Contributions to Plotly.js library for a bespoke Sankey diagram

This blog post is a guide on the changes I applied to Plotly.js to get a custom sankey diagram with special functionalities. It is related to my previous post titled "Dunham's Personnel Flow: Interactivity, Usability," where I anticipated that I manipulated a local copy of Plotly.js to customize a sankey diagram and meet Dunham project's (DD) requirements. Plotly.js is an excellent JavaScript library for interactive graphing. This library is built on top of D3.js and Stack.gl. It is free, open-source, and high-level declarative, and therefore, easy to use. In this post, I will be explaining in detail how I modified its source code to adapt it to my needs.

Firstly, I want to remark that it is important to correctly understand the mechanisms to extend a system's functionality. Javascript allows us to extend libraries, modules, functions, classes, objects, etc., and each of these components has its ways and best practices to do so. In the days when I started this task, I was not JavaScript-savvy and I dared to edit Plotly's source code directly. I do not recommend this, although to be completely fair to my procedure, the changes I applied would have remained the same, except that, if doing it properly, they would have been written in a different part of the code.

DD's sankey webpage is mainly composed of an HTML document, a JavaScript file with the plotting settings, and the local copy of Plotly.js—let us call them myHTML, myJS, and myPlotly, respectively, from now on. The latter is a huge file (almost 200,000 lines of code), not easy to deal with in terms of finding what you want. After two years working with this library, I only became slightly familiar to it. My first goal was to identify the part of the file devoted to sankey diagrams, and then, finding the parts of the code related to specific functionalities I wanted to tailor. The first and most visually evident modification of our customized sankey diagram is its layout: the performer nodes (on the edges) are snapped closer to their first and last check-ins (inner nodes), respectively. The distance is controlled by a "global" variable named *SNAP_DISTANCE*[1] defined in myHTML. I injected this variable in the function *persistOriginalPlace* of myPlotly, which I previously backed up and renamed with *persistOriginalPlace__2*[2] to prevent overwriting and losing the original code. The process was as follows. For each performer node on the left, I calculated its position ($xLeft_p$,$yLeft_p$), then got its first check-in, calculated the position of this check-in node ($xLeft_c$, $yLeft_c$), and made $xLeft_p = xLeft_c$ - *SNAP_DISTANCE*. For each performer node on the right, I made $xRight_p = xRight_c +$ *SNAP_DISTANCE*, where ($xRight_c$, $yRight_c$) is the position of the last check-in for that performer. This way, I am modifying every performer node position with respect to its X axis by setting a fixed distance to its first/last check-in.

The second special feature that I implemented was multiselection. Sankey diagrams typically only highlight individual nodes/links on hover and unhighlight them on mouse out, but they do not allow to highlight/unhighlight based on click/unclick, let alone to select/unselect multiple elements. Although there is a "select" event attached to nodes and links on Plotly.js, it is not ready to work by default (or at least it was not on the version I am using), so I had to implement

[1] Variable names are not always the best choice, but I kept them for historical reasons.
[2] The suffixes "__*X*" help me find my own additions in such a huge file.

my own event handlers for both node and link selection, whose codes can be found on *nodeSelect__4* and *linkSelect__4*, respectively. In essence, I reused the original codes, which basically highlighted nodes and links on mouse over, but I added an attribute named *selected* that stores the selection status of the element in question. One click sets this attribute to *true* and another click sets it back to *false*. On mouse out, the events *nodeUnhover* and *linkUnhover* are called. I replaced them with my own *__4* versions. They do the same as the original ones, but before performing any unhighlighting action, they check the value of the attribute *selected*: if it is *true*, it will not unhighlight the element[3]. Neat!

