Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 20, 2014

Visual Analysis of Websites

Giovanni Viviani

Abstract

The fast expansion of websites has led to a situation where websites have become hard to maintain due to their size and complexity.
We present a tool, named Beholder, that allows the visualization and analysis of websites. The tool is able to gather information about a website and visualize its structure. This allows us to identify problems or flaws in the design of the structure of the website.
We have used Beholder on websites of varying size. Thanks to the tool we have been able to identify flaws in their design and problems related to single pages.

Advisor
Prof. Dr. Michele Lanza

Advisor's approval (Prof. Dr. Michele Lanza): Date:

Acknowledgements

First of all I would like to thank my advisor, Prof. Dr. Michele Lanza , that never stopped pushing me to my best. I don't think I would have been able to achieve so much without your help.

I would also thank the REVEAL group as a whole, for supporting me whenever they have been able to. Our weekly meetings have been a source of inspiration for me.

Special thanks to my parents, Anselmo Viviani and Monique Bernasconi, and my sister, Camila Viviani. You never stopped believing in me and I couldn't have done it without your support.

I want to express my gratitude to all the friends I have met during my studies. In particular, I want to thank Antonio Azara, Alessio Aurecchia, Filippo Ferrario, Giorgio Gori and Jacques Dafflon. You guys have been the best companions I could have wished for and studying with you has been a pleasure.

Finally, I would also like to thank Ambra Garcea, Simone Bianchi and Vittoria Viganò. You have always been there to support me. Thanks friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The size and complexity of websites has grown over the years. This has lead many websites to become hard to maintain. What is missing is a proper way to analyse a website and this lack has caused an increment of the effort and cost required in the maintenance.

The user needs a tool that allows him to analyse his website and identify problems in it. Since for human beings it is simpler to understand concepts visually, the tool should be structured so that it provides a visual representation of the website.

We have created Beholder, a web-based tool that provides an interface for the user to see, interact and analyse the website. The tool allows the user to crawl any website and visualize it as a graph, using polymetric views[1] to display additional information.

The visual representation is interactive. This allows the user to adapt it to his needs by selecting which information he wants to display or hide. Through the use of our tool, any user is able to grasp the structure of a website and analyse it to identify its problems.

1.1 Structure of the Document

The remainder of this document is organized as follow.

In Chapter 2 we talk about the current state of the art of website analysis.

In Chapter 3 present Beholder, explaining how it is structured and what is possible to do with it.

In Chapter 4 we show how Beholder can be used on real websites.

In Chapter 5 we draw some conclusion about the project and discuss possible future works.

In Appendix A we talk about the implementation details of the tool.

Chapter 2

Related work

2.1 Validators

”A validator is a computer program used to check the validity or syntactical correctness of a fragment of code or document.”[1]

In case the aim is to get a feedback on the source code, validators allows the user to completely automatize the process. One of the most famous HTML validators is the Markup Validation Service[2] hosted by the World Wide Web Consortium (W3C) [3].

The final report of a validator generally consist only in a list of the errors and/or imperfection in the source code. Additional information might be provided, but they are generally limited to cross-browser warnings.

While validators can be really useful to identify syntax error, their major flaw is their complete inability to see the ”big picture”. This prevent the user to check anything about the final website.

2.2 Search Engine Optimization

Search Engine Optimization (SEO) is the process of analysing a website in order to find a way to increase its visibility. In particular, the aim is to improve the structure of the website so that search engine will rank it better.

Search Engine Optimization started in the mid-1990s, when webmasters realized the important of the rank of webpage in a search engine, but the term firstly appeared only in 1997 in a document from the Multimedia Marketing Group (MMG) [4].

Nowadays there are many Search Engine Optimization tools available on the web: by just using a search engine, it is possible to find hundreds of them. In most cases, their use requires some sort of payment, usually in the form of a monthly subscription. Among the best ones we can find tools like Customer Magnetism [5] and Boostability[6].

Search Engine Optimizations offers a more detailed analysis of a website. The analysis is done on the final content of the page, and not on the source code. This allow a more realistic analysis. Still, the result is far from a complete analysis. In addition, the tools usually do not provide access to the raw results of the analysis but they just present a final report. This can be a problem if the user is looking for specific information that is not present in the report.

[1]http://en.wikipedia.org/wiki/Validator
[2]`http://validator.w3.org/`
[3]`http://w3.org`
[4]`http://web.archive.org/web/19970801004204/www.mmgco.com/campaign.html`
[5]`http://www.customermagnetism.com/`
[6]`http://www.boostability.com/`

2.3 Website testing and reporting tool

In some cases there might be need of a more complex analysis of a website. This is the case when the main concern is not the page rank of the page, but its usability or its design.

In this case, the are not many options available. Few tools can be found on the web, but they all require a monthly fee to be used. In some cases a trial version is available, but it is always very limited. Unfortunately there is no valid free replacement for them.

An example for this kind of tool is Nibbler [7]. Nibbler scan a website and executes a set of tests on it, finally returning a detailed report on the results. While this kind of tools provides more information, it still lack a proper visualization. As we know, for humans it is easier to understand something presented with a graphical visualization. Even if the user is able to obtain all the information needed, the fact that the report is just a long list of text make it harder to understand it.

As we can see, there are many different kind of analysis, but they are all in some way incomplete. The main problem of the analysis we presented is the lack of a graphical visualization. This causes its results to be hard to understand. In addition, most of them do not provide all the information that the user might need. For this reason we created a tool that provide to the user a graphical visualization and direct access to all raw data, allowing him to get all the information he wants.

[7] `http://nibbler.silktide.com/`

Chapter 3

Beholder

In this chapter we present Beholder. We analyse each part of it in details. We can see a preview of the tool in figure 3.1.

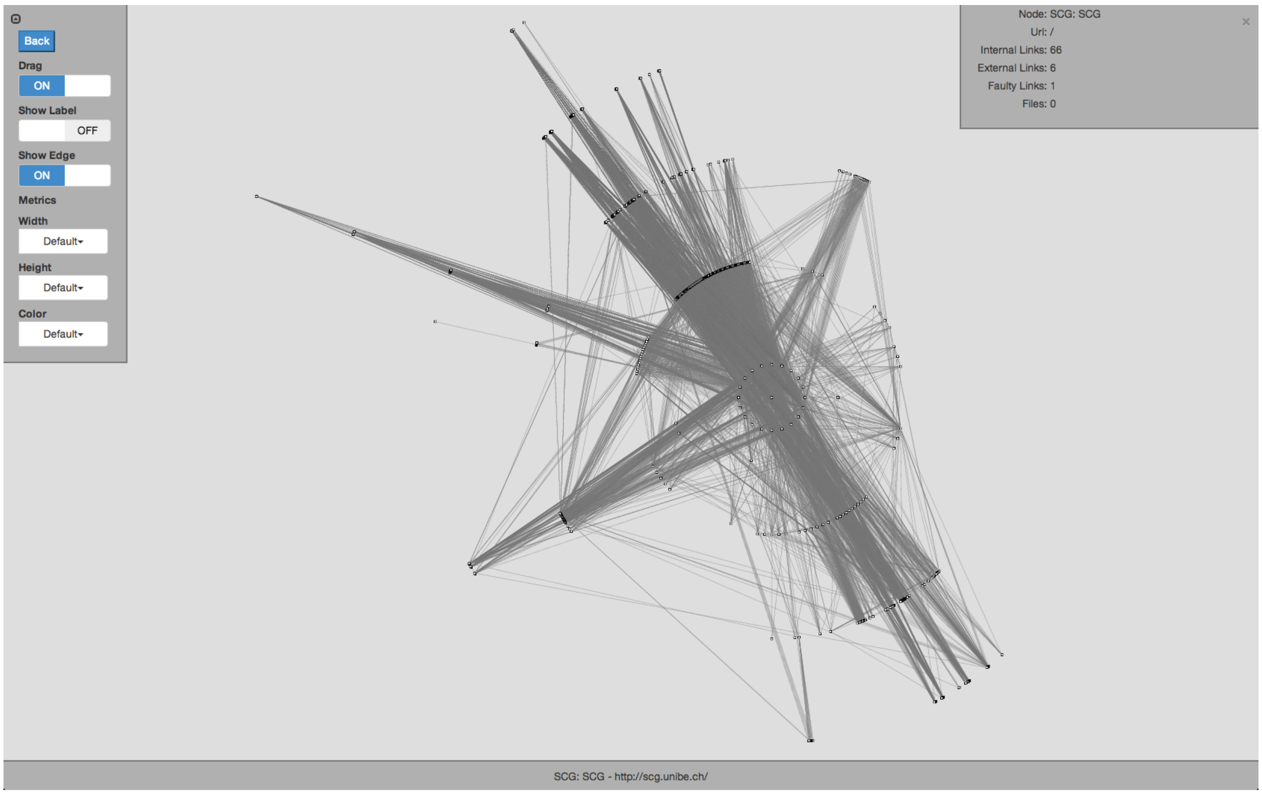


Figure 3.1. A preview of the graphical visualization of a website

Beholder is a web tool that provides to any user the ability to gather informations about a website and graphically visualize it at any time.

