

GSoC 2019

CC Catalog Visualization

Student: Maria Belen Guaranda

Mentors: Sophine Clachar and Breno Ferreira

Supporting Documents: [GSoC Proposal](#) and [Assessment](#)



Project Overview

There is no larger compendium of shared human knowledge and creativity than the Commons, including over 1.4 billion digital works available under CC tools. Creative Commons has released the “CC Search” project. Being able to access visualizations of all the indexed content is a good way for the community (and CC) to see how much data has been indexed and find and explore relationships between CC-licensed content on the web. The challenge is then to create visualizations of all the data that is stored in the Creative Commons catalog (over 250 million works and growing) and show how they link to each other.

Tech Stack: Python, JavaScript, [force-graph](#), [Highcharts](#)

Github Repo: <https://github.com/creativecommons/cccatalog-dataviz>

Initial Research

The project proposal contemplated the visualization of a traditional node-link diagram. Nevertheless, due to the huge amount of data to be managed, this diagram tends to become a hairball. This means, it is not easy to see the relationships. There are other types of visualizations and representations of graphs. The question to be answered is then: could we visualize CC-Catalog data in another (more scalable) way? This section reviews two case studies, as the state-of-art of graph visualizations.

Case study 1: visualization of a genealogical graph.

This is a special kind of graph, and its objects become genealogical entities. Genealogical graphs have their own standard data structures, and the data they manage is hierarchical. Its respective analysis tasks are classified in: topology-based, browsing-and-filtering and overview tasks.

Proposed solution: GeneaQuilts is a new visualization technique for representing large genealogies of up to several thousand individuals. It is inspired by the Quilts visualization, which is a matrix-based representation. The positions of nodes in the matrix are defined by the hierarchical order of the entities. Children will be placed at lower rows (or layers) than their parents.

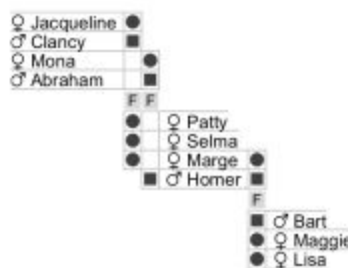


Fig. 5. GeneaQuilts Visualization of the Simpson Family.

Taken from [1].

Pros:

- Eliminate the node-link diagrams very long edges and too many crossings to be suitable for exploration or presentation
- Display layered graphs in a more compact manner
- More scalable visualization solution

© 2006 The Authors

Proposed solution: Papilio, a new visualization for the permissions data of mobile applications, in order to explore and analyze permission usage. Security analysts are the main targeted operators. Papilio is a two-sided visualization: on the right-side, partial order relations between supersets and subsets of e-classes (parent-child relations) are visualized ; on the left side, permissions are shown as e-class attributes.

Normal visualization:

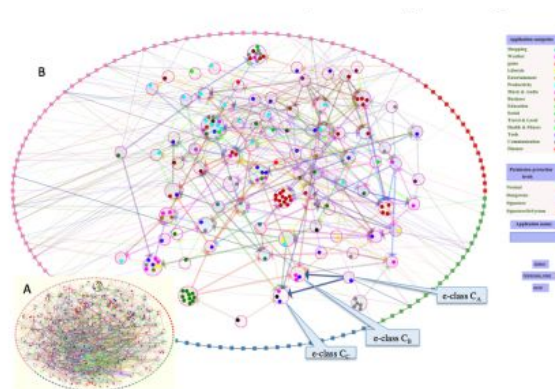


Figure 2: Applying force-directed layout on our dataset. Inset A: the force-directed layout on the entire dataset. B: the layout of one-third of applications in our dataset (reducing the applications gives a better view of the layout).

Taken from [2].

With Papilio:

