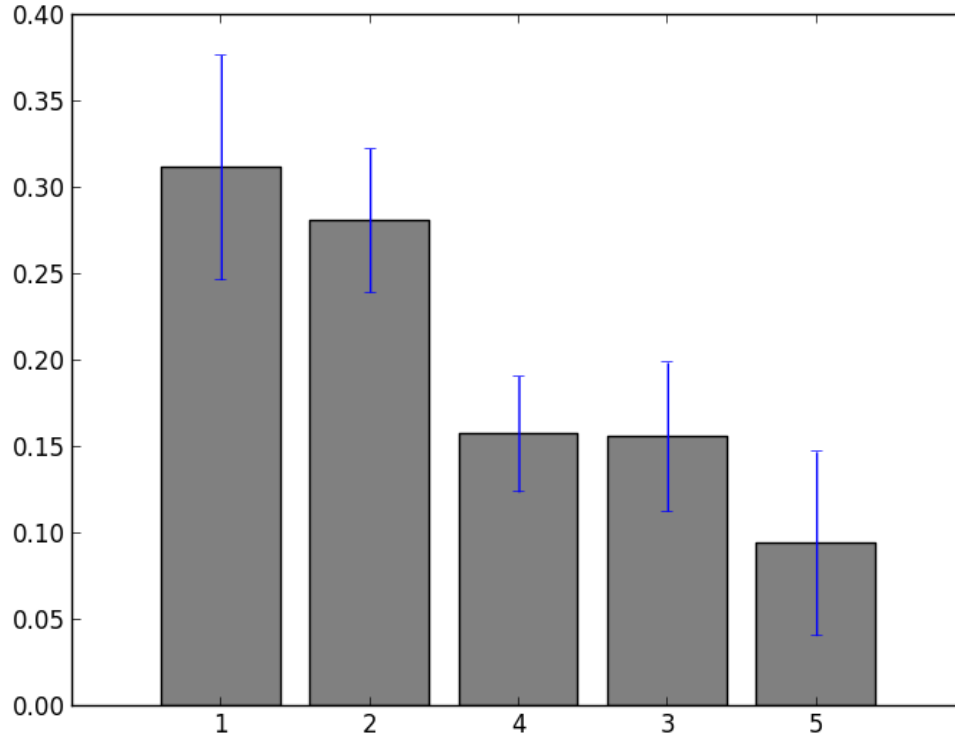
Figure 12. Important Feature Ranking

## Usage

To test this module a website with a decent amount of traffic was chosen. The data set used to test the implementation of the usage module was a raw access logs from a Norwegian website with about 50,000 unique users per month. The file was 653 MB containing 2,371,220 unique entries for the month of April 2014. If you collected data for a year with the same amount of traffic the file would contain over 28 million entries with a size of 7.6 GB.

Running the program on the April 2014 data set gave the results shown in Table 5. The results show that web access logs contain a lot of entries that are not necessary to determine a user session. From the 2,371,220 entries only 45,564 were kept for further

processing. That comes out to 1.9 percent of usable data. The results also show that from

5,222 unique IP addresses the program was able to extract 8,740 separate user sessions,

with 13,904 entries in the global navigation list.

Table 5. Usage module output

| Lines read | 2,371,220 |
|---|---|
| Lines in list | 45,564 |
| Lines skipped | 2,325,656 |
| IP addresses | 5,222 |
| User sessions | 8,740 |
| User nav | 13,904 |

## Path Analyzer

An example of testing a path would be from the start page to a news article titled

"Emerson Process to visit UND for student recruitment". The shortest path required the

user to first navigate to the section named "News" with a total of 2 clicks to arrive at the

desired page. This does not seem bad, as it was logically located on the news page. If this

was a high profile news article the website owner wanted to promote the path could be

shortened by providing a link on the home page, which would reduce the shortest path by

1 click. Another example is from the start page to the page with information about the

Masters program. This path only required 1 click (shown in Figure 13), which is very

good.

The path analyzer successfully identified the shortest path from a start node to an

end node. It is up to the website owner to interpret this information and decide if this is

the intended path or if it needs to be modified. As with the example of the news article it could easily be improved by adding a link on the start page to the news article in question.
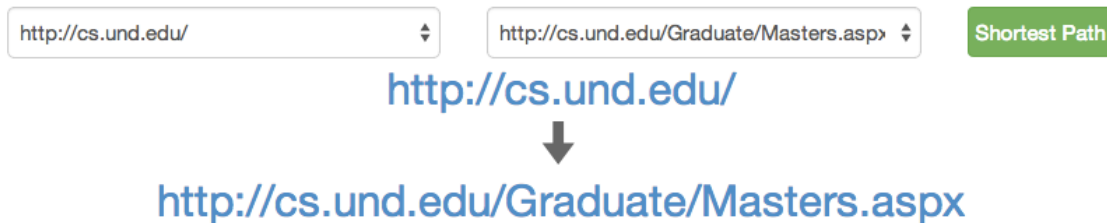


Figure 13. Path analyzer example

**Putting them all together**

Each module provides a different view of the usability of a website and can be used by itself. When the results from all the modules are combined it gives a better view of the total usability. One of the modules might give a high score, but when looked at in a larger context it might not be good enough. The full analysis view of the system is shown in Figure 14 and includes information from all the different modules in one screen. The path analyzer can be used from this screen as well. The results from all the generated test graphs are shown in Table 6.

The system successfully generated the data in this view to give a website owner the information needed to identify usability issues related to user navigation and information architecture.

