YACA and YASM: Command-Line Tools for Web Site Content Auditing and Site Mapping

Group 2

Reinhard Egger, Daniel Geiger, Lorenz Leitner and Peter Oberrauner

706.041 Information Architecture and Web Usability WS 2019/2020
Graz University of Technology

03 Feb 2020

Abstract

Websites are made up of individual pages which are interlinked to each other. When studying the content and structure of a website, the pages can be analyzed by creating a content audit and a (visual) site map. A content audit tries to list all available content of a website, mainly all existing pages, but additional information can be listed as well. A site map represents the hierarchical structure of the pages via parent-child relationships. A tree structure of this hierarchy can be visualized using a graph model. Existing paid proprietary solutions for these purposes have been shown to be inadequate especially for their price. This report shows two tools which offer this functionality free and open-source: Yet Another Content Auditor (YACA) and Yet Another Site Mapper (YASM). The development process is described and the usage of the tools is documented. An insight into past issues and potentials of future work is given as well.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Most websites consist of many individual pages, which are linked to each other in addition to containing links to the outside of the domain. The pages within a domain, that is the pages that belong to a website, are a point of interest when studying a website's content and structure. For this purpose, tools exist to generate content audits and (visual) site maps of websites. A content audit attempts to list all available content of a website, which is in its most basic form a list of all pages. Additional information can be stored as well, such as page titles, content types, et cetera. A site map shows the hierarchical structure of a website, in specific the parent-child relations of pages. A parent page links and therefore points to a child page. A child page is therefore one level deeper in the hierarchy than its parent page. Such a hierarchy can be expressed in a tree structure, which lends itself well to being represented visually as graphs or diagrams.

1.0.1 Existing Solutions

Several existing tools for either or both of these purposes exist, which have been listed in the survey paper of Ganster et al. [2019]. It is shown that most of these existing solutions are paid proprietary services and aimed at corporate use, in contrast to personal use. Even though these tools are paid, some do not offer satisfactory results in some ways.

1.0.2 New Solution

Ganster et al. [2019] provide a proof of concept solution for generating a content audit via Scrapy [Scrapinghub 2020], a free and open-source tool to crawl and download web pages. This piece of code was used to implement a more sophisticated solution for generating content audits via Scrapy, called Yet Another Content Auditor (YACA). From the results of YACA, a visual site map can be generated via the second tool, called Yet Another Site Mapper (YASM). Both of these tools are described and documented in this project report, as well as the process of developing the tools and potential future work.

1.0.3 Availability

Both YACA and YASM are free and open-source, available on GitHub. YACA can be found at `https://github.com/LoLei/yaca` and YASM can be found at `https://github.com/LoLei/yasm`. YACA is licensed under the MIT license [MIT] and YASM is licensed as CC BY-SA 4.0 [CC BY-SA 4.0].

Chapter 2

Development

The development of YACA and YASM took place over the course of about six weeks for the class Information Architecture and Web Usability (706.041) at Graz University of Technology in 2019/2020. The team of developers consisted of four people which tried to split the work between them, using the version control system Git as a collaboration tool. The following sections show the development process, issues which were encountered and lessons that were learned over the course of development.

2.1 Development Process

An initial proof of concept of Scrapy's content auditing capabilities was obtained from Ganster et al. [2019] at the beginning of development, as a starting point from which to develop further functionality. This piece of code created a content audit from `oebb.at` in form of a list of URLs in a CSV file.

The first step was to generalize this and remove the hardcoded URL in favor of a dynamic input, which also meant wrapping the pure Scrapy project with a Python wrapper. Through this Python wrapper, a dynamic input domain can be supplied as well as every other options which were developed in the future. This wrapper is what was to become YACA. Over the course of development, all code which stemmed from the proof of concept has been removed or rewritten. However, a lot of boilerplate code that is auto-generated from Scrapy for its project structure is of course the same.

It was quickly seen that the actual link wiring of pages on a website cannot be taken as-is to generate a page hierarchy, which is needed to generate a site map. Consider a menu present on top of all pages of a website, which is often the case. Since all pages have links to the pages in this menu, theoretically the menu pages would always be children of every page. This and other link wiring issues led to using only an approximation of the page hierarchy in the content audit via parent URLs and child URLs, and using a different method of generating an accurate page hierarchy in YASM itself.

