Basic Technical Protocol

Overview

1. Get data
    a. Query Discogs API via Postman, for all electronic music 12" vinyls, year by year.
2. Format and filter data
    a. Index and filter data to contain necessary metadata (title, label, styles, year, country, want/have community metrics and uri) with Alteryx
    b. Create node list file with metadata, and two cleaned adjacency list files (for artists and labels respectively) via custom python script
3. Create network visualizations
    a. Project adjacency list files via NetworkX and *bipartite projection*
    b. Combine with node list file to create final network file
    c. Spatialize with ForceAtlas2, isolate Giant Connected Component and colour nodes by modularity class.
4. Create a web application prototype as a Github page, through Sigma.js

Data Collection via Postman

The goal of using Postman, was to both make the process of obtaining authentication from Discogs and the making of API the queries, easier. Through cross referencing with Discogs' API's documentation and the site itselfs search function, we found that we had to refine our search queries to work around Discogs performance limitation and their pagination system. In our case we found the necessary specification to be: 12" Master releases on vinyl. Furthermore, to not get more than 100 pages per Search API call, we had to do each year's query separately.

1) Create an updating variable for the page number; to allow the query to call a different page of the pagination each time run.

```
pm.environment.set("page_number", 1 +
parseInt(pm.environment.get("page_number")));
```

2) Write the output of each query to the variable "r" in a JSON format; to allow collection of the data afterwards.

```
let responses = pm.collectionVariables.has('r') ?
JSON.parse(pm.collectionVariables.get('r')) : [];

console.log(responses);
```

```
responses.push(pm.response.json());

pm.collectionVariables.set('r', JSON.stringify(responses));
```

	VARIABLE	INITIAL VALUE	CURRENT VALUE	...	Persist All	Reset All
☑	r		[{"pagination":{"page":1,"pages":84,"per_page":100,"items":8388,"urls":{"last"...			

3) Run the “Search” API query through the postman interface with the following settings (variables marked in “{{}}” and defined in the “collection variables” settings):

GET {{url}}/database/search?per_page=100&type=master&page={{page_number}}&genre=electronic&year={{year}} **Send**

Params • Authorization Headers (9) Body Pre-request Script • Tests • Settings • Cookies

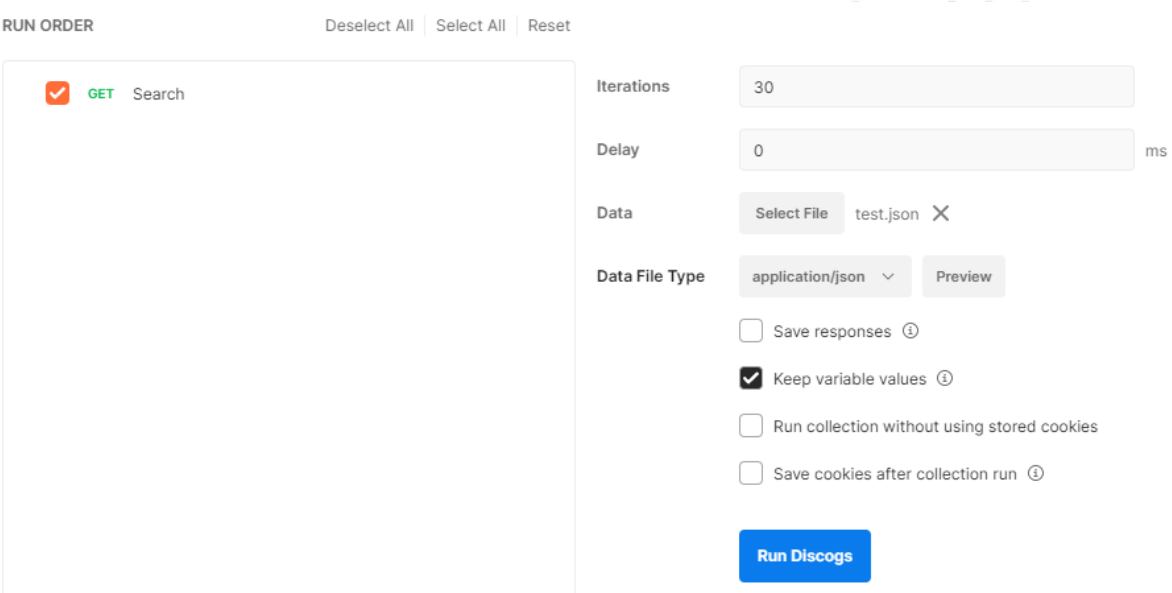
Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
☑	per_page	100			
☑	type	master			
☑	page	{{page_number}}			
☑	genre	electronic			
☑	year	{{year}}			