The information are gathered by a custom crawler that crawl the website ignoring cross-domain pages. All information are then stored in a database. In a second time, the website can be visualized under the form of a graph that represent its structure.

3.1 Crawler

The crawler has the responsibility to obtain all the information from a website. Failure in doing that would lead to lack of important details and/or pages.

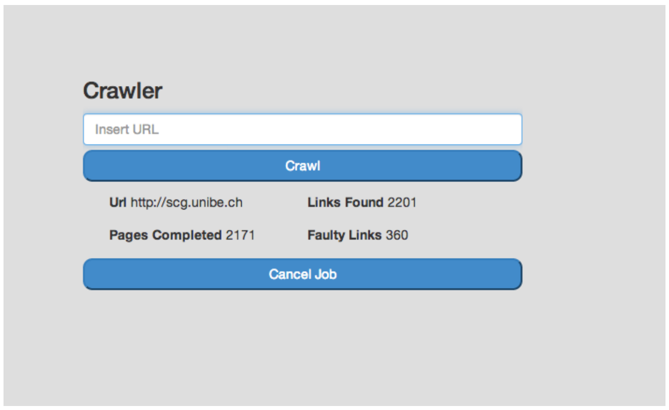
We put some effort in the development of a crawler able to gather all the information we are interested in. The decision of creating a custom crawler instead of using an already existing open source crawler is based on the needs of the tool. Usually crawlers are used in combination with a search engine: they crawl all the pages and store their content so that an indexer can then index them. In our case we do not need the full page, but just some information about it.

The base idea behind the crawler remains the same: the crawler start at a specific URL and fetch the webpage. Once it has retrieved it, the crawler parse the page to get all the URLs and repeat the same routine for each one of them.

Since our aim is not to map the internet but just to gather information, the crawler has been instructed to ignore cross-domain URLs. For those pages it just check if the link is working by submitting an HEAD request. In addition, while crawling it keeps track of additional informations like: number of links to/from a webpage, number of broken links from a page, etc.

Once the crawler has exhausted the pages of a website, it stores all the information gathered in a local database for future use. Each result is stored separately and it is possible to store multiple entries of the same website.

The crawler is hidden to the user. All communications between the user and the crawler are handled by the GUI. The user can submit an URL to the server, that will take care to pass it to the crawler and provide a way to obtain information about the state of the job. There is no way to predict how long a job will take or how many links are left to be crawled, so the feedback generated by the crawler consists in information about the current state, like how many links have been crawled and how many links have been found, as shown in figure 3.2



Url http://scg.unibe.ch **Links Found** 2201
Pages Completed 2171 **Faulty Links** 360

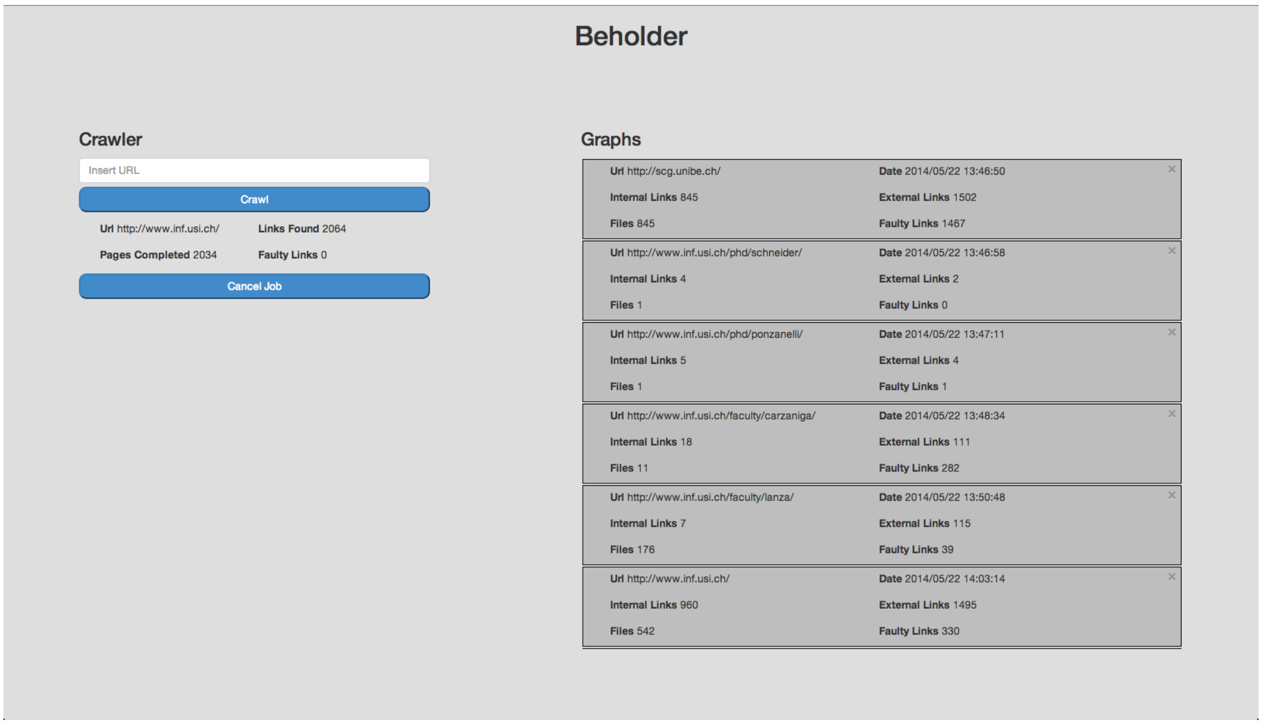
Figure 3.2. Example of a status of the job currently being executed by the crawler

3.2 User Interface

The interface has been designed to allow a fast access to both the crawler and the graphs. A registration is required in order to use the tool, and once the user has logged in a cookie is generated to maintain the authentication. Figure 3.3 show the main user interface, that can be divided in two parts: on the left side we have the crawler handler and on the right we can see a list of the websites already present in the database and ready to be analysed.

The crawler handler allows the user to submit a new URL to be crawled or to cancel an existing running job. In addition, it provides information about the current state of the URL currently being crawled. Below that we can find informations about errors that prevented the job to be executed.

The list on the right side contains all the URLs already crawled by the user. The list is fetched from the database and it is automatically updated any time the crawler finishes a job. In addition to the URL, it provides information about the complete website. Multiple results for the same URL can be saved since it is highly possible that the website has changed in the meantime. Finally, it is also possible to eliminate element from this list, that will be then removed from the database too. Figure 3.4 shows an example of a entry of the list.


Beholder

Crawler

Insert URL

Crawl

Url http://www.inf.usi.ch/ **Links Found** 2064
Pages Completed 2034 **Faulty Links** 0

Cancel Job

Graphs

Url http://scg.unibe.ch/ **Date** 2014/05/22 13:46:50
Internal Links 845 **External Links** 1502
Files 845 **Faulty Links** 1467

Url http://www.inf.usi.ch/phd/schneider/ **Date** 2014/05/22 13:46:58
Internal Links 4 **External Links** 2
Files 1 **Faulty Links** 0

Url http://www.inf.usi.ch/phd/ponzanelli/ **Date** 2014/05/22 13:47:11
Internal Links 5 **External Links** 4
Files 1 **Faulty Links** 1

Url http://www.inf.usi.ch/faculty/carzaniga/ **Date** 2014/05/22 13:48:34
Internal Links 18 **External Links** 111
Files 11 **Faulty Links** 282

Url http://www.inf.usi.ch/faculty/lanza/ **Date** 2014/05/22 13:50:48
Internal Links 7 **External Links** 115
Files 176 **Faulty Links** 39

Url http://www.inf.usi.ch/ **Date** 2014/05/22 14:03:14
Internal Links 960 **External Links** 1495
Files 542 **Faulty Links** 330

Figure 3.3. The user interface

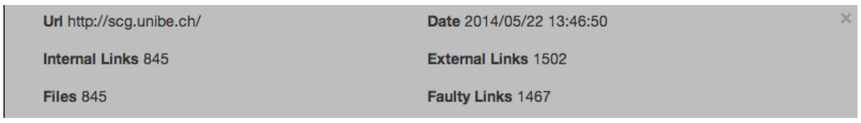

Url http://scg.unibe.ch/ **Date** 2014/05/22 13:46:50
Internal Links 845 **External Links** 1502
Files 845 **Faulty Links** 1467

Figure 3.4. An example of the resume of a website crawled

3.3 Visualization

Once the crawler has finished crawling a website, the tool allows to visualize it at any time. Figure 3.5 shows an example of the visualization of a website.