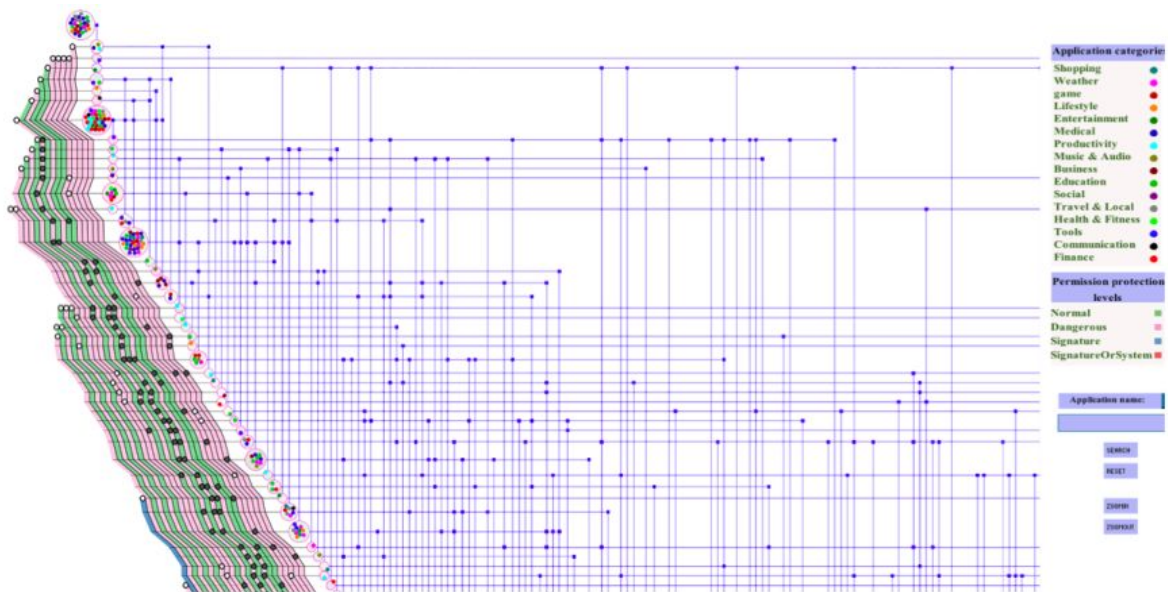


Figure 4: *Papilio* is a two-sided visualization showing ordered application e-classes through the center, with the parent-child relations among e-classes on the right side and the requested permissions of e-classes on the left side.

Taken from [2].

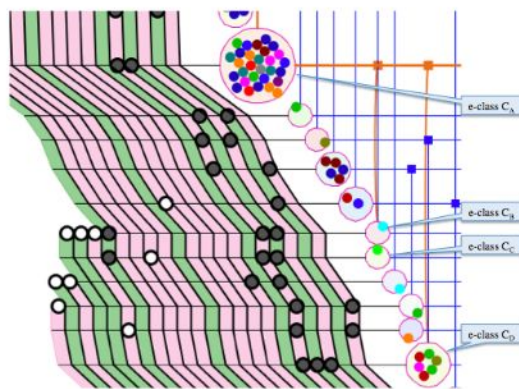


Figure 5: C_A is a parent e-class for C_B , C_C and C_D . Also C_B and C_C are siblings which are recognizable based on their positions.

Taken from [2].

130 permissions visualization:

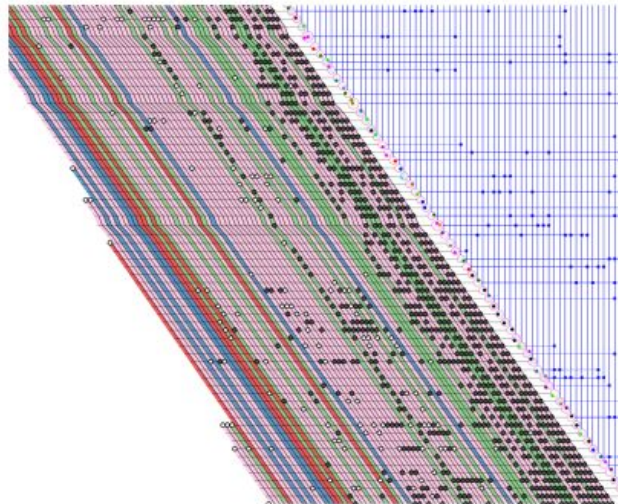


Figure 9: *Out of 130 permissions, just a small set of them are frequently requested by applications and the rest are either never requested or requested infrequently.*

Taken from [2].


Pros:

- Incorporate aspects of set membership, node-link diagrams and matrix layouts.
- New security findings.
- Ability to demonstrate hierarchical structure of data

Discussion and conclusions

Both use cases studied above had as goal to find a scalable visualization. Their proposed solutions nevertheless, were tailored specifically to their data and analysis rather than a general solution. Limitations to these solutions are:

- Data needs to be hierarchical (or have an attribute to use so to order or group the data) .
- Benefits are focused on the analysis tasks that are specific to those use cases.
- Visualizations still fail for thousands of elements (they do not meaningful insights unless you make a zoom).
- We would have to build a library from scratch.



The limitation of the node-link diagram we search to overcome is its low scalability. From this brief review, we conclude that none of the proposed solutions seem viable for the CC-Catalog visualization project, as they do not show huge improvements in scalability. In addition, they do not fit for our case study.

References:

1. Bezerianos, A., Dragicevic, P., Fekete, J. D., Bae, J., & Watson, B. (2010). Geneaquilts: A system for exploring large genealogies. *IEEE Transactions on Visualization and Computer Graphics*, 16(6), 1073-1081.
2. Loorak, M. H., Fong, P. W., & Carpendale, S. (2014, June). Papilio: Visualizing android application permissions. In *Computer Graphics Forum* (Vol. 33, No. 3, pp. 391-400).

Recommendation: Force Directed Graph (FDG). See some examples in the [useful links](#) below.

The Graphical User Interface [Front-End]

The development of the front-end of the application will be using the web standards: HTML5, CSS3 and Javascript. Additionally, Bootstrap will be used as front-end framework to give the application a responsive and good-looking design.

Visualization libraries/languages: Javascript, VivagraphJS, force-graph, Hlghcharts.

