Papers in Evolutionary Economic Geography

18.43

A network-based method to harmonize data classifications

Dario Diodato

Utrecht University
Urban & Regional research centre Utrecht

A network-based method to harmonize data classifications*

Dario Diodato

Center for International Development, Harvard University. dario_diodato@hks.harvard.edu

January 2, 2019

Abstract

A frequent problem in research is the harmonization of data to a common classification, whether that is in terms of – to name a few examples – industries, commodities, occupations, or geographical areas. Statistical offices often provide concordance tables, to match data through time or with different classifications, but these concordance tables alone are often not sufficient to define a clear methodology on how the matching should be performed. In fact, the concordance tables have, in numerous occasions, a many-to-many mapping of classifications. The issue is exacerbated when two or more concordance tables are concatenated. In this Jupyter notebook, I discuss a network-based abstraction of this problem and propose, as a general solution, a method that identifies the network components (or the network communities) to make data converge to a new classification. The method simplifies the issue and reduces greatly conversion errors.

Keywords: classification, concordance, harmonization, network, Python, Jupyter.
JEL categories: C65, C82, C88.

Acknowledgments

I gratefully thank Frank Neffke for suggesting the core idea behind this method.

Table of Contents

```
In [ ]: """
         . Packages required
         1 Introduction
         2 Concordance tables as matrices
         3 Concordance tables as networks
         4 Main methodology
         5 Concatenated concordances
         6 The complex case in a many-to-many mapping
         7 A compact program
         . Discussion and conclusions
        """
```

*This paper is written as ipython notebook with Jupyter. The interactive .ipynb file can be found on my website (dariodiodato.com). The download bundle also contains the required inputs and a compact .py program to rapidly and automatically harmonize any number of classifications.

Packages required

1) Standard packages

- numpy, pandas, matplotlib
- Use: pip install numpy pandas matplotlib

2) networkx

- Requires standard packages
- Supports: pip install networkx

3) py2cytoscape

- Required for this ipython notebook, not for the compact program *concordance_network.py*
- Requires installing Cytoscape software (Cytoscape needs to be running when using this Jupyter notebook)
- Requires installing python-igraph
- Requires installing requests: pip install requests
- Supports: pip install py2cytoscape

```
In [1]: import networkx as nx
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from py2cytoscape import util as cy
        from py2cytoscape import cytoscapejs as cyjs
        import requests
        import json
        from IPython.display import Image
```

```
<IPython.core.display.Javascript object>
```

1 Introduction

Researchers are very often faced with the problem of harmonizing data that have been recorded with different classification schemes. Statistical offices and other institutions many times go through the trouble of carefully constructing and distributing concordance tables, to help others in their research. However, these raw concordance tables typically have a many-to-many mapping across classifications, meaning that the researcher is left without a clear and unambiguous method to harmonize the data. To my knowledge, no best practice has emerged on how to deal with this situation. This Jupyter notebook[1], proceeding with many examples, discusses the problems of harmonizing data from many-to-many concordance tables. I will propose to abstract concordance tables first as matrices, then as networks. This allows to use well-known properties

[1] For the PDF printout of the notebook, I omitted some of the non-essential Python code to make this notebook/paper leaner and easier to read. Every time some code is omitted, the reader will find the sign "[...]" in the PDF. The actual Jupyter notebook (.ipynb file) contains all the code.

of matrices and network to get a deeper understanding of the potential issue in the operation of harmonization. It also allows to identify a clear network-based solution that provides an error-free harmonization methodology.

The most common problem when doing research on the economy, to my perception at least, is related to dealing with industrial classifications. Each country has one or more classifications in every given period, and each classification gets constantly updated and reorganized over time. As result, any time we need to compare two countries, two periods, or sometime even two datasets in the same country/period, we must harmonize industries first. For this reason, I will use the harmonization of industry classifications as the working example in this notebook.

But a reclassification of industries is not the only situation when we must deal with harmonization. Another pressing issue, especially for research in economic geography, is how to deal with the re-classifications of regional boundaries. Standard geographical units, such as Metropolitan Statistical Areas (MSAs) in the US or NUTS regions in Europe are frequently updated. Take as an example what the Wikipedia page on UK NUTS regions says about a reclassification in 2010.

"The combined area of UKD2 (Cheshire pre-2010) and UKD5 (Merseyside pre-2010) were replaced by UKD6 (Cheshire post-2010) and UKD7 (Merseyside post-2010), due to the transfer of Halton to the Merseyside NUTS region from Cheshire. This resulted in the following changes to the underlying NUTS 3 areas: UKD22 (Cheshire CC) being split into UKD62 (Cheshire East) and UKD63 (Cheshire West and Chester); The areas of Liverpool, Sefton and Wirral were not changed as NUTS 3 areas however to reflect their transfer within NUTS 2 areas were respectively renumbered from (UKD52 to UKD72; UKD53 to UKD 73 and UKD54 to UKD 74). The two areas of UKD51 (East Merseyside pre 2010) and UKD21 (Halton and Warrington) were amended by the transfer of Halton from the latter to former to form the new areas of UKD71 (East Merseyside post-2010) and UKD61 (Warrington)."

https://en.wikipedia.org/wiki/NUTS_statistical_regions_of_the_United_Kingdom (Accessed December 19, 2018)

The discussion in this paper, while using industries as examples, is general enough to be applied to any reclassification problem a researcher might face including those of geography reclassification.

As a first working example, let us say we are interested in exploring industry growth rates around the great recession. We want to know how the US economy had changed between 2005 and 2010. To focus the analysis, we restrict the question to one sector: information and cultural services (Naics 51). We ask, what happened to the 5-digit categories that make this sector?

We then go to the Census website and choose a source for data. Here we take County Business Patterns. From:

https://www.census.gov/programs-surveys/cbp/data/datasets.html

We download the "Complete U.S. File" for the years 2005 and 2010. (The files used in this notebook are already included in the download bundle; the online source is for reference)

1.1 Data import 2005

```
In [2]: # 2005 file
        input_file = "cbp05us.txt"
```

```
        data05=pd.read_csv(input_file)
        data05 =data05[['naics','empflag','emp']]

In [3]: # keep industry 51, keep 5-digit only
```

[...]

```
In [5]: # 2005 data, display
        data05.head()

Out[5]:       naics     emp
        1418  51111  380144
        1420  51112  158843
        1422  51113   83555
        1424  51114   57042
        1426  51119   25047
```

1.2 Data import 2010

```
In [6]: # 2010 file
        input_file = "cbp10us.txt"
        data10=pd.read_csv(input_file)
        data10 =data10[['naics','lfo','empflag','emp']]

In [7]: # all firms type
        data10=data10[data10['lfo'] == "-"]

In [8]: # keep industry 51, keep 5-digit only
```

[...]

```
In [10]: # 2010 data, display
         data10.head()

Out[10]:       naics     emp
         8705  51111  264833
         8719  51112  134432
         8733  51113   81511
         8749  51114   41489
         8763  51119   20424
```