Figure 14. Screenshot of website analysis

Table 6. Results from generated graphs

| Size | Connectivity | Nodes | Edges | Avg. in/out degree | Max depth | Avg. shortest path | Usability score |
|------|--------------|-------|-------|--------------------|-----------|--------------------|-----------------|
| Large | High | 169 | 15210 | 90 | 2 | 1.4643 | 4 |
| Large | Low | 47 | 1810 | 10 | 4 | 2.5041 | 1 |
| Small | High | 27 | 702 | 26 | 1 | 1.0 | 5 |
| Small | Low | 29 | 174 | 6 | 3 | 2.0074 | 3 |

# CHAPTER V
# CONCLUSION

Testing usability will continue to be a very challenging task because of all the different aspects that impact usability. Specialized tools and methods has been shown in previous work to give good results on specific areas, but there is not a general method or tool for testing everything.

A system utilizing multiple modules with very specific tasks to give a better view of usability issues with a website has been implemented and demonstrated. The crawler successfully automatically navigate a website and create a website graph representing the structure of how the pages of a website are connected. The website graph could also be manually created allowing for a website structure to be tested before implementation. To improve the information in the website graph a module was created to incorporate data related to actual use of a website. Parsing web access logs saved by the web server identified paths traversed by real users. The information on navigation patterns resulted in a better guess on most likely path a user would take from one page to another. The features extracted from the website graph like number of nodes, number of edges, in/out degree and radius was the basis of for training the machine-learning model and identifying issues in other website. The machine-learning model successfully identified usability issues when trained and tested with data automatically generated. The model could be trained with real world data at a later point if enough data could be gathered and

would then give better results when testing real websites. The results from all of the modules were combined in a system that presented the information to the tester, which could be used to improve the website.

While the modules are useful when used in isolation, the real potential of the system is when the modules are combined to give a better view of issues. The system mostly focuses on issues related to user navigation and organization of pages (information architecture). Navigation and organization on websites is very important as they both relate to how easily a user can access the information or perform certain actions. If a user is not able to perform the desired actions it might result in lost revenue for the website owner. The system presented is able to identify specific issues with navigation from one page to another, giving the website owner the most likely path a user would take. The data collected by each module and the combined results could have many other use cases, but the system presented has primarily focused on usability related to navigation and information architecture.

The machine-learning module could be improved in future work by changing the features used for training and prediction. The module could probably yield more accurate results with real world data as well. The data gathering would be very time intensive, as most machine learning models require a large amount of data to give accurate results. An efficient method to visualize paths through a large website graph is another area that could benefit from more research.

# APPENDICES

# Appendix A

## Crawler source code

```python
"""
Crawl a website and save it to the databse

"""
from bs4 import BeautifulSoup
import urllib2
import urlparse
import oursql
import sys
from OrderedSet import OrderedSet


def linkInDomain(link,domain):
    domain = urlparse.urlparse(domain)[1]
    linkDomain = urlparse.urlparse(link)[1]
    return domain == linkDomain

def withProtocol(url,protocols):
    #convert to list if not already
    if type(protocols) is str: protocols = [protocols]

    t = urlparse.urlparse(url)
    return t.scheme in protocols

#Remove the fragments at the end of urls
def cleanUrl(url,domain=None):
    if domain != None: aurl = urlparse.urljoin(domain,url)
    else: aurl = url

    t = urlparse.urlparse(aurl)
    s = t[0]+"://"+t[1]+t[2]
    if t[2] == '': s += '/'
    if t[3]: s += ";"+t[3]
    if t[4]: s += "?"+t[4]
    return s

def isWebPage(url):
    extensions = ('.asp','.aspx','.axd','.asx','.asmx','.ashx',
            '.cfm',
            '.yaws',
            '.html','.htm','.xhtml','.jhtml',
            '.jsp','.jspx','.wss','.do','.action',
            '.pl',
            '.php','.php4','.php3','.phtml',
            '.py',
            '.rb',
            '.cgi','.dll',
            '.adp','.r')
    t = urlparse.urlparse(url)
```

44

```python
        #If there is no extension, assume it's still a web page
        if "." not in t.path: return True
        if t.path.endswith('/') or t.path.endswith(extensions): return
True

        #More robust way to check the response (takes a little longer)
        try:
                r = urllib2.urlopen(url).info()
                if r != None and r['content-type'] == 'text/html': return
True
        except:
                return False

        return False



def getLinks(url,domain,crawledLinks=[],unCrawledLinks=[]):

        print "- Crawling",url
        try:
                html_doc = urllib2.urlopen(url).read()
        except:
                print "- Error: could not open",url
                return []

        soup = BeautifulSoup(html_doc)

        links = soup.select('a[href]')

        temp = []
        for l in links:
                href = cleanUrl(l['href'],domain)
                if href not in temp and \
                        href not in crawledLinks and \
                        href not in unCrawledLinks and \
                        linkInDomain(href,domain) and \
                        withProtocol(href,['http','https']) and \
                        isWebPage(href): temp.append(href)
        return temp

def crawlWithMaxLinks(startUrl,maxLinks,domain):

        struct = {}
        crawledLinks = OrderedSet()
        unCrawledLinks = OrderedSet([startUrl])
        """
        links = getLinks(startUrl,domain,crawledLinks,unCrawledLinks)
        for l in links:
                if l not in unCrawledLinks: unCrawledLinks.add(l)
        """
        duplicates = 0
        while len(crawledLinks) < maxLinks and len(unCrawledLinks) > 0:
                c = unCrawledLinks.pop(False)
                #print "Popping:",c
                struct[c] = {}
```

