ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Directed graph visualization of EPFL's course catalogue using D3.js

Center for Digital Education

Raphaël Steinmann

Supervisors
Prof. Patrick Jermann, Dr. Francisco Pinto

January 2018

1 Introduction

The course catalogue at EPFL is very broad, and it can be complicated for the students to browse it and find the courses they seek. This project addresses this issue by proposing a visual representation of the courses at EPFL and the relationships between them.

This visualization consists of a directed graph where a node represents a course and an edge represents a relationship between two courses. It was entirely built in HTML, CSS and Javascript, using Mike Bostock's famous D3.js library.

2 A word about the data

2.1 Where does it come from ?

The data about courses was provided by IS-Academia. The edges come from different sources. The **baseline edges**, which mean that a course is listed as a prerequisite for another course were directly scraped from the web by Maxime Coriou in the scope of his semester project. The **indicative edges** were provided by Dr. Kshitij Sharma and Indira Sen, who ran a topic extraction on the courses' description using the LDA algorithm in order to have a similarity metric between courses. This metric was then used to infer edges between the courses that share a high similarity.

In other words, an indicative edge means that two courses seem to speak about similar subjects. The reason we need indicative edges (at least for the moment) is because a big majority of the professors don't fill the *prerequisite* field of their course(s).

2.2 Data Processing

As the data comes from several different places and is not always consistent through the years (the different sources did not necessarily use the same unique identifiers, the courses' names and/or codes can change over the years, ...), it was necessary to explore and process the data. I tried my best to discard as little data as possible while preserving consistency. As the courses evolve every year, I also designed the code in a way such that it is very easy to import new data.

3 Functionalities and features

3.1 Dynamic resizing of the nodes and edges

Naturally, when the graph is zoomed in/out, the nodes and edges get bigger/smaller on the screen. However, this behaviour is not desirable in our case as we want the nodes to keep the same size as we zoom in or out. This way, although the impression of zooming is still there, what is actually growing are not the nodes but the space between them. This feature is achieved by listening to zooming events and updating the nodes' radius and edges thickness proportionally to the scaling factor, which can be obtained from the SVG element's transformation matrix.

3.2 Courses' names

One of the advantages of dynamically resizing the nodes when we zoom in is that it frees some space to display additional information. Here, the courses' names are displayed next to the nodes when there is enough space (i.e. when we zoom enough).

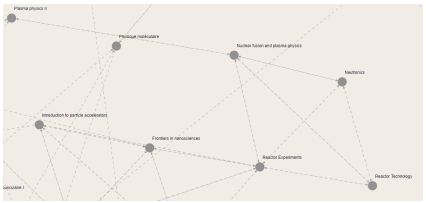

Figure 1: The courses' names appear when space permits it

3.3 Information panel

When a node is selected, an information panel pops at the bottom on the screen. It displays the title of the course, a short description and the sections that can enroll to this course. It also displays one clickable cell for each of the selected course's direct neighbors. Clicking on a cell selects the course it represents. Hovering a cell aslo triggers a micro-feedback that highlights the corresponding node on the graph (of course this is only available in the desktop version).

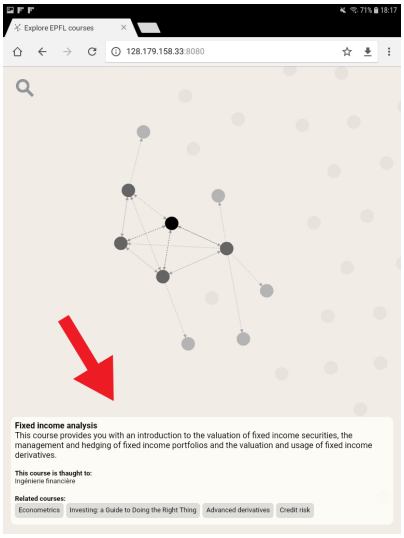

Figure 2: The information panel appears when a course is selected

3.4 Highlight a node and its neighborhood

Clicking a node progressively highlights it and its neighbors and their neighbors. Each layer is smoothly highlighted with transitions and increasing delays to express the notion of distance from the root node. The saturation of the nodes' and edges' color decreases as we go further from the root. This features relies on an implementation of the breadth-first search algorithm.

As shown on the picture, the selected node is colored in black, its neighbors in dark grey and their neighbors in light grey. The remaining barely visible nodes are the *background nodes*, i.e. nodes that do not share any first or second degree connection with the selected node.

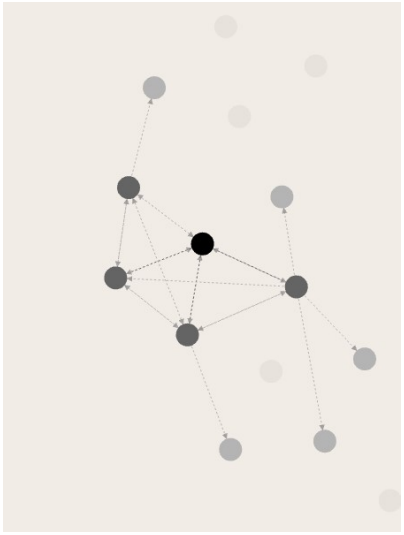

Figure 3: Selected node and its neighborhood

3.5 The search bar

Clicking on the search icon in the top left corner of the screen opens a text box to search a particular course. Each time a new letter is typed in the search box, the suggested results are updated and displayed just below the text box. Clicking on one of these results selects the corresponding node on the graph.