4) Define the necessary *Collection Variables*, which specifies which API to call, the authorization required to run queries and by creating our own application; and set other variables (*page number* and *year*) across all queries.

	VARIABLE	INITIAL VALUE	CURRENT VALUE
☑	url	https://api.discogs.com	https://api.discogs.com
☑	user_agent	PostmanDiscogs/1.0	PostmanDiscogs/1.0
☑	consumer_key		OJpqHVzrTditFkLWwNJ
☑	consumer_secret		iRnCkDkUDeKAdfKUCRI
☑	oauth_token		dedehkCQeLGdTUqyoxI
☑	oauth_token_secret		HtyxzdueZuUPhwliLwxy
☑	oauth_verifier		gGlfpSLBZM
☑	username		
☑	page_number	0	30
☑	year	0	1970

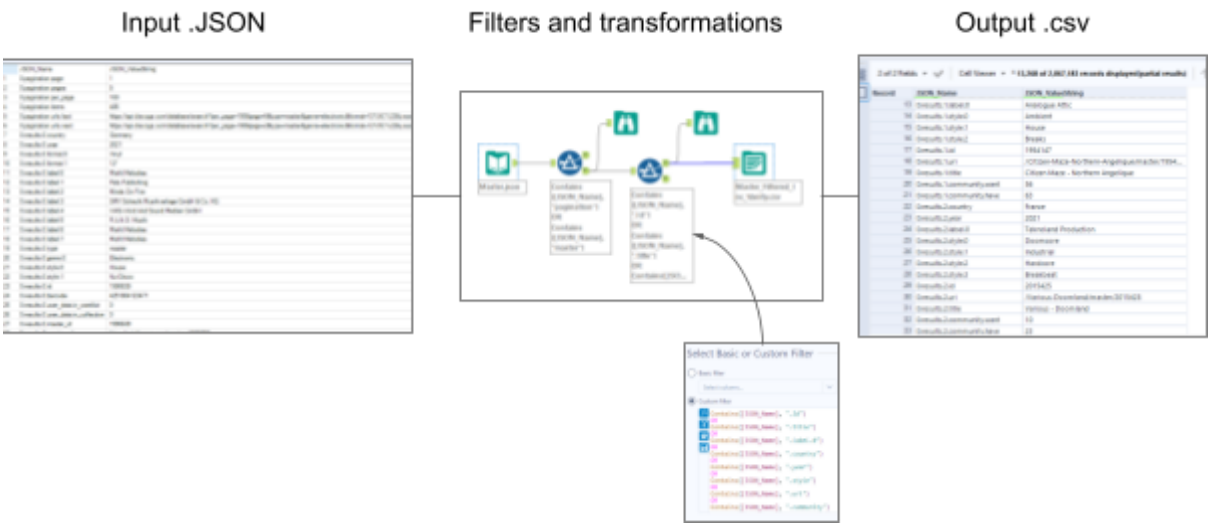
5) Use the *Runner* function, which allows for running one or more API queries multiple times. In this case, 30 runs (that is 30 pages of a 100 releases each) was the maximum size feasible, due to that being the maximum amount of data capable of being stored in a variable.



6) Finally, you get each query results in a JSON format data file containing information on each master release. Each file also contains information on the number releases as distributed via *pagination*. Apart from the *artist* and *title* on the same line (more about why that proved troublesome later) a query result also contains useful metadata about a release's attributes; such as *country, year, format, label* and *style*. Additionally, the result JSON also contains other useful information, such as a release's unique *master id*, the *uri* to its entry on Discogs or even community data such as how many people *want* or *have* the release in question.
7) After each *collection run*, each *result* JSON (from "r") can then manually be strung together to one big Master file, containing all Electronic Music 12'' vinyl master releases from 1970 to 2021 .