Force-Directed Graph (FDG)

Regarding the huge amount of data, it was necessary to make a research and comparison between existing visualization libraries with high performance. I searched for a library that matched with most of the following criteria:

- High performance
- Low learning curve
- High customization (and achieve a nice interface with relatively few lines of code)
- Availability of existing projects (in order to reutilize as much code as possible)

We ended-up with these two libraries:

1. **VivagraphJS** library proves to be one of the most performant in terms of layout calculation. This library is part of the ngraph family, and is an open-source library.
2. **force-graph** library. Uses HTML5 canvas for rendering and d3-force for the underlying physics engine.

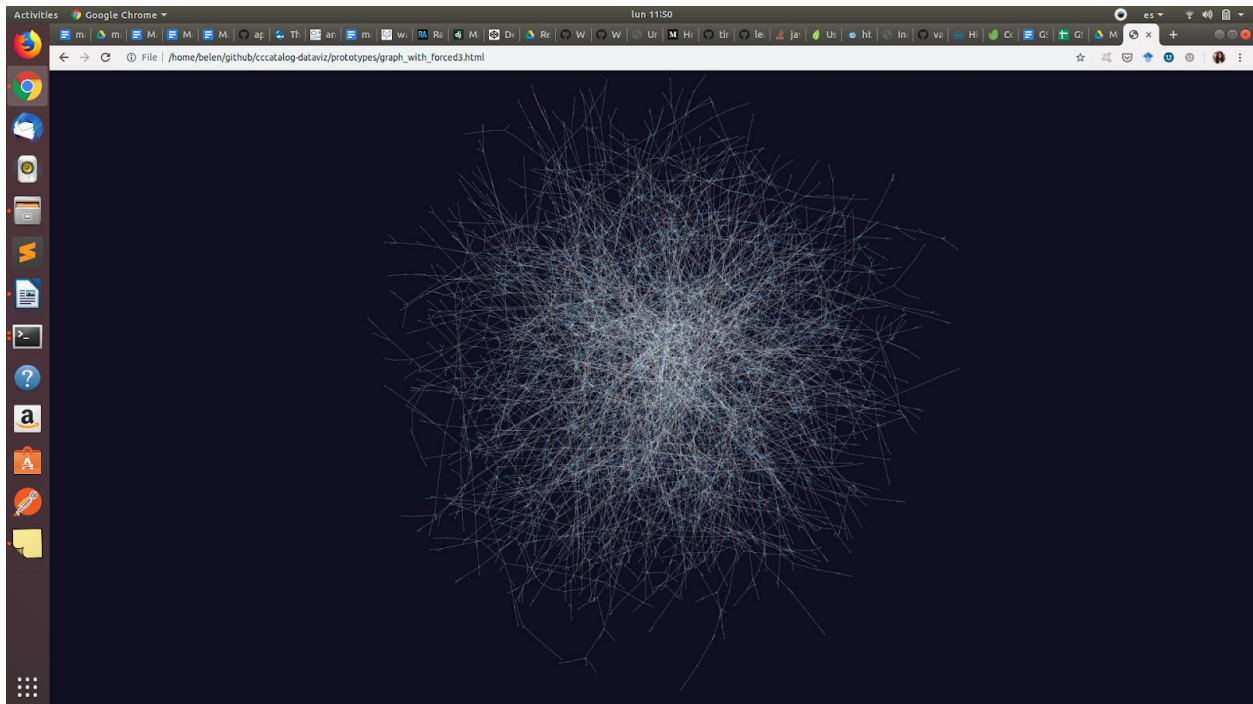
Two prototypes of the fdg were made with each library. A comparison between their rendering time is shown in the next table:

Node quantity vs Rendering time

	VivagraphJS	force-graph
10k nodes	5s	5s
30k nodes	25s	11s
50k nodes	1:30 min	15s
100k nodes	---	40s

Vivagraph does not have a function for creating a graph with a file, in contrast with force-graph. Also with Vivagraph, more code was needed to build a FDG with a simple style. Moreover, there is little documentation of Vivagraph; this library has few tutorials. One example showed that it must be used in complement with other libraries like d3 or vis.js.

Hence, the chosen library for the FDG is **force-graph**.



Graph prototype with force-graph. Browser: Chrome.

force-graph has a simple but powerful API for customization. There are general properties (this is, for the entire graph), and then there are properties for nodes and edges customization. I am going to review what was made for each of these parts.

Graph building

For building the graph there are two options:

1. Creating the nodes and edges programmatically in a script, when the HTML is generated.
2. Reading from a JSON file

In the proposal I contemplated the building of the graph following a json structure for the nodes and links. Hence I decided on the second option. The array structure that force-graph needs is:

```
{
```

```

"nodes": [
  {"id": "Domain1", "other_feature": 1, ...},
  {"id": "Domain2", "other_feature": 1, ...},
  ...
],
"links": [
  {"source": "Domain1", "target": "Domain2", "value": 6},
  {...},
  ...
]
}

```

Which is similar to the structure in the proposal. The nodes are in an array of python dictionaries. Each dictionary needs to have an *id* key; then any other features can be stored. The links array needs to have *source* and *target* keys, and the *value* as the link weight. The reading and data parsing is done with the *fetch* function. For building the graph, we instantiate a *ForceGraph()* object, followed by a series of attributes to customize all the parts of the graph.