1.3 Concordance

```
In [11]: data05.shape

Out[11]: (30, 2)
```

```
In [12]: data10.shape

Out[12]: (27, 2)
```

We can easily see by counting the rows in the two tables that the data in 2005 and 2010 do not match for 5-digit industries of the sector 'information.' The right concordance table can be dowloaded from the Census

https://www.census.gov/eos/www/naics/concordances/concordances.html

Concordance file: 2002_to_2007_NAICS.xls. (The file is already included, the online source is for reference)

```
In [13]: # import concordance and transform it to 5-digit

         input_file = "2002_to_2007_NAICS.xls"
         conc_raw=pd.read_excel(input_file, skiprows=2)

                                        [...]

         C = conc_raw[['naics2002','naics2007']]

                                        [...]

         C.head()

Out[13]:    naics2002  naics2007
         0      11111      11111
         1      11112      11112
         2      11113      11113
         3      11114      11114
         4      11115      11115

In [14]: C.shape

Out[14]: (741, 2)
```

1.4 The issue with raw concordance tables

The following table illustrates the issue with raw concordance tables. Even though the Census provides tables to harmonize data from 2002 to 2007, we can see, thanks to the table, that some naics2002 codes are assigned to multiple naics2007 codes. Moving to the opposite direction does not help either: multiple naics2002 are linked to some naics2007. In other words, the raw concordance table has a many-to-many mapping.

```
In [15]: def group_sum(pdframe,x,group,newvar):
             newframe = pdframe.groupby(group)[[x]].sum().reset_index()
             newframe = newframe.rename(columns={x:newvar})
             merged = pdframe.merge(newframe,on=group)
             return merged

In [16]: # count number of classes each naics2002 and naics2007 maps to

         C['ones']=1
         C = group_sum(C,'ones','naics2002','by2002')
         C = group_sum(C,'ones','naics2007','by2007')
         C = C[['naics2002','naics2007','by2002','by2007']]

In [17]: Cslice = C[C['naics2002'] >= 51520]
         Cslice.head(10)
```

Out[17]:

	naics2002	naics2007	by2002	by2007
470	51521	51521	1	1
471	51611	51913	1	2
472	51811	51913	3	2
473	51711	51711	1	3
474	51751	51711	1	3
475	51811	51711	3	3
476	51721	51721	1	1
477	51731	51791	1	3
478	51791	51791	1	3
479	51811	51791	3	3

How can we compute growth rates, with this web of reclassifications?

In the next two sections, I discuss two powerful abstractions: concordance tables as matrices and concordance tables as networks. With this I will also establish a notation and a language, which will be helpful for the rest of the notebook.

This notebook is organized as follows:

Section 2 abstracts concordance tables as matrices. In particular, I highlight the difference between a raw concordance table and a final transformation matrix, and look at the different properties that these two types of matrices have.

Section 3 abstracts concordance tables as networks. We show that this allows for a better comprehension of a raw concordance mapping than abstracting concordances as matrices.

Section 4 proposes the main methodology: first, we identify the connected components of the raw concordance network. Second, we assign to each connected component a new classification code. Finally, we create two transformation matrices, one for each original classification, to move from old to new codes.

Section 5 extends the main methodology to a situation when one has to harmonize several classifications. In the example of this section we show the harmonization of Naics 2002, 2007, and 2012.

Section 6 shows a further extension. Sometime the raw concordance has a mapping so interconnected that the whole concordance is one large connected component. However, when this happens the network seems to typically exhibit several separate components, weakly linked by

isolated nodes. This suggests that by identifying clusters (communities), we can quickly separate these weakly connected components automatically and then apply the main methodology.

Section 7 presents a compact program to efficiently apply the methods presented in this notebook.

A brief discussion is included in the closing remarks.

2 Concordance tables as matrices

2.1 Definitions

Let's define two sets of classifications (for instance, industry classifications) $i \in I$ and $j \in J$. As an example, we have

$$I = \{a, b, c\} \quad J = \{\alpha, \beta, \gamma, \delta\}$$

A **raw concordance table** is a matrix C_{ij}, mapping i and j classifications

$$C_{ij} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can notice that b maps to γ, and c maps to δ, uniquely. However a maps to both α and β. This means that we can easily transform data from j to i, but not from i to j.

As C_{ij} is a concordance table that does not always uniquely identify the mapping between classifications, let us introduce the concept of **transformation matrix**. We call this matrix T_{ij} and it allows to transform data from classification j to classification i.

If we collect employment data for α, β, γ, and δ in a vector x_j

$$x_j = \begin{bmatrix} 10 \\ 20 \\ 5 \\ 15 \end{bmatrix}$$

Then we have that

$$x_i = \sum_j T_{ij} x_j$$

Or in compact matrix form

$$\mathbf{x}_i = \mathbf{T}_{ij} \mathbf{x}_j \tag{1}$$

2.2 What is the relation between C and T?

By adopting a convention of keeping all data vectors as column vectors, we then have that $\mathbf{T}_{ij}$ must be a left stochastic matrix. That is all columns of $\mathbf{T}_{ij}$ must sum to 1.[2]

[2] Technically, left stochastic matrices are squared. The matrix T could be indeed thought as square, by stacking sets I and J in both the row and column dimensions. As it makes no substantial difference to perform this stacking, I refer to the non-squared transformation matrix as stochastic.

This is an important property that derives from the fact we set up the transformation between classifications as a system that distributes items from j to i. So if we are distributing the 20 workers in β across the three classes in I, the sum of what goes to a, b, and c (from β) has to be 20.

In an alternative conceptualization of the transformation matrix, let's denote it with $\mathcal{T}_{ij}$, the matrix generates an average between the variables. Imagine, for instance, that instead of the vector of employment $\mathbf{x}_j$, we have a vector of wages $\mathbf{w}_j$. The operation we do to transform wages from classification j to classification i is

$$\mathbf{w}_i = \mathcal{T}_{ij}\mathbf{w}_j \tag{2}$$

Only this time $\mathcal{T}_{ij}$ is a right stochastic matrix: all the rows sum to 1. This is because each resulting wage of industry i must be a weighted average of the relevant industries of j, so the weights must sum to one. We highlight, however, that the case of computing weighted averages with $\mathcal{T}_{ij}$ can be reduced to the general case of $\mathbf{T}_{ij}$, for instance transforming employment $(\mathbf{T}_{ij}\mathbf{x}_j)$ and remuneration $(\mathbf{T}_{ij}\mathbf{r}_j)$, and only after computing average wages. For this reason, hereafter, I focus only on left stochastic matrices ($\mathbf{T}_{ij}$) and ignore right stochastic ones ($\mathcal{T}_{ij}$)

So the important question is how to get to $\mathbf{T}_{ij}$ from $\mathbf{C}_{ij}$

We can easily see that $\mathbf{C}_{ij}$ is a left stochastic matrix in the example. We can then simply define $\mathbf{T}_{ij} = \mathbf{C}_{ij}$ and compute

$$\mathbf{x}_i = \mathbf{T}_{ij}\mathbf{x}_j = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 20 \\ 5 \\ 15 \end{bmatrix} = \begin{bmatrix} 30 \\ 5 \\ 15 \end{bmatrix} \tag{3}$$

However, what happens if we are given $\mathbf{x}_i$ and we want to compute $\mathbf{x}_j$?