Data Filtration via Alteryx

The goal of using Altery was two-fold. First, we could filter through the information in the data file, and select only what is needed in the output CSV file. Secondly, Alteryx also allowed us to index all the releases and their metadata separately - to make the process of creating the network files easier. In retrospect, however, using Alteryx was in many ways a remnant of our efforts to figure out how to extract data from Discogs data dumps. As such, having to do this process again, it would probably have been easier to either do the entire data processing in Alteryx, *or* in Python - not a combination of both.



Data Processing in Python

After having obtained the filtered and indexed files from Alteryx, the goal of using Python was to create the node- and edge list files to put into Gephi. In all actuality the easiest way of getting the data into Gephi, turned out to be creating a master node file that contained all the meta data, as well as the unique release-id from Discogs itself. Then, creating two seperate *adjacency list* files, one for label and one for artists, that only contained the names of their artists and label, as well as the unique id of the release they connect to. The final file, with both edges and nodes, could then be made by combining all three files.

0) Importing necessary libraries

```
#import libraries
import pandas as pd
import re
```

1) creating the *nodelist* file and creating the proper headers (*title, artist, recordlabels,* etc.).

```
#df for DataFrame
df = pd.read_csv('Master_Filtered.csv')

num_rows = df.shape[0] #number of rows

#create masternode file
node_file = open("masternodeslist_split_no_regex.csv","wb")

#creates header for masternode file
node_file.write("Label; title; artists; recordlabels; year; country; styles; url; want;
have\n".encode("utf-8"))
```

2) Creating the data frame, corresponding to the correct attributes from the header, and explicitly formatting to *utf-8*, to avoid encoding errors when Gephi has to read the file.

```
def write(id,a):
    try:
        # write to masternode file
        node_file.write("{}; {}; {}; {}; {}; {}; {}; {}; {};
{}\n".format(a[9],a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8]).encode("utf-8"))

    except:
        print("error: release skipped")
```

3) Using the indexes from the filtered file to separate information from one release to another, and to write/clear a new list for each release.

```
size = 10
res_data = [None] * size
res_id   = 0
cur_id   = 0

for i in range(num_rows):
    x = df.iloc[i,0].split('.') #split result number info

    # set id for current release
    cur_id = int(x[2]) + int(x[0])*100 #corresponds to centi / deca

    # check if new release
    if(cur_id != res_id):
        #save res_data to file
        write(res_id,res_data)
        res_id = cur_id
        res_data = [None] * size
```

4) Putting information from the appropriate attribute in the data, into the correct position in the array.

```
    # replace / remove symbols that breaks the .csv format
    val = val.replace("\"","")
    val = val.replace("'","")
    val = val.replace(",","/")
    val = val.replace(";","/")
```

(And remembering to remove or replace symbols that would mess with the output CSV format)

```
    # determine attributes (title, year etc.)
    attr = x[3]
    val  = df.iloc[i,1]
# do something depending on which attribute
    if attr == "title":
        title_val = val.split(" - ")
```

```
        # release title
        res_data[0] = title_val[1].strip()
        res_data[1] = title_val[0].strip()
    if attr == "label":
        res_data[2] = val
    if attr == "year":
        res_data[3] = val
    if attr == "country":
        res_data[4] = val
    if attr == "style":
        if res_data[5] is not None:
            res_data[5] =  res_data[5] + ", " + val
        else:
            res_data[5] = val
    if attr == "uri":
        res_data[6] = str('https://www.discogs.com'+val)
    if attr == "community":
        if x[4] == "want":
            res_data[7] = val
        else:
            res_data[8] = val
    if attr == "id":
        res_data[9] =val
```

5) For creating the *adjacency list* needed to make edges for the network, it is the same procedure as with the *node list* file - except this time we are creating three separate files, each with the *id* of a release, and a specific attribute (*label, artist* or *style*). In retrospect, we ended up using the *style* edge files more as a curiosity, than a part of the final network.

```
#create label edge file
edge_file_label = open("adjacencylist_label_only.csv","wb")

#create artist file
edge_file_artist = open("adjacencylist_artist_only.csv","wb")

#create style file
edge_file_style = open("adjacencylist_style_only.csv","wb")def write(id,a):
    try:
        # write to edge files
        edge_file_label.write("{}; {}\n".format(a[9],a[2]).encode("utf-8"))
        edge_file_artist.write("{}; {}\n".format(a[9],a[1]).encode("utf-8"))
        edge_file_style.write("{}; {}\n".format(a[9],a[5]).encode("utf-8"))

    except:
        print("error: release skipped")
```

6) The script for creating the adjacency lists also has two important additions. To get each artist as a separate entity in the network, a long list of *regular expressions* were needed to properly separate one artist from another. This list was obtained experimentally, by looking through a multitude of different releases on Discogs.