Properties:

width(px): set the width (in pixels) of the canvas that stores the graph.

height(px): set the height (in pixels) of the canvas that stores the graph.

*If we do not set these properties, the canvas will occupy the entire window.

backgroundColor(color): the color of the background; we can pass a color in any format (hex or rgb). The current color is: #07263b.

zoom(number, duration): initial zoom; we can pass a second argument that dictates the time at which the canvas gets to the desired zooming. force-graph comes by default with a zoom range of [0.001, 1000], which can literally make the graph disappear and makes a dizzy experience for the user. This is the reason I researched how to edit this behaviour in the default settings.

Source_file : /prototypes/js/force-graph.min.js

Search for the term *scaleExtent*. In the *second* match, we can edit this range:

```
) {return!!e.enableZoomPanInteraction&&!R.button}).scaleExtent([0.1,25]).on("zoom",function(){var t=Pn(this);[r,i]
```

Editing the file force-graph.min.js using Sublime Text.

The values I left for now are the ones that appear in the image above.

graphData(json): function that receives the data and builds the graph.

Note: the property *enablePointerInteraction* needs to be true or nodeHover and nodeClick won't work

Nodes styling

nodeLabel(str): displays the str when hovering over a node. Here I set the domain name of the node to be displayed.

onNodeClick(function): function is triggered when the user clicks over a node. Here I pass the *drawPieChart(node)* function, which displays the pie chart inside a modal.

onNodeHover(function or expression): here I set that the cursor changes to a hand appearance, so the user knows a node is clickable. Also, I call a function to highlight the neighbors of a node, called *traverseHighlight()* (this function is implemented in the script part of the index.html, and it was taken from an example made by the author of the library, [link 10.d](#)).

nodeCanvasObjectMode(node => 'mode'): in order to draw the nodes shape in a custom way, I need to set this property first. There are 3 modes : *replace*, *before* and *after*. I set the mode to *'replace'*, which means the nodes will be rendered using just nodeCanvasObject. Here I set the size of the nodes (since this is where the circles are drawn), and draw the labels of the nodes (the domain names). The node size is taken from the *node_size* field.

nodeCanvasObject((node, ctx, globalScale), function/expression): we can receive the node and its canvas element (ctx), and draw the node as desired.

Edges styling

onLinkHover: accessor for styling the edges when the user hovers over them. Here I change the color of the edges in order to highlight them.

linkWidth(number or expression): set the width of the edges; currently, I increase the width to 2 when the user hovers over an edge or when highlighting the neighbors of a node; otherwise it's 1.

linkColor(str or expression): accessor to set the color of the edges.

linkDirectionalArrowLength(number): link object accessor function for the length (in px) of the arrow head indicating the link directionality. The arrow is displayed directly over the link line, and points in the direction of source > target.

Current palette:

background: #07263b

nodes: #49fffd stroke: #004c4b

links: rgb(155, 216, 240) on_hover: 'white'

font: #6ab9c6

Pie chart

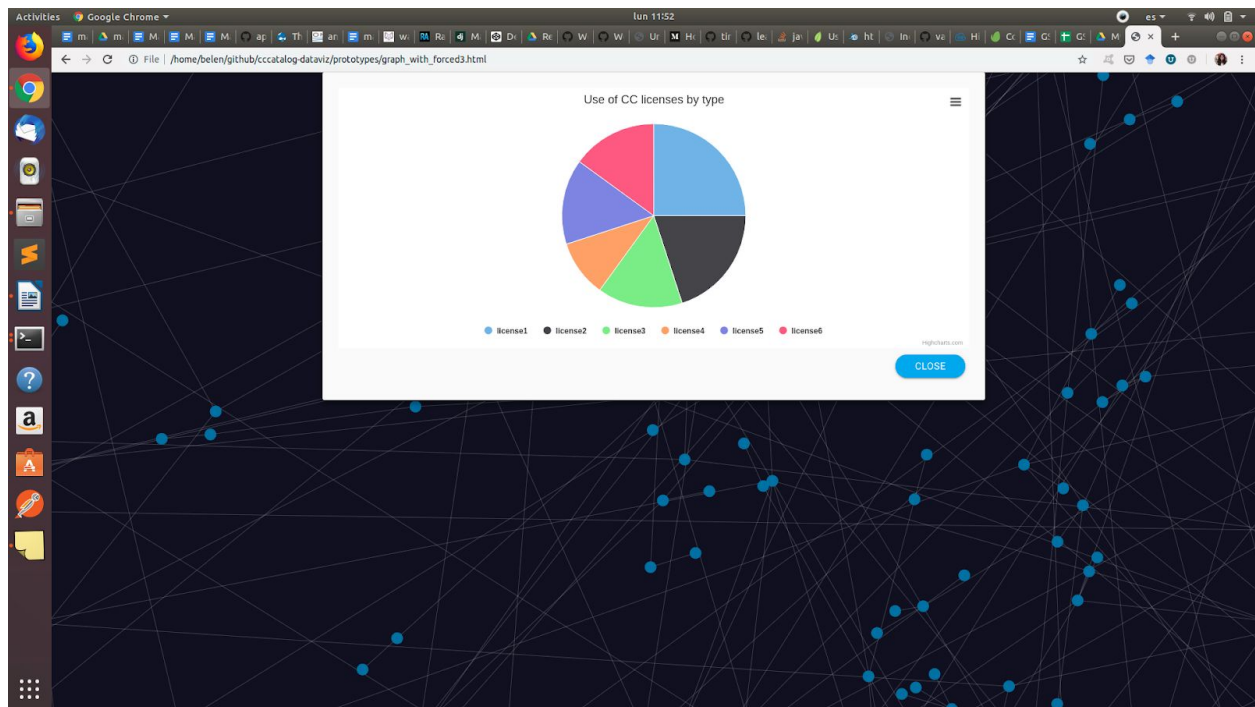
The idea is to display the pie chart inside a modal. The modal will be triggered and shown after the user clicks on a node in the graph.