45

```python
            newLinks = getLinks(c,domain)
            for l in newLinks:
                    #Add to the struct
                    if l in struct[c]: struct[c][l] += 1
                    else: struct[c][l] = 1

                    #Add to the uncrawled list
                    if l in unCrawledLinks or l in crawledLinks or l ==
c: duplicates += 1
                    else: unCrawledLinks.add(l)

            crawledLinks.add(c)

    return struct,crawledLinks,unCrawledLinks,duplicates


def insertPage(url,wsid):
    with dbConn.cursor(oursql.DictCursor) as db2:
            db2.execute("SELECT pid FROM pages WHERE url = ? AND wsid =
?",(url,wsid))
            existingPid = db2.fetchone()
            if existingPid != None:
                    return int(existingPid['pid'])
            else:
                    db2.execute('INSERT INTO pages (url,wsid) VALUES
(?,?)',(url,wsid))
                    return db2.lastrowid


if __name__ == '__main__':
    """

    To run crawler use:
    python crawl.py <uid> "<websiteName>" <domain> <startUrl>
<maxLinks>

    """
    args = sys.argv

    if(len(sys.argv) < 6): #Not enough parameters
            print "To run the crawler use: \n","python crawl.py <uid>
\"<websiteName>\" <domain> <startUrl> <maxLinks> "
            sys.exit()

    uid = args[1]
    websiteName = args[2]
    domain = args[3]
    startUrl = args[4]
    maxLinks = int(args[5])


    print "uid:",uid
    print "websiteName:",websiteName
    print "domain:",domain
    print "startUrl:",startUrl
```
46

```python
        print "maxLinks:",maxLinks

        s,c,u,d = crawlWithMaxLinks(startUrl,maxLinks,domain)


        #Connect to the database
        dbConn = oursql.connect(host='127.0.0.1', user='root',
passwd='root',
            db='thesis')
        db = dbConn.cursor(oursql.DictCursor)

        db.execute('INSERT INTO websites (name,domain,uid,start_url)
VALUES (?,?,?,?)',(websiteName,domain,uid,startUrl))
        wsid = db.lastrowid
        print wsid
        print "Crawled: ",len(c)
        print "Not crawled: ",len(u)
        print "Duplicates:",d

        for k in s:
            print "##",k,"##"

            fromPid = insertPage(k,wsid)
            for k2 in s[k]:
                toPid = insertPage(k2,wsid)
                #print "- saving link from",fromPid,"to",toPid,"with
count",s[k][k2]
                db.execute('INSERT INTO links (fromPid,toPid,count)
VALUES (?,?,?)',(fromPid,toPid,s[k][k2]))

                print str(s[k][k2]),":",k2
        #print s


        #Close database
        dbConn.close()
```

# Appendix B

## System index source code

```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import sys
sys.path.append('../crawler')
sys.path.append('../graph')

# enable debugging
import cgitb
cgitb.enable()
import cgi
sys.stderr = sys.stdout


from graph2 import *
from crawl import *

import oursql

#get our config for the app
from config import config

#Connect to the database
dbConn = oursql.connect(host='127.0.0.1', user='root', passwd='root',
db='thesis')
db = dbConn.cursor(oursql.DictCursor)



db.execute("SELECT * FROM users WHERE uid = ?",(config['uid'],))
user = db.fetchone()


print "Content-Type: text/html;charset=utf-8"
print


print """<!DOCTYPE html>
<html>
  <head>
    <title>Usability testing</title>
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
  </head>
  <body>

  <header class="row">
      <div class="col-xs-6">
```

48

```
            <a href="/index.py">
                    <h1><span class="glyphicon glyphicon-user"></span>
""",config['siteName'],"""</h1>
            </a>
    </div>
    <div class="col-xs-6 text-right">
            <aside>
                    <span class="user_pic"><span>
                    <span class="user_name">""",user['name'],"""</span>
            </aside>
    </div>

  </header>

  <div class="container">

  <div class="row">

"""

db.execute("SELECT * FROM websites WHERE uid = ?",(user['uid'],))

websites = db.fetchall()

print """
<div class="col-lg-12">
<table class="table">
    <thead>
        <tr>
                <th>Name</th>
                <th>Domain</th>
    <th>Added</th>
        </tr>
    </thead>"""
for ws in websites:
    print "<tr>"
    print '<td><a
href="analyze.py?wsid='+str(ws['wsid'])+'">'+str(ws['name'])+"</a></td>
"
    print "<td>",ws['domain'],"</td>"
    print "<td>",ws['added'],"</td>"
    print "</tr>"

print """

  </div>


    </div> <!-- /row -->
    </div> <!-- /container -->

  <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
  <script src="https://code.jquery.com/jquery.js"></script>
  <!-- Include all compiled plugins (below), or include individual
files as needed -->
```
49

```
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>"""
```

# Appendix C

## Website analysis source code

```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import sys
import os
import os.path #For checking if image exists

import networkx as nx

import numpy as np
import sklearn
from sklearn.neighbors import KNeighborsClassifier

sys.path.append('../graph')


# enable debugging
import cgitb
cgitb.enable()
import cgi
sys.stderr = sys.stdout


import generateGraph

import oursql

#get our config for the app
from config import config




def
predictScore(numNodes,numEdges,averageOutDegree,averageInDegree,radius)
:
        predictionEngine = KNeighborsClassifier() #seems to be giving
consistent results :)

        dataset =
np.genfromtxt(open('../learning/temp/new_train.txt','r'),
delimiter=',',dtype="i")

        target = np.array([x[0] for x in dataset])
        train = np.array([x[1:] for x in dataset])

        probas = predictionEngine.fit(train,target)
        return
probas.predict([numNodes,numEdges,averageOutDegree,averageInDegree,radi
us])
```