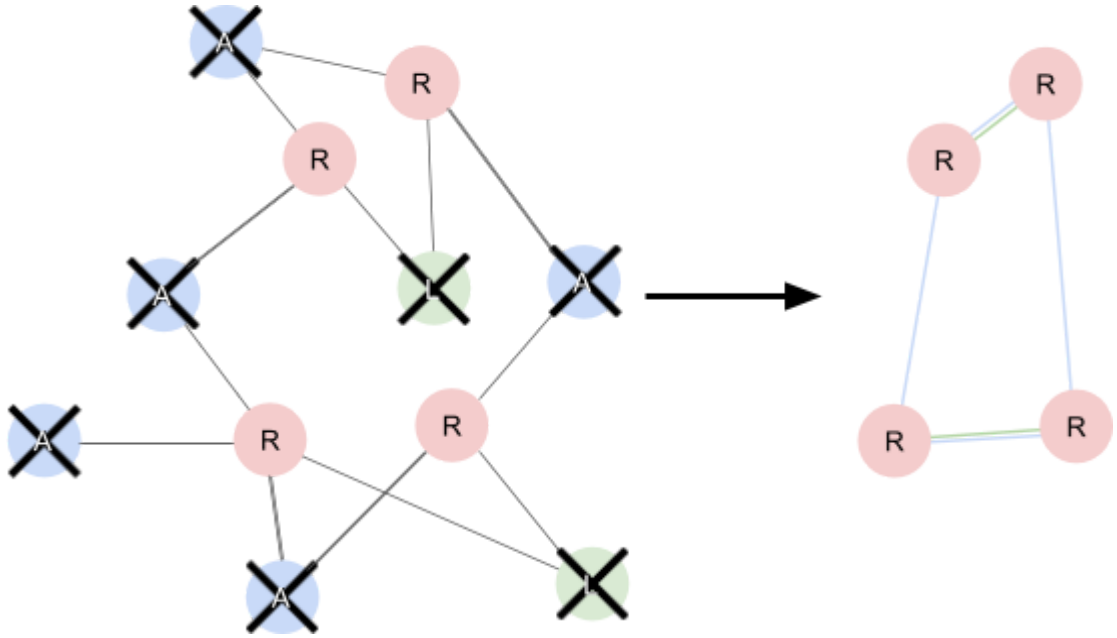
```
# artist(s) by splitting at regex
        artist_regex =
'(?i)\s+&\s+|\s+and\s+|\s*,\s*|\s+vs.?\s+|\s*/\s*|\s*\+\s*|\s+a.k.a.\s+|\s+ft.?\s+|\s+featuring\s+|\s+
presents\s+|\s+w/\s+|\s+feat.?\s+|\s+x\s+|\s*:\s*'
        artists = re.split(artist_regex,title_val[0])
```

Additionally, each attribute needed for the three different adjacency list files as well as the title of the release, had to have a have a prefix put in (e.g. “title:” before the release title) in order to be able to separate different types of nodes in the final network.

```
res_data[0] = "title:" + title_val[1].strip()

...

res_data[1] = "artist:"+"; artist:".join(["{}"]*len(artists)).format(*artists)

if attr == "label":
        res_data[2] = "label:" + val
...

if attr == "style":
        if res_data[5] is not None:
            res_data[5] =  res_data[5] + "; style:" + val
        else:
            res_data[5] = "style:" + val
```

Network Projection with NetworkX

To produce a more “readable” network, we opted to “project” the label and artist nodes of our network, onto the release nodes. To achieve network projection, we needed a bi-partite network. Luckily, our network was already partitioned after type, due to the prefixes we put in.