Having a list of all existing URLs via YACA was actually enough as initial input for YASM to generate a site map. YASM goes through these URLs and generates a tree structure hierarchy based on the slashes in the URLs. For example, `domain.com/p1/p2` and `domain.com/p1/p3` means `domain.com` points to `p1` and `p1` points to `p2` and `p3`. This tree structure in itself is a site map, and is exported as JSON. However, to get a human readable visual representation, this tree structure needed to be converted to the Graphviz textual format, which can then be used by Graphviz to generate a visual graph or site map as PDF and SVG consisting of the parts of the URLs as nodes. Fortunately there is a Graphviz Python library that makes it easier to use the tree structure instead of manually writing text to a file via Python.

2.2 Issues

During the process of developing these tools, a few issues occurred, as it is known to happen. Some were minor and easily fixed, therefore not worth mentioning here, but the larger ones might be interesting to know about.

2.2.1 Edge Bundling

One of the biggest issue when using Graphviz with orthogonal splines, that is edge connections between nodes with right angle corners, is that the automatic bundling of edges that run along each other is broken, and has been for a long time. It is possible with regular edges, through the option `concentrate=true` but this does not work for orthogonal splines, which has been reported since at least 2012 [Warner 2012]. So if it were easy to fix, someone would have already done it. Another option would be to use the Mingle Algorithm to use with Graphviz as done by Fröhlich et al. [2017], but this is also untested with orthogonal splines.

The most straightforward solution was just to offer an alternative to Graphviz, which can be used in cases Graphviz's output gets too messy. The alternative that was found is Blockdiag, which inherently uses orthogonal edges and bundles them as well. This leads to a much cleaner looking site map in most cases. However, there is no official Python library available for this, but since it was written in Python, it can simply be used as a module, and specific functions needed to generate the site map can be called. Since Blockdiag uses a very similar textual input format to Graphviz the way this was handled was to just generate the Graphviz textual format via Graphviz's Python library, and transform this with a few changes to one that can be used by Blockdiag.

Discovered later in the development process, a way to mitigate this in Graphviz is to group nodes from a section of a site together, which places them in a close proximity on the visual map, which prevent many overlapping edges, even if they are still running along in parallel.

2.2.2 Depth and Width

The depth of a website is the maximum number of links deep one can hop until the end is reached, that is there are no more links, or there are only links left pointing to an upper level page. The depth can be limited in both YACA and YASM via the `-d` flag. If a YACA run is done only to use the result as input for YASM, the depth should be limited, to not scrape an unnecessary number of unneeded pages. When using YASM, the depth should also be limited because displaying a large number of pages in a visual site map may be impossible, at least to look at without zooming far out.

The width of a website is the maximum number of pages in one level. Limiting this number does not make much sense as it would cut off arbitrary pages on one "side" of a website, even though they are equally important as the other nodes on that level. Deeper nodes tend to have a lesser importance than upper level nodes. Therefore limiting the depth in this way is more intuitive. Since the scraping in YACA is also done breadth-first this goes hand in hand. However, some websites may still have a large width, which makes viewing the visual output equally as difficult as having a large depth. In most cases, limiting the depth also limits the width somewhat, since higher levels tend to have fewer pages. Otherwise, to mitigate large widths the orientation of the YASM output can be changed via the `--orientation` flag. This can make it easier to zoom in and scroll through nodes like a simple list in a PDF, from top to bottom. To generally save space, the padding in between nodes can be altered and reduced as well, by using the `-wp` and `-hp` flags, so as many nodes as possible can fit onto the visual output format. A visual sitemap generated with these options can be seen in Figure 2.1.