51

```python
#Connect to the database
dbConn = oursql.connect(host='127.0.0.1', user='root', passwd='root',
db='thesis')
db = dbConn.cursor(oursql.DictCursor)

db.execute("SELECT * FROM users WHERE uid = ?",(config['uid'],))
user = db.fetchone()

print "Content-Type: text/html;charset=utf-8"
print


print """<!DOCTYPE html>
<html>
  <head>
    <title>Usability testing</title>
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
  </head>
  <body>

  <header class="row">
      <div class="col-xs-6">
            <a href="/index.py">
                  <h1><span class="glyphicon glyphicon-user"></span>
""",config['siteName'],"""</h1>
            </a>
      </div>
      <div class="col-xs-6 text-right">
            <aside>
                  <span class="user_pic"><span>
                  <span class="user_name">""",user['name'],"""</span>
            </aside>
      </div>

  </header>

  <div class="container">

"""

form = cgi.FieldStorage()
wsid = form.getvalue("wsid")
if wsid == None:
      print "No wsid was specified"
else:
      wsid = int(wsid)
      db.execute("SELECT * FROM websites WHERE uid = ? AND wsid =
?",(user['uid'],wsid))

      website = db.fetchall()[0]
```

```python
        print "<h1>",website['name'],"</h1>"

        g = generateGraph.fromDB(db,wsid)



        db.execute("SELECT pid FROM pages WHERE url LIKE ? AND wsid =
?",(website['start_url'],wsid))
        start_url_id = db.fetchone()['pid']

        bfs_tree = nx.bfs_tree(g,start_url_id)

        graphImage = "images/graphs/graph-"+str(wsid)+".png"

        numNodes = nx.number_of_nodes(g)
        numEdges = nx.number_of_edges(g)
        averageOutDegree =
(sum(g.out_degree().values()))/float(g.number_of_nodes()))
        averageInDegree =
(sum(g.in_degree().values()))/float(g.number_of_nodes()))
        averageWeightedOutDegree =
(sum(g.out_degree(weight='weight').values()))/float(g.number_of_nodes())
)
        averageWeightedInDegree =
(sum(g.in_degree(weight='weight').values()))/float(g.number_of_nodes()))
        radius = nx.radius(bfs_tree.to_undirected())
        numberAttractingComponenets = nx.number_attracting_components(g)
        averageShortestPath = nx.average_shortest_path_length(g)


        def stars(n,maxStars=5):
            o = '<span class="starRating">'
            for i in range(maxStars):
                if (i < n): o += '<span class="glyphicon glyphicon-
star"></span>'
                else: o += '<span class="glyphicon glyphicon-star-
empty"></span>'
            o += '</span>'
            return o

        predictionLabel = {0:'Extreamly bad', 1:'Really bad', 2:'Bad',
3:'Okay', 4: 'Good', 5: 'Really good'}

        prediction =
predictScore(numNodes,numEdges,averageOutDegree,averageInDegree,radius)


        print '''
        <br/><br/>
        <div class="row"><div class="col-sm-8">

        <table class="table website-stats">
            <thead>
                <tr>
                    <th colspan="2" class="text-center">Website
graph</th>
```
53

```
                                      </tr>
                        </thead>
                        <tbody>
                                <tr>
                                      <td>Domain</td>
                                      <td>''',website['domain'],'''</td>
                                </tr>
                                <tr>

                                      <td>Root url</td>
                                      <td>''',website['start_url'],'''</td>
                                </tr>
                                <tr>

                                      <td>Number of nodes</td>
                                      <td>''',numNodes,'''</td>
                                </tr>
                                <tr>

                                      <td>Number of edges</td>
                                      <td>''',numEdges,'''</td>
                                </tr>
                                <tr>

                                      <td>Average in/out degree</td>
                                      <td>''',round(averageInDegree,4),'''</td>
                                </tr>
                                <tr>

                                      <td>Average in/out degree (weighted)</td>

            <td>''',round(averageWeightedInDegree,4),'''</td>
                                </tr>
                                <tr>

                                      <td>Max depth (from root)</td>
                                      <td>''',radius,'''</td>
                                </tr>
                                <tr>

                                      <td>Attracting components</td>
                                      <td>''',numberAttractingComponenets,'''</td>
                                </tr>
                                <tr>
                                      <td>Average shortest path</td>
                                      <td>''', round(averageShortestPath,4) ,'''</td>
                                </tr>
                                <tr>
                                      <td>Usability score</td>
                                      <td>''',stars(prediction),'''
('''',predictionLabel[prediction[0]],''')</td>
                                </tr>
                        </tbody>
                </table>

                </div></div>

                <br/><br/>


                '''


                db.execute("SELECT * FROM pages WHERE wsid = ?",(wsid,))
```
54

```python
        pages = db.fetchall()

        shortestPathFrom = form.getvalue('shortestPathFrom')
        shortestPathTo = form.getvalue('shortestPathTo')
        if shortestPathTo != None and shortestPathFrom != None:
                shortestPathFrom = int(shortestPathFrom)
                shortestPathTo = int(shortestPathTo)

        print '<form method="post" role="form">'
        print '<div class="row">'

        print '<div class="col-sm-5">'
        print '<select name="shortestPathFrom" class="form-control">'
        for p in pages:
                print '<option value="'+str(p['pid'])+'"'
                if shortestPathFrom == p['pid']: print '
selected="selected"'
                print '>'+str(p['url'])+'</option>'
        print '</select>'
        print '</div>'

        print '<div class="col-sm-5">'
        print '<select name="shortestPathTo" class="form-control">'
        for p in pages:
                print '<option value="'+str(p['pid'])+'"'
                if shortestPathTo == p['pid']: print ' selected="selected"'
                print '>'+str(p['url'])+'</option>'
        print '</select>'
        print '</div>'

        print '<div class="col-sm-2">'
        print '<button class="btn btn-block btn-success"
type="submit">Shortest Path</button>'
        print '</div>'

        print '</div>'
        print '</form>'

        if shortestPathTo != None and shortestPathFrom != None:

                try:
                        sp =
nx.shortest_path(g,shortestPathFrom,shortestPathTo)

                        s = ''
                        for p in sp:
                                s += str(p)+","
                        s = s[:-1]

                        db.execute("SELECT * FROM pages WHERE pid IN ("+s+")
ORDER BY FIELD(pid,"+s+")")

                        pages = db.fetchall()
                        print '<div class="shortestPath">'
                        i = 1
                        prevPid = None
```
55

```python
                        totalProb = None
                        for p in pages:
                                print "<div>"
                                print '<div class="link"><a target="_blank"
href="'+str(p['url'])+'">',p['url'],"</a></div>"
                                if i < len(pages): print '<div class="arrow
glyphicon glyphicon-arrow-down"></div>'
                                i += 1
                                print '</div>'

                        print "</div>"


                        #print "Total probability of reaching this node:
",format(totalProb,'.5f'),"%"
                except nx.exception.NetworkXNoPath:
                        print '<br><div class="alert alert-danger">There is
no path between the pages selected</div>'


        print '<img src="'+graphImage+'">'

print """

        </div>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://code.jquery.com/jquery.js"></script>
    <!-- Include all compiled plugins (below), or include individual
files as needed -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>"""
```