Since $\mathbf{C}'_{ij} \equiv \mathbf{C}_{ji}$ is not a left stochastic matrix, as it maps more than one j class to $i = a$, we cannot use it 'as-is' to define $\mathbf{T}_{ji}$

2.3 Concordance functions

I define here a **concordance function** as a function that takes a raw concordance matrix as inputs and gives, as output, a left stochastic matrix $\mathbf{T}_{ij}$.

$$\mathbf{T}_{ij} = f\left(\mathbf{C}_{ij}\right) \tag{4}$$

The simplest example of f is the following, which I call Equal Split (ES)

$$f^{ES}\left(\mathbf{C}_{ij}\right) = \frac{C_{ij}}{\sum_i C_{ij}} \tag{5}$$

Note that going from j to i, we get

$$\mathbf{T}_{ij} = f^{ES}(C_{ij}) = C_{ij} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

However going from i to j, we get

$$\mathbf{T}_{ji} = f^{ES}(C_{ji}) = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

See that, with this method, we introduce an error if we move data from i to j.

$$\mathbf{x}_j = \mathbf{T}_{ji}\mathbf{x}_i = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 30 \\ 5 \\ 15 \end{bmatrix} = \begin{bmatrix} 15 \\ 15 \\ 5 \\ 15 \end{bmatrix} \tag{8}$$

In many contexts the error can be quite severe. In the example of computing growth rates, for instance, it would be impossible to make sense of the growth rate of imputed industries. As an additional example, imagine, we calculate employment vectors for several US states and we measure the degree to which industries coagglomerate (as in **? ?** or **? ?**) by taking the correlation of industry employment across states. If we were to use the imputed vectors we would find perfect coagglomeration for all industries which have been split.

Sometimes, more sophisticated functions are possible. Let's say we are interested in employment in j, $\mathbf{x}_j$, and we have $\mathbf{x}_i$, $\mathbf{C}_{ij}$, and $\mathbf{y}_j$ – where the latter is a measure of output in j. We can use, then, a Weighted Split (WS) function

$$f^{WS}\left(\mathbf{C}_{ij}\right) = \frac{C_{ij}y_j}{\sum_j C_{ij}y_j} \tag{9}$$

With, for instance

$$\mathbf{y}_j = \begin{bmatrix} 25 \\ 75 \\ 20 \\ 40 \end{bmatrix} \tag{10}$$

we get

$$\mathbf{x}_j = \mathbf{T}_{ji}\mathbf{x}_i = \begin{bmatrix} 0.25 & 0 & 0 \\ 0.75 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 30 \\ 5 \\ 15 \end{bmatrix} = \begin{bmatrix} 7.5 \\ 22.5 \\ 5 \\ 15 \end{bmatrix} \tag{11}$$

which has improved on the previous outcome, but still has an error. Note that if one had the exact weights, it could be possible to use f^{WS} with no error. However, having the exact weight in most cases require knowing vector $\mathbf{x}_j$, thus not having a concordance problem to begin with.

In the next section we introduce an alternative conceptualization: we imagine concordance matrices as networks. This is a powerful abstraction that will be useful to develop a concordance function $f(C_{ij})$ that can do error-free transformations

3 Concordance tables as networks

We can think of concordance tables as networks or graphs. Let's take the stylized concordance table above C_{ij} and represent it. Recall that

$$C_{ij} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We are first going to create a network with networkx package. Node attributes such as industry name or classification type (i or j) are going to be attached to the networkx graph. Then, we are going to link the networkx graph to Cytoscape. (Cytoscape needs to be open for this notebook to work).

3.1 Networkx graph

The following code generates a network using the Python package networkx

In [18]:
```
"""
generate graph/network
"""

# empty graph/network
G = nx.Graph()

# add nodes
s1 = ["a","b","c"]
s2 = ["alpha","beta","gamma","delta"]
for i in s1:
    G.add_node(i)
for j in s2:
    G.add_node(j)

# add edges
G.add_edge("a","alpha",key=0)
G.add_edge("a","beta",key=0)
G.add_edge("b","gamma",key=0)
G.add_edge("c","delta",key=0)
```

In [19]:
```
# attributes
node_name = {}
node_type = {}
# node attributes
for k in s1:
    node_name[k]=k
    node_type[k]=1

for k in s2:
    node_name[k]=k
    node_type[k]=2
```

In [20]:
```
# assign attributes to networkx G
nx.set_node_attributes(G,node_name,"n_name")
nx.set_node_attributes(G,node_type,"n_type")
```

3.2 Cytoscape graph

Now that we have created the networkx graph – inclusive of node attributes – we can make Cytoscape dialog with this notebook. To do so it is necessary to have Cytoscape software open in the background. We can then run the following code

```
In [21]: # move network from networkx to cy
         cytoscape_network = cy.from_networkx(G)
```

```
In [22]: # Basic Setup
         PORT_NUMBER = 1234
         IP = 'localhost'
         BASE = 'http://' + IP + ':' + str(PORT_NUMBER) + '/v1/'
         HEADERS = {'Content-Type': 'application/json'}
         requests.delete(BASE + 'session')
```

```
Out[22]: <Response [200]>
```

```
In [23]: res1 = requests.post(BASE + 'networks', data=json.dumps(cytoscape_network), headers=HEA
         res1_dict = res1.json()
         new_suid = res1_dict['networkSUID']
         requests.get(BASE + 'apply/layouts/force-directed/' + str(new_suid))
```

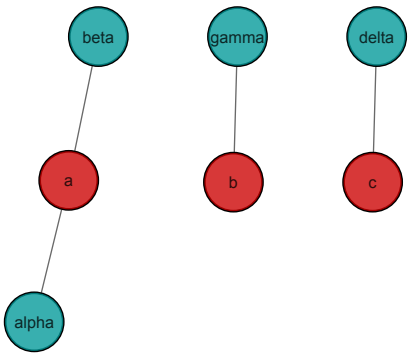
```
Out[23]: <Response [200]>
```

```
In [24]: style_name = 'Basic_Style'
```

[...]

```
         # Display
         Image(BASE+'networks/' + str(new_suid) + '/views/first.png')
```

```
Out[24]:
```



We observe that (as we have seen in the previous section) b maps 1:1 to γ, and c maps 1:1 to δ. However a maps to both α and β.

3.3 Example: Naics concordances

For simple mapping, such as this one, drawing a network does not add much to the understanding that we get by looking at the concordance table. But for more complex mapping, the visualization is useful. Let's go back to our motivating example to compute industry growth rate in the US economy between 2005 and 2010.

```
In [25]: # import concordance and transform it to 5-digit
         input_file = "2002_to_2007_NAICS.xls"
```