A few additional mitigations were proposed, such as stacking pages on top of each other, or changing directions after one or two levels, so the upper level nodes points down to a string of nodes which are actually on the same level. However, doing this dynamically via Graphviz was deemed out of scope for a project of this scale. Also, once one starts stacking pages and doing similar workarounds to have a

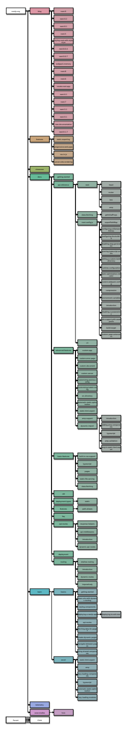


Figure 2.1: YASM output of the YACA content audit of `nextjs.org`, a website with about 100 pages. [Generated with `./yasm.py ../yaca/output/nextjs-org.json -e blockdiag --orientation landscape -wp 30 -hp 10`]

prettier graph, the actual structure of the website gets lost, while every output seems to be transformed into a generic balanced tree. So this generally is a trade-off between aesthetically pleasing output and actual representation of a website's structure.

Chapter 3

Documentation

YACA and YASM both have multiple features and options beyond their basic capability, which can be specified as input arguments when starting a tool. The following technical documentation sections shall list all these and describe them where further explanation is required. Also worth noting are the prerequisite requirements that need to be installed before the tools can be used, as well as which platforms are supported.

3.1 Supported Platforms

YACA and YASM were designed to run across all major operating systems, as only OS-agnostic Python functions were used when dealing with matters such as the file system. In addition to that, the development of both tools was distributed across multiple machines and operating systems as well.

Therefore, the supported operating systems are Linux, Windows and MacOS. These are the ones which were tested at least. It might very well be the case that the tools also work on BSD variants and whatnot.

3.2 Prerequisites

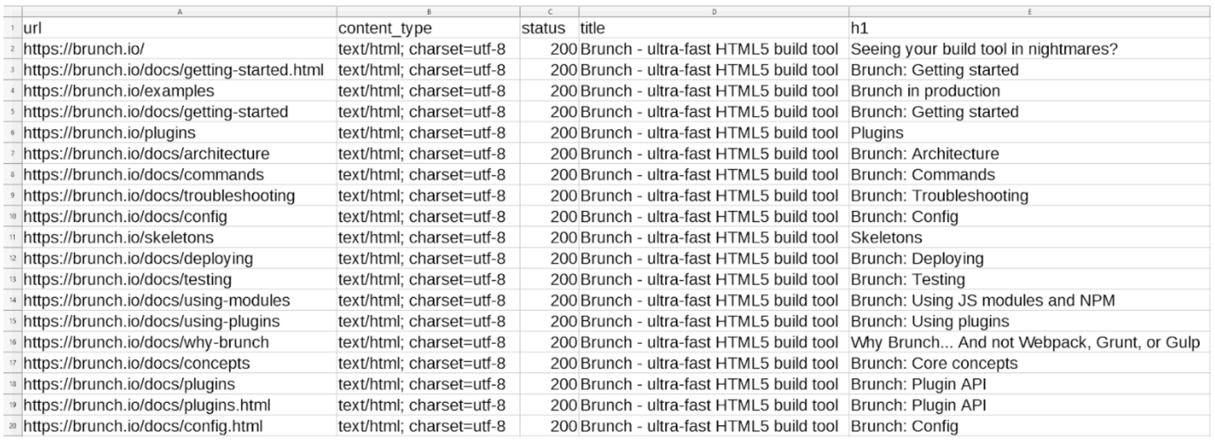
The following lists dependencies that need to be installed on a machine in order to run YACA and YASM:

- Python 3.x [Python Software Foundation 2020] (Tested with 3.7 and 3.8)
- Scrapy [Scrapinghub 2020]
- Graphviz [AT&T Labs Research 2020]
- Blockdiag [Komiya 2020] (With PDF support)
- Python Colorspace [Stauffer 2020]

These programs are all open-source and under permissible licenses to use. Installing them for usage with Python is in most cases a simple installation via pip, for example `pip install graphviz`, for Graphviz. However, Blockdiag requires a special version for PDF support, therefore its installation must be done with `pip install "blockdiag[pdf]"`. In some cases, Python Colorspace's pip installation doesn't work and requires cloning its repository and running `python setup.py install` from within.