# Appendix D

## Training data generation source code

```python
"""
Generate test data for the machine-learning model

"""
import networkx as nx
import numpy as np
import sys

import matplotlib.pyplot as plt

import random

from networkx import expected_degree_graph

import oursql

def fromDB(db,wsid):
  g = nx.DiGraph()
  db.execute('SELECT * FROM pages WHERE wsid = ?',(wsid,))
  for p in db.fetchall():
    g.add_node(int(p['pid']))

    db.execute('SELECT fromPid,toPid,count FROM links WHERE fromPid =
?',(int(p['pid']),))
    r = db.fetchall()
    if r != None:
      for l in r:
        #w = float(random.randint(0,100))/100.0
        w = int(l['count'])
        g.add_edge(int(l['fromPid']),int(l['toPid']),weight=w)
  return g

def small(maxN=50,maxWeight=3,connectivity='high'):
  return
randomGraph(np.random.random_integers(maxN/2,maxN),maxWeight,connectivi
ty)

def large(maxN=200,maxWeight=5,connectivity='high'):
  return
randomGraph(np.random.random_integers(maxN/2,maxN),maxWeight,connectivi
ty)


def randomGraph(n=50,maxWeight=5,connectivity='high'):
  g = nx.complete_graph(n,create_using=nx.DiGraph());

  #remove some random edges so the graph is not complete
  edges = g.edges()
  numEdges = len(edges)
```

57

```python
    if(connectivity == 'high'): numEdgesToRemove = numEdges -
g.number_of_nodes() * np.random.random_integers(50,200)
    elif(connectivity == 'low'): numEdgesToRemove = numEdges -
g.number_of_nodes() *  np.random.random_integers(2,10)

    random.shuffle(edges)
    g.remove_edges_from(edges[:numEdgesToRemove])

    #Add random weights to the edges
    for (f,t) in g.edges():
      g[f][t]['weight'] = np.random.random_integers(1,maxWeight)
    return g



if __name__ == '__main__':
  """

  To run generateGraph.py use:
  python generateGraph.py [<numGraphs> [<maxNodes> [<maxWeight>]]]

  """
  args = sys.argv

  if(len(sys.argv) < 2): #Not enough parameters
    print "To run the Graph generator use: \n"," python
generateGraph.py <numGraphs>"
    sys.exit()

  numGraphs = int(args[1])

  data = []
  goodBad = None
  for i in range(numGraphs):
    choice = np.random.random_integers(0,3)
    if(choice == 0):
      g = small(connectivity='low')
      goodBad = np.random.random_integers(3,4)
    elif(choice == 1):
      g = small(connectivity='high')
      goodBad = np.random.random_integers(4,5)
    elif(choice == 2):
      g = large(connectivity='low')
      goodBad = np.random.random_integers(0,2)
    elif(choice == 3):
      g = large(connectivity='high')
      goodBad = np.random.random_integers(2,4)

    numNodes = nx.number_of_nodes(g)
    numEdges = nx.number_of_edges(g)
    averageOutDegree =
(sum(g.out_degree().values()))/float(g.number_of_nodes()))
    averageInDegree =
(sum(g.in_degree().values()))/float(g.number_of_nodes()))
```

58

```python
    averageWeightedOutDegree =
(sum(g.out_degree(weight='weight').values())/float(g.number_of_nodes())
)
    averageWeightedInDegree =
(sum(g.in_degree(weight='weight').values())/float(g.number_of_nodes()))
    root = np.random.random_integers(0,numNodes-1)
    while (len(g[root]) <= 0): root =
np.random.random_integers(0,numNodes-1)
    bfs_tree = nx.bfs_tree(g,root)
    radius = nx.radius(bfs_tree.to_undirected())


#data.append((goodBad,numNodes,numEdges,averageWeightedOutDegree,averag
eWeightedInDegree,radius))
    print
str(goodBad)+","+str(numNodes)+","+str(numEdges)+","+str(averageWeighte
dOutDegree)+","+str(averageWeightedInDegree)+","+str(radius)
```