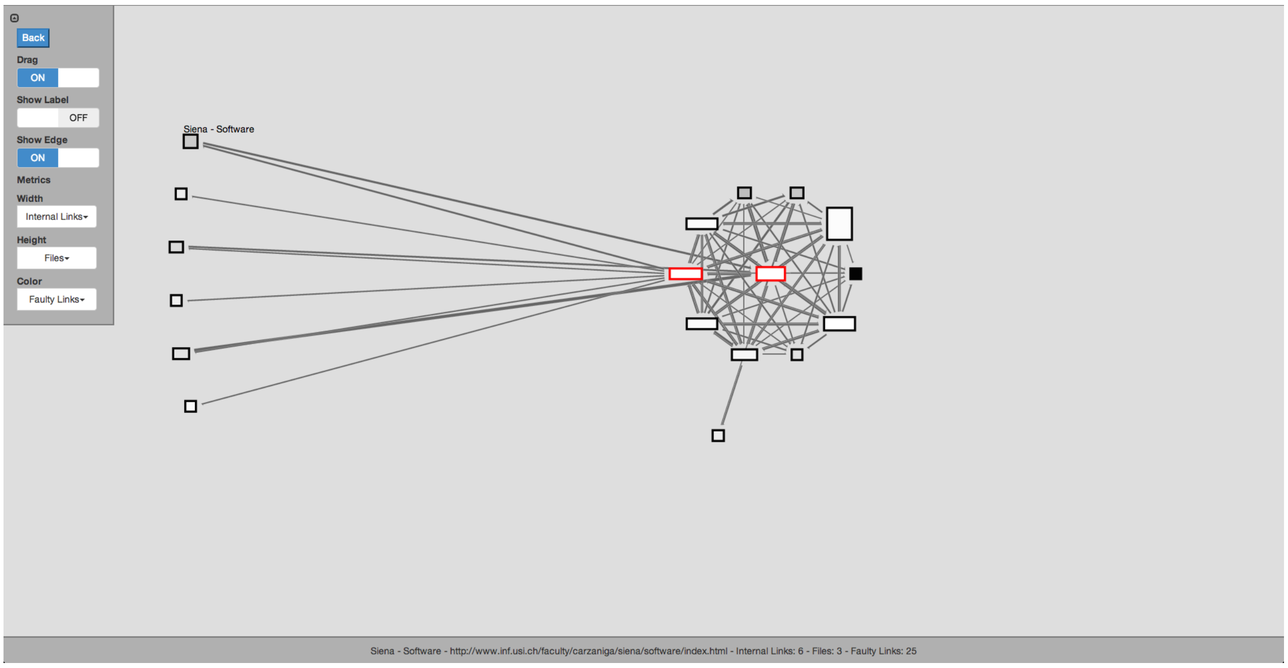
The website structure is visualized in the form a of a directional radial graph. Each rectangular node represent a webpage and each connection means that the webpage contains at least one link to the other page. The thickness of the connection indicates the number of link going from the same page to another one. Each node can be identified by its label, that it is displayed on request.

The graphs contains only internal links and does not display cross-domain links. Files are hidden by default, but it is possible to visualise them by interacting with the graph.

3.3.1 Metrics

In order to display additional information on the graph, we applied the concept of polymetric views to the nodes[1]. By using rectangles instead of other shapes we can use two dimension instead of one, allowing us to display more metrics at the same time. A third metric is displayed as the intensity of color of the node. The metrics that are currently usable are the following;

- Number of links pointing to a page belonging to the same website
- Number of links pointing to a page belonging to a external website
- Number of links pointing to a file



Drag ON
Show Label OFF
Show Edge ON
Width Internal Links
Height Files
Color Faulty Links

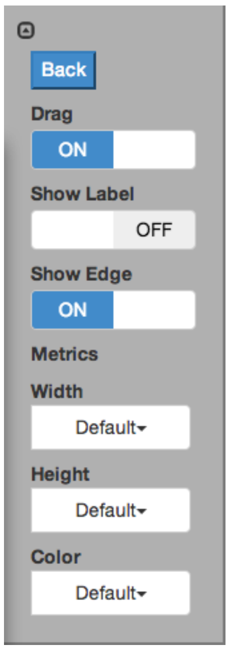
Siena - Software - http://www.inf.usi.ch/faculty/carzaniga/siena/software/index.html - Internal Links: 6 - Files: 3 - Faulty Links: 25

Figure 3.5. An example of the visualization of website

- Number of broken links

Any metric can be applied to different dimension at any time by just interacting with the side menu shown in figure 3.6.

3.3.2 Additional options



Drag ON
Show Label OFF
Show Edge ON
Width Default
Height Default
Color Default

Figure 3.6. The option menu of the visualization

Figure 3.6 shows the option menu that provides additional functionalities to the graph. The current functionalities allow the user to execute the following modification of the graph:

- Set the metrics to be displayed on the three dimensions

- Display or hide all the node labels. By default the labels are visible only when the pointer hovers a node
- Display or hide all the edges. In case of complex graph it might be hard to see the nodes due to the presence of a large amount of edges.
- Switch between drag and select mode. In drag mode it is possible to drag the graph around. In select mode it is possible to draw an area and select all the nodes inside of it. This allows to execute operations on all of them

3.3.3 Interacting with the graph

Is it possible for the user to interact directly with the graph in order to modify it or obtain additional information about it. Currently the following operation are available:

- Move a node to a different position by drag and dropping it in the desired position
- Move the complete graph while in drag mode by dragging it around
- Zoom in and out using the scroll wheel
- Hovering over a node highlights all the nodes linked to it
- Select multiple node while in select mode. This allow to execute actions on all of them
- Right click a node to visualize the context menu. The menu allows to eliminate the node from the graph or visualize a list of files or a list of broken links

3.4 Performance

The graph of a website is rendered using Scalable Vector Graphics (SVG).[1]. Being vector graphics, the performances of the tool might degenerate with the increase in complexity and size of websites. In particular, after crossing a certain threshold it is highly possible to encounter slowdowns in the execution of commands. This threshold is not well defined and vary depending on the specifications of the machine that is rendering the graph, on the current use of the CPU and on the dimension of the website.

[1]`http://www.w3.org/TR/SVG/`

Chapter 4

Application

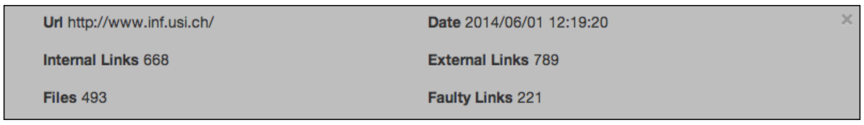
In this chapter we are provide some example of the usage of Beholder on some real websites. We have used the tool on multiple websites varying both in size and complexity. The websites we have analysed are the following:

- **http://www.inf.usi.ch/** The website of the faculty of informatics at the Università della Svizzera Italiana.
- **http://scg.unibe.ch/** The websites of the Software Composition Group of the University of Bern.
- **http://www.inf.usi.ch/faculty/lanza/** The website of the dean of the faculty of informatics at the Università della Svizzea Italiana.

For all websites, we will start by highlighting any information that can be obtained by just looking at the graph. In many cases this is enough to identify some design flaws. We then proceed to add some metrics to the visualization and analyse the resulting graph. We will finally draw some conclusion about what we discovered.

4.1 USI-INF Website

We can see a resume of the website in figure 4.1. The website contains a fairly large number of pages. The number of broken links is actually reasonable compared to the number of pages. The website is also pointing to a large number of external pages, roughly more than one for each internal page.



Url http://www.inf.usi.ch/ **Date** 2014/06/01 12:19:20
Internal Links 668 **External Links** 789
Files 493 **Faulty Links** 221

Figure 4.1. Resume of the USI-INF website

Figure 4.2 shows the graph of the USI-INF website. It is immediately clear that the website is over-complex. There are too many connections between the nodes, up to the point that it is not possible to distinguish them. We can also see that most of the nodes resides in the inner part of the graph, with only few nodes being on the outers circles. In order to simplify the graph, we are going to reduce the graph to the two most inner levels of the graph. Figure 4.3 sows the resulting subgraph.

Even though we are just considering only the inner part of the website, the graph still looks too complicated. A visible problem is related to cross-branch links: while it is expected to see some of them, in some part of the graph there are too many of them. For example, in the highlighted areas we can see thick groups of connections from a branch to another one. Considering the thickness of them and the fact that it happens at least twice with the same branch, we can conclude there must be some sort of design flaw.