3.3 Usage

The basic usage of both tools is to either execute the Python file with Python, or let your shell choose the interpreter, since both files were given the executable permission. One additional argument is required, the domain and the input file, for YACA and YASM respectively. Further usage differs according to which tool is being used.



url	content_type	status	title	h1
https://brunch.io/	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Seeing your build tool in nightmares?
https://brunch.io/docs/getting-started.html	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Getting started
https://brunch.io/examples	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch in production
https://brunch.io/docs/getting-started	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Getting started
https://brunch.io/plugins	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Plugins
https://brunch.io/docs/architecture	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Architecture
https://brunch.io/docs/commands	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Commands
https://brunch.io/docs/troubleshooting	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Troubleshooting
https://brunch.io/docs/config	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Config
https://brunch.io/skeletons	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Skeletons
https://brunch.io/docs/deploying	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Deploying
https://brunch.io/docs/testing	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Testing
https://brunch.io/docs/using-modules	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Using JS modules and NPM
https://brunch.io/docs/using-plugins	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Using plugins
https://brunch.io/docs/why-brunch	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Why Brunch... And not Webpack, Grunt, or Gulp
https://brunch.io/docs/concepts	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Core concepts
https://brunch.io/docs/plugins	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Plugin API
https://brunch.io/docs/plugins.html	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Plugin API
https://brunch.io/docs/config.html	text/html; charset=utf-8	200	Brunch - ultra-fast HTML5 build tool	Brunch: Config

Figure 3.1: CSV output of YACA of the `brunch.io` website. [Generated with `./yaca.py brunch.io`]

```
$ ./yaca.py -h
usage: yaca.py [-h] [-d DEPTH] [-dl DELAY] [-c] [-ah] [-p] [-sd] [-ct
    CONTENTTYPE] domain

positional arguments:
  domain                Domain/URL - Starting point. <example>.<tld>

optional arguments:
  -h, --help            show this help message and exit
  -d DEPTH, --depth DEPTH
                        Maximum depth of the crawl (default 0=unlimited)
  -dl DELAY, --delay DELAY
                        Delay between individual page downloads in seconds (
                            float supported)
  -c, --children        Include links within the domain in JSON output
  -ah, --ahrefs         Include links outside the domain in JSON output
  -p, --parent          Include parent in CSV output
  -sd, --subdomains     Include subdomains
  -ct CONTENTTYPE, --contenttype CONTENTTYPE
                        Content-Type of pages that should be crawled (default =
                            only text/html)
```

Listing 3.1: The help text shown when supplying the `-h` flag to YACA.

3.3.1 YACA

YACA takes an input domain and produces a content audit from this root starting domain, producing it as CSV and JSON output files. The CSV output of YACA can be seen in Figure 3.1. All usage parameters can be see in Listing 3.1. This help text can be generated by supplying the `-h` flag to YACA.

3.3.2 YASM

YASM takes YACA JSON as input, that is the output of a YACA run, as described above. From this, it generates a (visual) sitemap, as PDF, SVG, JSON and GV, which is the textual Graphviz format. The output engine can be changed from Graphviz to Blockdiag, as it tends to produce better formatted visual sitemaps in some cases, but the choice is left up to the user. All usage parameters can be seen in Listing

```
$ ./yasm.py -h
usage: yasm.py [-h] [-v] [-i] [-d DEPTH] [-wp WIDTHPADDING] [-hp HEIGHTPADDING]
               [-e {dot,blockdiag}] [-o {landscape,portrait}] [-t {pdf,svg}]
               [-s] file

positional arguments:
  file                  Input file - YACA JSON

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Verbosity of the output
  -i, --instant         View the output file immediately
  -d DEPTH, --depth DEPTH
                        Maximum depth of the sitemap (default unlimited)
  -wp WIDTHPADDING, --widthpadding WIDTHPADDING
                        Padding width between adjacent nodes
  -hp HEIGHTPADDING, --heightpadding HEIGHTPADDING
                        Padding height between adjacent nodes
  -e {dot,blockdiag}, --engine {dot,blockdiag}
                        Engine for output
  -o {landscape,portrait}, --orientation {landscape,portrait}
                        Orientation for blockdiag output
  -t {pdf,svg}, --type {pdf,svg}
                        Output file type
  -s, --sdsp            Treat subdomains same as slash path URL parts
```

Listing 3.2: The help text shown when supplying the -h flag to YASM.