# Appendix E

## Usage processing code

**ChrisPy.py**

```python
import urlparse
import datetime
import re
import sys

userSessionPeriod = 15*60 # 15 min
regex = '([(\d\.)]+) - - \[(.*?)\] "(GET|POST) (.*?) HTTP/\d+\.\d+"
(\d+) (\d+|-) "(.*?)" "(.*?)"'
extensions = ('.asp','.aspx','.axd','.asx','.asmx','.ashx',
        '.cfm',
        '.yaws',
        '.html','.htm','.xhtml','.jhtml',
        '.jsp','.jspx','.wss','.do','.action',
        '.pl',
        '.php','.php4','.php3','.phtml',
        '.py',
        '.rb',
        '.cgi','.dll',
        '.adp','.r')
months = {'Jan': 1,'Feb': 2,'Mar': 3,'Apr': 4,'May': 5,'Jun': 6,'Jul':
7,'Aug': 8,'Sep': 9,'Oct': 10,'Nov': 11,'Dec': 12}


data = []
lines = []
numberOfLinesRead = 0
linesInList = 0
numberSkipped = 0
numberOfUserSessions = 0

users = {}
userNav = {}



# Determine if the url is a web page or another resource like
images,javascript,video,etc.
def isWebPage(url):
    t = urlparse.urlparse(url)

    #If there is no extension, assume it's still a web page
    if "." not in t.path: return True
    if t.path.endswith('/') or t.path.endswith(extensions): return True

    return False


def startProgram():
```

```python
    if len(sys.argv) < 2:
        print "To run: python",sys.argv[0],"<inputFileName>
[<outputFileName>]"
        sys.exit();

    inputFileName = sys.argv[1]
    outputFileName = None
    if len(sys.argv) == 3: outputFileName = sys.argv[2]

    print "Input file: ",inputFileName
    print "Output file: ",outputFileName

    return (inputFileName,outputFileName)

#Custom parseDate. Faster than using the "built in" parseDate
def parseDate(d):
    date = d[:d.find(":")].split("/")
    time = d[d.find(":")+1:d.find(" ")].split(":")
    return
datetime.datetime(int(date[2]),months[date[1]],int(date[0]),int(time[0]
),int(time[1]),int(time[2]))



def filterEntries(data,entries):
    """
    Filtering:
    - Only POST and GET
    - Everything over 3XX, which is an error
    - Exclude image,css,javascript,etc

    Converting:
    - Datetime to timestamp (int)
    """
    numberSkipped = 0
    for l in data:
        m = re.match(regex, l)
        if m != None:
            m = list(m.groups())
            del(m[2]) #Deleting POST/GET value since we don't really
need it anymore
            if int(m[3][0]) <= 3: # If status code is below 3xx
                if isWebPage(m[2]) and not
m[2].startswith(('/custom/aktivitetskalender/','/wp-cron.php')): #
check extension. TODO: remove kongsberg.no special case
                    m[1] = int(parseDate(m[1]).strftime('%s'))
                    #del(m[3:]) # Remove these for now TODO: remove
this later
                    entries.append(m)
                else: numberSkipped += 1
            else: numberSkipped += 1
        else: numberSkipped += 1
    return numberSkipped
```

61

```python
def identifyUserSessions(users,userNav):
    """
    User sessions
    1. Order them by IP address
    2. Sort items by time
    3. If time between two requests are over a limit: mark it as a new
session
    4. We don't care about two requests to the same page in a row, so
ignore them
    """
    numberOfUserSessions = 0
    for u in users:
        if len(users[u]) > 0:
            users[u] = sorted(users[u], key=lambda k: k[1]) # Sort by
time
            sessions = []
            temp = []
            prevTime = users[u][0][1]
            for l in users[u]:
                temp.append(l)
                if (prevTime + userSessionPeriod) < l[1]:
                    #print "-----------------------"
                    numberOfUserSessions += 1
                    sessions.append(temp)
                    temp = []
                #print
datetime.datetime.fromtimestamp(l[1]).strftime('%Y-%m-%d %H:%M:%S'),
l[2]
                prevTime = l[1]

            if len(temp) > 0: sessions.append(temp)

            #Put them into a dict: (fromURL,toURL) = numberOfNavs
            for session in sessions:
                if len(session) > 1: #If only 1 item, they user does
not navigate anywhere, so we don't care
                    for l in range(len(session)-1):
                        if session[l][2] != session[l+1][2]:
                            key = (session[l][2],session[l+1][2])
                            if key in userNav: userNav[key] += 1
                            else: userNav[key] = 1
    return numberOfUserSessions




def output(outputFileName,userNav):
    if outputFileName != None:
        with open(outputFileName,"w") as FileObj:
            for key in userNav:
                FileObj.write(str(userNav[key])+"|"+str(key)+"\n")
    """else: #or the screen
        for n in userNav:
            print userNav[n],":",n
    """
```

```python
def
programStatus(linesRead,linesInList,linesSkipped,IPaddresses,userSessio
ns,userNav):
    print 'Number of lines read: ', linesRead
    print 'Number of lines in list: ',linesInList
    print 'Number of lines skipped: ',linesSkipped
    print 'Number of IP addresses: ',IPaddresses
    print 'Number of user sessions: ', userSessions
    print 'Number of user nav: ', userNav


def orderByIP(lines,users):
    for l in lines:
        if l[0] not in users: users[l[0]] = []
        users[l[0]].append(l)
```

## processLog.py

```python
#!/usr/bin/python

from ChrisPy import *


inputFileName,outputFileName = startProgram()

with open(inputFileName) as FileObj:
    for l in FileObj: data.append(l)

#Process each entry
numberSkipped = filterEntries(data,lines)
numberOfLinesRead = len(data)
del data


#Order them by IP address
orderByIP(lines,users)
linesInList = len(lines)
del lines

#Idenitfy user sessions
numberOfUserSessions = identifyUserSessions(users,userNav)


output(outputFileName,userNav)
programStatus(numberOfLinesRead,linesInList,numberOfLinesRead-
linesInList,len(users),numberOfUserSessions,len(userNav))
```