This script uses NetworkX to do network projection on the networks created via the *adjacency lists* only. One set of nodes is projected onto another set of nodes (called *top-* and *bottom-nodes* or *0* and *1* in the script), by referring to the "bipartite" attribute of the network. This attribute was made manually in Gephi, by making a new column and calling all nodes with the "title:" prefix (that is releases) 0, and the other type of node (in the example below: styles)

1.

```
import networkx as nx
from networkx.algorithms import bipartite

B = nx.read_gexf("Style_Release_final.gexf").to_undirected()

top_nodes = set(n for n,d in B.nodes(data=True) if d['bipartite']==0)
# releases
bottom_nodes = set(B) - top_nodes
# label/style/artist etc.

G = bipartite.weighted_projected_graph(B, top_nodes)
#project "1" onto "0", weighted


nx.write_gexf(G, "bipartite_style.gexf")
```

Creating the final network visualization in Gephi

Creating the final network file was then a matter of combining the two projected networks created from the adjacency list files with the master node list file containing all the metadata. Before this could be done however, we also went back to the projected network files, and labeled the edges themselves, so that when all the files were combined, it would be possible to view whether an edge came from an artist or a label.

Nodes Edges | Configuration | Add node | Add edge | Search/Replace

Source	Target	Type	Id	Label	Weight
45176	2120031	Undirected	5550948	Label	1.0
201899	2120031	Undirected	5550947	Label	1.0
45030	2120031	Undirected	5550943	Label	1.0
2100646	2120031	Undirected	5550931	Artist	1.0
2113656	2120031	Undirected	5550930	Label	1.0
2109440	2115444	Undirected	5269143	Label	2.0
2109440	2102394	Undirected	4898036	Label	1.0
2132332	2109440	Undirected	6291099	Artist	1.0
2123512	2150406	Undirected	7340373	Artist	1.0
2140960	2234065	Undirected	12276833	Label	1.0

Modularity Report

Parameters:

Randomize: On
Use edge weights: On
Resolution: 1.0

Results:

Modularity: 0.805
Modularity with resolution: 0.805
Number of Communities: 108

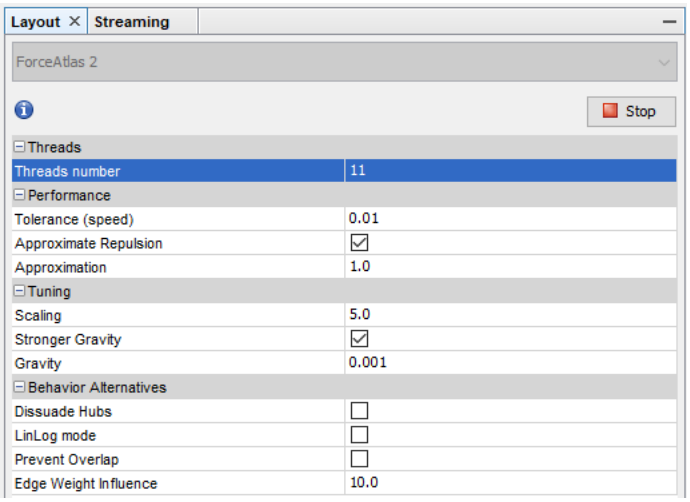
Another addition to the file we wanted to make, was to find the difference between how many people wanted a certain release and how many of that release that was actually for sale (via the Column Calculator plugin). While it didn't change much in terms of ranking, it would, however, seem a better metric for how *coveted* a release is - compared to just looking at how many people want it.
Two final adjustments were done before subjection the network to the ForceAtlas2 algorithm: First, running the “Modularity Class” algorithm to find clusters in our network based on structure, rather than layout. Secondly, we filtered the network down to only the “Giant Connected Component” to avoid “satellites” in the network.
In the end, the final network file looks like this in the data laboratory:

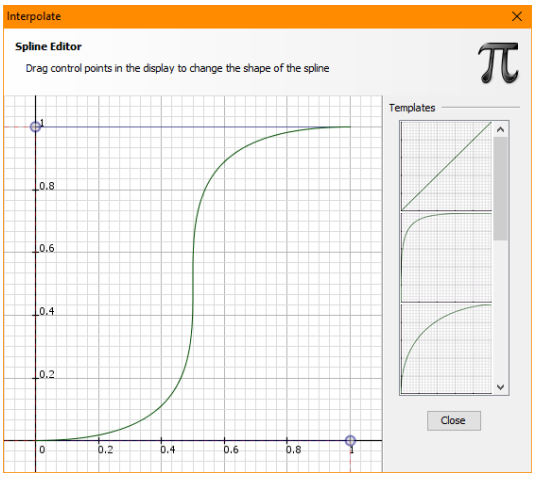
Nodes Edges | Configuration | Add node | Add edge | Search/Replace | Import Spreadsheet | Export table | More actions

Id	Label	...	title	artists	recordlabels	year	country	styles	want	have	Popularity Metric	url	Modularity Class
2211510	29410		Music Sound...	Stardust	Roulé	1998	France	House	33257	23682	9575.0	https://www.discogs.com/Star...	88
193276	80049		The Age Of ...	Age Of L...	DiKi Records	1990	Belgium	Trance, Tec...	29052	16527	12525.0	https://www.discogs.com/Age-...	28
190099	94963		Gypsy Wom...	Crystal W...	Mercury	1991	US	House, Gar...	22529	13062	9467.0	https://www.discogs.com/Crys...	74
208350	49387		Spacer Woman	Charlie	Mr. Disc Or...	1983	Italy	Italo-Disco, ...	16504	5428	11076.0	https://www.discogs.com/Char...	41
201107	5108		Acid Tracks	Phuture	Trax Records	1987	US	Acid House,...	16117	4907	11210.0	https://www.discogs.com/Phut...	75
2132619	350827		Big Fun	Innercity	KMS	1988	US	House, Tec...	16106	14147	1959.0	https://www.discogs.com/Inne...	90
189192	79590		Go	Moby	Outer Rhythm	1991	UK	Trance, Tec...	16095	12385	3710.0	https://www.discogs.com/Mob...	4
190138	6424		Papua New ...	The Futur...	Jumpin & P...	1991	UK	Breakbeat, ...	15973	14120	1853.0	https://www.discogs.com/The-...	27
2130405	5568		The Promise...	Joe Smoo...	D.J. Intern...	1987	US	House	15775	7983	7792.0	https://www.discogs.com/Joe-...	17
2208811	401133		Deep Inside	Hardrive	Strictly Rhy...	1993	US	House, Dee...	15253	6568	8685.0	https://www.discogs.com/Hard...	0
173804	100665		The Bomb! (...	Kenny Do...	Henry Stre...	1994	US	House	14662	11610	3052.0	https://www.discogs.com/Ken...	90
210379	134087		Keep The Fir...	Gwen Mc...	Atlantic	1982	US	Soul, Disco,...	14239	2439	11800.0	https://www.discogs.com/Gwe...	21

In terms of spatializing the network, we tried to promote the formation of clusters as much as possible. As such, we increased the edge weight from 1 to 10, set the scale relatively low (compared to the size of the network) and specifically turned off “Prevent overlap”.

Similarly, we set size by our “Popularity Metric”, but with a range from 0.1 to 100 and a spline setting to promote making the big nodes visible, and the rest small enough to still cluster.





Creating a Web Application Prototype with Github and Sigma.js

After our initial design process, we were left with a set of design prescriptions (see “Design Prescriptions for the Web Application Prototype”). As our easiest route to making a functioning prototype seemed to be through the “SigmaExporter” plugin for Gephi, this was the approach we went with. However, lacking the necessary competencies in JavaScript, CSS and HTML competencies - save for a few cosmetic changes such as custom legend and hyperlinking the url to the Discogs pages - we had to settle for making a basic network visualization through the plugin. The biggest constraint, however, came in the form of hosting the data through Github, as even through large file storage, the maximum size of the network file we could show as a Github page, is only 100 MB. For comparison, the “final” projected network turned out to be 600 MB exported as a JSON file. Thus, after cleaning the “original” file of unnecessary metadata and removing nodes with 2 or fewer edges, we ended up with this network visualization on a Github Page.