[...]

```
         C.head()
```

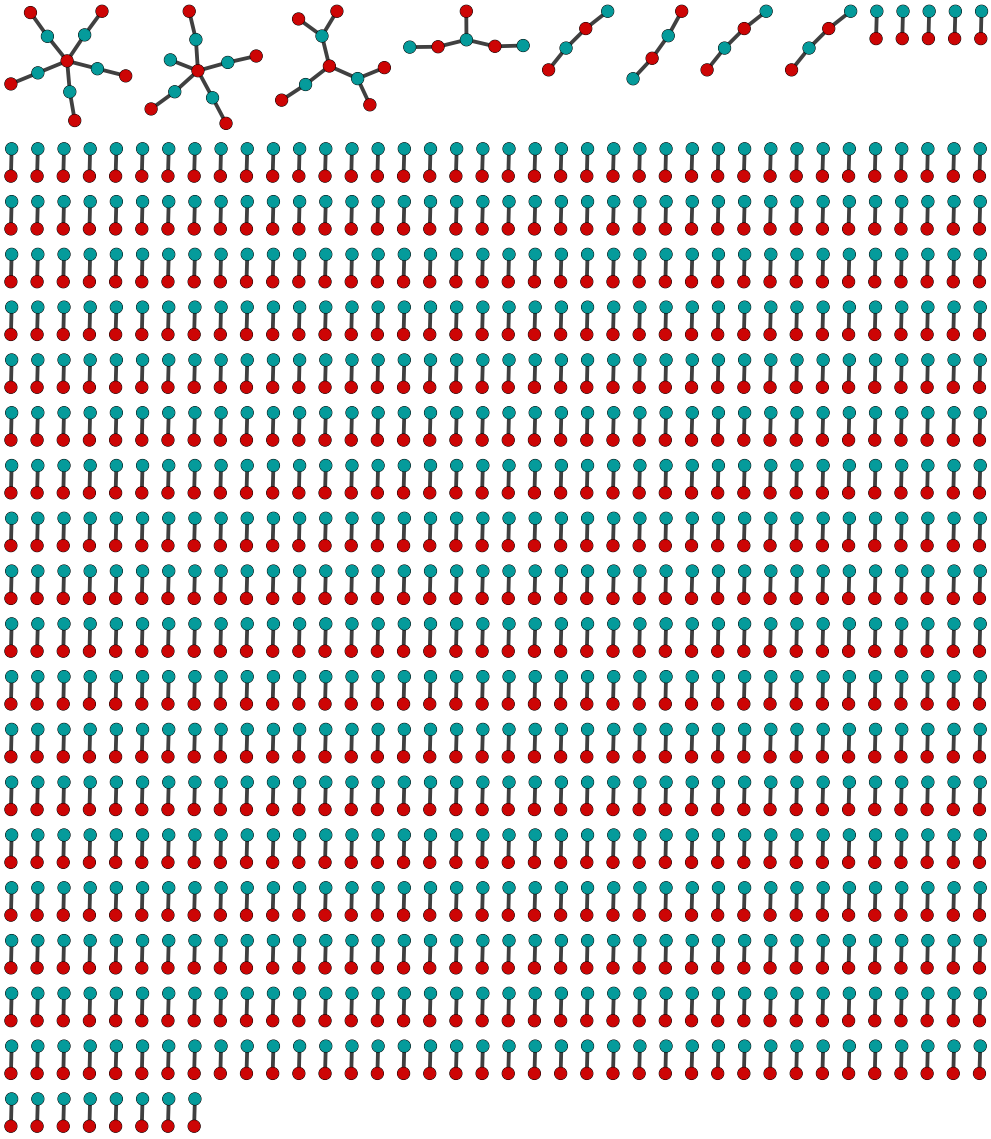
```
Out[25]:    naics2002  naics2007                         name2002  \
         0     111111     211111                  Soybean Farming
         1     111112     211112  Oilseed (except Soybean) Farming
         2     111113     211113          Dry Pea and Bean Farming
         3     111114     211114                    Wheat Farming
         4     111115     211115                     Corn Farming

                              name2007
         0                  Soybean Farming
         1  Oilseed (except Soybean) Farming
         2          Dry Pea and Bean Farming
         3                    Wheat Farming
         4                     Corn Farming
```

[...]

```
In [27]: """
         generate and display graph/network here
         """
```

[...]

```
   Out[35]:
```



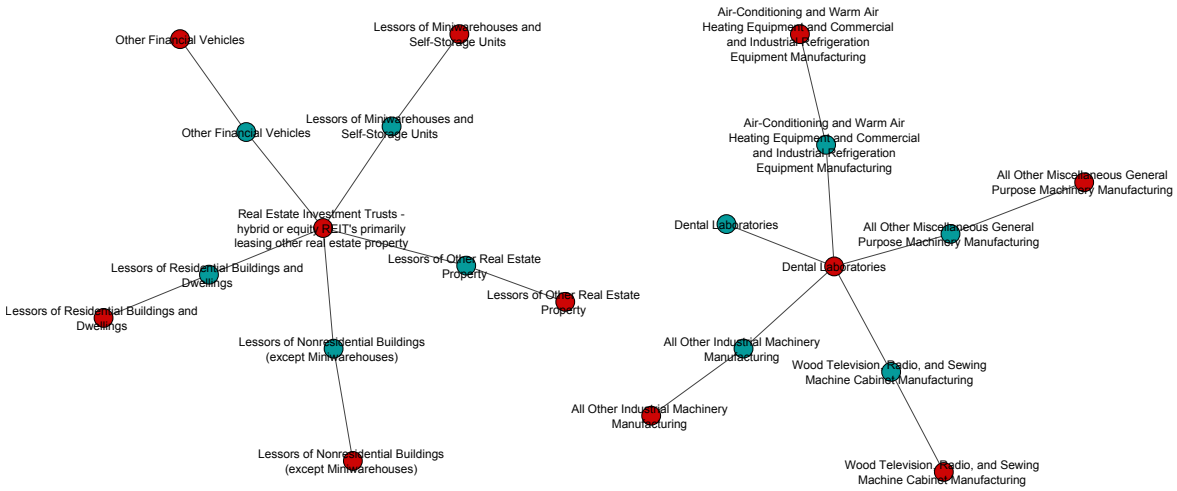
We can see from the network above that most Naics industries have a one-to-one mapping between 2002 and 2007, however some industries have a complex many-to-many concordance.

Note that the image rendered above can also be explored in Cytoscape. One can zoom-in, move nodes or entire components. The link between this notebook and Cytoscape is two-way. Once we are satisfied with the view, we can re-run the Python code to get an updated image. See here below, a zoomed view on the first two components.

```
In [36]: style_name = 'Basic_Style'
```

[...]

```
Out[36]:
```



Note that the many-to-many mapping we observe in the network implies that neither $\mathbf{C}_{ij}$ nor $\mathbf{C}_{ji}$ are left stochastic matrices. We need to use a concordance function to obtain a transformation table, such as $\mathbf{T}_{ij}$

$$\mathbf{T}_{ij} = f\left(\mathbf{C}_{ij}\right) \tag{12}$$

In the next section, we show a network-based concordance function f that, unlike many alternative options (such as the ones proposed in section 2: f^{ES} and f^{WS}), does not introduce any error.

4 Main methodology

4.1 An error-free solution to the many-to-many problem

The principle behind the method is rather simple: we are going to define a new, fictional, target classification, for each of the connected components in the network. So for instance, in the case of Naics 2002-2007, depicted in the Cytoscape visualization above we have

```
In [40]: nc = nx.number_connected_components(G)
         nc
```

```
Out[40]: 705
```

The new classification will have 705 industry classes. Each of the Naics 2002 and each of the Naics 2007 will converge towards one (and one only) of these 705 new classes.