## processLogMPI.py

```python
#!/usr/bin/python

import math
```
63

```python
from ChrisPy import *
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()


if rank == 0:

    inputFileName,outputFileName = startProgram()

    l2 = []
    with open(sys.argv[1]) as FileObj:
        for l in FileObj:
            l2.append(l)
            numberOfLinesRead += 1

    chunks = size
    n = int(math.ceil(float(len(l2))/float(chunks)))
    for i in xrange(0, len(l2), n):
            data.append(l2[i:i+n])
    del l2

data = comm.scatter(data, root=0)

#Process each entry (in parallel)
numberSkipped = filterEntries(data,lines)

del data
data = []


l2 = comm.gather(lines, root=0)

if rank == 0:
    lines = []
    for l in l2:
        lines += l
    del l2

    #Order them by IP address
    orderByIP(lines,users)
    linesInList = len(lines)
    del lines

    chunks = size
    data = []
    for i in range(chunks): data.append({})
    n = int(math.ceil(float(len(users))/float(chunks)))
    i = -1
    c = 0
    for k in users:
        if c % n == 0: i += 1
        data[i][k] = users[k]
        c += 1
```

64

```python
users = comm.scatter(data, root=0)
del data

#Identify user sessions (in parallel)
numberOfUserSessions = identifyUserSessions(users,userNav)


root_userNav = comm.gather(userNav, root=0)
root_numberOfUserSessions = comm.gather(numberOfUserSessions, root=0)
root_users = comm.gather(users, root=0)

if rank == 0:
    userNav = {}
    for i in root_userNav: userNav.update(i)

    numberOfUserSessions = 0
    for i in root_numberOfUserSessions: numberOfUserSessions += i

    users = {}
    for i in root_users: users.update(i)

    output(outputFileName,userNav)
    programStatus(numberOfLinesRead,linesInList,numberOfLinesRead-
linesInList,len(users),numberOfUserSessions,len(userNav))
```
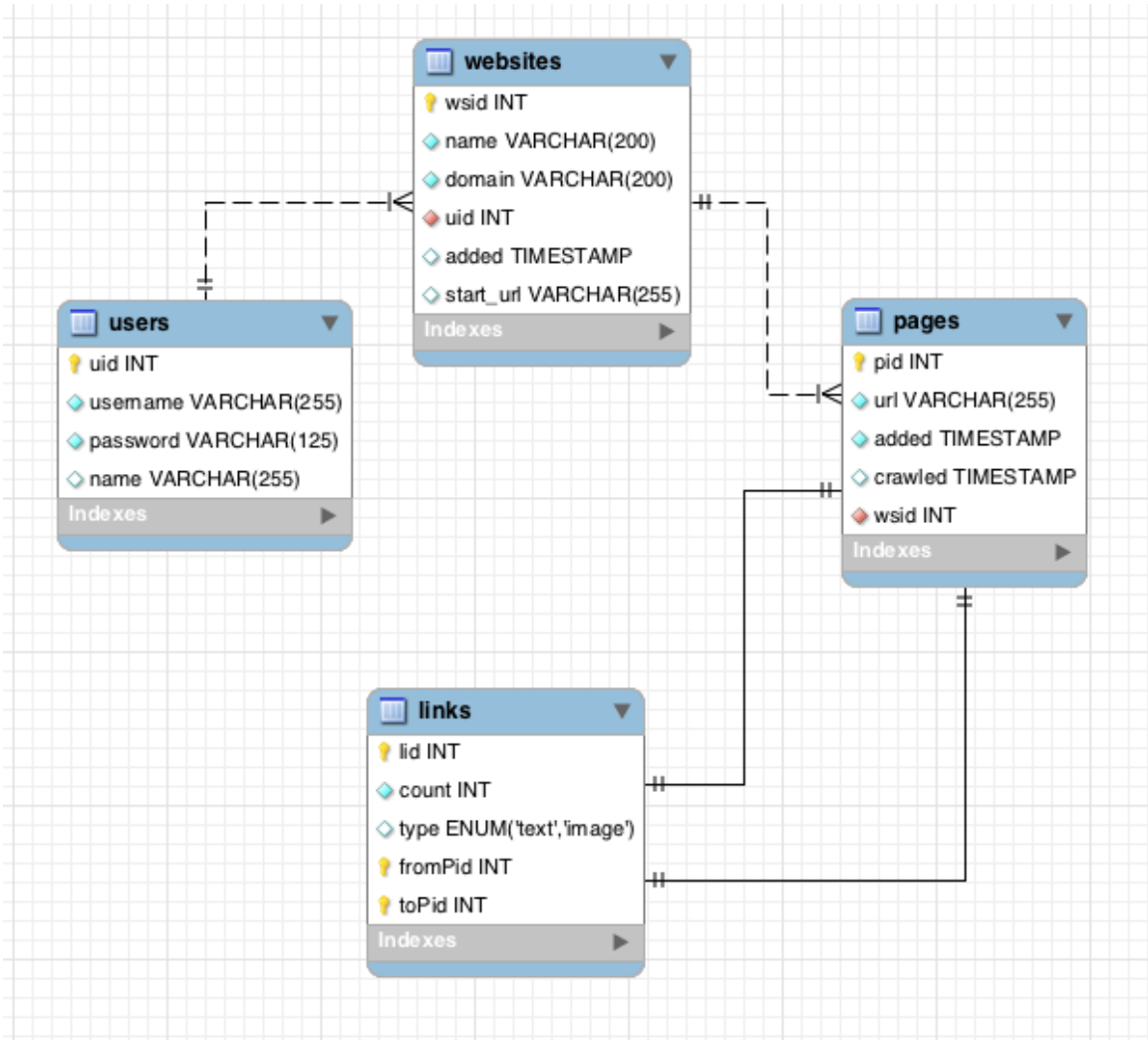
# Appendix F