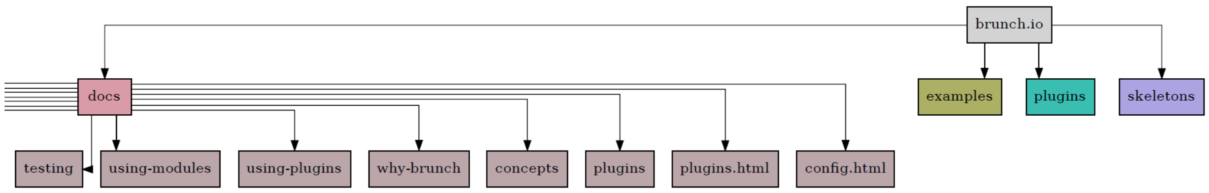


Figure 3.2: Graphviz output of `brunch.io` website. [Generated with `./yasm.py ../yaca/output/brunch.io`]

3.2, which can be produced by supplying the `-h` flag as well.

3.4 Graphviz

Graphviz is a popular package which consists of various open source tools that can generate or process DOT files. DOT is a graph description language. Defining such a graph in DOT is very simple. An example of an output of Graphviz can be seen in Figure 3.2.

3.5 Blockdiag

Blockdiag can produce in some cases a much better looking visual site map instead of Graphviz as can be seen in Figure 3.3. Mainly due to that edge bundling is working in Blockdiag. Blockdiag's textual

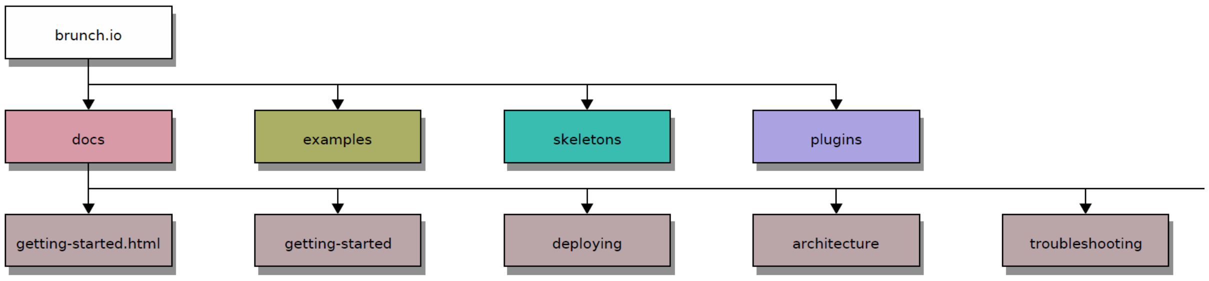


Figure 3.3: Blockdiag output of `brunch.io` website. [Generated with `./yasm.py ../yaca/output/brunch.io`]

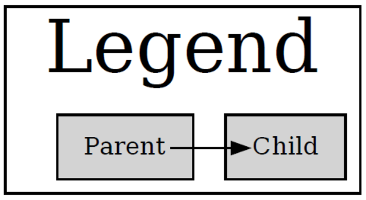


Figure 3.4: Legend of a site map of the engine Graphviz

format is very similar to that of Graphviz's textual format called DOT, therefore only slight changes are necessary for Blockdiag to work.

3.6 Node Colors

There exist many different color spaces, most known is the RGB color space, but for a sitemap a color space is needed, where the colors are easily distinguishable. The HCL color space is based on the human color perception and can be controlled by the hue (color tone), chroma (colorfulness) and luminance (lightness). This makes it much easier to dynamically set a color palette for the 1st level nodes of the sitemap, and reducing the chroma with each level. To use the HCL color space in python, Stauffer 2020 is used.

3.7 Legend

For both Graphviz and Blockdiag a legend is added to the visual sitemaps, thus explaining how the sitemap is structured. A node which is a parent,that is connected to their child node with an arrow can be see in Figure 3.4 for Graphviz and in Figure 3.5 for Blockdiag.