With Naics 2002 denoted with i, Naics 2007 with j, and the new classification with k. The concordance function is in this case

$$f^{CC}\left(\mathbf{C}_{ij}\right) = \begin{bmatrix} T_{ik} \\ T_{jk} \end{bmatrix} \tag{13}$$

That is, in this case, the concordance function (CC for connected components) has two outputs: one transformation matrix for i and one for j. In practical terms we need to identify which of the Naics 2002 and 2007 belong to which component. The following code does that, returning the two transformation matrices T_{ik} and T_{jk}

```
In [41]: # identify connected components
         list_comp=sorted(nx.connected_components(G), key = len, reverse=True)

         # number of digit in classification =5
         nd = 5

In [42]: #  concordance function
         C123=pd.DataFrame(columns= ["code","code_3","s"])

         i = 0
         for c in range(nc):
             code_3 =c+1
             for n in list_comp[c]:
                 C123.loc[i] = 0
                 C123.loc[i]["code"] = n
                 C123.loc[i]["code_3"] = code_3
                 C123.loc[i]["s"] = np.floor(n / (10**(nd) ) )
                 i += 1

         T13=C123.loc[C123["s"]== 1]
         T23=C123.loc[C123["s"]== 2]

         T13.columns=["code_1","code_3","s"]
         T23.columns=["code_2","code_3","s"]

         T13["code_1"] -= 10**(nd)
         T23["code_2"] -= 2*(10**(nd))
```

We now have 2 Transformation tables T13 and T23 (T_{ik} and T_{jk} in the general notation). They both converge to a 3rd new-virtual classification. Let us see for instance T23 (T_{jk}).

In [43]:

```
T23.head(20)
```

Out[43]:

	code_2	code_3	s
1	52599	1	2
7	53111	1	2
8	53112	1	2
9	53113	1	2
10	53119	1	2
11	33911	2	2
13	33712	2	2
15	33329	2	2
17	33399	2	2
20	33341	2	2
24	51913	3	2
25	51791	3	2
29	51711	3	2
31	32619	4	2
33	32629	4	2
34	33661	4	2
36	11251	5	2
38	11199	5	2
40	31521	6	2
42	31499	6	2

Data transformation Let us now transform the data and compute the growth rates

In [44]:
```
T13.rename(columns={'code_1': 'naics'},inplace=True)
T23.rename(columns={'code_2': 'naics'},inplace=True)
```

In [45]:
```
# compute transformation Tx for 2005 data
merged_05 = data05.merge(T13,on='naics')
transf_05 = merged_05.groupby('code_3')['emp'].sum()
```

In [46]:
```
# compute transformation Tx for 2005 data
merged_10 = data10.merge(T23,on='naics')
transf_10 = merged_10.groupby('code_3')['emp'].sum()
```

In [47]:
```
# compute growth rate of code_3
g_all = (transf_10 - transf_05)/transf_05
g_all
```

Out[47]:
```
code_3
3     -0.117559
439   -0.303335
```

```
440   -0.153680
441   -0.024463
442   -0.272659
443   -0.184573
444    0.112382
445   -0.009126
446    0.688269
447    0.004478
448    0.030424
449   -0.238538
450    0.070963
451    0.223861
452    0.088723
453   -0.149466
454   -0.151867
455   -0.060747
456    0.060797
457    0.011361
458   -0.469858
459    0.013437
460   -0.312108
461   -0.080038
462   -0.205233
Name: emp, dtype: float64
```

The above growth rates have no error. In particular code_3[3] is the third largest component in the concordance network and is the result of the aggregation of sevaral industries (6 from 2005, 3 from 2010).

The remaining codes, code_3[439] to code_3[462], are the 439th (to 462nd) largest connected components. In this particular case they all are made of only 2 nodes (1 from 2005, 1 from 2010)

Use of functions such as f^{ES} and f^{WS} would create an error for all industries belonging to code_3[3], while they would be accurate for code_3[439] to code_3[462].

As this new coding numbering has no correspondance with the original coding, one needs to design a system to retrive what a code is. The following Python function is a handy option

```
In [48]: # what is code_3[k]?
         def show_code(k):
             a = T13[T13['code_3'] ==k]
             b = T23[T23['code_3'] ==k]
             frames = [a,b]
             show_codek = pd.concat(frames)
             return show_codek

In [49]: show_code(439)

Out[49]:      naics code_3  s
         913  51111    439  1
         912  51111    439  2
```

```
In [50]: show_code(3)
```

```
Out[50]:     naics code_3  s
         21  51711      3  1
         22  51811      3  1
         23  51751      3  1
         26  51791      3  1
         27  51731      3  1
         28  51611      3  1
         24  51913      3  2
         25  51791      3  2
         29  51711      3  2
```

Note that column s indicates if the naics code is from 2005 (1) or 2010 (2)

5 Concatenated concordances

Sometimes a researcher might need to concatenate concordances. For instance there might be no concordance tables from SITC to Naics. But one could go from SITC to ISIC and from ISIC to Naics. Or more simply one might want to follow a Naics industry over time, but Naics gets redifined every five years.

If one avails transformation matrices for every step, the problem is simple. We can in fact concatenate transformation matrices

$$\mathbf{x}_z = \mathbf{T}_{zq}\mathbf{T}_{qk}\mathbf{T}_{kj}\mathbf{T}_{ji}\mathbf{x}_i \tag{14}$$

However, if the transformation matrix is obtained with error (using for instance an Equal Split concordance function on a many-to-many concordance table), we would accumulate errors at each step.

The network-based method proposed in this workbook can accomodate the use of concatenated concordances: we just need to consider all the different raw concordance tables (e.g. $\mathbf{C}_{zq}, \mathbf{C}_{qk}, \mathbf{C}_{kj}, \mathbf{C}_{ji}$) as part of the same network.

As a working example we use the concodances of NAICS for the following changes

https://www.census.gov/eos/www/naics/concordances/concordances.html

Concordance file: 2002_to_2007_NAICS.xls
Concordance file: 2007_to_2012_NAICS.xls
(The files are already included, the online source is for reference)

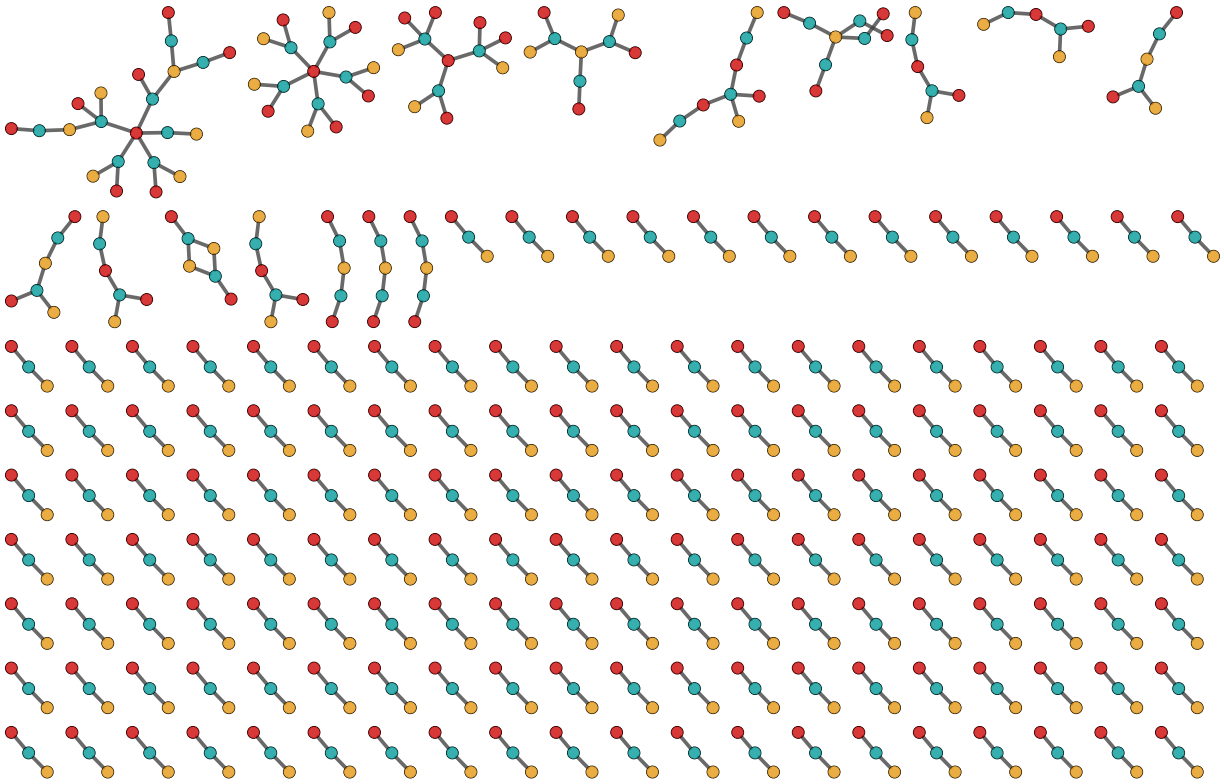
```
In [51]: # import concordances
```

[...]