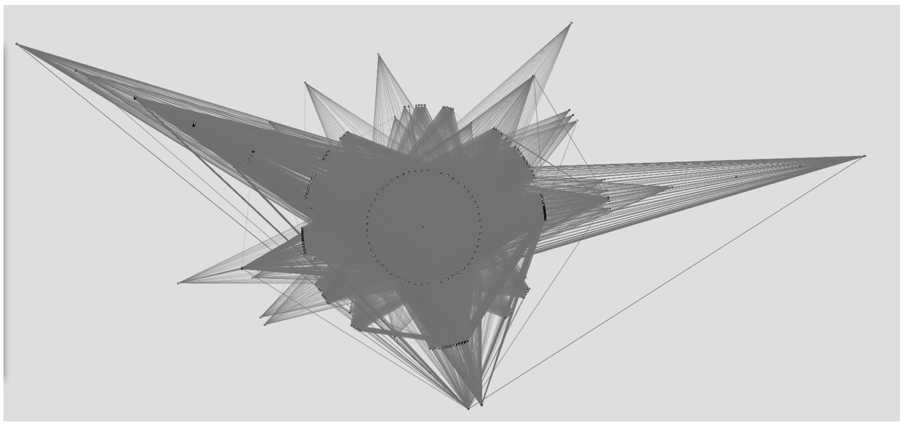


Figure 4.2. Full graph of the USI-INF website

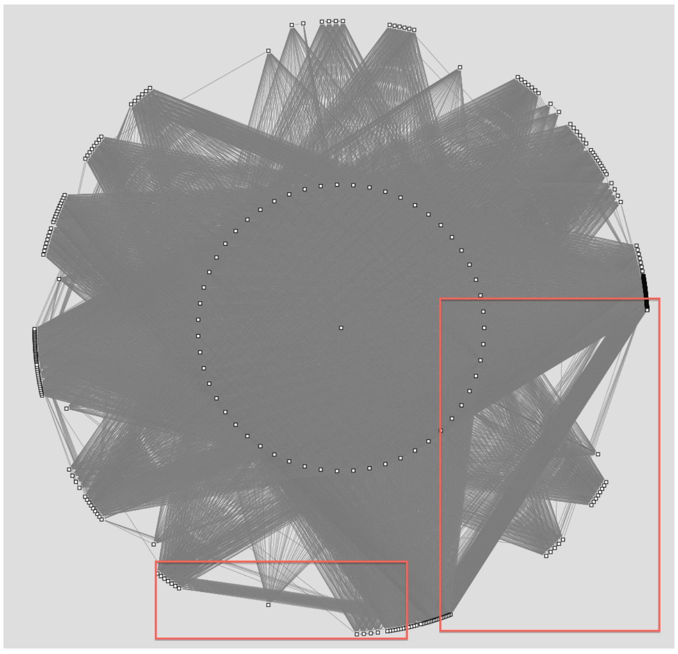


Figure 4.3. Detail of USI-INF

Another problem can bee seen in the fact that the edges of many outer nodes create a huge pyramid below them. It is not normal to have that many connection between an outer node and inner nodes. This problem might be related to the fact that the website contains a big navigation bar: having a menu with many links on every page will lead to a situation similar to the current one.

Let us have a look at some more details. Figure 4.4 show the same graph as figure 4.3 with some metrics applied to the nodes. For the sake of clarity we have removed all the edges of the graph. The width of each node represent the number of internal links, the height the number of external links and the color the number of broken links.

We can see how some nodes have different sizes now, depending on the number of links they contain. Some nodes are actually completely black, indicating that they contains various broken links. Still, this remains difficult to analyse. We need to remember that all values are scaled, therefore if a node contains a really high value for any of those metrics, it will reduce the others node to the minimum size. The best way to analyse is then to reduce the size of the graph again. We can see that on the left side of the graph there are two sector with nodes of different color, highlighted in red. We are going to examine that part of the graph to better see the difference between nodes.

Figure 4.5 shows the two sectors one near the other. The metrics displayed are the same as the previous graph, but we can finally see some details. First of all we notice that most nodes have roughly the same size, with a bunch of smaller nodes and a really big node. The number of broken links also vary between the nodes, but we can see how most of the nodes have a low number of them, if none. Figure 4.6 shows the information about the big node

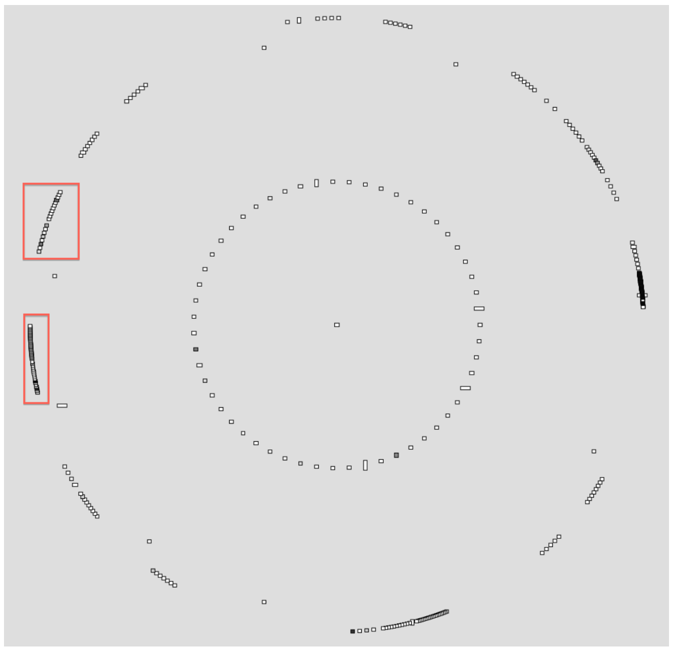


Figure 4.4. Detail of USI-INF with metrics

and the complete black node, that represent the nodes with the highest value for each metric.

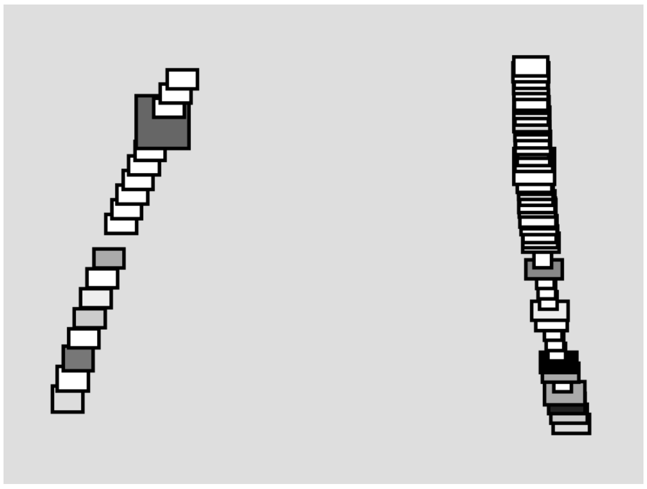


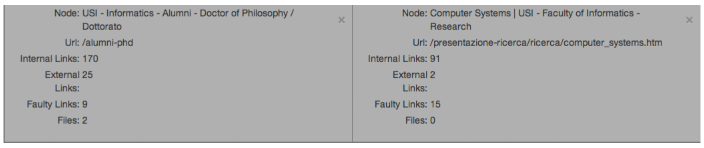
Figure 4.5. Detail of Figure 4.4

We can see how the highest number of faulty links is fifteen. Knowing that the value are scaled, we can say that the rest of the nodes contains a low number of broken links. While it is still a low number, it is worrying that so many pages contains that number of broken links, and this is an issue that should be investigated.

The height of nodes is particularly interesting. We can see how the maximum number of external links is twenty-five, which is pretty low. Still, the big node is way higher than most of the others node. We can then assume that the rest of pages have little if none external links.

The width of nodes less interesting. Most of the the nodes have an average width, with some small nodes. The big node is a special case, having a really high number of internal links. This results in it obfuscating the other nodes and flattening the differences between them.

In conclusion, we can say that the website of the Faculty of Informatics is definitely over-complex and requires a redesign. The complex structure of the graph reflect the state of the website, where it is hard to find the information needed due to every page pointing to every other page. Still, the average number of broken links in the various page is pretty low and the depth level is constant in the various branches.



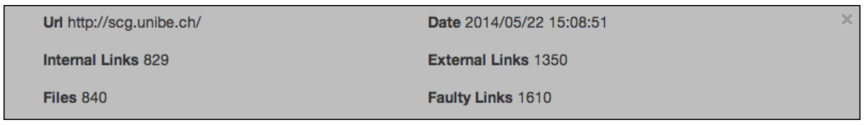
Node: USI - Informatics - Alumni - Doctor of Philosophy / Dottorato
Url: /alumni-phd
Internal Links: 170
External Links: 25
Faulty Links: 9
Files: 2

Node: Computer Systems | USI - Faculty of Informatics - Research
Url: /presentazione-ricerca/ricerca/computer_systems.htm
Internal Links: 91
External Links: 2
Faulty Links: 15
Files: 0

Figure 4.6. Details of nodes of Figure 4.5

4.2 SCG Website

In figure 4.7 we see the information about the SCG website. We can see that this website is even larger than the previous one. It immediately comes to attention the number of broken links, being nearly twice as much as the number of pages of the website. This definitely indicates some sort of problem in some part of the graph that should be investigated.