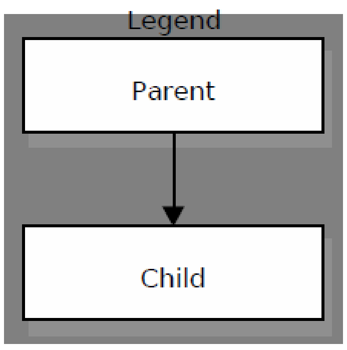


Figure 3.5: Legend of a site map of the engine Blockdiag.

Chapter 4

Future Work

Some of the issues mentioned in Chapter 2 could be worked on if additional development on YACA and YASM were to be done. The edge bundling issue might be solved in a more sophisticated manner, even if the current mitigations don't necessarily require it, as well as the problem with large websites, for example with a large width.

Another interesting aspect is the extensibility and further usage of the results that YACA and YASM provide. YACA provides two output formats, CSV and JSON. The results in these files, which is a list of all URLs with some additional information like the page title, the content type, et cetera, could very easily be used by another program, as it is done when using it with YASM. The outputs of YASM can also be used by other programs, since not only the visual site map in PDF and SVG is produced, but it is also generated as a JSON tree hierarchy and textual Graphviz output (GV/DOT). This JSON tree hierarchy can be for example imported into D3.js [Bostock 2020], a visualization engine using JavaScript. This has been tested insofar that it can be imported and read by D3.js using `d3.json`, however already existing solutions which map a JSON hierarchy into a visual D3.js output such as `d3.hierarchy` require additional rewriting of the YASM JSON format, in specific adding `name` and `children` keys everywhere.

What also could be improved is the content of nodes in the visual sitemaps created with YASM. Currently the only content in the nodes is the part of the URL identifying the node but it would be possible to, for example, display the titles of pages or any other information that could be crawled with YACA. This would, of course, blow up the sizes of individual nodes which might make the sitemap harder to reason about but it could be a valuable configuration option for some users.

Chapter 5

Concluding Remarks

Content audits and visual sitemaps are useful tools when analyzing the content and structure of websites. Due to the poor quality and high costs of existing commercial tools, a reliable solution based on open source technologies could prove beneficial for many users. During this project, two tools have been implemented that can be used to create comprehensive content audits as well as customizable, intelligible visual sitemaps. Both tools have been implemented using exclusively open source frameworks and the results have shown to be of, at least, comparable quality as those of commercial tools.

Bibliography

AT&T Labs Research [2020]. *Graphviz*. `https://www.graphviz.org/` (cited on page 7).

Attribution-ShareAlike 4.0 International [2020]. Creative Commons, 02 Feb 2020. `https://creativecommons.org/licenses/by-sa/4.0/` (cited on page 1).

Bostock, Mike [2020]. *D3.js*. `https://d3js.org/` (cited on page 13).

Fröhlich, David, Markus Kammerhofer, Samuel Kogler, and Stephan Stiboller [2017]. *Edge Bundling*. Technical report. Graz University of Technology, 2017. `https://courses.isds.tugraz.at/ivis/surveys/ss2017/ivis-ss2017-g4-survey-edge-bundling.pdf` (cited on page 4).

Ganster, Paul, Peter Grassberger, Magdalena Mayerhofer, and Ana Lopez Camarero [2019]. *Sitemap Generators*. Technical report. Graz University of Technology, 2019 (cited on pages 1, 3).

Komiya, Takeshi [2020]. *Blockdiag*. `http://blockdiag.com/` (cited on page 7).

Python Software Foundation [2020]. *Python*. `https://www.python.org/` (cited on page 7).

Scrapinghub [2020]. *Scrapy*. `https://scrapy.org/` (cited on pages 1, 7).

Stauffer, Reto [2020]. *Python Colorspace*. `https://python-colorspace.readthedocs.io` (cited on pages 7, 10).

The MIT License [2020]. Massachusetts Institute of Technology, 02 Feb 2020. `https://opensource.org/licenses/MIT` (cited on page 1).

Warner, Kip [2012]. *Problem With Ortho Splines And Concentrate*. 2012. `http://graphviz.996277.n3.nabble.com/Problem-with-ortho-splines-and-concentrate-td165.html` (cited on page 4).