In [54]:
```
"""
generate graph/network here
"""

# empty graph/network
G = nx.Graph()

# add edges
for k in C_12.index:
    i = tuple(C_12.iloc[[k]]["naics2002"].values)[0]
    j = tuple(C_12.iloc[[k]]["naics2007"].values)[0]
    G.add_edge(int(i),int(j))

for k in C_23.index:
    i = tuple(C_23.iloc[[k]]["naics2007"].values)[0]
    j = tuple(C_23.iloc[[k]]["naics2012"].values)[0]
    G.add_edge(int(i),int(j))
```

[...]

Out[63]:



With three codes, the same concepts apply. First, many codes have unique mapping: whenever we see a component with three nodes, all of differnt colors, it means that the industry has one-to-one mapping over time (actually in the concordance there are more one-to-one components than depicted; they are omitted for visual clarity).

We now proceed as before, only that our output now consists of three transformation matrices

With Naics 2002 denoted with i, Naics 2007 with j, Naics 2012 with h, and the new classification with k. The concordance function is in this case

$$f^{CC}\left(\mathbf{C}_{ij}, \mathbf{C}_{jh}\right) = \begin{bmatrix} T_{ik} \\ T_{jk} \\ T_{hk} \end{bmatrix} \tag{15}$$

```
In [64]: # identify connected components
         list_comp=sorted(nx.connected_components(G), key = len, reverse=True)

         # number of digit in classification =5
         nd = 5

In [65]: nc = nx.number_connected_components(G)
         nc

Out[65]: 691

In [66]: #  concordance function
         C123N=pd.DataFrame(columns= ["code","code_N","s"])

         i = 0
         for c in range(nc):
             code_N =c+1
             for n in list_comp[c]:
                 C123N.loc[i] = 0
                 C123N.loc[i]["code"] = n
                 C123N.loc[i]["code_N"] = code_N
                 C123N.loc[i]["s"] = np.floor(n / (10**(nd) ) )
                 i += 1

         T1N=C123N.loc[C123N["s"]== 1]
         T2N=C123N.loc[C123N["s"]== 2]
         T3N=C123N.loc[C123N["s"]== 3]

         T1N.columns=["code_1","code_N","s"]
         T2N.columns=["code_2","code_N","s"]
         T3N.columns=["code_3","code_N","s"]


         T1N["code_1"] -= 10**(nd)
         T2N["code_2"] -= 2*(10**(nd))
         T3N["code_3"] -= 3*(10**(nd))
```

```
In [67]: # export to csv final output
         T1N.to_csv("concordance_naics2002.txt", index=False)
         T2N.to_csv("concordance_naics2007.txt", index=False)
         T3N.to_csv("concordance_naics2012.txt", index=False)
```

While in this example we presented a case of two concordances across three classifications, there is no limit to the number of raw concordances that can be concatenated to the network: the method works with an arbitrary number of classifications.

6 The complex case in a many-to-many mapping

6.1 When the network has a large connected component

Let's consider a case where the concordance network has one large connected component. It may not be the only component, but let us say that it dominates the network significantly. In this case the method outlined this far is not suited to harmonize the classifications, as we would only have one (or very few) final classification.

We can however apply a modification of this method – still based on network analysis – that can provide an elegant solution, even though it is not error free.

The core idea behind the modified method is to identify the clusters (communities, more precisely) in the concordance network, and assign a new "target" classification to each of the clusters.

Let us take a real example, where this might become useful. **?** provide a classification to move from the Harmonized System (HS) to Naics (PS, henceforth).

Their table is 10-digit HS to 6-digit Naics and, being many-to-1, requires no modifition to be used (it is already **transformation matrix** "as is")

We verify this hereafter

```
In [68]: # import PS concordance

         input_file = "PS_hs10_naics6.txt"
         PS=pd.read_csv(input_file)
         PS.head()
```

Out[68]:

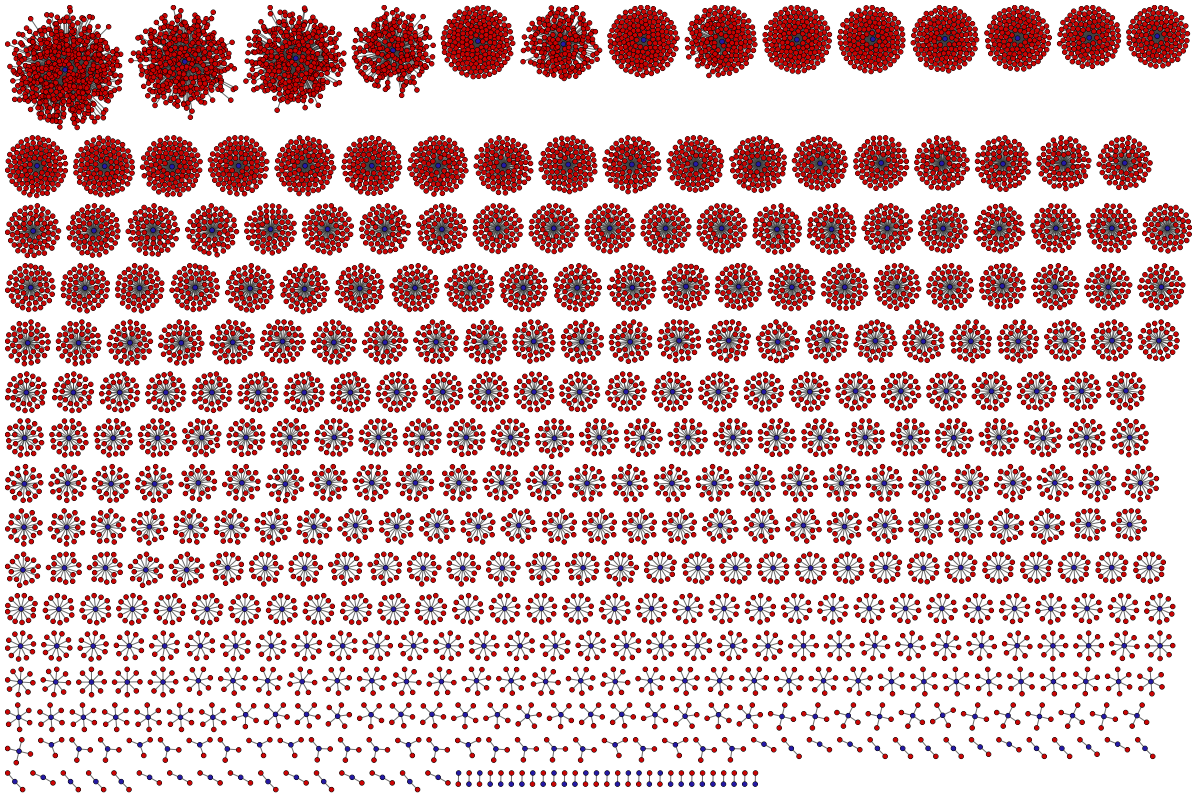
	hs	naics
0	101100000	112920
1	101110000	112920
2	101190000	112920
3	101200000	112920
4	101901000	112920

```
In [69]: """
         generate graph/network here
         """
```

[...]