Url http://scg.unibe.ch/
Date 2014/05/22 15:08:51
Internal Links 829
External Links 1350
Files 840
Faulty Links 1610

Figure 4.7. Resume of the SCG website

Figure 4.8 shows the graph of the website. While still being a complex website, it is a bit more clear than the previous one. We can already obtain some information from the full graph. We can see different branches that expands in depth, but the connections between them are scarce. This indicates that the pages are properly grouped by argument. The pyramids below the outer nodes are small, indicating a small and simple navigation bar pointing only to the main pages. We can also see how the nodes are divided in two inner level and two outer level, with a few scattered nodes outside of them.

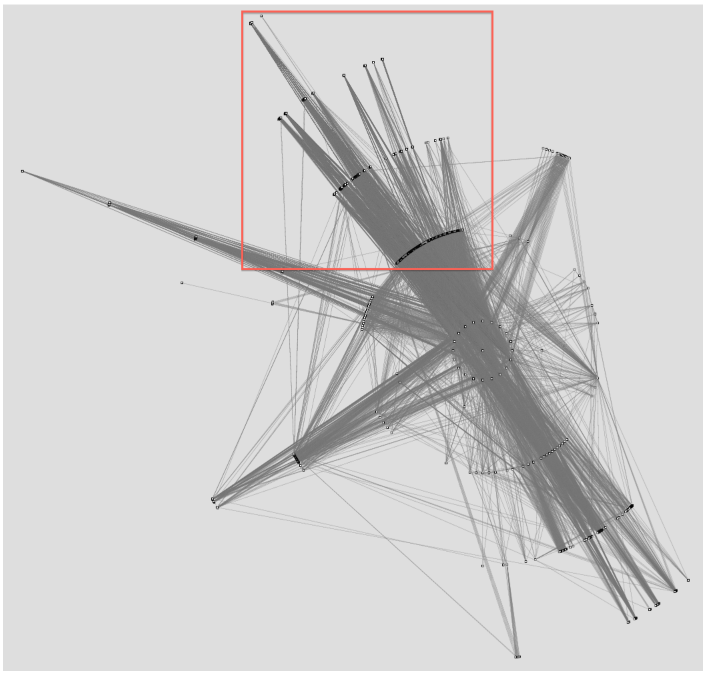


Figure 4.8. Full graph of the SCG website

Now, let us analyse a bit more in detail the graph. The most interesting branch is the one of the wiki, highlighted in red. In figure 4.9 we see its subgraph.

We can see various things about this subgraph. First of all, we see how there are really few cross-branch links. This means that the pages are divided in a meaningful way. Still, we need firstly to apply some metrics to it before being able to see the important details.

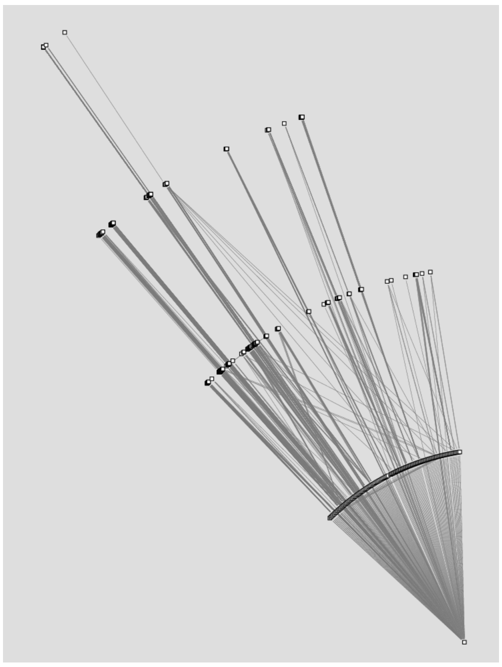
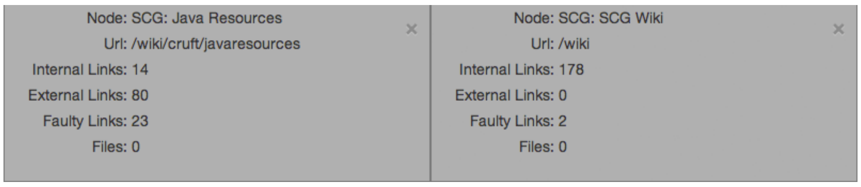


Figure 4.9. Detail of Figure 4.8

In figure 4.11 we see the same graph with some metrics applied. We decided to reduce the distance between the nodes in order to be able to better see the differences between them. We are displaying the number of internal and external links on the width and the height of the nodes. We can immediately see how most of the nodes still have roughly the same size, with a couple of them are bigger than the others. In figure 4.10 we see the information about two of those nodes, the one with the highest number of internal links and the one with the highest number of external links. The first one, having a relatively high number of external links, is a webpage pointing at external references about Java.The page actually does not have any other responsibility, so is not really a problem. The other one is the main page of the wiki, residing on the first level of the graph. Its job is to provide access to the wiki, so we cannot argue about the number of internal links that it contains.



Node: SCG: Java Resources
Url: /wiki/cruft/javaresources
Internal Links: 14
External Links: 80
Faulty Links: 23
Files: 0

Node: SCG: SCG Wiki
Url: /wiki
Internal Links: 178
External Links: 0
Faulty Links: 2
Files: 0

Figure 4.10. Detail of the nodes of figure 4.11

On the extreme left we also see a node that is completely black, meaning that it is full of broken links. With four hundred and more broken links, it would look like to be a major problem of the website. This turn out to be only a momentary problem though, because by crawling again the website the problem disappears leaving a node with many files instead of broken links. This shows us how each information obtained has to be doublechecked. The web is not a static entity: webpages can change at any time and server might go down at any time. A consequence of this is that this error is obfuscating all the faulty links in the graph due to the scaling of the value, making it harder to identify the nodes with broken links.

In conclusion, we can say that despite the dimension the SCG website maintains an organized structure. Unfortunately the analysis is not completely accurate this time due to a web problem during the crawling that lead hundred of links to be seen as broken.

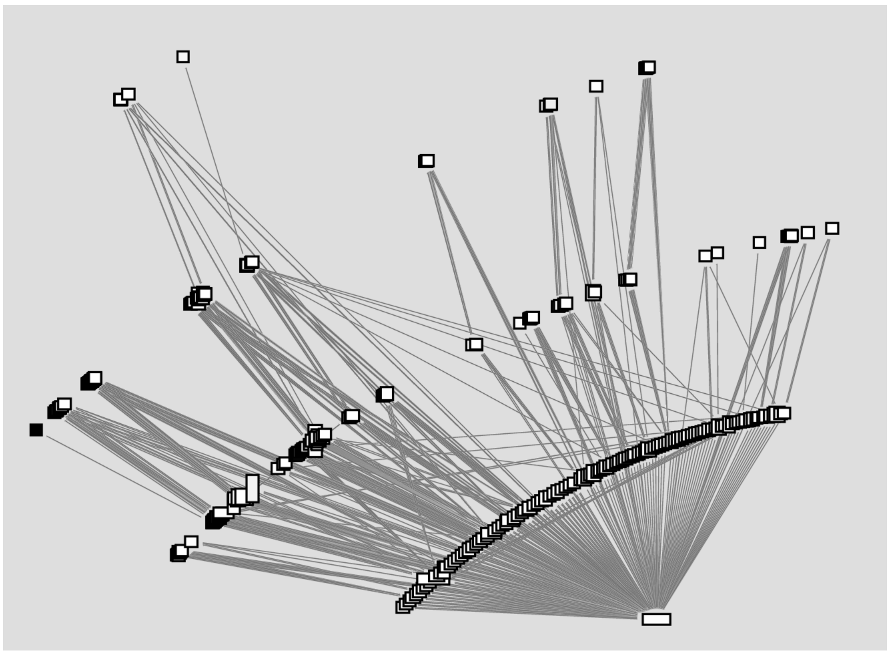
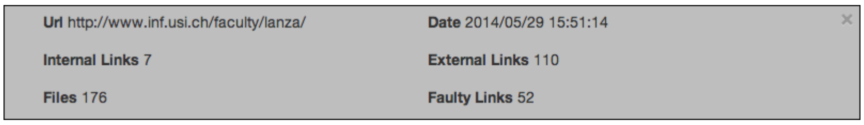


Figure 4.11. Figure 4.9 with metrics

4.3 Michele Lanza's Website

Finally, we are going to see a simpler website. For this purpose we chose the website of Professor Michele Lanza at the Università della Svizzera Italiana. In figure 4.12 we can see the results of the crawling on this website. The number of broken links is definitely too large for the few pages of the website. There is also a high number of files linked to the website, probably publications of the subject.



Url http://www.inf.usi.ch/faculty/lanza/	**Date** 2014/05/29 15:51:14
Internal Links 7	**External Links** 110
Files 176	**Faulty Links** 52

Figure 4.12. Resume of the SCG website

The website is really small and the tool generate a simple graph, as we can see in figure 4.13. This allows us to see in a clear way how single nodes can provide multiple information.