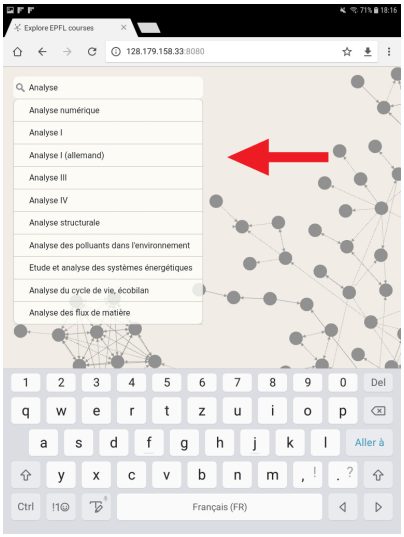

Figure 4: The search bar and suggested results

3.6 Meaning of an edge

Some edges are solid, while others are dashed edges. A solid edge is a baseline edge, whereas a dashed edge is an indicative edge. Recall that a baseline edge means that a course is referenced as a prerequisite for another course on IS-Academia. An indicative edge means that two courses seem to speak about similar subjects according to a concept extraction algorithm.

You may have noticed that the edges are directed: an edge from A to B means that course B is a pre-requisite of course A. However, it's important to know that at the moment the indicative edges actually have no information of posteriority, and thus their direction is purely random.

3.7 Smooth transitions

Instead of applying changes instantaneously, transitions smoothly interpolate the DOM from its current state to the desired target state over a given duration. This allows the user to follow more easily the changes that are taking place on the screen. This project subtly uses transitions without slowing down the visualization or making it laggy. Transitions occur for example when nodes are highlighted, or when a node is selected and the camera moves its position.

4 Performance issues and solutions

Each edge on the graph is actually an SVG *line* object. For some reason, drawing and updating a significant number of lines is very costly (way more than *circle* objects, used to represent nodes). Thus, since the visualization must run on a tablet with limited computing resources, I had to find several workarounds to keep things running smoothly.

4.1 Hide edges when the graph is moved

The first thing is to hide all the edges whenever the graph is zoomed or dragged, and draw them all again when the zoom-event is over. This is done by listening to the beginnig and end events of any zoom event.

4.2 Batch drawing of the edges

Unfortunately, the drawback of this method is that redrawing the edges after a zoom event takes a few seconds, and every other interaction is blocked during this time. To overcome this problem the edges are separated in batches of n edges each at the moment of drawing them. A timer function loops over the batches and draws them one by one, with a stopping condition that interrupts the drawing if another zoom event takes place. This method also has the advantage that the user can see the edges getting drawn on the screen batch by batch, which gives a feedback on what is happening.

4.3 Only update visible elements

Last but not least, one of the most important optimizations consists of detecting which nodes/edges appear on the viewport and update only these elements. Detecting if a node is visible is quite straight-forward, one must just compute its new coordinates using the SVG element's transformation matrix. Detecting visible edges is slightly more crafty, since an edge can cross the screen while none of its endpoint is visible. The trick is to consider each side of the screen as a segment and check if the edge intersects at least one of these segments.

4.4 Nodes layout

Placing the nodes on the screen is less straight-forward than it seems and actually requires some non-negligible computing time. Thus, in order to spare some precious rendering time in the browser, I decided to pre-compute the position of the nodes. To that end, I used a very complete and convenient software called Gephi. Gephi offers a wide variety of algorithms to place the nodes of a graph in visually convenient ways. I combined the *Yifan Hu* (for a tree-like structure) and the *Noverlap* (to set a minimum distance between each pair of nodes) algorithms. In order to export this data in a Javascript-friendly format while keeping the position information, I chose to export the nodes and edges in Gephi's .gexf files.

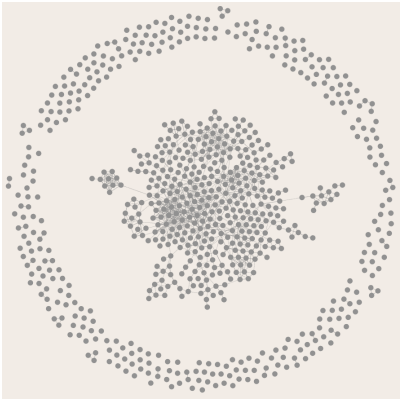

Figure 5: Layout

5 Compatibility

This visualization was developed and tested on **Google Chrome**, both desktop browser (version 63.0.3239.132) and Android mobile browser (version 63.0.3239.111). It was also tested on **Safari** (desktop version, V9.1.2). **There is no guarantee that it will work flawlessly neither on other browsers, nor on an iPad.** One could think that Chrome is the same on Android and iOS, but in reality Apple restricts how third-party developers build a web browser and forces them to use Apple's WebKit rendering engine. So it is actually completely different under the hood.

The reason I chose to conduct this project on an Android device and put iOS aside in simply because it is impossible to debug a web application on an iPad without being on MacOS (which is not my case).

6 Conclusion and future work

Finally, I would say that my work on this project laid down some solid basis, as it resulted in a fully working web application for Desktop and Android. However there is still a lot of room for future work and advanced refinement, for example:

- Ensuring the Application is compatible on other browsers (Firefox) and on iPad
- Adding some advanced functionalities, like filtering the courses by section
- Improving the layout of the nodes

To facilitate the work of any eventual student that would continue this project in the future, I fully documented my code and organized it in a clear and logical manner.

To conclude, I would say that it was very interesting to conduct this project. I had the chance to access interesting real-life data and use it to build a useful application. It was also a great opportunity to familiarize myself with front-end development.

7 References

- **D3.js**: https://d3js.org/
- **Gephi**: https://gephi.org/
- **gexf plugin for D3.js**: https://github.com/Yomguithereal/gexf