One can explore in Cytoscape all the different components to check that indeed multiple HS codes converge to 1 and 1 only NAICS code (in blue at the center of each component). So far, there is no problem.

Out[76]:



What if then we are in need of matching data at a different level of aggregation? This is in fact a problem incurred in **?**, where 6-digit COMTRADE data needed to be matched with information from BLS recorded according to 4-digit Naics.

We quickly notice that things get more complicated.

In [77]:
```
# import PS concordance, trunkated at hs6 and naics4

input_file = "PS_hs6_naics4.txt"
PS2=pd.read_csv(input_file)
PS2.head()
```

Out[77]:

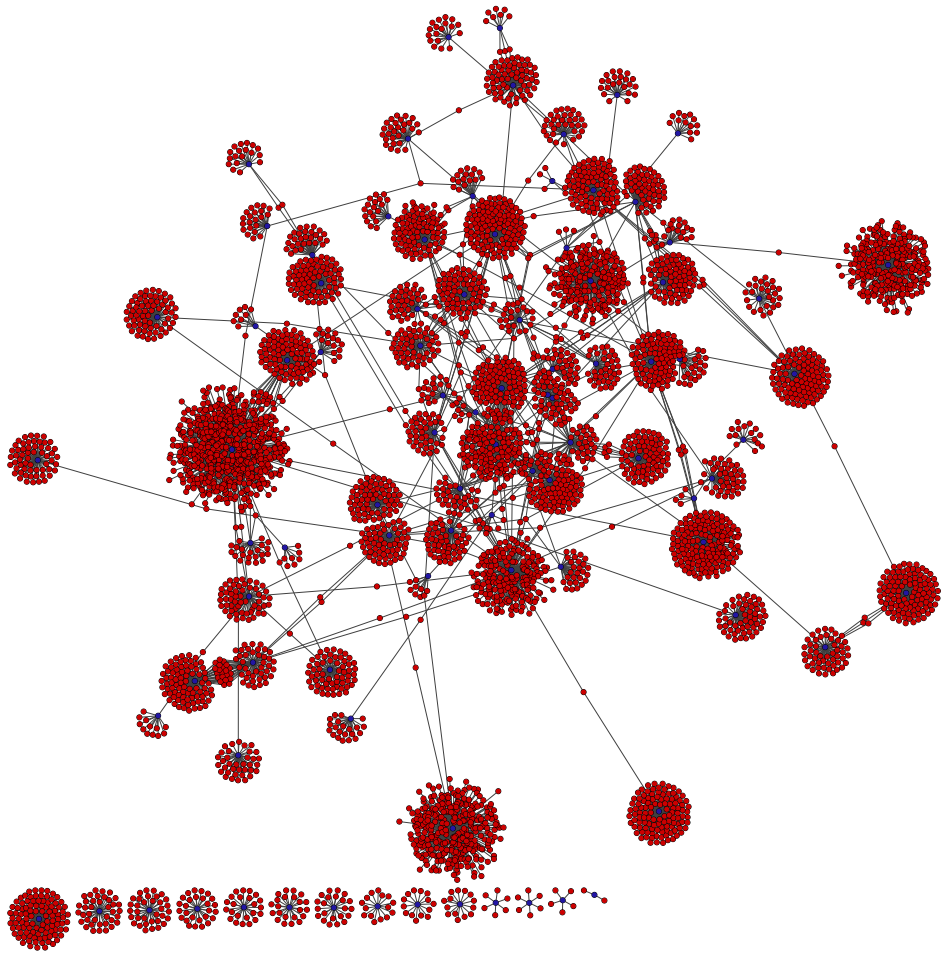
	hs	naics
0	440334	1133
1	270900	2111
2	270111	2121
3	261210	2122
4	250621	2123

[...]

```
In [79]: """
         generate graph/network here
         """
```

[...]

Out[86]:



While a handful of components are small isolates, the majority of the concordance network is made of one large component, which is kept together only by few bridging nodes.

One solution is to use Cytoscape to identify this bridges and disconnect the components by hand. A quicker way to obtain the same result is to run a community discovery algorithm.

In this example I use 'label propagation' algorithm, but several options are possible.

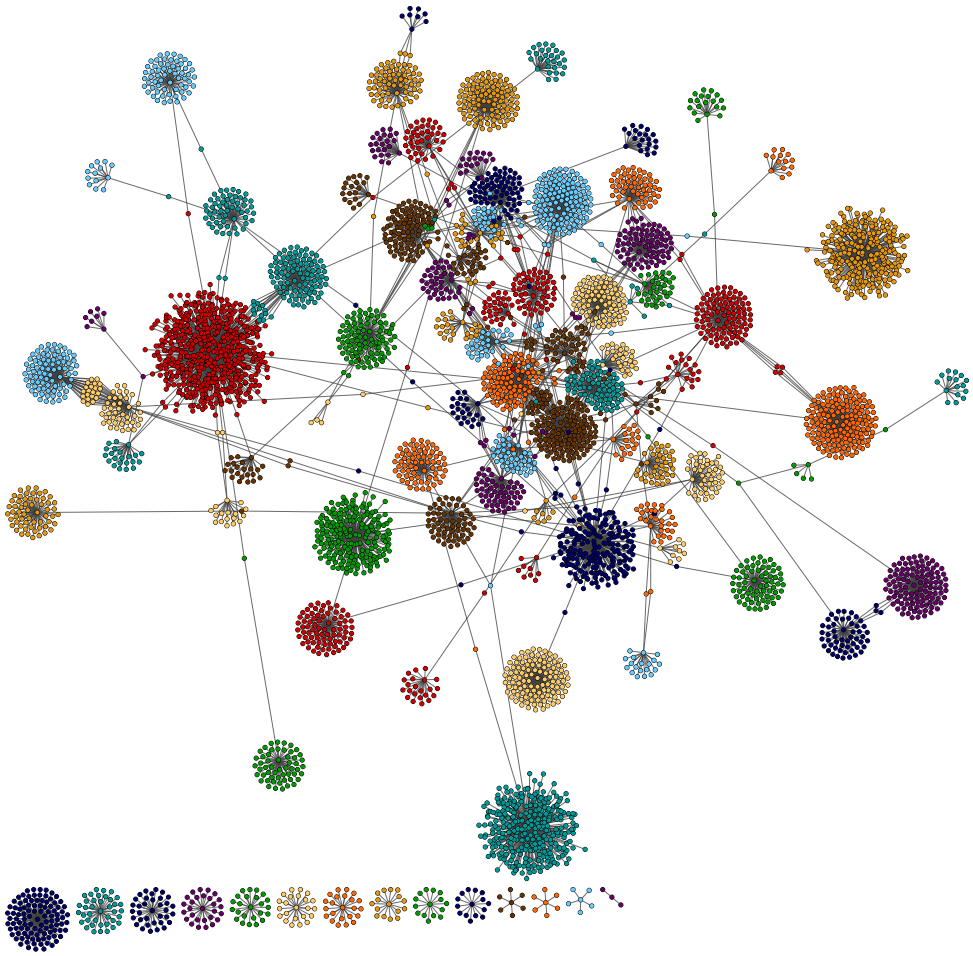
```
In [87]: from networkx.algorithms import community

In [88]: C2 = community.label_propagation_communities(G)
```

```
In [89]: list_comp2=sorted(C2, key = len, reverse=True)
         nc2 = len(list_comp2)
         nc2

Out[89]: 90

In [90]: # final concordance tables generation
         C123=pd.DataFrame(columns= ["code","code_3","s"])

         i = 0
         for c in range(nc2):
             code_3 =c+1
             for n in list_comp2[c]:
                 C123.loc[i] = 0
                 C123.loc[i]["code"] = n
                 C123.loc[i]["code_3"] = code_3
                 C123.loc[i]["s"] = np.floor(n / (10**(6) ) )
                 i += 1

         T13=C123.loc[C123["s"]== 1]
         T23=C123.loc[C123["s"]== 2]

         T13.columns=["hs","new_code","s"]
         T23.columns=["naics","new_code","s"]

In [91]: # node attributes
         node_type = T13.set_index("hs").new_code.to_dict()
         node_type.update(T23.set_index("naics").new_code.to_dict() )
         for k in node_type:
             node_type[k]=int(node_type[k])

In [92]: T13['hs']-=1000000
         T23['naics']-=2000000

In [93]: T13.head()

Out[93]:       hs new_code  s
         0  290241        1  1
         1  290242        1  1
         2  290243        1  1
         3  290250        1  1
         4  284110        1  1

In [94]: T23.head()

Out[94]:     naics new_code  s
         115   3251        1  2
         699   3132        2  2
         1060  3311        3  2
         1323  3399        4  2
         1423  3152        5  2
```

```
In [95]: # assign attributes to networkx G
         nx.set_node_attributes(G,node_type,"n_type")

In [96]: # move network from networkx to cy
         cytoscape_network = cy.from_networkx(G)
```