The features that make Highcharts a good choice are:

- Neat documentation.
- Options for downloading the graphics.
- Open-source
- Data structure similar to the ones that will be managed with the graph.

When designing the modal, its HTML tag needs to be defined after the FDG graph div in order to be displayed, otherwise it stays behind the graph. In addition, a feature should be implemented

to erase the contents of the modal prior to displaying the chart; otherwise the content will be appended, and information from past nodes will be also displayed.



Prototype of the pie chart with Highcharts. Browser: Chrome.

Data Review

I first analyzed a sample data file with 8MB of data, in CSV format. The dataset contains the following columns:

- `provider_domain`: URL of the domain with licensed content.
- `cc_license`: the path to the license deed used by `provider_domain`
- `Images`: number of images in the URL of the `provider_domain`
- `links`: dictionary that contains domains as keys and the values are the number of times the `provider_domain` has referenced the domain.

Observations:

1. `cc_license` is not well referenced by some domains.
2. There are cases where the domains reference certain license, but in their license description they write about another license type.
3. Including `creativecommons.org` as a node in the graph will instantly turn it into the most important node, as all the domains reference it. This can make the graph look messy, and other influential nodes could be overshadowed. Therefore, we will not include `creativecommons.org` as a node in the final graph.
4. Most of the domains contain a few dozen or less of images

For the development of the source code, we are going to have a `utils` module, and a main file.

These files will be inside the `http` folder, in the AWS virtual machine. In the repository, they will be under the `GSOC2019/src/data_processing` folder.

`utils.py`: it will contain helper functions for data processing and transformation.

`data_parser.py`: contains the data processing steps. This file contains the main function that takes as input the tsv files, and outputs the final json file in the format required by force-graph.

Step 1: cleansing and filtering

In this step, I develop rules to exclude data that does not conform to a certain standard. These rules also help to make the dataset smaller and have a legible and accurate final visualization of the data.

1. Exclude rows that contain “`creativecommons.org`” as `provider_domain`
2. Exclude circular links (e.g. provider domains that link to itself)

3. Exclude entries with a non-valid links dictionary
4. Exclude rows with no licences
5. Only take the top 10 outgoing links from each provider

There are validations that we must make when reading the files, regarding the data format, amount of fields, etc.

1. Rows with an invalid json in the *links* column. The following example was found in the file CC_MAIN_2018_51.tsv:

```
{
  "www.blablacar.com": 1,
  "hopper.com": 1,
  "www.travelfreedompodcast.com": 1,
  "www.responsibletraveltourism.com": 1,
  "globalhelpswap.com": 1,
  "www.airbnb.com": 2,
  "www.secretflying.com": 1,
  "www.lyft.com": 1,
  "candicedoestheworld.com": 1,
  "www.suitcasescholar.com": 1,
  "How to Travel on a Budget If you're thinking about heading out and spending a large chunk of your money, there are resources to keep you traveling longer and options while traveling to stretch the budget. Teaching Abroad Many opt to teach English and travel slower, spending time in one country for an extended length (as long as that lovely visa allows) and then heading on to the next destination. In some countries, a TEFL or CELTA isn't even necessary, and a person can make a decent income simply offering private conversational English. Often times, if someone is taking a course in a foreign country, placement after completing the program also comes with the package. Being a Digital Nomad There is also the digital nomad lifestyle, which allows a person to work remotely and offer a service no matter where in the world they are. It is important to keep in mind that this lifestyle requires a lot of hard work, and isn't something which only takes up a little time, allowing the rest of the time to spend traveling. Unless a person has a solid foundation and business set up before they travel, working as a digital nomad can take years before a real profit is made, and that means living in countries where the cost of living is relatively less than the home country (Thailand is a great example of this, although lately longer term visas have been harder to obtain). Short-Term Work": 1,
  "www.agoda.com": 1,
  "theygetaround.com": 1,
  "atuktuk.com": 1,
  "www.saveelephant.org": 1,
  "plus.google.com": 2,
  "findingthefreedom.com": 1,
  "twitter.com": 4,
  "toomanyadapters.com": 1,
  "www.hecktictravels.com": 1,
  "facebook.com": 2,
  "whatsdavedoing.com": 1,
  "creativecommons.org": 1,
  "www.uber.com": 1,
  "globetrottergirls.com": 1,
  "http://www.pinterest.com": 2,
  "www.hostelworld.com": 1,
  "www.legalnomads.com": 1,
  "indietravelpodcast.com": 1,
  "www.trustedhousesitters.com": 1,
  "www.wanderingeducators.com": 1,
  "foodandphotosrtw.com": 1,
  "indecisivetraaveler.com": 2,
  "www.helpx.net": 1,
  "www.dpbolvw.net": 1,
  "www.couchsurfing.org": 1,
  "www.spaghettraveller.com": 1,
  "amzn.to": 3,
  "www.worldnomads.com": 1,
  "www.travelfreak.net": 1,
  "www.google.com": 1,
  "instagram.com": 2,
  "workaway.info": 1,
  "www.coseats.com": 1,
  "www.rome2rio.com": 1,
  "www.bigworldsmallpockets.com": 1,
  "www.aussieontheroad.com": 1,
  "<a href='http'": 1,
  "www.egnite.biz": 1,
  "expatexperiment.com": 1
}
```