Technical steps

1) After reducing the file size to a 100 MB JSON, export the network through Sigma.js plugin for Gephi.
2) Upload to Github
3) Customize config. file for performance and readability:

```
 "type": "network",
  "version": "1.0",
  "data":
"https://github.com/AlfredFelumb/A.A.P.P.---Discogs-Music-Exploration-Tool/blob/main/network/data.json?raw
=true",
  "logo": {
```

```
    "file":
"https://media.discordapp.net/attachments/808294432586989588/855784063121096744/Powered_by.png?width=180&h
eight=100",
    "link": "",
    "text": "A. Felumb"
  },
  "text": {
    "intro": "This visualization has been developed in conjunction with a project group at
Techno-Anthropology at Aalborg University, Copenhagen. <a target=\"_blank\"
href=\"https://github.com/AlfredFelumb/A.A.P.P.---Discogs-Music-Exploration-Tool\">Github.<a> \n Version
0.1",
    "more": "Network of electronic music releases on 12'' vinyl from www.discogs.com with connections to
accompanying artists and labels. It's recommended that you explore the network by searching for a specific
release, artist, or label. Using the search function, you get a given node with edge links out to a depth
of 2. \n The network has 186131 nodes and 324915 edges. The network was also reduced to only include nodes
with 3 or more connections, to accomodate Github file storage limitations.",
    "title": "<br> Discogs Electronic Music Exploration Tool"
  },
  "legend": {
    "edgeLabel": "",
    "colorLabel": "Releases <br> Labels <br> Artists",
    "nodeLabel": "\n",
    "releasecolor": "Release"
  },
  "features": {
    "search": true,
    "groupSelectorAttribute": false,
    "hoverBehavior": "dim"
  },
  "informationPanel": {
    "groupByEdgeDirection": false,
    "imageAttribute": false
  },
  "sigma": {
    "drawingProperties": {
      "defaultEdgeType": "line",
      "defaultHoverLabelBGColor": "#002147",
      "defaultLabelBGColor": "#ddd",
      "activeFontStyle": "bold",
      "defaultLabelColor": "#000",
      "labelThreshold": 2,
      "defaultLabelHoverColor": "#fff",
      "fontStyle": "bold",
      "hoverFontStyle": "bold",
      "defaultLabelSize": 18,
      "batchEdgesDrawing": "true",
      "canvasEdgesBatchSize": "50",
      "webglEdgesBatchSize": "10",
      "hideEdgesOnMove": "false",
      "labelSize": "fixed",
      "labelSizeRatio": "2"
    },
    "graphProperties": {
      "maxEdgeSize": 3,
      "minEdgeSize": 1,
      "minNodeSize": 1,
      "maxNodeSize": 1
    },
    "mouseProperties": {
      "maxRatio": 20,
      "minRatio": 0.0001
    }
  }
}
```


4) Modify the main file script to include a nodes “neighbors” to a depth of 2 instead of the default 1:

```
function findNeighborsAndNeighborsNeighbors(a)
{
    incoming={},outgoing={};
    sigInst.iterEdges(function (b) {
        b.attr.lineWidth = !1;
        b.hidden = !0;

        n={
            name: b.label,
            colour: b.color
        };


       if (a == b.source || a == b.target)
       {
        var neighborNode = a == b.target ? b.source : b.target
        if (a==b.source) outgoing[neighborNode]=n;
        else incoming[neighborNode]=n;

        sigInst.neighbors[neighborNode] = n;


        // neighbors neighbors
        sigInst.iterEdges(function (c) {
            c.attr.lineWidth = !1;
            c.hidden = !0;

            n={
                name: c.label,
                colour: c.color
            };

            if (neighborNode==c.source) outgoing[c.target]=n;
            else if (neighborNode==c.target) incoming[c.source]=n;
            if (neighborNode == c.source || neighborNode == c.target)
            {
                sigInst.neighbors[neighborNode == c.target ? c.source :
c.target] = n;
            }
           c.hidden = !1, c.attr.color = "rgba(0, 0, 0, 1)";
        });

       }
       b.hidden = !1, b.attr.color = "rgba(0, 0, 0, 1)";
    });
    return [incoming,outgoing];
}
```

5) Customize the info box with suitable text, “Powered by Discogs” logo, and a custom legend for release, label and artist nodes.