[...]

Out[99]:



In the figure above, 10 colors are used to highlight the 90 clusters identified by the community discovery algorithm. Visual inspection confirms that the algorithm correctly severs weak ties that were connecting otherwise separate components.

Notice that the network has 15 connected components. This means that using the community discovery algorithm buys us an additional 75 industry classes. This operation can be written as

$$f^{CD}\left(\mathbf{C}_{ij}\right) = \left[\begin{array}{c} T_{ik} \\ T_{jk} \end{array}\right] \tag{16}$$

with CD meaning community discovery, i indicating 4-digit Naics, j 6-digit HS and k the new classification.

7 A compact program

Working with a Jupyter notebook has the advantage of allowing a careful exploration of the network, working in parallel with Cytoscape.

However some might find the process a little laborious, especially if one is working on a project the requires several harmonizations. A program that takes the raw concordance table as input and returns as output the required transformation matrices, without any further input from the researcher, is just what it is needed in this case.

Such program also has the advantage of requiring many less packages and no connection to Cytoscape, meaning quicker set up time for first time user.

Attached to the download bundle of concordances, one can find the file

```
concordance_network.py
```

Basic use The file is a very streamlined version of this notebook, and it can accommodate the main methodology, as well as the extensions. In essence, one just need to run the program in a shell

```
python concordance_network.py concordance.txt
```

where concordance.txt is the raw concordance table input (C_{ij}), provide by the user in comma separated, edge-list form. The program gives two outputs: Transform_1.txt and Transform2.txt, which are T_{ik} and T_{jk}, the two transformation tables.

Multiple concatenated concordances If there is the need to harmonize more than one concordance, simply type all concordance files as arguments

```
python concordance_network.py concordance1.txt concordance2.txt... concordanceN.txt
```

Community discovery algorithm Finally, the program creates the new classes from connected components as default. This is the recommended option, as one has a full cognition of what is happing and it is fully error-free. However, as shown in the previous section, sometime it makes sense to run a community discovery algorithm instead. After being run, the program will ask:

```
"Default option finds connected components.
 Use community discovery algorthm, instead? Type y/n: "
```

Type "n" to carry on with default option, or "y" to run a community discovery algorithm.

Please find more details on the use of the program in the headings of concordance_network.py file itself

Discussion and Conclusions

Many-to-many raw concordance tables can force the researcher to introduce unwanted errors when transforming data. The network-based approach proposed in this notebook allows for an error-free transformation of data.

More specifically, the use of network components detection algorithm makes the procedure entirely error free. This is therefore the recommended methodology to harmonize data.

The downside of this method is that in some cases the raw concordance network might not have many components. This usually happens when a number (sometime a small number) of classes gets a drastic re-assigment. A clear case of this happens when a code is reassigned outside, for instance, its 2-digit class. This code would then act as a bridge between two otherwise disconnected components.

One option is two visually check each of these bridging nodes, choose to which components the node is most related to, and then manually remove the unwanted tie. While we do not dwell on this option here, this notebook provides the perfect tool to organize a workflow to do so. In fact one can load the raw concordance network here and launch it in Cytoscape. With node colors reflecting classifications (e.g. naics2002, red; naics2007, blue; naics2012, yellow) and node labels reflect the name of the class (e.g. "dental laboratories"), the researcher can use Cytoscape the make sense of this bridging reclassification. Once a choice to sever a tie is made, it is then easy to drop the corresponding row from the Pandas table of raw concordance or an edge from a networkx graph. After removing the unwanted ties, one can re-run the network components detection algorithm and get many more classes compared to before.

A second option is to let the computer do that. In this notebook, we show how a community discovery algorithm works well to identify bridging nodes and split a weakly connected component into two separate clusters. The obvious advantage of this procedure is that it runs in matter of seconds. Compared to manually check hundreds of individual edges and nodes, the procedure is very appealing when the network structure appears as a complex web of links. However, one needs to be careful as the algorithm may remove a bridge in a different way than a human would. Unlike in the case network components detection, where the outcome is predictable, with community discovery it is not obvious that the result will be as desired. In an extreme case, it is possible that the algorithm identifies a cluster that does not contain any node from a particular class (for instance a community that has codes for naics2002 and naics2007, but not for naics2012).

For these reason, I recommend that the use community discovery only in conjunction with additional checks. In the Naics-HS concordance problem in **?**, for instance, we use community discovery, but we manually check for issues, and validate as well the results with an alternative method.

References

Diodato, Dario, Frank Neffke, and Neave O'Clery (2018), "Why do industries coagglomerate? how marshallian externalities differ by industry and have evolved over time." *Journal of Urban Economics*.

Diodato, Dario and Ulrich Schetter (2018), "Pathways to prosperity: Economic development with a network of industries." Mimeo.

Ellison, Glenn, Edward L Glaeser, and William R Kerr (2010), "What causes industry agglomeration? evidence from coagglomeration patterns." *American Economic Review*, 100, 1195–1213.

Pierce, Justin and Peter Schott (2009), "Concording US harmonized system categories to US SIC and NAICS industries." NBER Working Paper 15548, National Bureau of Economic Research.