Here, there have been several ways in which a json could be wrong. In the screenshot above, there was a paragraph taken as a key, instead of a domain. Other errors have been found:

- HTML tags as json keys
 - No ending curly bracket (malformed json)
 - Keys with no values
2. Rows with fields separated with more than one tab between them.
 3. Rows without data in any of the 4 columns.

Step 2: extract the CC License

CC Catalog has an existing way of extracting the CC license from domains that correctly link to creativecommons.org. I will copy [this function](#) and make some modifications in order to exclude certain cases. For example, malformed license like the following will be excluded:

/licenses/by-nc-sa/2.5/deed.

/licenses/hola-by/2.5/

If there is a way to correct the license format, in a meaningful way, that is preferred instead of excluding the data. However, rows that contain a bad license (no license, no version), will be excluded.

Step 3: create a method to format the domain name in the 'provider_domain'

Package used: [tldextract](#)

I created a function called *getDomainName()* that takes a URL, as input. It then calls the *extract* function from *tldextract*, which returns the URL parts, which are:

- Sub domain
- Domain (this is what we want to extract)
- Suffix

In the following pieces of code:

```
def getDomainName(_provider_domain):  
    res = extract(_provider_domain)  
    return res.domain
```

```
#Step 3. Create a method to format the domain name in the 'provider_domain'  
df["source"] = df["source"].apply(getDomainName)
```

I apply the function to all URLs in the data. For example, if 'www.flickr.com' is passed to the function *getDomainName*, it will return 'flickr'. The name that is returned by this function is the formatted name that will be displayed in the nodes in the FDG.

Step 4: aggregate the data

There are two data aggregation steps. In both, we aggregate the rows by *provider_domain*. This is necessary because a domain may have multiple webpages but we are interested in the statistics across all the pages.

First data aggregation

In the first data aggregation, the following is performed:

- Images column is summed.
- cc_licenses are accumulated and stored in a dictionary format because this format is an expressive way to store the various license and version (since domains may utilize multiple license). The format is: `{('license_name','license_version'):quantity}` . The quantity indicates the number of licenses and version that the *provider_domain* has used.
- Links dictionary is rebuilt to reflect the total links across the entire domain.

- The value of the **creativecommons.org** link is extracted from the Links dictionary and put in a new column, called *licenses_qty*¹.

At this point, the data frame is saved into several TSV files. Each file will be processed in memory and from them I will extract the nodes and links arrays for the graph visualization.

Second data aggregation

Now that I have all the links of a provider_domain, I can calculate the number of **outgoing links** per each one. This is done using the `extract_links_len(links_col)` function in the *utils* module. But before I can do this, I need to read the TSV file and load it into a data frame. The links and cc_licences columns contain a special type of data (json), for which I need to pass to Pandas customized parser functions. They are called `LinksReader()` and `LicensesReader()`. These functions also exclude rows without a proper JSON format in their respective fields (links or cc_licences). Once I have the data frame, I apply the `extract_links_len` function and put the result in a new column, *links_qty*.

```
#Calculate total outgoing links by domain
df["links_qty"] = df.apply(extract_links_len, axis=1)
#delete domains with less than 10 outgoing links
```

I also define the node size, which is proportional to the amount of licensed work the provider_domain has. The range of the size goes from 10 to 100. I divide the *licenses_qty* value by the maximum value of that column in the Data Frame, and multiply that by 100. I then take the maximum between 10 and the number calculated before.

```
#apply scaling factor for the licenses qty to determine the node size
max_licenses_qty = float(df["licenses_qty"].max())
max_node_size = 100
min_node_size = 10
df["node_size"] = df["licenses_qty"].apply(lambda col: max(min_node_size, int((col/max_licenses_qty) * max_node_size)))
```

I will refer to this Data Frame as the **provider_domain Data Frame**. As described in [Step 1](#), rows with no licenses or links are excluded. In addition, I create another filtering rule, which is to include only the top 10 neighbors of a node. In order to achieve that, I need to process each row, and extract all the target domains that are the keys of the *links* field.