## Database model

**websites**
- 🔑 wsid INT
- ◇ name VARCHAR(200)
- ◇ domain VARCHAR(200)
- 🔴 uid INT
- ◇ added TIMESTAMP
- ◇ start_url VARCHAR(255)
- Indexes ▶

**users**
- 🔑 uid INT
- ◇ username VARCHAR(255)
- ◇ password VARCHAR(125)
- ◇ name VARCHAR(255)
- Indexes ▶

**pages**
- 🔑 pid INT
- ◇ url VARCHAR(255)
- ◇ added TIMESTAMP
- ◇ crawled TIMESTAMP
- 🔴 wsid INT
- Indexes ▶

**links**
- 🔑 lid INT
- ◇ count INT
- ◇ type ENUM('text','image')
- 🔑 fromPid INT
- 🔑 toPid INT
- Indexes ▶

# REFERENCES

Ben-David, A., & Frank, E. (2009). Accuracy of machine learning models versus "hand crafted" expert systems–A credit scoring case study. *Expert Systems with Applications, 36*(3), 5264-5271.

Berners-Lee, T., Masinter, L., & McCahill, M. (1994). Uniform resource locators (URL).

Campbell, D. J. (1988). Task complexity: A review and analysis. *Academy of Management Review, 13*(1), 40-52.

Catledge, L. D., & Pitkow, J. E. (1995). Characterizing browsing strategies in the world-wide web. *Computer Networks and ISDN Systems, 27*(6), 1065-1073.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* MIT press Cambridge.

Cothey, V. (2004). Web-crawling reliability. *Journal of the American Society for Information Science and Technology, 55*(14), 1228-1238.

Dalcín, L., Paz, R., & Storti, M. (2005). MPI for python. *Journal of Parallel and Distributed Computing, 65*(9), 1108-1115.

67

Dalcín, L., Paz, R., Storti, M., & D'Elía, J. (2008). MPI for python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing, 68*(5), 655-662.

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI: Portable parallel programming with the message-passing interface* MIT press.

Harty, J. (2011). Finding usability bugs with automated tests. *Commun.ACM, 54*(2), 44-49.

Hwang, W., & Salvendy, G. (2010). Number of people required for usability evaluation: The 10±2 rule. *Communications of the ACM, 53*(5), 130-133.

JACKO, J. A., & SALVENDY, G. (1996). Hierarchical menu design: Breadth, depth, and task complexity. *Perceptual and Motor Skills, 82*(3c), 1187-1201.

Jeffries, R., & Desurvire, H. (1992). Usability testing vs. heuristic evaluation: Was there a contest? *ACM SIGCHI Bulletin, 24*(4), 39-41.

Joshi, A., & Krishnapuram, R. (2000). *On Mining Web Access Logs,*

Korvald, C., Kim, E., & Reza, H.Evaluation and implementation of machine learning techniques in usability testing for web sites.

Kung, D. C., Liu, C., & Hsia, P. (2000). An object-oriented web test model for testing web applications. Paper presented at the *Quality Software, 2000. Proceedings. First Asia-Pacific Conference On,* 111-120.

Liu, H., Janssen, J., & Milios, E. (2006). Using HMM to learn user browsing patterns for focused web crawling. *Data & Knowledge Engineering, 59*(2), 270-291.

Marchetto, A., Tiella, R., Tonella, P., Alshahwan, N., & Harman, M. (2011). Crawlability metrics for automated web testing. *International Journal on Software Tools for Technology Transfer, 13*(2), 131-149.

Masseglia, F., Poncelet, P., & Teisseire, M. (1999). Using data mining techniques on web access logs to dynamically improve hypertext structure. *ACM Sigweb Newsletter, 8*(3), 13-19.

Najork, M., & Wiener, J. L. (2001). Breadth-first crawling yields high-quality pages. Paper presented at the *Proceedings of the 10th International Conference on World Wide Web,* 114-118.

Nielsen, J. (1994). Usability inspection methods. Paper presented at the *Conference Companion on Human Factors in Computing Systems,* 413-414.

Oztekin, A., Delen, D., Turkyilmaz, A., & Zaim, S. (2013). A machine learning-based usability evaluation method for eLearning systems. *Decision Support Systems,*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Dubourg, V. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research, 12,* 2825-2830.

Rukshan, A., & Baravalle, A. (2012). Automated usability testing: Analysing asia web sites. *arXiv Preprint arXiv:1212.1849,*

Scholtz, J. (2004). Usability evaluation. *National Institute of Standards and Technology,*

Seffah, A., Donyaee, M., Kline, R. B., & Padda, H. K. (2006). Usability measurement and metrics: A consolidated model. *Software Quality Journal, 14*(2), 159-178.

Sonderegger, A., & Sauer, J. (2009). The influence of laboratory set-up in usability tests: Effects on user performance, subjective ratings and physiological measures. *Ergonomics, 52*(11), 1350-1361.

Srivastava, J., Cooley, R., Deshpande, M., & Tan, P. (2000). Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter, 1*(2), 12-23.

Wu, Y., & Offutt, J. (2002). Modeling and testing web-based applications. *GMU ISE Technical ISE-TR-02-08,*

Zhou, B., & Chen, J. (2002). User behavior based website link structure evaluation and improvement. Paper presented at the *ICWI,* 168-175. Retrieved from http://dblp.uni-trier.de/db/conf/iadis/icwi2002.html#ZhouC02