We can immediately see how the structure of the website is really simple, with all pages pointing to each other While this could be seen a problem in a big website, we need to consider the context. This is a personal website, and all the link to its pages are contained in the navigation bar. The limited number of pages allows this to be done without creating an enormous navigation bar.

We can also see how nearly all the edges are of the same size, except for a couple. This indicates that all the pages containing only the link on the navigation bar to the other pages, except for few of them that contains a link in the body too.

Let us start analysing the details of this graph. In figure 4.14 we can see the same graph with some metrics applied. In this cases we decided to apply the number of internal links to the width of nodes, the number of external links to the height and the number of files to the intensity of the colour. We can immediately see that there are different type of nodes. Some have really few connections while others contains a large number of links. We notice a the bottom right node that contains a large number of external links. This is the *service* webpage, that contains the list of the conferences that the subject attended.

For the internal link the biggest node is the homepage, since it contains a presentation with multiple links to other part of the website. We can see a relation to the edges: all thick edges of the graph are connected to the

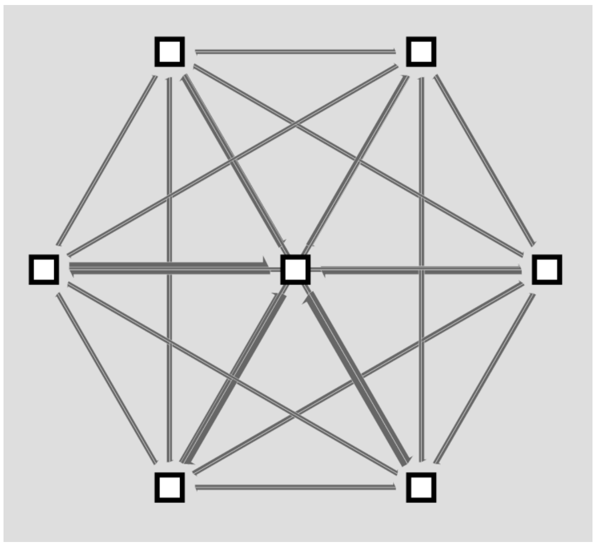


Figure 4.13. Full graph of the website of Michele Lanza

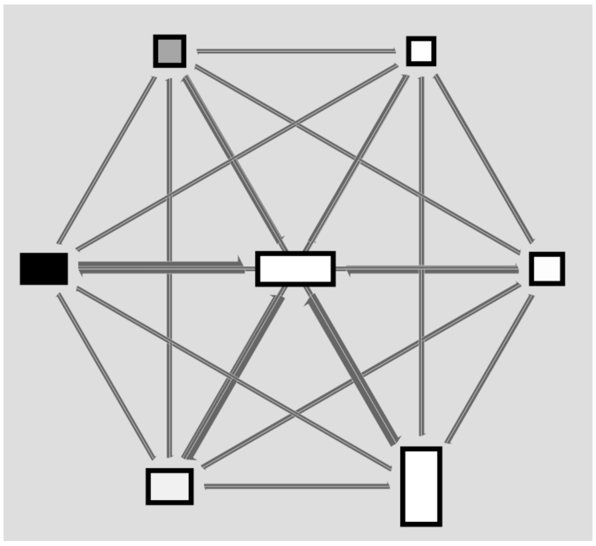


Figure 4.14. Full graph of the website of Michele Lanza with a set of metrics

homepage, so it makes sense that it is the node with the highest number of internal links. Files are scarce around the website and concentrate themselves in two nodes, the one containing all publications and the one about research.

Finally, in figure 4.15 we see the same set of metrics but the number of faulty links instead of the number of files on the intensity of the color. Here we see the first problem of the website: there is a node that has a really high number of faulty links. If we check the details, we see the that it contains twenty-seven broken links, that is roughly equivalent to one ninth of the total number of links of the website. If we check the page, we see how this is caused by links to the websites of past conferences. In this case the decision could be either to keep these links or remove them and leave only plain text. The number of broken links in the rest of the website is low, and while they should still be fixed, it cannot be considered a major problem.

In conclusion we can say that this website is well structured. There are not many problem in the website except for the broken links in one of the pages. The structure is really simple and it conforms to a website about a single person.

These examples show how Beholder is useful for analysing a website. In the three example we managed to analyse the structure of the website and its pages. We have seen that often websites can be complex, up to the point that is hard to even identify how they are structured. But even in those cases we managed to analyse them by focussing only on a subgraph of them.

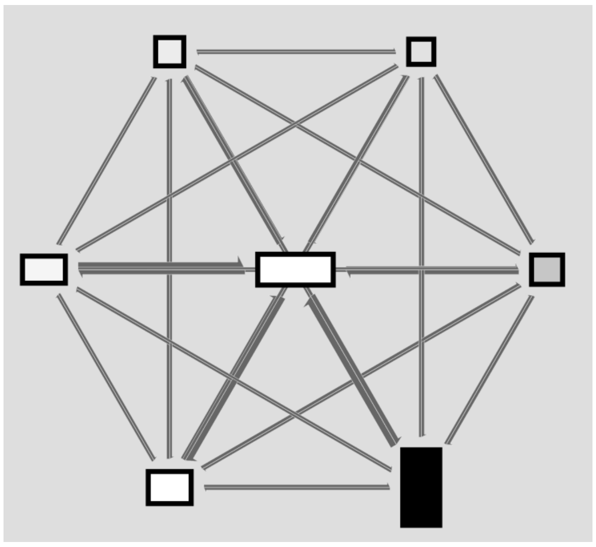


Figure 4.15. Full graph of the website of Michele Lanza with a set of metrics

We have seen how a graphical visualization allows the user to immediately get some information about the website by simply looking at the graph. In addition, the application of metrics to the nodes of the graphs allow to immediately identify problematic pages. This has shown how a simple text report is inefficient in comparison to a proper visualization.

The drawback of the tool is the possibility of a web problem to corrupt the final result. As we have seen in the SCG website, it is possible that a URL generate an error that corrupt the information obtained by the crawler about that page. This error does not directly affect the other URLs crawled, but it might end up modifying the metrics of the node, leading to a possible obfuscation of the values of the other nodes in the graph.

Chapter 5

Conclusion

It has become clear in the last years that there is a need to improve the analysis of websites. Websites have become way too complex to be handled manually, and this had lead to major complication in the task of maintaining them. There is a need for a tool that provides a visualization more understandable than the raw data.

In chapter 2 we saw that some way of analysing websites already exist, but they are not enough yet. Those kind of analysis are restricted to determined aspects of a website, like the quality of the source. While they are helpful to analyse a specific aspect, they lack the capability to offer a view of the big picture. From there comes the need of a better tool to provide a way to analyse a website in its fullness.

In chapter 3 we presented Beholder, a tool that allows the user to analyse completely a website. We explained in details how it is structured and what is possible to see with the tool. We emphasize the fact that it provides a visual analysis, that allows a better understanding than a text report.

Finally in chapter 4 we presented some cases of usage of the tool with some real websites. We have shown how it is possible to obtain various information. We have seen how the tool is able to provide details by both a visual representation and raw data. We have been quickly able to detect some major flaws in the websites proposed and identify a possible solution to them.

In conclusion, we managed to build a tool that allows a visual analysis of websites. In fact Beholder:

1. Provides a visual representation of the structure of a website in the form of a interactive graph
2. Allows to modify the visualization do display different kind of information
3. In addition, it also allows direct access to the raw data

The tool has been designed in order to be deployed on a server and accessed from any browser. Its flexibility and the possibility to crawl and analyse any websites make it possible to adopt it in any real world case.

The tool can be found and used at: `http://beholder.inf.usi.ch`.

5.1 Future Work

The current state of the tool is sufficient to analyse many websites, but there is still space for further development. In particular, there following features are still missing:

1. The tool currently does not support pages generate on the client side. The crawler just fetch the HTML page, therefore any content generate dynamically, for example through JavaScript, is not found.
2. Even if it is possible to modify the graph, those modifications are not saved. Therefore the user has to start the analysis from scratch every time.

3. The list of graphs is not completely editable. It is possible only to add or remove graphs. The next step would consist in allowing to group graphs or share them between users.
4. Currently the graph is created with only a radial layout. It might be interesting to the user if it would be possible to select between various layouts.
5. More complex metrics might be gathered from the crawling task and then displayed on the visualization.

Additional work is also needed in some existing part to improve performances and usability, specifically for the following problems:

1. The generation of the graph takes a long time in case the graph is considerably large. A way to reduce the computation required to generate the graph would speed up this task.
2. The visualization with SVG of large graphs has pretty low performances. This issue should be investigated in order to find a way to lighten the amount of computation needed for the rendering.