I also need to extract those target domains in order to build the nodes and links lists that force-graph requires.

¹ This should have been called cc_links

Nodes and links generation

Force-graph needs to be passed a single json file with two lists: one containing information about the nodes, and the other containing the links. They are both arrays of dictionaries. So I need to build a DataFrame of unique nodes. At first I had to process bigger TSV files by chunks. Here were the challenges faced:

- A source node can also be a target node.
- I can delete duplicate entries per column, but as I process the data in chunks, my scope is limited to the chunk size.
- A domain can be repeated not only within a chunk, but in different chunks too.
- Source and target must have licensed content (or at least in some field I need to alert that that row does not have licenses information, so in the HTML no pie chart will be rendered).

But then, when I had several smaller TSV files, although the data analysis was extended, the coding complexity decreased.

I first started by formatting the data into source and target columns to generate the unique nodes for the graph. I create the function `get_sources_links` I iterate through each row of the current DataFrame I have (the one with provider_domain, cc_licences, links column, etc), and by reading the links column, I load the json of each row. For each key in the json, I create a new row with provider_domain as source, the key as target, and the value of the key as a value feature. I append that new row to a new DataFrame. I build a new row each time I read a line, so I have a DataFrame with all the links of a single provider_domain. Here is where I apply the filtering rule: get the top 10 neighbors of a node. I implemented this by taking the rows with the 10 largest values in the **value** column.

```
chunk = pd.DataFrame(rows)
chunk["value"] = chunk["value"].astype(int)
chunk = chunk.nlargest(10, "value") #take the top 10 outgoing l
SOURCES TARGETS value
```

When I finish iterating over the rows, I convert the DataFrames to list and save the output. That is how I get a new Data Frame containing all the existing links of the graph, with source, target and value columns. This is the **links Data Frame**. Then, I extract the **target** column in order to get the **target nodes**, which must be included in the final nodes list; I convert this column into a new Data Frame. Nevertheless, in this Data Frame could exist target nodes that are also provider_domains. Hence, I make a left join between this new Frame and the Data Frame of provider_domains, by their *domain_name*, and keep only the rows that are not in the latter Data

Frame. This is like applying a difference operation over sets, but there is no straight forward function for this in Pandas.

```
#Get the target nodes that are not in the nodes DataFrame
nodes_to_concat = targets.merge(df_nodes["domain_name"], left_on="target", right_on="domain_name", how='left', )
nodes_to_concat = nodes_to_concat[nodes_to_concat["_merge"]=="left_only"]
```

Next, I create and use the function `convert_targets_to_nodes` and, using the resulting rows from the last operation, I append to them the columns the following columns, which are the ones present in the `provider_domain` Data Frame:

- `Provider_domain`: this field is filled with the string: “Domain not available”
- `Node_size`: 10 is the minimum value for a node size.

The following columns were all filled with zeros:

- `Images`
- `cc_licenses`
- `Links_qty`
- `Licenses_qty`

Finally, I append these rows to the *provider_domains Data Frame*.

Both the *links* and the *provider_domain* Data Frames are transformed to a dictionary style “records”, which is the most similar to what force-graph expects from the source file. The final structure is the one detailed in the subsection “*Graph Building*” of this document, section “*Graphical User Interface (front-end)*”.

Pie chart

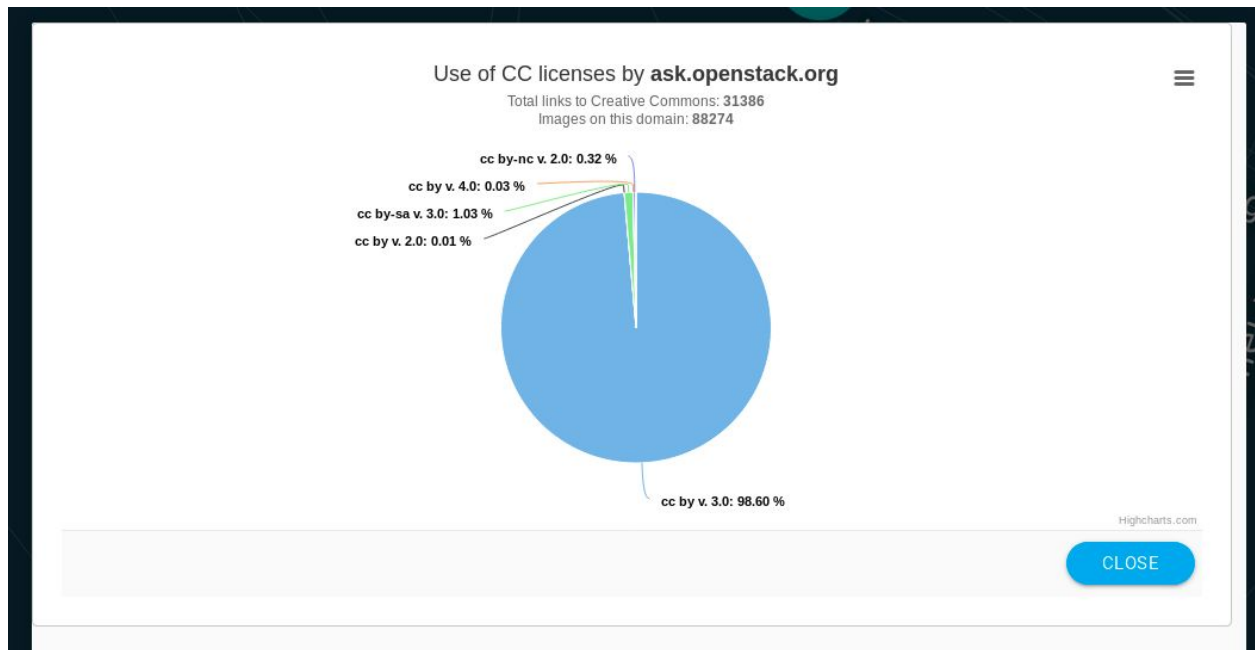
After all the data processing, a final json file, with a list of nodes and a list of links, is generated. The nodes list stores information for each node of the graph. In this information, a *cc_licenses* field is present. This field contains a dictionary with the types of CC licenses as keys, and the amount of licenses as values.