The third modification that I brought in myPlotly concerns tooltips. They pop up on mouse over and show information with different formats and styles depending on the type of element pointed at. On a regular sankey diagram by Plotly.js, node and link tooltips display their respective background colors that help distinguish between a node and a link. This distinction is in principle a good design decision as nodes and links are conceptually different because they represent different concepts. In DD's sankey diagram, unlike, links mirror performer nodes and the real distinction is made between performer nodes (outer) and check-in nodes (inner), as explained in my previous post. I gave link tooltips the same format and style as performer node tooltips by redefining the variable *tooltip* in the function *linkHoverFollow*. In particular, I gave the parameters *text*, *color*, and *borderColor* of the function *Fx.loneHover* (that creates the link tooltips) the same values as their equivalent in the function *Fx.loneHover* that creates the performer node tooltips:

text:	*<performer's name>*
	'Between ' + <first date> + ' and ' + <last date>
color:	*'rgb(0, 128, 0)'*
borderColor:	*'rgb(255, 255, 255)'*

More important than its format and style is its content. On a regular sankey diagram, a node tooltip shows the node's label and the number of incoming and outgoing links, while a link tooltip shows the link's label as well as the source and target nodes' labels. In DD's sankey diagram, we need to retrieve the type of node (performer vs. check-in), the check-in date, city, and country, and the performer's first and last check-in dates. I created three new attributes on myPlotly, *nodeType__2*, *city__2*, and *country__2*, that store the necessary data. These attributes are shared and store different types of data depending on the type of element in question. This is described in the following table:[4]

[3] In my latest update, I replaced the true/false values with a count on how many times an element was selected (an element can be directly selected by clicking on it or indirectly if it is on the same path of another selected element). The procedure is still the same: I check if *selected* is greater than 0 before unhighlighting.

[4] A variable or attribute should store only the type of data it was meant for. For example, *city__2* should store names of cities only, and not dates. This may be seen as a bad decision design but it saved me from creating two extra attributes.

	label	***nodeType__2***	***city__2***	***country__2***
Check-in node	check-in's date	*check-in*	city's name	country's name
Performer node	performer's name	*cast*	first check-in's date	last check-in's date

These three attributes needed to be injected in different parts of myPlotly code for them to be recognized and used by the library (*label* was already included). The specific locations in the code are (let us take *nodeType__2* for illustration):

- In the definition of the attributes as sub-dictionaries of the dictionary *node* of the variable *attrs*:

```
nodeType__2: {
    valType: 'data_array',
    dflt: [],
},
```

- When processing nodes, inside the for loop:

```
var nt__2 = nodeSpec.nodeType__2[i];
```

- When processing nodes, in the dictionary pushed into the variable *nodes*:

```
nodeType__2: nt__2,
```

- When coercing the node to accept new attributes:

```
coerceNode('nodeType__2');
```

From now on, the attribute *nodeType__2* is available for every node element of the sankey diagram. The same steps need to be followed to include *city__2* and *country__2* in the nodes. The main use of these attributes is providing information to the tooltips and formatting them accordingly. This is an excerpt of the code of the function *Fx.loneHover* for node tooltips that illustrates a case use:[5]

```
if (d.node.nodeType__2 == 'cast') {
    text = [
        d.node.label,
        'Between ' + d.node.city__2 + ' and ' + d.node.country__2[6]
    ]
}
else { // d.node.nodeType__2 == 'checkin'
    text = [
        d.node.label,
        d.node.city__2 + ' ('+ d.node.country__2 +')'
    ]
}
```

[5] The structure of the code has been slightly modified for clarity.
[6] Let us remember that *city__2* and *country__2* contain the performers' first and last check-in's dates, respectively.

Layout, multiselection, and tooltips are only the three main changes I applied to Ploty.js to obtain a custom library that meets DD's needs. Many more subtle changes scattered in hundreds of lines of code were introduced in myPlotly to fine tune the library. It would not be possible to address all of them, but hopefully these three major cases can be used as a starting guide for new users that wish to tailor their own sankey diagrams. The consequences of working with a local copy of Plotly.js are that the changes applied will remain locally and that new updates of Plotly.js will not be available at DD. Plotly.js is licensed under the MIT license, therefore, code modifications and reuse are permitted. Finally, understanding these changes is not intuitive, but the curious reader can contact me for further explanation and details.