Appendix A

Implementation

In this appendix we are going to see in details about the implementation of Beholder. The server and the crawler are written in Java while the client part is in JavaScript.

A.1 External Libraries

To develop Beholder we used different libraries. In this section we are going to see which libraries we used, dividing them between Java and JavaScript libraries.

A.1.1 Java

- **Apache Tomcat**[1] Tomcat is an implementation of a Java Servlet. In serves as base for our server.
- **Jersey**[2] Jersey is a framework used to develop RESTful web services.
- **MongoDB Java Driver**[3] The official Java driver for MongoDB, used to interface with the database.
- **Jstl**[4] Jstl allow us to dynamically generate part of the content of the Jsp templates.
- **Jsoup**[5] We use Jsoup to handle the parsing of a webpage after the crawler has fetch it. Mostly we use this library to get all the URLs contained in a webpage and to convert relative to absolute URLs.
- **Apache Common Validator**[6] When the crawler receive a new URL to be crawled, it needs to check if it is a valid URL or just a normal string. Therefore we use the Apache URL validator to parse the string.

A.1.2 JavaScript

- **Bootstrap**[7] Bootstrap is a framework that allows an easier creation of websites thanks to a set of HTML and CSS templates for various elements.
- **JQuery**[8] JQuery is a famous JavaScript library that allows to interact with HTML and event handler in a simpler way.

[1]http://tomcat.apache.org/
[2]https://jersey.java.net/
[3]http://docs.mongodb.org/ecosystem/drivers/java/
[4]https://jstl.java.net/
[5]http://jsoup.org/
[6]http://commons.apache.org/proper/commons-validator/
[7]http://getbootstrap.com/
[8]http://jquery.com/

- **VivaGraphJS**[9] VivaGraphJS is a graph drawing library designed to support multiple rendering engines. In our case, we use it to render in SVG the final graph.
- **When**[10] When is n implementation of JavaScript Promises , used in replacement of callbacks. The decision of using promises instead of callbacks is based on the fact that promises are more powerful than callbacks and allow to maintain an higher clarity in the code.

A.2 Crawler

The crawler, written in Java, has been created completely from scratch in order to satisfy some requirements of the project. The crawler is not aware of the server: the server just communicate with the crawler by starting it and adding jobs to the queue.

The crawler has been designed to use multiple threads in order to speed up its work. In total, 31 threads are spawned: 1 thread runs the crawler itself and the remaining 30 threads belong to a thread pool that takes care of parsing every webpage.

The crawler work as follow: first all, the starting page (from now on called *origin*) gets checked in order to asses if it actually exist and it is available. If the check passes, the page is fetched through an HTTP request and an object is created. This object contains all the information about the page and it will be later stored in the database. The page is then parsed using JSoup to get all the links in the source. For each link in the list, the following actions are executed: first of all, if it is a relative URL, it is converted to an absolute URL. Then, depending on the URL, one of the following action is executed:

1. If the URL has already been crawled, an edge between the two is saved to be later pushed in the database
2. If the URL has not yet been crawled, but it is already scheduled to be crawled a reference is saved so that the edge will be created once the URL has been crawled
3. If the URL has neither been crawled and put in schedule, the crawler notify the thread pool about this URL. As soon as a thread is free, the URL will be crawled.

The same procedure is repeated for all URLs until all the URLs have been crawled. In addition, for any URL that is not the origin, two additional actions are executed:

1. The URL is not parsed and saved if it is a cross-site URL. In such case, only an HEAD request is issued to check if the link is broken or not.
2. All edges that are missing are added. An edge is missing if the URL was scheduled to be crawled when the edge was discovered.

Once the crawler has finished crawling all URLs, all information gathered are passed to the database handler in order to be stored. Once this is done, the job is considered completed. Therefore all collections and the thread pool are cleared, the state of the crawler is reset to the original state and a new job is fetched from the job queue. In case the job queue is empty, the crawler will just idle until a new job is available.

A.3 Database

After the crawling is done we need a non-volatile way to store the information gathered. We therefore opted for a local database, and we decided to use MongoDB. The advantage of using MongoDB is the fact that entries are stored in a format similar to JSON. This allows to use them directly as JSON object on the client side without having to convert them.

[9] `https://github.com/anvaka/VivaGraphJS`
[10] `https://github.com/cujojs/when`

The database is accessed both by the crawler and by the server using the official Java Driver. The crawler mostly execute only write operation on the database, while the server executes both read and write

The crawler access the database only when one job is finished and it needs to store the information gathered. Once a job is finished, multiple list are passed to the database handler, containing information about URLs parsed, edges, broken links and so on. The handler then create a new graph id and proceed to store all the information in the database.

The server, on the other hand, access the database constantly. Many of the commands that the user submit on the client will require an access to the database. Most of them consist in read access and are executed, for example, when an user requires a new graph or additional information about some webpage. Write operation are only executed in order to add a new user to the database or to delete and existing graph.

A.3.1 Schemas

Table A.1. User Schema

user	
username	String
password	String

Table A.2. Graph Schema

graph	
url	String
id	String
user	[User]
links	Integer
externalLinks	Integer
internalLinks	Integer
faultyLinks	Integer
files	Integer
date	String

Table A.3. Html Node Schema

htmlNode	
url	String
graph	String
id	String
type	String
label	String
externalLinks	Integer
internalLinks	Integer
faultyLinks	Integer
binaryNodes	Integer

Table A.4. Binary Node Schema

binaryNode	
url	String
graph	String
id	String
type	String
label	String
parent	[String]
size	Long

Table A.5. Faulty Link Schema

faultyLink	
url	String
graph	String
code	String
parent	[String]

Table A.6. Edge Schema

edge	
id	String
source	String
target	String
graph	String
value	Integer

Table A.7. Counter Schema

counter	
collection	String
counter	Integer

A.4 Server

The server consist in a Tomcat servlet using Jersey to have a RESTful system. The server handles all communication between the client and the crawler or the database. In addiction, the server also take care of generating the graph before it is passed to the drawing library.

The server can be seen as the core of the tool. Once it is started, it firstly start the crawler thread and connect to the database. All messages directed to the crawler are then passed to an handler that communicates with it. The same happens for the communications with the database.

All messages from the server to the client are firstly encoded as JSON strings. This decision was taken in order to simplify the transmission of messages. In this way, the client can just parse the message, converting it directly in an object.

As said above, the server also take care in generating the graph accordingly to a radial layout. This computation has been moved from the client to the server in order to reduce the amount of computation needed on the client. It also allows us to save the generate graph so that it needs to be generated only once. Once the client request a graph, if this graph has not yet been loaded, the server will fetch all its information from the database and arrange the nodes in a radial layout. This is done by running a Breath First Search algorithm[11] on the list of nodes, so that each node is child of the first node that has a link to it. This allow us to maintain the original hierarchy in most cases, even though the graph is not a tree. Once the generation is finished, the graph is stored for future uses and it is converted in a JSON string representing the complete object, ready to be sent it to the client.

A.5 Client

The client provide the user an user interface that allows him both to interact with the crawler and to visualized previously crawled websites. We can divide the client in two main parts:

- The main interface, where the user interact with the crawler and see the list of websites already crawled.
- The graph visualization, where the user can analyse a website.

A.5.1 Main Interface

The main interface has been designed to provide only the necessary elements in a clear way. It is divided into two areas, each one covering roughly half of the screen, that are separately scrollable.

The left side shows only a text box when the crawler is idling. The box allows the user to submit a job to the crawler through a POST request. When the crawler is not idling, below he box is possible to see the state of the current job. Every second a request is issued to the server to fetch an update on the status of the current job. When the server answers that the crawler is idling, the client stop asking for updates and instead check every 10 seconds if a new job has started. In case it is so, it resume asking for updates. This allow the user to be uptodate all the time on the state of the crawler.

Below the update there is more empty space. This space is reserved for error messages from the crawler. In most cases those errors are about not valid URLs or duplicate request, and contains the type of error with the URL that generated that specific error. Messages do not fade over time, but remain there until the user decide to delete them. Figure A.1 shows an example of such message.

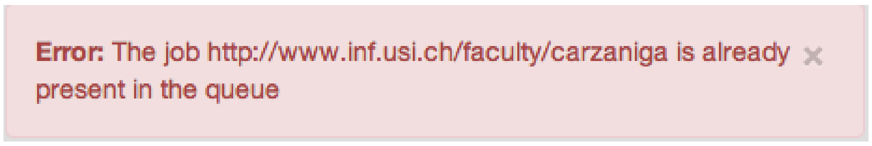

Error: The job http://www.inf.usi.ch/faculty/carzaniga is already present in the queue

Figure A.1. An example of error message form the crawler

The right area contains the list of website, crawled by the user, that can be seen at any time. The list is generated when the page loads through a query to the server that fetches the information from the database. Whenever the crawler finish a job, an entry it is added to this list, so that the list is always uptodate. Clicking on one of the entries open the graph visualization for that website.