Here is an image to illustrate the above:

```
{"cc_licenses": {"('by-nc', '4.0')": 1, "('by-nc', '3.0')": 1 },
```

As you can appreciate, the keys are a tuple that contains the (license_name,version). That format is not very user-friendly, so before rendering the pie chart, I change the string format and beautify

it a little bit, removing the parentheses and appending the "cc" prefix before every license name. The final look of the pie chart for a node is the following:



The Final Graph

Visit the production website to view a demo of the force-directed graph in [2D](#) or [3D](#). Also my [final blog](#) post documents some of my final tweaks to optimize rendering the graph.



Future Enhancements

Filtering domains by country suffix

Some domains have suffixes that represent countries, for example: domain.au, which corresponds to a domain from Australia. One proposed criteria for filtering nodes in the visualization could be by country.

To note:

- Extract suffixes with `tlxextract` package, and put them in another column
- Check how many unique values of the column are; if there are a lot, it might not be suitable to use countries suffixes as a category for filtering
- It is more user-friendly to show them country names instead of suffixes. A challenge then is, to create a dictionary and map the suffixes with the countries they reference to.

Filtering domains by name

A user might just want to check if a specific domain has licensed content, and how does it use it. Another enhancement could be then to add a search bar and offer to the user the possibility to search for a node, given a domain name and/or URL.

Useful Links

Common Crawl & the ETL Process

1. <http://commoncrawl.org/the-data/get-started/>
2. <https://tech.marksblogg.com/petabytes-of-website-data-spark-emr.html>
3. <https://engineeringblog.yelp.com/2015/03/analyzing-the-web-for-the-price-of-a-sandwich.html>

Force-Directed Graph

4. [Wikipedia Description](#)
5. [Distributed Graph Analytics](#)
6. <https://observablehq.com/@d3/force-directed-graph>
7. <http://getspringy.com/>
8. <http://js.cytoscape.org/demos/colajs-graph/>
9. <https://ialab.it.monash.edu/webcola/examples/unconstrainedsmallworld.html>
10. GitHub Repositories
 - a. [3d-force-graph](#)
 - b. <https://github.com/vasturiano/force-graph>
 - c. <https://github.com/anvaka/VivaGraphJS>
 - d. Highlighting nodes and neighbors with force-graph :
<https://bl.ocks.org/vasturiano/321a770c3f4b041d1a4f40ff4ecbb6c>

Initial Research

2. GeneaQuilts: <https://ieeexplore.ieee.org/document/5613445>
3. Papilio: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12395>

Pie Chart

4. <https://jsfiddle.net/gh/get/library/pure/highcharts/highcharts/tree/master/samples/highcharts/demo/pie-legend/>
5. <https://jsfiddle.net/gh/get/library/pure/highcharts/highcharts/tree/master/samples/highcharts/export-data/multilevel-table>
6. <https://www.highcharts.com/demo/accessible-pie>
7. <https://www.highcharts.com/demo/pie-basic>



Documentation guidelines

8. <https://realpython.com/documenting-python-code/>

Data processing and performance

9. <https://stackoverflow.com/questions/7837722/what-is-the-most-efficient-way-to-loop-through-dataframes-with-pandas>
10. <https://stackoverflow.com/questions/52673285/performance-of-pandas-apply-vs-np-vectorize-to-create-new-column-from-existing-c>
11. <https://stackoverflow.com/questions/10715965/add-one-row-to-pandas-dataframe>
12. <https://ys-l.github.io/posts/2015/08/28/how-not-to-use-pandas-apply/>

GSoC Blogs

The GSoC experience was also documented in various blog posts.

- [Visualize CC Catalog data](#)
- [Visualize CC Catalog data - data processing](#)
- [Visualize CC Catalog data - data processing part 2](#)
- [Visualize CC Catalog data - data processing part 3](#)
- [The Linked Commons graph: the final vis](#)