A.5.2 Graph Visualization

The largest part of the window is occupied by the graph itself, with small windows on the side that provides additional information or options. The use can interact directly with the graph in order to move or modify it as it pleases. Additional option allows to show more or less details at any time.

In the central part of the window there is the graph. The graph is composed by multiple rectangular nodes connected by arrow-shaped edges. Each node represent a different webpage and the directional edges represent the

[11] http://en.wikipedia.org/wiki/Breadth-first_search

links connecting them. Nodes start with default sizes but can change dimension when the user requires to display some metrics. Edges are designed so that is it possible to show both direction in a single link: one side of the edge indicates one of the direction and, if there exist a connection in the other side too, the other side represent the opposite direction. The thickness of the edge indicates the number of links going from one node to another one. Figure A.2 show how a bidirectional edge appears.

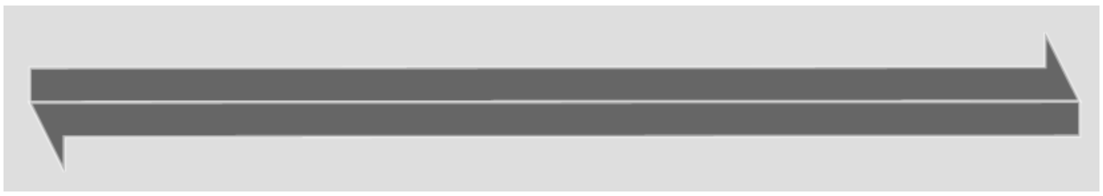
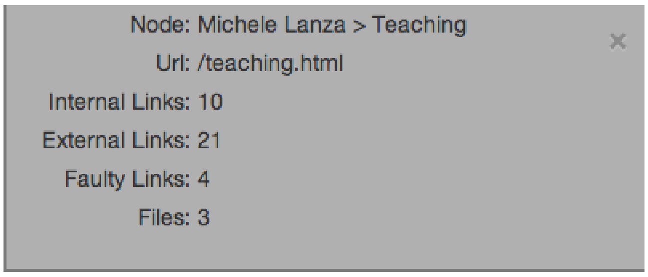


Figure A.2. An example of a bidirectional edge

The graph is displayed in a radial layout. Each circle represent a depth increase in the hierarchy of the website. Being websites' hierarchies more complex than a tree, the graph end up having multiple cross-branch links. Therefore the parent of each node is fixed to be the first node to point to that node when iterating over all the nodes with a Breath First Search algorithm[12]. The distance between each circle is calculated singularly for every sector of it, where with sector we indicates all the children of the same parent on that level. The distance depends on the number of children, in order to maintain them at a minimum distance in between them. The distance cannot be less that 100 pixels or more than 500 pixels though, otherwise the nodes will be placed too far away to be seen or too close to the previous level.

There are two information box in the interface, one dynamic that is always visible and one static that can be hidden.

- **The status bar** Placed at the bottom of the screen, the status bar is always visible and contains information about the last hovered node. The information displayed depends on which metrics the user is currently displaying. It also contains the complete URL of that node and its title.
- **The information box** Placed at the top right corner, the box is shown when a node is clicked. It contains all the information of then node and can be hidden at any time. It also contains the relative URL of that node and its title.



Node:	Michele Lanza > Teaching
Url:	/teaching.html
Internal Links:	10
External Links:	21
Faulty Links:	4
Files:	3

Figure A.3. An example of the information box displaying information about a webpage

Figure A.3 show an example of the information box with the various information contained.

On the top left corner we can find the option menu. This is a collapsable box that allows the user to modify the way the graph is displayed. The main feature is the possibility to set the dimension of all node to the value of a specific metric. The current version of the tool calculate four different metrics from the information gathered from the graph and allow the use to set them on the width, height or color intensity of each node. The values are scaled between a minimum and a maximum value in order to provide a better visualization of the difference.

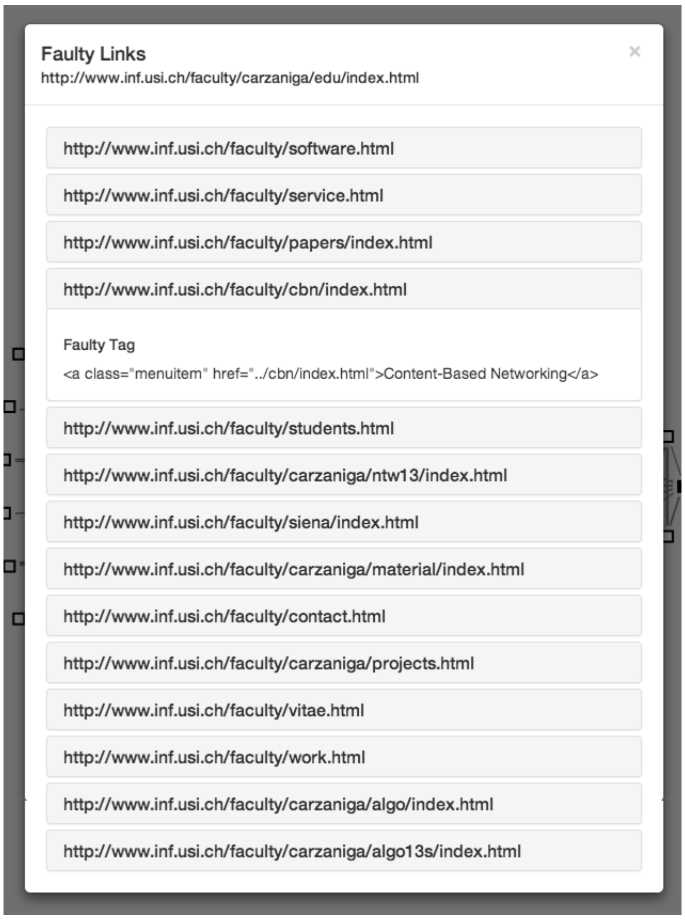
Another option provided is the possibility to keep all the labels of the nodes visible all the time. In the normal state, labels are drawn only when the node is hovered, in order not to fill the visualization with too much text. But

[12]http://en.wikipedia.org/wiki/Breadth-first_search

in case the user desires it, is it possible to display all the labels all the time. Labels are drawn few pixels above the nodes and are automatically scaled with the rest of the graph.

A similar option is also present for the visibility of edges. Normally, all edges are visible all the time. But in case the graph contains hundreds of nodes, the edges might become too many to be displayed, rendering the graph hard to understand. If that wasn't enough, a larger number of edges can negatively effects the performances of the visualization. Therefore there is the possibility to hide all the edges temporarily in order to interact more easily with the graph.

The last option allows the user to switch between the *select mode* and the *drag mode*. Drag mode is the default mode. While nodes can be dragged around the graph at any time, the graph itself can be moved only when the user is in drag mode. Select mode disable the possibility to drag the graph around but introduce the use of selector. By dragging an area on the graph is it possible to select all the nodes inside that area and execute some operation on all those nodes. Currently the only operation available is to delete those nodes from the graph.



Faulty Links
http://www.inf.usi.ch/faculty/carzaniga/edu/index.html

http://www.inf.usi.ch/faculty/software.html
http://www.inf.usi.ch/faculty/service.html
http://www.inf.usi.ch/faculty/papers/index.html
http://www.inf.usi.ch/faculty/cbn/index.html

Faulty Tag
<a class="menuitem" href="../cbn/index.html">Content-Based Networking</a>

http://www.inf.usi.ch/faculty/students.html
http://www.inf.usi.ch/faculty/carzaniga/ntw13/index.html
http://www.inf.usi.ch/faculty/siena/index.html
http://www.inf.usi.ch/faculty/carzaniga/material/index.html
http://www.inf.usi.ch/faculty/contact.html
http://www.inf.usi.ch/faculty/carzaniga/projects.html
http://www.inf.usi.ch/faculty/vitae.html
http://www.inf.usi.ch/faculty/work.html
http://www.inf.usi.ch/faculty/carzaniga/algo/index.html
http://www.inf.usi.ch/faculty/carzaniga/algo13s/index.html

Figure A.4. An example of a modal window containing the broken link of a webpage

Finally, right clicking any node open a context menu that allows to execute actions on that node or obtain additional information. Right now, same as with the selector, it is possible to delete the node. Additionally is it possible to obtain a list of the files pointed from the webpage and a list of the faulty links inside the webpage. When one of those list is requested, the client submit a query to the server in order to obtain the information and then create a modal window where the information will be displayed. For the files, is it possible to see the name and the size of each file, while for the broken links the link itself and the HTML tag containing it. Figure A.4 shows an example of the modal for the broken links.

Bibliography

[1] S. Ducasse, M. Lanza, and R. Bertuli. High-level polymetric views of condensed run-time information. In *Proceedings of CSMR 2004 (8th European Conference on Software Maintenance and Reengineering)*, pages 309–318. IEEE CS Press, 2004.